



# SOFTWARE ENGINEERING

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики  
Кафедра технологий программирования

*Software*

*Engineering*

Минск 2015

Над книгой работали:

- Белый Антон
- Борисевич Павел
- Гетьман Святослав
- Григорьев Антон
- Грушевский Андрей
- Ипатов Алексей
- Лебедев Николай
- Михальцова Анна
- Ровдо Дарья
- Трубач Геннадий
- Щавровский Святослав
- Ярошевич Яна

## Содержание

<b>Глава 1. IT-проект .....</b>	5
IT-проект и его структура .....	5
Жизненный цикл IT-проекта .....	14
Моделирование жизненного цикла .....	17
Мозговой штурм .....	24
Управление жизненным циклом .....	29
Классификация программного обеспечения .....	30
<b>Глава 2. Менеджмент IT-проекта.....</b>	41
Менеджмент IT-проекта .....	41
Команда менеджмента проекта .....	59
Роли в команде и их функции .....	71
Управление IT-проектами .....	80
Совместная работа в проектировании .....	100
Дистанционное сотрудничество .....	105
Стратегии и история развития крупных IT-компаний.....	108
<b>Глава 3. Архитектура компьютера и мобильных устройств .....</b>	149
Архитектура компьютера .....	149
Операционные и вычислительные системы.....	154
Мобильные операционные системы.....	157
Энергосбережение .....	166
<b>Глава 4. Языки программирования .....</b>	188
Языки программирования и их краткий обзор.....	188
JavaScript.....	195
Трансляторы.....	213
<b>Глава 5. Базы данных .....</b>	224
Реляционные базы данных. SQL.....	224
Нереляционные базы данных .....	228
Обработка больших данных. Подходы к формированию больших данных.....	236
<b>Глава 6. Разработка программного обеспечения .....</b>	244
Разница между разработкой и производством программного обеспечения .....	244
Парадигмы программирования .....	264
Стили программирования.....	271

Clean code .....	279
Рефакторинг .....	281
Паттерны проектирования.....	286
Разработка мобильных приложений.....	293
Тестирование .....	300
Стандартизация .....	303
<b>Глава 7. Инновационные концепции и технологии .....</b>	<b>306</b>
Искусственный интеллект.....	306
Виртуальная реальность .....	311
Искусственные нейронные сети.....	323
Роботы и роботизация человека.....	345
Интернет вещей .....	369
Облачные вычисления .....	373
3D печать .....	378
<b>Глава 8. Компьютерная и информационная безопасность .....</b>	<b>388</b>
Компьютерная и информационная безопасность.....	388
Антивирусы .....	406
<b>Приложение .....</b>	<b>415</b>
Список вопросов, возникнувших в ходе обсуждения .....	415

# Глава 1. ИТ-проект

## ИТ-проект и его структура

### Структура проекта

**Проект** – это временное действие, которое выполняется для создания уникального продукта или услуги. Временное обозначает, что каждый проект имеет свои определенные начало и конец. Уникальный обозначает, что продукт или услуга принципиально отличается от других аналогичных продуктов или услуг, так как принципиально отличаются условия создания этих продукта или услуги в каждом проекте.

**Участниками** проекта являются:

- собственник, заказчик, инвестор;
- менеджеры проекта;
- исполнители работ проекта;
- «окружающая» организация;
- службы контроля (технического, финансового и пр.)
- финансирующие организации, банки.

Несмотря на все многообразие существующих проектов, в команде можно выделить ряд более или менее стандартных ролей.

В первую очередь, это *менеджер (руководитель) проекта* — физическое лицо, несущее личную ответственность за успех проекта и осуществляющее оперативное руководство.

Как правило, в компаниях назначают *куратора проекта* — представителя высшего руководства, который хоть и не вникает в тонкости текущего положения дел в проекте, но контролирует его ход, следит, чтобы проект соответствовал стратегическим целям компании, а если у менеджера проекта не хватает полномочий, — помогает ему своим авторитетом.

*Проектный комитет* создается в компаниях, в которых бизнес построен по проектному типу. Это орган, задачи которого — отбирать проекты и контролировать их выполнение на высшем уровне, принимать ключевые решения.

В технически сложных проектах важна роль *главного инженера проекта (ГИП, командный лидер)*, который порой по статусу равен менеджеру проекта.

В крупных проектах могут выделяться *менеджеры по различным функциональным областям*, например, по управлению финансами, персоналом, рисками и т. п.

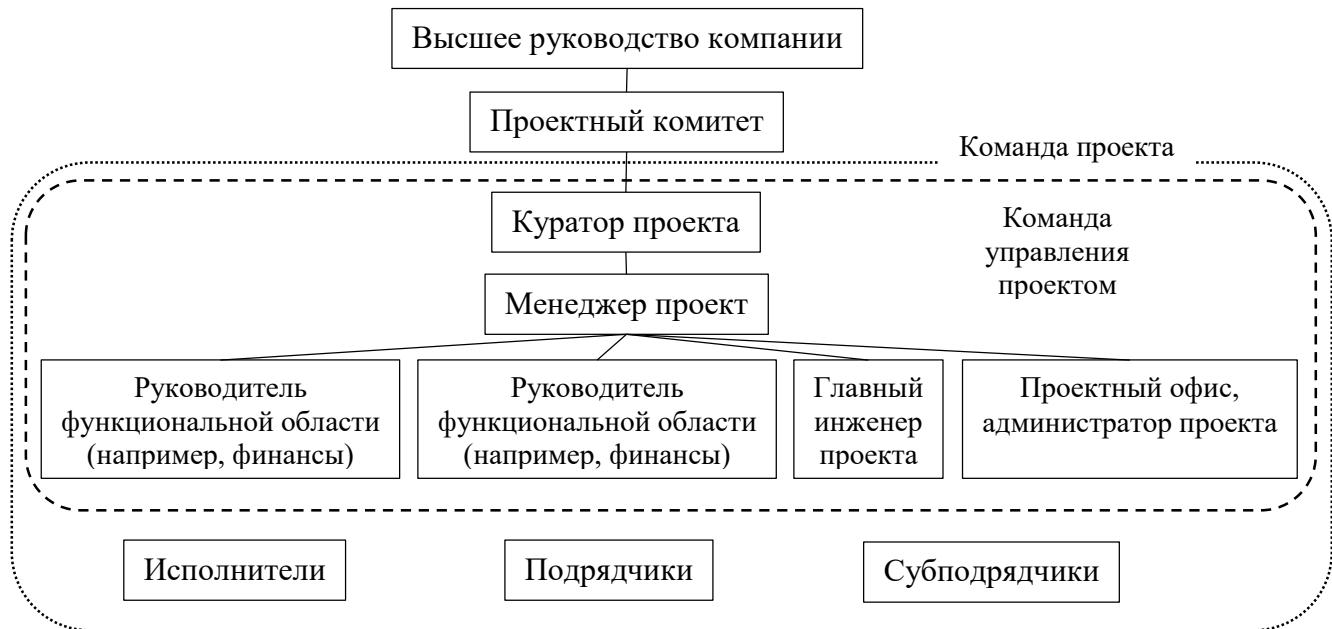
Все вышеперечисленные роли образуют *команду управления проектом*, которая входит в *команду проекта*. Также участниками команды проекта являются *исполнители* как из числа штатных сотрудников компании, так и нанятые специально для реализации конкретного проекта. Иногда в нее включают *подрядчиков* и *субподрядчиков*.

Отдельно стоит выделить *проектный офис*. В простейшем случае это своего рода секретариат, в котором хранится вся документация по проекту. Он может состоять как из одного, так и из

нескольких сотрудников. В более продвинутых компаниях *проектный офис* также играет роль методологического центра, обслуживающего все проекты организации.

Основная команда формируется исходя из потребностей проекта. Должны ли сотрудники:

- работать полный или неполный рабочий день?
- отчитываться менеджеру проекта?
- тратить на проект 100% своего рабочего времени?



## Эффективная деятельность

Показатели эффективной деятельности команды:

- Ясное понимание цели проекта и нацеленность на конечный результат;
- Четкое распределение функций и ответственности;
- Наличие плана развития команды;
- Командная солидарность;
- Взаимопонимание и бесконфликтность;
- Посещаемость рабочих совещаний и активное участие в решении проблем.

Для обеспечения эффективного руководства командой проектный менеджер должен:

- Определить организационную структуру команды, подобрать ее состав, распределить функции и обязанности;
- Назначить руководителей и ответственных за отдельные направления;
- Своевременно спланировать, распределить и скоординировать работу;
- Четко объяснить цели и задачи;
- Преодолевать препятствия и избегать конфликтов;

- Обеспечить трудовую активность команды силой личного авторитета, заинтересовать каждого члена команды, оказывать им помощь и проявлять участие, поддержать перспективу команды;
- Привлекать всех к решению задач;
- Обеспечить поддержку проекта со стороны руководства и регулирование отношений с окружением команды, создавать привлекательный имидж команды.

**Менеджер проекта** – лицо, отвечающее за успех проекта, а также за подбор и работу своей команды и завершение проекта. Это происходит в рамках ограничений, наложенных партнерскими и другими организациями, внешними по отношению к проектной команде.

Менеджер проекта должен быть назначен как можно раньше. Обычно менеджера проекта назначает **Заказчик**.

Искусство управления человеческими ресурсами и координация их в проекте реализуется менеджером посредством применения административных и поведенческих знаний для достижения определенных проектных целей в содержании, затратах, времени, качестве и удовлетворении участников проекта.

Менеджер проекта является ключевой фигурой в команде проекта. От его лидерских качеств, организационных способностей, харизмы, умения вовремя принимать решения и сглаживать конфликты напрямую зависит успех проекта.

#### Определение термина ИТ-проект

Термин "**ИТ-проект**" обычно используется для обозначения деятельности, связанной с использованием или созданием некоторой информационной технологии. Это приводит к тому, что ИТ-проекты охватывают очень разнообразные сферы деятельности: разработку программных приложений, создание информационных систем, развертывание ИТ-инфраструктуры и пр.

С одной стороны, эти работы соответствуют классическому определению проекта "Проект – это комплекс усилий, предпринимаемых с целью получения конкретных уникальных результатов в рамках отведенного времени и в пределах утвержденного бюджета, который выделяется на оплату ресурсов, используемых или потребляемых в ходе проекта". С другой стороны, они обладают известными отличительными особенностями:

- разделение на уровне идеологии заказчика и исполнителя: заказчиком, как правило, является бизнес, а исполнителем – ИТ-специалисты, и есть трудности в выявлении требований, ожиданий от проекта, в формировании технического задания. Существует также проблема эффективных коммуникаций;
- ответственность за результат проекта имеет "солидарный" характер. То есть здесь нельзя возложить ответственность за успех проекта только на исполнителя, точно так же, как нельзя говорить, что исключительно заказчик виновен в том, что проект не удался. В ИТ-проекте должны создаваться определенные условия для взаимодействия сторон, и стороны, участвующие в нем, несут равную ответственность за результаты проекта;
- зачастую реализация ИТ-проекта предусматривает изменение существующих организационных структур на предприятии;

- обычно в ИТ-проект вовлечено множество подразделений организации;
- существует высокая вероятность конфликтов между руководителем проекта, высшим руководством, руководителями подразделений и персоналом организации;
- многие ИТ-проекты имеют колоссальные бюджеты. В крупных компаниях масштабы проектной деятельности в области информационных технологий (ИТ) измеряются миллионами долларов, причем реализация новых проектов происходит постоянно. Если, например, промышленное предприятие достаточно один раз построить – и оно будет работать, не требуя регулярных инвестиций, то развитие ИТ-инфраструктуры в растущих компаниях требует больших и регулярных вложений. Большие бюджеты, в свою очередь, подразумевают больший уровень ответственности и, соответственно, больший уровень компетенции тех людей, которые этими проектами управляют.

## [Планирование бюджета](#)

*Планирование бюджета* на ИТ-проекту можно разделить на три этапа:

### I. Сбор информации

Для составления (и обоснования) бюджета на ИТ нужно знать:

1. Что компания уже купила в прошлом и за что продолжает платить в настоящем, а именно:
  - используемое и находящееся в резерве оборудование,
  - лицензии на программное обеспечение,
  - сервисные контракты,
  - операторские услуги,
  - расходные материалы и затраты на них.
2. Какие информационные технологии используются в компании и для чего. Полный список услуг (ИТ-сервисов), которые использует бизнес в своей работе, начиная от банальных «электронная почта», «печать документов», «телефонная связь», заканчивая системами управления, безопасности и специфическими бизнес-приложениями.
3. Какие цели, планы и задачи у компании на ближайший финансовый период, какие проблемы должны быть решены? Реорганизация подразделений, увеличение персонала, появление новых задач, изменение требований по производительности, надежности и безопасности работы информационных систем. На данном этапе полезно пообщаться как с руководством компании, так и с руководителями структурных подразделений, чтобы понять их потребности по изменению качества обслуживания, надежности, удобства работы с системами, а также уточнить список используемых ими ИТ-сервисов. Дополнительно можно провести анкетирование пользователей, собрать полученные за последнее время обращения.

### II. Анализ

Задача данного этапа – найти те точки, которые мешают в текущий момент достичь поставленных целей или могут помешать в будущем. Для этого рекомендуется проанализировать следующие показатели:

## **1. Производительность**

Хватает ли производительности систем в настоящий момент всем пользователям? Соответствует ли текущий уровень производительности систем ожиданиям бизнеса? Что сейчас является слабым звеном? Хватит ли производительности, если нагрузка вырастет в соответствии с планами по развитию?

## **2. Надежность**

Достаточные ли меры приняты для обеспечения сохранности данных? Допустимо ли менять оборудование по мере выхода из строя или стоит его обновить заранее? Сможете ли вы в случае сбоя восстановить работоспособность в требуемые сроки или требуется заранее приобрести дополнительное оборудование или программное обеспечение для этого? Есть ли какие-то известные проблемы, которые влияют на безотказность работы ИТ-систем?

## **3. Функциональность**

Решают ли существующие приложения задачи пользователей и бизнеса? Решают ли они их эффективно? Что компания недополучает сейчас? Что нужно изменить, чтобы соответствовать будущим требованиям? Актуальны ли существующие бизнес-приложения вообще?

## **4. Безопасность**

Защищены ли данные компании от внешних угроз? А от внутренних? Соответствует ли система защиты уровню угроз? Как изменятся требования к информационной безопасности в обозримом будущем?

## **5. Удобство**

Создает ли что-нибудь дискомфорт в работе пользователей с компьютерной техникой? Удобно ли расположены принтеры в офисе, сильно ли шумят компьютеры, все ли интерфейсы и системы понятны для пользователей, жалуются ли они еще на что-то? Можно ли это улучшить?

## **6. Операционные расходы**

Оптимальны ли существующие операционные расходы? Соответствуют ли они рыночной стоимости? В какие суммы ежемесячно обходится компании тот или иной сервис? Из чего состоят эти расходы? Можно ли их уменьшить без ущерба для компаний?

## **7. Запасы**

Есть ли необходимые расходные материалы? Сколько нужно будет дополнительного оборудования и лицензий в случае расширения? Нужны ли будут дополнительные разовые или постоянные услуги в случае планируемого развития бизнеса?

## **III. Формирование бюджета и обоснование**

Собственно, вся основная работа выполнена на прошлых этапах. Последняя задача — представить полученные выводы руководству в понятном и удобном для принятия решений виде. Обычно разделяют все расходы по следующим категориям:

1. Операционные расходы на поддержание деятельности: расходные материалы, сервисные контракты, услуги, оплата труда специалистов.

2. Необходимые капитальные вложения, в случае отсутствия которых возможны серьезные потери для бизнеса. Сюда относятся расходы, которых компании не избежать и вопрос лишь в том, будет она инвестировать деньги в это заранее, или, когда уже понесет обозначенные потери.
3. Рекомендуемые инвестиции – в сочетании с необходимыми позволяют значительно повысить показатели работы, а также устраниТЬ риски, которые могут оказать негативное влияние на бизнес.
4. Расходы, связанные с развитием – необходимый объем инвестиций для обеспечения работы систем и поддержания качественных показателей в случае реализации планов по росту бизнеса.
5. Возможные инвестиции, позволяющие улучшить функциональные возможности и/или удобство работы сотрудников с ИТ-системами. Данный пункт является красной тряпкой для финансистов, позволяя им при согласовании бюджета отказать вам в этой части и тем самым с честью выполнить свой долг, не опасаясь за последствия.
6. Обоснование каждой из статей бюджета. Это самый важный подпункт. Бизнес, к сожалению, не оперирует понятиями «шестилетний сервер» и ничего не понимает в технике – он оперирует только категориями потребностей в ИТ-сервисах, возможностями, рисками и их стоимостью для бизнеса. Каталог услуг (ИТ-сервисов), который вы делали в самом начале, нужен вам для того, чтобы общаться с руководством компании на одном языке – это ваша точка взаимопонимания. Выявив потребность в оборудовании, программном обеспечении, персонале и пр. обосновывайте необходимость в них конкретными показателями работы конечных ИТ-сервисов (риски, качество, скорость реакции и пр.), которые получает или получит бизнес.

## Реализация ИТ-проекта

В **реализации ИТ-проектов** следует обратить внимание на следующие особенности:

- зачастую в компании заказчика одновременно выполняются несколько ИТ-проектов;
- приоритеты выполнения проектов постоянно корректируются;
- по мере реализации проектов выполняется уточнение и корректировка требований и содержания проектов;
- велико влияние человеческого фактора: сроки и качество выполнения проекта в основном зависят от непосредственных исполнителей и коммуникации между ними;
- каждый исполнитель может принимать участие в нескольких проектах;
- налицо трудности планирования творческой деятельности, отсутствуют единые нормативы и стандарты;
- сохраняется повышенный уровень риска, вплоть до непредсказуемости результатов;
- происходит постоянное совершенствование технологии выполнения работ.

## Прибыль в Open Source проектах

Зачастую в ИТ проектах заказчик оплачивает всю его стоимость, однако следует отметить, что в **Open Source** проектах *финансирование* может происходить и другими способами:

- Продажа профессиональных услуг

Финансовая отдача от затрат на программное обеспечение с открытым исходным кодом может исходить от продажи услуг, таких как обучение, техническая поддержка, или консультации, а не самого программного обеспечения.

Другая возможность предлагает open source программное обеспечение только в виде исходного кода, при этом предоставляя исполняемые бинарные файлы только платным клиентам, предлагая коммерческие услуги по компиляции и созданию инсталляционных пакетов программного обеспечения. Кроме того, предоставление open source как коммерческий товар на физическом носителе (например, DVD).

Успешные Open source компании, использующие эту бизнес-модель: RedHat и IBM; более специализированным примером является Revolution Analytics.

- Продажа фирменных товаров

Некоторые FOSS организации, например, Mozilla Foundation или Wikimedia Foundation, пытаются продавать фирменные товары: футболки, кофейные кружки. Это может рассматриваться в качестве дополнительной услуги для сообщества пользователей.

- Продажа программного обеспечения как услуги

Платная подписка на онлайн-аккаунты и доступ к серверу для клиентов является способом получения прибыли на базе программного обеспечения с открытым исходным кодом. Кроме того, комбинация настольных ПК с сервисом, называется программное обеспечение плюс услуги. Предоставление услуг облачных вычислений и программного обеспечения как услуги (SaaS) без предоставления самого программного обеспечения с открытым исходным кодом, ни в двоичной ни в исходной форме соответствует большинстве лицензий с открытым исходным кодом (за исключением AGPL).

- Партнерство с финансирующими организациями

Прочие финансовые ситуации включают партнерские отношения с другими компаниями. Правительства, университеты, компании или другие неправительственные организации могут разрабатывать у себя или нанять подрядчика для внутренних пользовательских модификаций программного обеспечения, а затем выпустить этот код под открытой open source лицензией. Некоторые организации поддерживают разработку программного обеспечения с открытым исходным кодом грантами или стипендиями, например, Google Summer of Code initiative основанную в 2005.

- Добровольные пожертвования

Появление систем Интернет микроплатежей в 2000-х годах таких, как PayPal, Flattr и Bitcoin помогает этому.

- Денежное вознаграждение за выполнение задачи (bounty)

Пользователи конкретного программного обеспечения могут объединиться вместе и собрать деньги для open source проекта для разработки желаемого функционала.

- Предварительный заказ / Crowdfunding / модель обратная bounty

Новая возможность финансирования проектов СПО – Crowdfunding, модель похожа на пред-заказ, а также на перевернутую модель bounty. Как правило, организуется на базе веб-платформ, таких как Kickstarter, Indiegogo, Catinca или Bountysource. Пример успешного финансирования: австралийской программист Timothy Arceri, который предложил за \$2500 реализовать в течение двух недель расширение OpenGL 4.3 для библиотеки Mesa.

- Программное обеспечение, содержащее рекламу

С целью коммерциализации FOSS, многие компании (в том числе Google, Mozilla и Canonical) перешли к экономической модели заработка на рекламе в программном обеспечении.

Например, приложение с открытым исходным кодом AdBlock Plus получает деньги от Google за расширение белого списка разрешенных рекламных блоков в обход блокировщика рекламы в браузере.

- Продажа дополнительных проприетарных расширений

Некоторые компании продают собственные дополнительные расширения, модули, плагины к open source программному продукту. Это может соответствовать свободным лицензиям, если сделано технически достаточно тщательно.

- Продажа необходимых проприетарных частей программного продукта

Вариант подхода выше, заключается в хранении нужного контента (например, аудио, видео для игр, графику или другие художественные активы) в закрытом программном продукте, выпуская сам исходный код под открытой лицензией. Хотя такой подход вполне совместим с большинством open source лицензий, клиенты должны купить контент, чтобы иметь полную и работающую версию программного продукта.

Похожий на этот прием привязывания open source программного продукта к проприетарной аппаратной части называется «тивоизация» и проходит с большинством open source лицензий за исключением GPLv3, которая прямо запрещает подобное использование.

- Пере-лицензирование под проприетарной лицензией

Если программный продукт использует только собственное программное обеспечение и программное обеспечение с открытым исходным кодом под разрешительной свободной лицензией, то компания может повторно лицензировать конечный программный продукт под проприетарной лицензией и продать продукт без исходного кода и софтверных свобод. Например, Apple Inc. является активным эксплуататором этого подхода, используя исходный код и программное обеспечение из различных open source проектов, например, ядро операционной системы BSD Unix под лицензией BSD было использовано в компьютерах Mac, которые продаются как патентованных продукты.

- Обfuscация исходного кода

Подход состоящий в запутывании исходного кода, для коммерциализации с некоторыми открытыми лицензиями, защищая при этом важные коммерческие тайны, интеллектуальную собственность и технические ноу-хай. Этот подход был использован в ряде случаев, например, Nvidia в своих драйверах для графических карт.

- Задержка с выпуском open-source software

Некоторые компании предоставляют самую свежую версию только для платных клиентов. Далее вендор делает ответвление от программного проекта без авторского права, добавляет к нему дополнения с закрытым кодом и продает конечный программный продукт. После некоторого периода времени патчи интегрируются обратно в приложение под той же лицензией, что и остальной часть кода. Эта бизнес-модель называется версия с отставанием.

Экстремальный вариант такой модели является бизнес-практика, которую популяризовали Id Software и 3D Realms, которые выпустили несколько своих программных продуктов под свободной лицензией после долгого коммерческого периода, в течение которого произошел возврат инвестиций.

- FOSS и экономика

По правовым исследованиям предпринимательства в Гарвардской школе права, свободное программное обеспечение является наиболее видимой частью новой экономики на основе общего равного производства информации, знаний и культуры. В качестве примеров они приводят ряд проектов FOSS, которые включают как free, так и open source ПО.

Эта новая экономика уже на стадии разработки. С целью коммерциализации FOSS, многие компании, Google является самым успешным, движутся в направлении экономической модели программного обеспечения, содержащего рекламу (AdWare).

### Аналоги среди IT-проектами

**Анализ статистики** показывает, что примерно 90 процентов ИТ-проектов аналогичны уже выполненным. У руководителя проекта имеется опыт реализации таких задач и понимание возможных проблем. В этих случаях иерархическая структура проекта и работ (ИСП/ИСР) формируется с применением подхода Top-down (сверху вниз), используется типовая структура проектной команды, планы проекта (план управления рисками, план коммуникаций и пр.) аналогичны планам предыдущих проектов. Однако 10 процентов проектов – инновационные, реализуемые "с нуля" и требующие творчества, нестандартных решений и управленческой смелости. Принятие решений в таких проектах характеризуется высокими рисками, что требует от руководителя глубоких знаний методики проектного управления и понимания особенностей её применения в сфере информационных технологий.

### Управление IT-проектом

Применение **методологии управления проектами** позволяет зафиксировать цели и результаты проекта, дать им количественные характеристики, определить временные, стоимостные и

качественные параметры проекта, создать реальный план выполнения проекта, выделить, оценить риски и предотвратить возможные негативные последствия во время реализации проекта.

Для эффективного управления проект должен быть хорошо структурирован. Суть этого процесса сводится к выделению следующих основных элементов:

- фазы жизненного цикла проекта, этапов, работ и отдельных задач;
- организационная структура исполнителей проекта;
- структура распределения ответственности.

### Жизненный цикл IT-проекта

**Жизненный цикл** – это последовательность фаз проекта, через которые он должен пройти для гарантированного достижения целей проекта, в нашем случае – для реализации некоторой информационной технологии.

Организационная структура подразумевает выделение ролей исполнителей, которые необходимы для реализации проекта, определение взаимоотношений между ними и распределение ответственности за выполнение задач.

### Жизненный цикл IT-проекта

#### Жизненный цикл проекта

Любой проект имеет ограниченный отрезок времени существования. Наличие этого отрезка времени означает, что у проектов есть жизненный цикл. Жизненный цикл последовательно проходит через четыре стадии:

1. Определение
2. Планирование
3. Выполнение
4. Сдача

Все начинается, когда появляется идея проекта. Происходит его определение. Здесь поднимается вопрос о цели проекта, о его целесообразности. Далее проект переходит в стадию планирования, где планируется бюджет, принимаются ключевые решения. Бюджет проекта имеет большое значение. Так, например, на воплощение проекта с большим бюджетом, в среднем затрачивается меньше времени, чем на проект с меньшим бюджетом. К этой стадии необходимо относиться ответственно, потому что если на этом этапе совершается архитектурная ошибка, то в случае, если ошибку невозможно исправить, проект либо меняет свою структуру (а это тянет за собой дополнительные расходы ресурсов), либо прекращает свое существование. На стадии выполнения происходит непосредственное воплощение проекта. Здесь проект может претерпевать значительные изменения. После стадии выполнения проект сдается. Под сдачей подразумевается введение проекта в эксплуатацию.

### Жизненный цикл IT-проекта

В IT-проектах можно выделить следующие особенности:

- по мере реализации проектов выполняется большое количество уточнение и корректировка требований и содержания;

- крайне велико влияние человеческого фактора на сроки выполнения;
- из-за трудности планирования творческой деятельности, отсутствуют единые нормативы и стандарты;
- сохраняется повышенный уровень риска, вплоть до непредсказуемости результатов;
- происходит постоянное совершенствование технологии выполнения работ, что тянет за собой постоянное изменение структуры проекта.

Что касается стандартизации, то современные стандарты не предписывают четких и однозначных схем построения структуры жизненного цикла проекта. Это сделано намеренно, поскольку достаточно жесткие схемы препятствуют использованию более прогрессивных технологий разработки, которых появилось очень много и которые продолжают интенсивно развиваться.

Исходя из этих особенностей, можно построить структуру жизненного цикла it-проекта:

- **Начальная стадия** - цель: определить границы системы и собрать требования высокого уровня;
- **Стадия уточнения** - цель: создать архитектурную основу системы;
- **Стадия конструирования** - цель: создание финального продукта;
- **Стадия передачи и сопровождения** - цель: внедрение продукта на предприятии заказчика, обучение персонала, сопровождение и обновление установленной информационной системы.

**Начальная стадия жизненного** цикла it-проекта не отличается по структуре от любого другого проекта. На стадии уточнения производится выбор технологий, определение необходимых ресурсов (как денежных, так и человеческих), построение команды. Также здесь происходит подписание необходимых документов. Таких как техническое задание и документов, которые призваны защитить компанию-разработчика от внезапного отказа заказчика от проекта. Уже на стадии уточнения может начинаться тестирование. Такой подход называется *test-driven development* (TDD, Разработка через тестирование). Он заключается в написании тестов раньше, чем написание кода. Далее следует **стадия конструирования**, которую можно условно разделить на фазы части:

- **Проектирование** - определение характеристик архитектуры системы, компонентов, составляющих, интерфейсов и других частей. Созданное описание, в свою очередь, является фундаментом для реализации;
- **Реализация** - создание программного продукта, исходя из созданного на этапе проектирования программного проекта;
- **Интеграция** - объединение программных компонентов и интегрирование их в среду.

Основная часть стадии конструирования, **реализация**, может быть поделена еще на 6 частей:

- **Пре-Альфа** - период времени со старта разработки до выхода стадии Альфа (или до любой другой, если стадии Альфа нет). Также так называются программы, не вышедшие еще в стадию альфа или бета, но прошедшие стадию разработки, для первичной оценки функциональных возможностей в действии. В отличие от альфа- и бета-версий, пре-альфа может включать в себя не весь спектр функциональных возможностей программы. В этом

случае, подразумеваются все действия, выполняемые во время проектирования и разработки программы вплоть до тестирования. К таким действиям относятся — разработка дизайна, анализ требований, собственно разработка приложения, а также отладка отдельных модулей.

- **Альфа** - внутреннее тестирование — стадия начала тестирования программы в целом специалистами-тестерами, обычно не разработчиками программного продукта, но, как правило, внутри организации или сообществе разрабатывающих продукт. Также это может быть стадия добавления новых функциональных возможностей. Программы на данной стадии могут применяться только для ознакомления с будущими возможностями.
- **Бета** - публичное тестирование — Стадия активного бета-тестирования и отладки программы. Программы этого уровня могут быть использованы другими разработчиками программного обеспечения для испытания совместимости. Тем не менее, программы этого этапа могут содержать достаточно большое количество ошибок.
- **Релиз-кандидат** - иногда «гамма-версия» — стадия-кандидат на то, чтобы стать стабильной. Программы этой стадии прошли комплексное тестирование, благодаря чему были исправлены все найденные критические ошибки. Но в то же время существует вероятность выявления ещё некоторого числа ошибок, не замеченных при тестировании.
- **Релиз** - издание продукта, готового к тиражированию. Это стабильная версия программы, прошедшая все предыдущие стадии, в которых исправлены основные ошибки, но существует вероятность появления новых, ранее не замеченных, ошибок.
- **Пост-релиз** - издание продукта, у которого есть несколько отличий от релизного и помечается как самая первая стадия разработки следующего продукта. Такие релизы не выпускаются на продажу, а раздаются бета-тестировщикам. Эта стадия встречается редко и присуща проектам, которые делятся на отдельно реализуемые версии.

Третья по счету фаза конструирования - **интеграция**. На этой фазе происходит интеграция разрабатываемого проекта с уже созданным окружением, с которым этот проект будет функционировать.

И последняя, четвертая, стадия жизненного цикла it-проекта - **стадия передачи и сопровождения**. Передача проекта заказчику - очень сложный процесс и требует тщательного подхода и ответственности. Какие здесь возникают проблемы? Часто возникают спорные вопросы из-за того, что требования к системе были сформулированы абстрактно либо недостаточно хорошо. Качественное техническое задание - оружие компании-разработчика. Однако наличие технического задания не является ни необходимым, ни достаточным основанием для полноценного закрытия работ. Если исполнитель придерживается ГОСТов, к приемочным испытаниям на основании технического задания должен быть разработан и согласован с заказчиком дополнительный документ "Программа и методика испытаний". В нем должны быть прописаны принципы оценки реализации требований технического задания. Также на этой стадии производится поддержка и сопровождение проекта. Длительность сопровождения зависит от сложности, целесообразности, выгодности и области применения проекта. Так, например, социальные сети поддерживаются и сопровождаются на протяжении всего времени использования, а проект, который перестал приносить прибыль, быстро теряет поддержку со стороны производителя.

## Особенности open-source проектов

У open-source проектов выделяется ряд специфических особенностей:

- техническое задание либо размытое, либо и вовсе отсутствует;
- большое количество исполнителей (contributors);
- отсутствие четко сформированного будущего проекта;
- непредсказуемость продолжительности жизни.

## Моделирование жизненного цикла

Перед началом любой работы каждый человек пытается максимально подробно разобраться в деталях, мелочах, с которыми ему предстоит столкнуться, разобраться в том, с чем придется иметь дело, изучить способы и стратегии исследования предмета деятельности. Этим и занимается такое учение, как методология. И как работа в данном случае рассматривается жизненный цикл IT-проекта.

При работе с IT-проектами использование специализированных методов управления проектами будет очень полезным для непрерывного прогресса, приводящего к большей успешности в результате. Планирование и выполнение IT-проектов может быть трудным по разным причинам, что делает умение успешно их завершать очень ценным качеством для любого работодателя. В наше время в интернете можно получить достаточно информации по каждой из методологий, но для особо любознательных IT-компаний проводят специальные тренинги по изучению различных методологий жизненного цикла. Так, например, компания "Проектный офис" часто проводит тренинги по Agile, Scrum в Минске продолжительностью 2-3 дня.

Как известно, нет единого способа решения различных проблем. И наш проект здесь не будет исключением. Эффективность каждого из проектов достигается не всегда использованием одной и той же методологии. Здесь и возникает проблема в ее выборе.

Выбор определенной модели жизненного цикла зависит, в основном, от содержания и целей проекта, а также от размера его финансирования. Рассмотрим некоторые из них.

### Waterfall

Одна из самых "старых", и уже традиционных, и, как следствие, популярных моделей, которую сейчас принято считать каскадной. Была упомянута в статье американского информатика Уинстона У. Ройса в 1970 году. Модель "водопада" определена как последовательная модель разработки с явно выраженным конечным результатом для каждой фазы. Стандартные фазы "водопада" представляют собой:

- 1) Requirements (Определение требований)
- 2) Design (Проектирование)
- 3) Development (Разработка)
- 4) Testing (Тестирование и отладка)
- 5) Maintenance (Инсталляция/ поддержка)

Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей. Переходов назад либо вперёд или перекрытия фаз — не происходит.

### Преимущества:

- Такая жёсткая последовательность делает процесс разработки чрезвычайно прозрачным, а, следовательно, максимально удобным для заказчика (благодаря последовательному переходу от этапа к этапу управление масштабными проектами осуществляется гораздо проще, а команда, в свою очередь, работает слаженнее);
- Жёсткая последовательность позволяет дать точную оценку стоимости разработки и ее сроков, что позволяет точно спрогнозировать эффект, полученный от запуска приложения;
- Удобство управления проектом и возможностью полного контроля над каждым этапом его создания;
- Эта модель представляет базовый подход, который может применяться в любом проекте (универсальная модель; Другие модели жизненного цикла могут оказаться более результативными и эффективными в зависимости от характеристик проекта. Например, если устанавливается пакет программного обеспечения, то пропускаются фазы проектирования и реализации.).
- Классический «водопадный» подход – это модель жизненного цикла, которую любой обыватель может применить, практически ничего не зная о методологии и планируя проект «с чистого листа».

Что может быть проще? Даже если у обычного человека очень маленький проект, всё равно он проходит эти базовые шаги, хотя бы даже проделывая некоторые из них в голове. К примеру, если у него 40-часовой (на одну рабочую неделю) проект разработки или улучшения документа, может показаться что он сразу же бросается в фазу "Разработка". Но так ли это? Наиболее вероятно, что он получил какого-либо рода поручение с требованиями или пожеланиями, которые придется осмыслить (Определение требований) и трансформировать в замысел будущего содержания (Проектирование). Затем он воплощает замысел (Разработка), проверяет результат (Тестирование) и передает для использования (Инсталляция).

#### Недостатки:

- При необходимости внесения поправок в документацию разработка продукта останавливается вплоть до момента повторного согласования документов (следовательно, при недостаточном уровне проработки требований существует риск увеличить сроки разработки до абсолютно неприемлемых величин);
- Накопление возможных на ранних этапах ошибок к моменту окончания проекта и, как следствие, возрастание риска провала проекта, увеличение стоимости проекта.

Забавно, что сам Ройс использовал итеративную модель разработки и даже не использовал термин "водопад". Оказалось, что он предусматривал обратные связи между этапами на одном уровне. И в отличие от привычного метода, здесь речь уже шла о параллельных работах по двум последовательным этапам, что дает возможность на более ранних стадиях выявить ошибки предыдущего этапа жизненного цикла и решить один из весомых недостатков классического "водопада" — невозможность возврата на предыдущий этап. К примеру, если рассмотреть параллельные работы по двум последовательным фазам - Coding и Testing, становится очевидным, что часть программы тестируется, в то время как другая часть все еще находится на стадии разработки. То есть получается, что речь действительно идет об итеративной методологии. В таком

случае, остается вопросом, почему методология в широких кругах разработчиков и тестировщиков воспринимается ошибочно.

Водопадный подход хорошо работает в проектах, где требования к программному продукту чётко определены и не должны меняться, вовлечение заказчика в процесс разработки не требуется. Каскадная модель подходит для разработки сложных и больших проектов и систем со строго определённой функциональностью (например, разработка транзакционной системы, где есть чётко определённый результат). Также удобно использовать её при разработке больших государственных заказов или научных разработках. А вот для разработки бизнес или веб-приложений крайне нежелательно, так это относительно креативный проект, ход работы которого может часто меняться (например, по просьбе заказчика), в то время как в "водопадной" методологии всё чётко и последовательно.

### Iterative

Итеративный подход (англ. *iteration* — повторение) — выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Проект при этом подходит в каждой фазе развития проходит повторяющийся цикл: Планирование — Реализация — Проверка — Оценка (англ. *plan-do-check-act cycle*). Этот метод является альтернативой последовательной модели.

Модель предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает "мини-проект", представляющий собой небольшие фрагменты функциональности, по сравнению с проектом в целом. Цель каждой итерации заключается в том, чтобы, постепенно добавляя больше опций, получить работающую версию программной системы, включающей функциональность, основанную на содержании всех предыдущих и текущей итерации. Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации продукт получает приращение — инкремент — к его возможностям, которые, следовательно, развиваются эволюционно. По сравнению с водопадом, итеративный метод является более гибким.

Основные преимущества итеративного подхода:

- Снижение воздействия серьёзных рисков на ранних стадиях проекта, что ведет к минимизации затрат на их устранение;
- Акцент усилий на наиболее важные и критичные направления проекта;
- Непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом;
- Раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта;
- Более равномерная загрузка участников проекта;
- Реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении.

Этот метод имеет и свои отрицательные стороны:

- Общее представление о возможностях и ограничениях проекта очень долгое время отсутствует;
- При итерациях приходится отбрасывать часть сделанной ранее работы;

- Добросовестность специалистов при выполнении работ всё же снижается, что психологически объяснимо тем, что у них появляется ощущение, что "всё равно всё можно будет переделать и улучшить позже".

Различные варианты итерационного подхода реализованы в большинстве современных методологий разработки (RUP, MSF, XP).

Пример реализации итеративного подхода — методология разработки программного обеспечения, созданная компанией Rational Software.

Итеративную либо быструю (Agile development) разработку часто используют при реализации проектов в области информационных технологий.

### Spiral

Спиральная модель была разработана в 1986-ом году Барри Боэмом. Она основана на классическом цикле Деминга PDCA (plan-do-check-act). При использовании этой модели создается в несколько итераций (витков спирали) методом прототипирования (процесс создания прототипа программы — макета (черновой, пробной версии) программы, обычно — с целью проверки пригодности предлагаемых для применения концепций, архитектурных и/или технологических решений, а также для представления программы заказчику на ранних стадиях процесса разработки).

Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем, в то время, как при итеративной методологии после каждой итерации чётко видно "приращение". При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла - определение момента перехода на следующий этап.

### Agile

Зачастую бывает необходимо модернизировать ИТ-решения или разрабатывать новые программные продукты в самые сжатые сроки. При этом требования к функциональности программного обеспечения не оговорены и постоянно пересматриваются. В таких условиях выбирают для реализации проектов модель гибкой разработки программного обеспечения – Agile.

Agile (и ее вариации – методологии Scrum, Extreme Programming и т.д.) подразумевает разработку программного обеспечения короткими циклами. Каждый этап – это уменьшенный вариант ИТ-проекта: проводится планирование, анализ требований, проектирование, кодирование, тестирование и документирование. По окончании этапа заказчик получает работающую версию ИТ-системы, если требуется, пересматривает дальнейшие приоритеты проекта, и цикл разработки запускается снова. В итоге создается решение, которое на 100% соответствует ожиданиям компании.

На практике методология Agile может использоваться в нескольких интерпретациях. В проектах для заказчиков наиболее часто применяют: Scrum, Extreme Programming, Lean Software Development (LSD), Dynamic Systems Development Method (DSDM), Open Unified Process (OpenUP), Agile Project Management (APM), Microsoft Solutions Framework для Agile (MSF) и др.

Можно выделить следующие преимущества Agile:

- Быстрая и постоянная обратная связь команды разработчиков с заказчиком (изменения в требованиях пользователей оперативно учитываются);
- Гибкий график реализации функциональности;
- Отсутствие затрат на составление и согласование подробной технической документации (Agile позволяет избежать затрат на работы, которые не всегда нужны (например, составление и согласование подробной технической документации));
- Возможность остановки проекта без ущерба для осуществленных вложений в разработку ИТ-системы (каждая стадия проекта, за исключением начальных, заканчивается созданием вполне готовой версии ИТ-системы; есть возможность (например, при замораживании бюджета) остановить проект и при этом получить работоспособное решение).

Рассмотрим несколько разновидностей Agile-метода.

### *Scrum*

Одна из самых популярных методологий гибкой разработки. Одна из причин ее популярности - простота.

Sprint — итерация в scrum, в ходе которой создаётся функциональный рост программного обеспечения. Ее длительность составляет 1 месяц (30 дней), но может и варьироваться. Результатом Sprint является готовый продукт (build), который можно передавать (deliver) заказчику (по крайней мере, система должна быть готова к показу заказчику).

В методологии Scrum всего три роли:

- Scrum Master
- Product Owner
- Team

*Scrum Master* - самая важная роль в методологии. Scrum Master отвечает за успех Scrum в проекте. По сути, Scrum Master является интерфейсом между менеджментом и командой. Как правило, эту роль в проекте играет менеджер проекта или тимлид. Важно подчеркнуть, что Scrum Master не раздает задачи членам команды, он проводит совещания (Scrum meetings) следит за соблюдением всех принципов Scrum, разрешает противоречия и защищает команду от отвлекающих факторов. Данная роль не предполагает ничего иного, кроме корректного ведения Scrum-процесса. Руководитель проекта скорее относится к владельцу проекта и не должен фигурировать в качестве Scrum-мастера.

*Product Owner* - это человек, отвечающий за разработку продукта. Представляет интересы конечных пользователей и других заинтересованных в продукте сторон. Product Owner - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один

человек, а не группа или комитет. Product Owner ставит задачи команде, но он не вправе ставить задачи конкретному члену проектной команды в течении спринта.

В методологии Scrum (Team) команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Product Owner. Работа команды оценивается как работа единой группы. Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные размер команды варьируются от 5 до 9 человек.

Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды.

Для облегчения коммуникаций команда должна находиться в одном месте. Предпочтительно размещать команду в одной общей комнате для того, чтобы уменьшить препятствия для свободного общения. Команде необходимо предоставить все необходимое для комфортной работы.

Заказчик получает возможность гибко управлять рамками системы, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в список Product Backlog, где они выстраиваются по приоритету наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов.

Каждый спринт представляет собой маленький "водопад". В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта. Границы спринта должен быть фиксированными. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в спринте.

В начале каждого спринта проводится планирование спринта. В планировании спринта участвуют заказчики, пользователи, менеджмент, Product Owner, Scrum Master и команда.

Остановка спринта производится в исключительных ситуациях. Спринт может быть остановлен до того, как закончатся отведенные 30 дней. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить Product Owner, если необходимость в достижении цели спринта исчезла. После остановки спринта проводится встреча с командой, где обсуждаются причины остановки спринта. После этого начинается новый спринт: производится его планирование и начинаются работы. Команда демонстрирует изменения продукта, созданные за последний спринт. Product Owner, менеджмент, заказчики, пользователи, в свою очередь, его оценивают. Команда рассказывает о поставленных задачах, о том, как они были решены, какие препятствия были у них на пути, какие были приняты решения, какие проблемы остались нерешенными. Аналогично происходит переход между спринтами.

Срыв спринта грозит в большей степени только финансовой стороне.

#### *XP (экстремальное программирование)*

Название методологии исходит из идеи применить полезные традиционные методы и практики разработки программного обеспечения, подняв их на новый "экстремальный" уровень. Так, например, практика выполнения исследования кода, заключающая в проверке одним программистом кода, написанного другим программистом, в "экстремальном" варианте

представляет собой "парное программирование", когда один программист занимается кодированием, а его напарник в это же время непрерывно просматривает только что написанный код.

Экстремальное программирование предлагает готовое решение: делать всё максимально просто и быстро, держать заказчика при себе (в основе экстремального программирования лежат не конкретные методики, как принято считать, а лишь четыре базовых принципа: общение, простота, обратная связь и храбрость.), позволяя ему активно следить за процессом разработки.

Двенадцать основных приёмов экстремального программирования могут быть объединены в четыре группы:

- Короткий цикл обратной связи (Fine-scale feedback)
  - Разработка через тестирование (Test-driven development)
  - Игра в планирование (Planning game)
  - Заказчик всегда рядом (Whole team, Onsite customer)
  - Парное программирование (Pair programming)
- Непрерывный, а не пакетный процесс
  - Непрерывная интеграция (Continuous integration)
  - Рефакторинг (Design improvement, Refactoring)
  - Частые небольшие релизы (Small releases)
- Понимание, разделяемое всеми
  - Простота (Simple design)
  - Метафора системы (System metaphor)
  - Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership)
  - Стандарт кодирования (Coding standard or Coding conventions)
- Социальная защищенность программиста (Programmer welfare):
  - 40-часовая рабочая неделя (Sustainable pace, Forty-hour week)

### *Lean*

Известна как бережливая разработка программного обеспечения, так как использует методы концепции бережливого производства.

Основными Lean-принципами являются:

- Исключение потерь. Потерями считается всё, что не добавляет ценности для потребителя. В частности, излишняя функциональность, ожидание (паузы) в процессе разработки, нечёткие требования, медленное внутреннее сообщение.
- Акцент на обучении. Короткие циклы разработки, раннее тестирование, частая обратная связь с заказчиком.
- Предельно отсроченное принятие решений. Решение следует принимать не на основе предположений и прогнозов, а после открытия существенных фактов.
- Предельно быстрая доставка заказчику. Короткие итерации.
- Мотивация команды. Нельзя рассматривать людей исключительно как ресурс. Людям нужно нечто большее, чем просто список заданий.

- Интегрирование. Передать целостную информацию заказчику. Стремиться к целостной архитектуре. Рефакторинг.
- Целостное видение. Стандартизация, установление отношений между разработчиками. Разделение разработчиками принципов бережливости.

Таким образом, нет универсальной модели. Её выбор, как правило, зависит от нескольких факторов, таких как характер работы (есть ли чётко определенный результат? Разработка веб-сайтов является творческим делом и поэтому подходит для гибкого метода. Разработка транзакционной системы, где есть четко определенный результат, больше подходит для водопадного метода.), тип клиента (согласен ли он работать итеративно, есть ли у него время на проверку и комментирование регулярных итераций?), мнение вашего руководителя, внутренний или внешний клиент (водопадный метод подходит для внешних клиентов, от которых можно потребовать соблюдения подписанных договоров, в отличие от внутренних клиентов, способных вынудить вас внести изменения, получив поддержку руководства). Но есть ряд методологий, которые являются излюбленными у крупных компаний. Так, например, Microsoft нередко использует MSF (Microsoft Solutions Framework), которую сама и предложила. А Apple отдаёт предпочтение Agile, Scrum, Kanban и Lean.

Каждому проекту своя методология!

## Мозговой штурм

### Основная идея

Сущность метода мозгового штурма заключается в том, что отбирается группа квалифицированных экспертов, но оценки и выводы делаются в ходе заседания. Все эксперты делятся на две группы: первая генерирует идеи (выставляет оценки), а вторая — их анализирует. При этом запрещается критиковать ту или иную идею. Идея, с которой согласится большинство экспертов, считается правильной.

Метод "мозгового штурма":

- достаточно оперативен и надежен;
- это максимум идей за короткий отрезок времени;
- это отсутствие какой-либо критики;
- это развитие, комбинация и модификация как своих, так и чужих идей.

Этот метод специально разработан для получения максимального количества предложений. Его эффективность поразительна: 6 человек за полчаса могут выдвинуть 150 идей. Бригада проектировщиков, работающая обычными методами, никогда не пришла бы к мысли о том, что рассматриваемая ими проблема имеет такое разнообразие аспектов.

### Техника проведения мозгового штурма

Техника мозгового штурма такова. Собирается группа лиц, отобранных для генерации альтернатив. Главный принцип отбора — разнообразие профессий, квалификаций, опыта (такой принцип позволяет расширить фонд априорной информации, которой располагает группа). Сообщается, что приветствуются все идеи, возникшие как индивидуально, так и по ассоциации при выслушивании предложений других участников, в том числе и лишь частично улучшающие чужие идеи (каждую идею рекомендуется записывать на отдельной карточке). Категорически запрещается любая

критика — это важнейшее условие мозгового штурма: сама возможность критики тормозит воображение. Каждый по очереди зачитывает свою идею, остальные слушают и записывают на карточки новые мысли, возникшие под влиянием услышанного. Затем все карточки собираются, сортируются и анализируются, обычно другой группой экспертов.

Число альтернатив можно впоследствии значительно увеличить, комбинируя сгенерированные идеи. Среди полученных в результате мозгового штурма идей может оказаться много глупых и несуществимых, но глупые идеи потом легко исключить последующей критикой.

## Условия проведения мозгового штурма

### Категории участников

- Жестких ограничений нет, но лучше включать в группу работников с относительно небольшим опытом работы — они еще не имеют выработанных стереотипов.
- При решении специфических задач необходимо приглашать специалистов (но они будут приглашенными, а не участниками).
- Рекомендуется формировать смешанные группы (из мужчин и женщин). Как правило, наличие представителей разного пола оживляет атмосферу работы.
- При проведении мозгового штурма желательно, чтобы количество активных и умеренных членов группы было примерно поровну.
- Необходимо, чтобы разница в возрасте, служебном положении между членами группы была минимальной. Присутствие начальства также сдерживает и ограничивает ход протекания мозгового штурма.
- Не рекомендуется приглашать на проведение мозгового штурма скептически настроенного руководителя, даже при условии его участия в роли наблюдателя.
- Целесообразно время от времени вводить в группу новых людей, новые люди вносят новые взгляды, идеи, стимулирующие мышление.

### Количество участников

- Оптимальный состав группы от 6 до 12 человек. Оптимальное участников число — 7.
- Не рекомендуется разбивать участников группы на более мелкие (2 и более).
- Количество людей в группе также зависит от количества в ней активных и умеренных членов. Если больше активных, то количество людей в группе должно быть меньше, больше умеренных — наоборот.

### Обстановка, место проведения

- Для проведения мозгового штурма целесообразно место проведения использовать аудиторию или отдельную комнату, вдали от постороннего шума. На стене рекомендуется повесить плакат с основными правилами проведения мозгового штурма.
- Желательно иметь доску, которую участники могут использовать для отображения своих идей. Столы и стулья рекомендуем расположить в виде буквы П, О, круга или полуэллипса. Это облегчает контакт участников и повышает коммуникабельность. Если группа небольшая (5 — 6 человек) — наиболее удобен круглый стол.
- Желательно иметь магнитофон: человек может не успеть вникнуть в идею и упустить ее.

- Не забывайте, что юмор во время собрания необходим. Это способствует созданию непринужденной обстановки и творческой атмосферы.

### **Продолжительность и время**

- Как правило, продолжительность проведения мозгового штурма и время колеблется в пределах 40 — 60 минут. Это наиболее эффективный промежуток времени.
- При решении простых проблем или при ограничении по времени наиболее подходящая продолжительность обсуждения — 10-15 минут.
- Наиболее подходящее время для проведения мозгового штурма — утро (с 10 до 12 ч), но также можно проводить его и после обеда (с 14 до 18 ч).

### **Типы проблем, решаемые методом мозгового штурма**

- Метод мозгового штурма позволяет решать любую проблему, решаемые методом имеющую несколько возможных вариантов решений. Проблемы, мозгового штурма имеющие только один ответ или ограниченное число возможных решений, не подходят для решения этим методом.
- Необходимо также избегать решения слишком общих, абстрактных проблем.
- Рекомендуется избегать полного решения проблемы за одну сессию. Если начальная формулировка слишком широка и обобщена, следует подразделить ее на ряд подпроблем.
- Метод мозговой атаки можно с успехом использовать для сбора информации, а не идей, т. е. для выяснения источников или формирования вопросов анкеты.
- Проблемы для обсуждения рекомендуется формулировать просто и ясно.

### **Озвучивание проблемы**

- Тема мозгового штурма раскрывается участникам заранее, за несколько дней до обсуждения. В этом случае ведущий (председатель) представляет краткое изложение темы или проблемы (до 5 мин, объемом на пол-листа), раздает ее участникам заранее.
- Ознакомление участников мозговой атаки с темой или проблемой непосредственно при проведении мозгового штурма.
- Существует также и смешанный способ подачи темы или проблемы для мозговой атаки. То есть заранее сообщается частичная, а не полная информация по проблеме.

*Рекомендуется использовать три правила представления идеи или проблемы:*

- Показать или проиллюстрировать путь развития проблемы или ситуации. Если это возможно, то лучше графически.
- Дать рекомендации по выбору основных точек соприкосновения. Использовать диаграммы, модели и все, что наилучшим образом подходит для этой цели. Желательно все это показать и объяснить просто и четко.
- Суммировать имеющиеся точки зрения, показать их преимущества и недостатки. Еще раз подчеркнуть необходимость решения.

### **Роль руководителя (лидера)**

- Основные функции руководителя заключаются в информировании всех участников о правилах мозговой атаки, контроле за их соблюдением, а также в общем контроле за дискуссией, чтобы она оставалась в рамках или границах обсуждаемой темы или проблемы.
- Важно, чтобы руководитель сам участвовал в генерировании идей. Он одновременно должен выполнять роль стимулятора или катализатора в случае замедления темпа генерирования идей. Хороший руководитель, как правило, должен заранее иметь список возможных решений проблемы.
- Роль руководителя заключается также в подборе участников мозгового штурма как минимум за 2 дня до ее проведения.
- Эффективный руководитель постоянно подбрасывает «дикие» и безрассудные идеи и предложения, чтобы продемонстрировать, что они поощряются.
- Иногда бывает, что группе участников трудно избавиться от традиционных подходов, стереотипов в решении проблемы. В этом случае рекомендуем использовать маленькую хитрость: руководитель останавливает ход мозгового штурма и вводит ограничения: в течение 2-3 минут предлагать только непрактичные, самые необычные идеи.
- Часто бывает, что участники продолжают генерировать интересные идеи и после проведения собрания. В этом случае задача руководителя — собрать группу через несколько дней и зафиксировать эти идеи.

### Оценка идей

- Для оценки идей необходимо выбрать критерии. Критериями оценки могут быть актуальность, практическая реализация, решаемость собственными силами, новизна и т. д.
- Оценка идей может осуществляться той же или другой группой по составу. Если оценка осуществляется той же группой участников, то, как правило, она производится через несколько дней.

### Правила проведения мозгового штурма

**Правило 1:** Запрещается всякая критика идей, высказываемых во время проведения мозгового штурма

Принцип проведения мозгового штурма заключается в приоритете количества высказанных идей над их качеством. Высказываемые участниками идеи, пусть даже самые сумасшедшие, могут служить отправной точкой для развития мыслительного процесса других участников. В этом и заключается преимущество коллективного мышления над индивидуальным. Любая, даже самая малая, оценка высказанной идеи может повлиять на весь процесс проведения мозгового штурма. Он будет успешным, если каждый участник направит свои усилия в конструктивное русло.

**Правило 2:** Свободный полет мыслей и поощрение самых «безумных» идей

Целью мозгового штурма, как коллективного творческого процесса, является поиск нестандартных, нетрадиционных идей. В противном случае этот процесс может превратиться в обычное совещание, на которых чаще всего предлагаются и обсуждаются именно стандартные идеи и решения, которые не всегда являются результативными и эффективными.

Для появления творческих идей необходим определенный настрой, когда мысли свободно проносятся в нашей голове. Это состояние характеризуется включением в работу нашего подсознания. Для появления такого настроя участников мозгового штурма следует

проводить специальную разминку с задачами на анализ и синтез, ассоциативные связи и т.д.

Высказывая свои идеи, участникам необходимо помнить, что совершенно не имеет значения, применимы они на практике или нет, так или иначе, многие из них, возможно, помогут найти эффективное решение.

#### **Правило 3:** Выдвижение как можно большего количества идей

Как уже упоминалось, для проведения мозгового штурма наиболее важно количество высказанных идей, чем их качество. Та как генерировать идеи участники должны (и могут) в течение небольшого ограниченного времени, то они должны научиться использовать уже высказанные другими участниками идеи для быстрого обдумывания и предложения новых.

В практике работы таких групп можно отметить, что целью проведения мозгового штурма является выдвижение более 100 идей за 20 минут. Самым продуктивным (успешным) мозговым штурмом является тот, при проведении которого за 20 минут предлагается 200 — 250 идей.

#### **Правило 4:** Обязательная фиксация всех идей

При проведении мозгового штурма должна быть зафиксирована каждая идея, даже если она повторяется. Все участники группы должны видеть все зафиксированные идеи, поэтому следует заранее к этому подготовиться.

Обычно идеи записывают маркерами на больших листах бумаги. Развесить их лучше заранее, перед началом мозгового штурма и разместить на стенах таким образом, чтобы они были хорошо видны каждому участнику.

#### **Правило 5:** Инкубация идей

После того, как все идеи высказаны и зафиксированы, необходимо время для того, чтобы их обдумать и оценить. Зачем нужен этот этап? Дело в том, что инкубационный период позволяет человеку оправиться от усталости, связанной с решением проблемы. Перерыв в трудной проблеме позволяет также забыть несоответствующие подходы к ней.

Решению проблемы может мешать функциональная закрепленность, и не исключено, что во время инкубационного периода человек забывает старые и безуспешные способы ее решения. Опыт показывает, что в период инкубации человек продолжает работать над задачей бессознательно. Кроме того, во время перерыва в процессе решения проблемы может происходить реорганизация материала.

#### **Брейнрайтинг**

Эта методика основана на технике мозговой атаки, но участники группы выражают свои предложения не вслух, а в письменной форме [11]. Они пишут свои идеи на листках бумаги и затем обмениваются ими друг с другом. Идея соседа становится стимулом для новой идеи, которая вносится в полученный листок. Группа снова обменивается листками, и так продолжается в течение определенного времени (не более 15 минут).

Правила мозговой атаки распространяются и на записи мыслей: стремиться к большему количеству идей, не критиковать выдвинутые предложения до окончания занятий, поощрять «свободные ассоциации».

Рассмотрим пример.

Менеджеры парфюмерной фирмы решили применить метод записи мыслей в поисках новаторских идей для развития бизнеса. Каждый участник заседания записал свою идею на листке и обменялся с соседом. Один из менеджеров подумал о производстве нового сорта мыла и стирального порошка, в то время как другой внес в список предложение разработать новую линию по производству шампуня и бальзама для волос. Ну, а третий, когда к нему попал этот листок с этими двумя идеями, соединил их и предложил создать уникальный продукт: мыло, шампунь и кондиционер в одном флаконе.

### Мозговая атака на доске

В рабочих помещениях можно повесить на стене специальную доску, атака на доске чтобы сотрудники размещали на ней листки с записями тех творческих идей, которые придут им в течение рабочего дня. Повесить эту доску следует на видном месте. В центре ее должна быть написана — большими яркими (разноцветными) буквами — требующая разрешения проблема. Любой, у кого возникнет интересная мысль, способная помочь в решении данной проблемы, может приколоть на доску листок с зафиксированной на нем идеей.

### Управление жизненным циклом

Хотя управление и соответствие нормативным требованиям встречаются на протяжении всего жизненного цикла, однако их представление, область действия и цели зависят от конкретного этапа. Например, действия по управлению изменениями на этапах «Планирование» и «Эксплуатация» будут иметь другую значимость и отличаться по составу участников и используемым факторам.

### Внутренние меры контроля

Процедуры и меры контроля следует разделить между несколькими людьми, каждый из которых будет выполнять свою часть. В этом случае внутренние меры контроля должны обеспечить надлежащее объединение результатов, полученных разными людьми, и гарантировать, что никому не удалось уклониться от выполнения. В финансовых вопросах проблемы контроля являются еще более важными. Отсутствие эффективного контроля может привести к ошибкам в бухгалтерском учете или даже мошенничеству и хищению.

Внутренние меры контроля представлены во всех областях, с которыми работает ИТ-подразделение. Одни меры контроля предназначены для физической среды, в которой находится инфраструктура центров данных, а другие используются непосредственно для технологий (например, определяют конфигурацию и перечень лиц, которым предоставлен доступ к административным функциям). Некоторые меры контроля используются при доступе к данным и применяются на различных этапах жизненного цикла данных — от шифрования до авторизации, восстановления и защиты данных.

Подтверждением того, что ИТ-услуга фактически контролируется на протяжении всего жизненного цикла, является следующее:

- Определение общих целей для каждого этапа жизненного цикла
- Определение рисков, связанных с достижением этих целей

- Определение методов управления рисками в виде мер внутреннего контроля по смягчению последствий рисков

Руководство несет ответственность за выработку целей, оценку хода работ и достижение результатов. В частности, управление включает процессы принятия решений (меры контроля), помогающие руководству выполнять эти требования. Каждый этап жизненного цикла ИТ-услуги содержит одну или несколько процедур управленческого анализа, функционирующих как управленческие меры контроля. Это означает, что нужные люди будут собраны вместе в надлежащее время и обеспечены информацией, необходимой для принятия управленческих решений.

*Контроль исполнения проекта* - процесс сравнения показателей плановых и фактических показателей выполнения проекта, анализ отклонений и их причин, оценка возможных альтернатив и принятие, в случае необходимости, решений о корректирующих действиях для ликвидации нежелательных отклонений.

Контроль проекта может включать следующие процедуры:

- Сбор отчетности о ходе работ по проекту
- Анализ текущего состояния проекта относительно основных базовых показателей (результаты, стоимость, время)
- Прогнозирование достижения целей проекта
- Подготовка и анализ последствий корректирующих воздействий
- Принятие решений о воздействиях и изменениях

Организация контроля может следить за:

- Качеством работ
- Ходом и темпом работ
- Стоимостью и сроками

## Классификация программного обеспечения

Каким бывает ПО

Как и любой другой продукт человеческой деятельности, программное обеспечение может быть, как коммерческим, так и некоммерческим. **Коммерческое программное обеспечение** - программное обеспечение, созданное с целью получения прибыли от его использования другими, например, путем продажи экземпляров.

По способу распространения и использования программное обеспечение принято делить на следующие три основных вида:

- Несвободное (или проприетарное)
- Свободное
- Открытое

Рассмотрим их подробнее, дабы узнать различия между ними.

## Проприетарное ПО

**Проприетарное программное обеспечение** (*proprietary software*) — программное обеспечение, являющееся частной собственностью авторов или правообладателей и не удовлетворяющее критериям свободного ПО (наличия открытого программного кода недостаточно). Правообладатель проприетарного ПО сохраняет за собой монополию на его использование, копирование и модификацию, полностью или в существенных моментах.

Предотвращение использования, копирования или модификации могут быть достигнуты правовыми и/или техническими средствами.

Технические средства включают в себя выпуск только машинно-читаемых двоичных файлов, ограничение доступа к читаемому человеком исходному коду (закрытый исходный код), затруднение использования собственноручно сделанных копий. Доступ к закрытому коду обычно имеют сотрудники компании-разработчика, но могут применяться и более гибкие условия ограничения доступа, в которых предоставление исходного кода разрешено партнёрам компании, техническим аудиторам или другим лицам в соответствии с политикой компании. Правовые средства могут включать в себя коммерческую тайну, авторское право и патенты.

Надо заметить, что любое программное обеспечение по умолчанию является проприетарным, так как по умолчанию действует закон об авторском праве. Но этого можно избежать благодаря лицензированию.

## [Примеры проприетарного ПО](#)

Примером может послужить большая часть продукции таким компаний как *Microsoft* и *Adobe*.

## Свободное ПО

**Свободное программное обеспечение** (*free software*) — программное обеспечение, пользователи которого имеют права («свободы») на его неограниченную установку, запуск, а также свободное использование, изучение, распространение и изменение (совершенствование), и распространение копий и результатов изменения.

Отцом движения свободного программного обеспечения является Ричард Столлман. 27 сентября 1983 года в Массачусетском технологическом институте им был запущен проект GNU (*GNU IS NOT UNIX*). Изначальной целью проекта было «разработать достаточно свободного программного обеспечения <...>, чтобы можно было обойтись без программного обеспечения, которое не является свободным». В октябре 1985 Ричард основал Фонд свободного программного обеспечения (*Free Software Foundation*) для продвижения свободного программного обеспечения. А в феврале 1986 года им была опубликована статья "*The Free Software Definition*", где он описал, что под свободой ПО он имеет в виду свободу копировать, распространять и изменять его. Современная версия статьи, с переводом на 39 языков, опубликована на сайте Фонда свободного ПО. Согласно Столлману, «Свобода ПО» означает «право пользователя свободно запускать, копировать, распространять, изучать, изменять и улучшать его». Его современная версия определения свободы ПО состоит из четырёх пунктов, пронумерованных от 0 до 3:

- Свобода запускать программу в любых целях (свобода 0).
- Свобода изучения работы программы и адаптация её к вашим нуждам (свобода 1). Доступ к исходным текстам является необходимым условием.

- Свобода распространять копии, так что вы можете помочь вашему товарищу (свобода 2).
- Свобода улучшать программу и публиковать ваши улучшения, так что всё общество выигрывает от этого (свобода 3). Доступ к исходным текстам является необходимым условием.

Следует заметить, что не все организации согласны с определением Столлмана. Например, дистрибутив Debian использует свои собственные критерии для определения свободы ПО, которые несколько отличаются от четырёх пунктов Столлмана. В результате, Debian считает лицензию 1-ю версию Artistic License свободной, а GNU Free Documentation License с неизменяемыми разделами несвободной.

### *Примеры свободного ПО*

Вот некоторые примеры:

**Audacity** — свободный многоплатформенный аудиоредактор звуковых файлов, ориентированный на работу с несколькими дорожками. Программа была выпущена и распространяется на условиях GNU General Public License.

**Blender** — свободный, профессиональный пакет для создания трёхмерной компьютерной графики, включающий в себя средства моделирования, анимации, рендеринга, постобработки и монтажа видео со звуком, компоновки с помощью «узлов» (Node Compositing), а также для создания интерактивных игр. В настоящее время пользуется наибольшей популярностью среди бесплатных 3D редакторов в связи с его быстрым и стабильным развитием, которому способствует профессиональная команда разработчиков.

**Media Player Classic** (MPC) — свободный проигрыватель аудио- и видеофайлов для операционной системы Windows. Программа имеет интерфейс, аналогичный Windows Media Player версии 6.4, однако основан на совершенно другой кодовой базе.

Языки **Perl, PHP, Python, Ruby, Free Pascal, FreeBASIC**.

Фреймворки **Spring, Hibernate**.

### *Свободные операционные системы*

Конечно же всем известный **Linux**. В этом семействе ОС существует множество дистрибутивов. Вот самые популярные среди них:

- Mint
- Ubuntu (Canonical предлагает Ubuntu бесплатно, но также продаёт коммерческую техническую поддержку)
- Debian
- Fedora
- Mageia

Конечно же существуют и менее известные ОС.

**BSD** (англ. Berkeley Software Distribution) — система распространения программного обеспечения в исходных кодах, созданная для обмена опытом между учебными заведениями.

К семейству BSD относятся: NetBSD, FreeBSD, OpenBSD, ClosedBSD, MirBSD, DragonFly BSD, PC-BSD, DesktopBSD, SunOS, TrueBSD, Frenzy, Ultrix и частично Darwin (ядро Mac OS X).

**Darwin** — это открытая POSIX-совместимая операционная система, выпущенная Apple Inc. в 2000 году. Она совмещает код, написанный самой Apple, с полученным от NeXTSTEP (система выпущена в 1989), FreeBSD (выпущена в 1993) и прочих свободных проектов.

Darwin — наследник разработанной в NeXT операционной системы NeXTSTEP, первая версия которой вышла в 1989 году. После того, как Apple поглотила NeXT в 1997 году, она объявила, что сделает свою следующую операционную систему на основе OpenSTEP API системы NeXTSTEP. Эта система разрабатывалась в рамках проекта Rhapsody с 1997 года и в 1999 году вышел основанный на этих разработках Mac OS X Server 1.0. В 2000 году Rhapsody был выделен в Darwin, выпущенный как свободное программное обеспечение в рамках публичной лицензии на исходники Apple (APSL) и компоненты Darwin присутствуют в Mac OS X по сей день.

**OpenSolaris** — операционная система с открытым исходным кодом, созданная корпорацией Sun Microsystems на базе Solaris.

Процесс разработки OpenSolaris ведётся на добровольной и неоплачиваемой основе сообществом разработчиков OpenSolaris, однако направляется и координируется с участием специалистов Sun. При этом установлен чёткий протокол организации разработки — так называемый OpenSolaris Community Process.

**ReactOS** — международный проект свободной и бесплатной операционной системы с открытым кодом. ReactOS не является точным клоном Windows, но операционной системой, совместимой с приложениями и драйверами Windows.

## Открытое ПО

**Открытое программное обеспечение** (*open-source software*) — программное обеспечение с открытым исходным кодом. Исходный код таких программ доступен для просмотра, изучения и изменения, что позволяет пользователю принять участие в доработке самой открытой программы, использовать код для создания новых программ и исправления в них ошибок — через заимствование исходного кода, если это позволяет совместимость лицензий, или через изучение использованных алгоритмов, структур данных, технологий, методик и интерфейсов (поскольку исходный код может существенно дополнять документацию, а при отсутствии таковой сам служит документацией).

Термин open-source был введен в 1998 г. Эриком Реймондом и Брюсом Перенсом, которые утверждали, что термин свободное программное обеспечение неоднозначен и отпугивает коммерческих предпринимателей. В феврале этого же года ими была основана организация *Open Source Initiative*, посвящённая продвижению открытого программного обеспечения. Эта организация занимается определением степени соответствия лицензии на программное обеспечение стандартам открытого программного обеспечения. Что интересно, эти стандарты основываются на директивах Debian для свободного программного обеспечения (*Debian Free Software Guidelines*), которые большей частью написаны Брюсом Перенсом.

Требования к лицензиям на открытое ПО в редакции *Open Source Initiative*:

- **Свободное распространение.** Это значит, что лицензия не должна налагать ограничений на продажу и распространение ПО.

- **Доступные исходные тексты.** Даже если ПО не поставляется с исходными текстами, эти тексты должны быть легко доступны. Это должны быть именно редактируемые человеком исходные тексты, а не выход обфускаторов, препроцессоров и тому подобные промежуточные формы.

Таким образом, freeware не является open-source.

- **Возможность модификации.** Простая возможность читать исходные тексты не позволяет экспериментировать с ними и выпускать модификации. Лицензия, претендующая на звание «открытой», должна разрешать не только чтение кода, но и модификацию, использование частей кода в других проектах и распространение получившихся программ на условиях той же лицензии.

Компания id Software выпустила исходные тексты (но не данные) Doom в 1998 году под «образовательной» лицензией. Через год тексты были перелицензированы под GPL.

- **Даже в случае неприкосновенности авторского исходного текста, производные программы и их исходные тексты должны свободно распространяться.** Чтобы не запутывать пользователя, свободные лицензии могут оставлять за автором какие-то права — например, производная программа обязана нести другое имя или версию; либо она должна состоять из авторских исходных текстов и патчей к ним. Тем не менее, автор должен разрешать распространять откомпилированные двоичные файлы и исходные тексты производной программы в том или ином виде.

Компания Netscape, выпуская исходные тексты браузера, оставила имя Netscape за собой. Несмотря на этот пункт, Mozilla Public License является открытой.

- Отсутствие дискриминации против людей и групп людей. Некоторые страны, например, США, имеют некоторые ограничения на экспорт ПО. Свободная лицензия может напоминать, что такие правила есть, но не может ставить свои.

Одна из «почти открытых» лицензий, созданных во время апартеида, запрещала использование программы полицией ЮАР. Апартеид пал, а требование осталось.

- **Отсутствие дискриминации по цели применения.** Свободная лицензия должна разрешать все виды деятельности, включая генетические и ядерные исследования, коммерческое применение и т. д. Про коммерческое применение говорится особо: «Мы хотим, чтобы коммерческие пользователи подключались к сообществу, а не считали себя отрезанными от него».

Как и со свободным ПО, личные убеждения автора не должны мешать делу, и пункты наподобие «нельзя использовать в клиниках для абортов» запрещены. Ведь один может запретить абORTы, другой — ругательства, третий — и то, и другое, а четвёртый — какой-нибудь из этих запретов, ничего от свободы не оставил.

- **Распространение лицензии.** Права, связанные с открытым ПО, должны быть применимы ко всем пользователям программы без заключения дополнительных соглашений, например, соглашения о неразглашении.

- **Лицензия не должна быть привязана к конкретному продукту.** Права на программный код не должны зависеть от того, является ли программа частью какого-то продукта. Человек, распространяющий программу в отрыве от сборника или перенёсший часть кода в другой продукт, имеет такие же права, какие давал сборник. Это требование закрывает некоторые лицензионные лазейки.

ReactOS и Wine активно обмениваются кодом. На основе ядра Linux строят прошивки различных устройств. Это возможно, потому что ни одна строчка кода, ни один файл исходного текста не привязан ни к какой программе.

- **Лицензия не должна ограничивать другие программные продукты.** За исключением банальной несовместимости, пользователь имеет право выбирать, чем пользоваться. Например, нельзя требовать, чтобы остальные программы, поставляемые вместе с данной, также были открытыми.

Свежие версии Ghostscript имели лицензию, которая запрещала использовать программу вместе с закрытым ПО (устаревшие версии выпускались под GPL). От этой практики отказались в 2007 году. Часть лицензий Microsoft Shared Source допускают создание ПО только под Windows.

- **Лицензия должна быть технологически нейтральной.** То есть, лицензия не должна требовать что-либо от интерфейса или технологий, применяемых в производной программе.

Например, непригоден пункт «пользователь должен принять лицензию, нажав на определённую кнопку» — это не даст использовать ПО в режиме командной строки без участия пользователя. Этот пункт также служит для того, чтобы закрыть лицензионные лазейки.

Существуют также программы, исходный код которых можно видеть, но которые не подходят под определения открытого или свободного ПО, например, UnRAR, распаковщик RAR-архивов. Его исходный код находится в открытом доступе, но лицензия запрещает использовать его для создания RAR-совместимых архиваторов. Другим популярным примером может быть программа шифрования TrueCrypt: её лицензия отнюдь не свободная, но исходный код при этом открыт, хотя менять его нельзя, можно лишь проверять работоспособность и «честность».

#### *Примеры открытого ПО*

**Hadoop** — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов. Используется для реализации поисковых и контекстных механизмов многих высоконагруженных веб-сайтов, в том числе, для Yahoo! и Facebook. Разработан на Java в рамках вычислительной парадигмы MapReduce, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат. В этом году EMC, Oracle и даже Microsoft объявили о коммерческой поддержке или производстве продуктов, которые работают с Hadoop, а Yahoo отложил HortonWorks, чтобы сфокусироваться на Hadoop. Легче перечислить те компании, которые не работают с Hadoop, чем те, которые пользуются данным фреймворком.

**Git** — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Программа является свободной и выпущена под лицензией GNU GPL версии 2.

**Apache Cassandra** — распределённая система управления базами данных, относящаяся к классу noSQL-систем и рассчитанная на создание высокомасштабируемых и надёжных хранилищ огромных массивов данных, представленных в виде хэша.

**jQuery** — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML.

**Node** или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

**Apache OpenOffice** — свободный пакет офисных приложений.

#### [В чём же разница?](#)

Отличие между движениями открытого ПО и свободного ПО заключается в основном в приоритетах. Сторонники термина «open-source» делают упор на эффективность открытых исходников как метода разработки, модернизации и сопровождения программ. Сторонники термина «free software» считают, что именно права человека на свободное распространение, модификацию и изучение используемых им программ являются главным достоинством свободного открытого ПО.

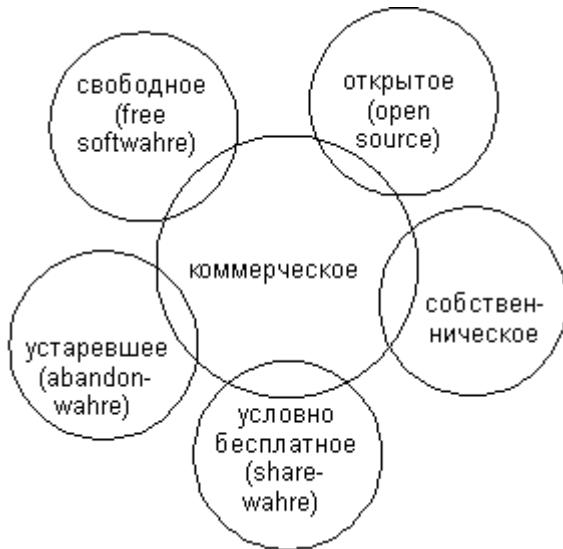
Опять-таки, приведённые выше определения свободного и открытого ПО придуманы конкретными организациями. И, при этом, данные термины не являются чьим-либо товарным знаком. То есть вся эта классификация достаточно условна.

Как уже говорилось, это битва взглядов и приоритетов. Если сподвижники свободного ПО всеми силами борются за права и свободы пользователя, забывая о правах и свободах разработчиков, то приверженцы открытого ПО стараются дать свободу и возможности как раз-таки разработчикам. Что ближе для вас, решать вам.

#### [Немного о заблуждениях](#)

Множество людей ошибочно считают коммерческое и свободное противоположностями. Но это мнение ошибочно, ведь свободное ПО может быть платным, а proprietарное может быть бесплатным (**freeware**). Хорошими примерами коммерческих программ, относящихся к разряду свободных, могут служить компилятор GNU ADA или многие операционные системы на основе GNU/Linux. Существует много бизнес-моделей, где не надо платить за каждую копию ПО, например, платная сервисная поддержка или коммерческая лицензия для использования свободного кода в собственническом ПО.

То есть, и proprietарное, и свободное, и открытое ПО может быть, как коммерческим, так и некоммерческим. Всё зависит от лицензирования конкретного продукта и выбранной бизнес-модели.



### О лицензиях

В соответствии с современным законодательством большинства стран, программный продукт и его исходный код охраняются авторским правом, которое даёт авторам и правообладателю (чаще всего правообладателем является организация-наниматель автора служебных произведений) власть над изменением, распространением, способом использования и поведением программы, включая случаи, когда исходный код опубликован. Сила власти авторских прав в современном обществе настолько велика, что даже изучение или попытки исправления ошибок программ путём дезассемблирования могут преследоваться уголовным правом.

Чтобы избавить пользователей программ от проблем, вызванных перекосом законодательства об охране результатов интеллектуальной деятельности в сторону правообладателя, авторы и правообладатели могут передать пользователям права на четыре вышеперечисленные свободы действий. Это достигается путём выпуска исходного кода программного обеспечения на условиях одной из особого рода лицензий, называемых **свободными лицензиями**.

Выделяют три группы свободных лицензий по масштабу ограничений:

#### Основанные на общественном достоянии

Максимальная свобода, в таких странах как США можно даже не указывать авторство при использовании такого контента

#### Пермиссивные (разрешительные)

Пермиссивные лицензии на свободное ПО — лицензии на программное обеспечение, которые практически не ограничивают свободу действий пользователей ПО и разработчиков, работающих с исходным кодом. В частности, пермиссивные лицензии сами по себе не ограничивают выбор лицензии для работ, производных от работы с пермиссивной лицензией.

Некоторые примеры:

- Лицензия BSD
- Лицензия MIT

- Mozilla Public License
- Creative Commons Attribution
- Apache

Лицензия MIT (англ. MIT License) — лицензия свободного программного обеспечения, разработанная Массачусетским технологическим институтом (МТИ). Лицензия MIT является одной из самых ранних свободных лицензий, так как она относительно проста и иллюстрирует некоторые из основных принципов свободного лицензирования. Она является разрешительной лицензией, то есть позволяет программистам использовать лицензируемый код в закрытом ПО при условии, что текст лицензии предоставляется вместе с этим ПО. Лицензия является GPL-совместимой, то есть разрешает программистам комбинировать и распространять GPL-продукты с программным обеспечением под лицензией MIT.

## Копилефтные

Копилефт лицензия — лицензия, которая:

- позволяет использовать оригинальные (исходные) работы при создании новых (производных) работ без получения разрешения владельца авторского права;
- требует, чтобы два пункта этого списка присутствовали в лицензии производной работы.
- Используя «копилефт» лицензии авторы и правообладатели предоставляют права на распространение копий оригинального произведения и его изменённых версий. Авторы производного произведения обязаны распространять его с сохранением тех же самых прав.
- Некоторые из них:
- GNU General Public License (GNU GPL)
- GNU Lesser General Public License
- GNU Affero General Public License

GNU General Public License (переводят как Универсальная общественная лицензия GNU, Универсальная общедоступная лицензия GNU или Открытое лицензионное соглашение GNU) — лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г., по которой автор передаёт программное обеспечение в общественную собственность.

Особенностью общественной лицензии GNU является наличие правила «копилефт», которое представляет собой условие распространения свободного ПО: ни один пользователь не имеет права, сделав модифицированную версию свободной программы, распространять ее, не соблюдая всех принципов свободного ПО. То есть нельзя модификацию свободной программы сделать несвободной. По этой причине лицензию GNU прозвали «вирусной лицензией»: она как бы «заражает» программу, становясь ее неотъемлемой частью.

Свободные лицензии так же могут разделяться на лицензии для свободного и открытого ПО. Но такое разделение опять-таки очень условно, так как лицензии на открытое ПО часто совпадают с лицензиями на свободное ПО.

Нарушение лицензий тех или иных программных продуктов преследуется законом. То есть, вам может грозить судебное разбирательство и денежный штраф. Поэтому очень важно законно (в

соответствии с лицензией) использовать программное обеспечение, а также правильно лицензировать собственные продукты. И необходимо быть очень внимательным при работе с нелицензованными продуктами.

Саймон Фиппс (Simon Phipps), президент организации Open Source Initiative (OSI), ранее руководившего направлением open-source в компании Sun Microsystems, опубликовал заслуживающее внимания исследование, согласно которому половина проектов на GitHub опубликованы без явного указания лицензии. Если проект опубликован без лицензии, то по умолчанию все права на код остаются собственно автором (All rights reserved). Попытка интеграции подобного кода в сторонние проекты может негативно отразиться на их лицензионной чистоте и в последствии привести к возможным юридическим претензиям. GitHub представил новый сайт [choosealicense.com](http://choosealicense.com), созданный для упрощения принятия решения по выбору той или иной лицензии при создании репозитория с кодом. На сайте в краткой форме описаны особенности основных открытых лицензий. Кроме сайта, на основной странице регистрации нового репозитория в GitHub появилась форма выбора лицензии, позволяющая автоматически сформировать файл с выбранным типом лицензии (ранее текст лицензии нужно было копировать вручную).

### Что нас ждёт?

С 2006 года тестинговой компанией Coverity совместно с американским Отделом национальной безопасности проводили исследования как в открытом, так и закрытом секторе разработки ПО, по результатам года они публиковали отчёт. По результатам 2011 года оказалось, что открытый исходный код не уступает по качеству проприетарному. Самыми качественными проектами были признаны Linux 2.6, PHP 5.3 и PostgreSQL 9.1, качество которых определялось по дефектной плотности (числу дефектов на тысячу строк кода), которая была равна 0.62, 0.20, и 0.21 соответственно.

По мнению аналитической фирмы Gartner, в ближайшие годы многие компании станут смешивать проприетарное ПО и ПО с открытым исходным кодом.

Слиянию проприетарного и open-source ПО на сервере отчасти способствует изменение отношения Microsoft, которая недавно начала смиряться с тем, что Linux и ПО open source не исчезнут. «В последние два года Microsoft как организация заметно изменилась и стала внутренне более зрелой, — сказал Досон. — Эти существенные перемены в Microsoft означают появление систем, в которых на платформе Windows исполняются такие открытые приложения, как Apache. Аналогично, комплекс ПО open source не надо рассматривать как комплекс ПО, работающего исключительно на Linux. Мы наблюдаем, что основным направлением роста являются коммерческие приложения на Linux».

Согласно аналитическому отчету компании 451 Group, за последние 3 года количество приобретений компаний, занимающихся разработкой свободного ПО удвоилось. Сохранится ли эта тенденция в обозримом будущем? Другая компания, Gartner Inc, на днях опубликовала прогноз, согласно которому, к 2012 году около 80% всего ПО будет распространяться под свободными лицензиями. Gartner связывает это с неуклонным ростом числа программистов, способных работать над проектом, используя Интернет в качестве коммуникационной среды. Еще одна важная причина заключается в желании сэкономить. Аналитики Gartner уверены, что экономическая выгода и легкость разработки свободного ПО неминуемо приведут к вытеснению коммерческих программ. Уже сейчас мы можем наблюдать не только рост числа свободных проектов, но и заметное

улучшение их качества: дистрибутивы Linux стали проще устанавливаться и легче обновляться. Многие коммерческие компании стали использовать Linux и свободное ПО в своих устройствах и программах.

В прошлом году Гейб Ньюэлл, руководитель компании Valve сделал ставку на Linux как игровую платформу. Вскоре компания начала воплощать в жизнь новую стратегию: вышел Steam под Linux, а с ним все части Half-Life 2, Left 4 Dead 2, Portal и многие другие игры. Разработчики Valve объяснили, почему Linux с технической точки зрения — более предпочтительная платформа для игр, чем Windows 8.

«Немного странно мне выступать здесь и рассказывать вам, ребята, что за Linux и open source будущее игровой индустрии, — сказал Гейб Ньюэлл. — Это вроде как приехать в Рим и учить католицизму Папу Римского».

Ньюэлл и раньше называл Windows 8 «катастрофой для индустрии персональных компьютеров», и подтвердил эти слова сейчас. Закрытые платформы будут уступать место открытым, считает он.

Как мы видим, сфера открытого программного обеспечения очень перспективна и востребована. Она даёт пользователям и разработчикам многие возможности и преимущества, на которые не способен проприетарный софт. Впрочем, последний никто не спешит списывать со счетов.

## Глава 2. Менеджмент IT-проекта

### Менеджмент IT-проекта

Менеджмент, как техпроцесс, является основным и неотъемлемым фактором развития проектов.

В подавляющем большинстве случаев для стартапов нанять опытного менеджера представляется сложным — услуги достойного специалиста стоят недешево, да и доверять на раннем этапе постороннему лицу участникам стартапа будет сложно. Поэтому менеджментом стартапов занимаются, как правило, сами участники проекта.

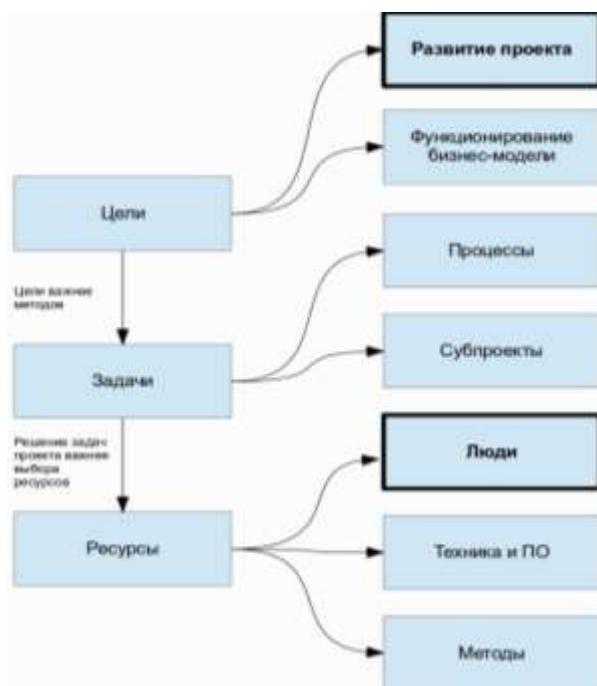
Кратко обозначены, выделены и пояснены основные, наиболее важные моменты.

- Чем управлять
- Основной процесс менеджмента
- Нет делегирования? И это хорошо
- Малая боевая диверсионная группа
- Системный анализ
- Divide et Empera
- Логгирование
- Оценка множественных и неявных факторов
- Время, как основной фактор
- Два в одном
- Риски

#### Чем управлять

Менеджмент — это управление. В нашем случае, управление проектом.

Понятно, что управление проектом — это работа над его составляющими.



Важнейшие исключения из любых правил.

Основная ценность — это люди.  
Основной приоритет — развитие проекта.

## Основной процесс менеджмента

Менеджер работает с процессами. Процессы являются составной частью проектов. Процесс может быть разовым или непрерывным, но он в любом случае итеративен. Это означает, что у каждого процесса есть циклические свойства — он легко может быть повторен, и даже для начала нового процесса возможно применять наработанный опыт — академические методики, личный опыт, опыт коллег и так далее.

До начала процесса необходимо формализовать исходные данные и выделить цели.

Этап анализа является опциональным. Он проводится, в зависимости от масштабов и цены процесса. Если процесс дорогой — все исходные данные подвергаются детализации, информация дополняется схемами и резюме.

На этапе планирования выбираются методы решения задачи, определяется, как именно будет осуществляться процесс.

Для обеспечения корректности приемки еще на этапе планирования составляется чеклист — список критериев, который однозначно дает понять, что проект завершен.

Естественно, исполнителю должна быть доступен максимальный объем информации, связанный с процессом, в котором он участвует — исходные данные, цели и требования в чеклисте.

Если процесс не является непрерывным — по достижению целей он может быть завершен.

При повторном выполнении процесса к исходным данным добавляются результаты предыдущей итерации.



Нет делегирования? И это хорошо

Один из самых популярных и в тоже время противоречивых методов традиционного менеджмента — делегирование. Есть куча академической информации о том, как, кому и когда поручать задания. В условиях стартапа, как правило, делегирование в общем понимании недоступно. Слишком мало денег, слишком мало людей.

Непосредственных участников у новорожденного проекта, как правило, мало. Нанимать экспертов со стороны — дорого, да и к тому же чревато утечкой информации и дополнительными временными затратами.

Поэтому, для обеспечения эффективности менеджмента (удачного управления проектом) целесообразно уделять большее внимание другим доступным методикам:

- **системный анализ** (методологию, теорию и практику исследования систем), которая исследует методологические, а часто и практические аспекты и использует практические методы (математическая статистика, исследование операций, программирование и др.);
- **интерактивный контроль** (*широко используется в строительной сфере*);
- **управление рисками** (принятие и выполнение управлеченческих решений, направленных на снижение вероятности возникновения неблагоприятного результата и минимизацию возможных потерь проекта, вызванных его реализацией).

### Малая боевая диверсионная группа

На самом деле работа в условиях ограниченных ресурсов является более эффективной.

1. Избыточные ресурсы расхолаживают
2. В малой группе короче и эффективнее коммуникации
3. Малую группу легче настроить на цель
4. В малой группе эффективнее контролируются процессы
5. В малой группе проще охранять коммерческую тайну

Сильные корпорации, такие как Google, используют метод малых групп для решения практически всех ключевых задач. Выделяется коллектив заинтересованных специалистов, который работает над проектом. Как показывает опыт, задачи решаются, и проекты «выстреливают». Не надо стесняться малого размера вашей команды. Вообще, никаких комплексов! Только энтузиазм, только объективизм.

### Системный анализ

Благодаря методу системного анализа малые рабочие группы решают сложнейшие задачи. Причем делают это быстро и дешево.

Я не призываю вас создавать тонны трудночитаемой документации. Но использование даже некоторых основных методик даст вашему проекту жизнь. Вот эти методики:

- разделение задач на подзадачи;
- выделение **субпроектов**;
- запись (**логгирование**) всего.

Цели системного анализа таковы:

- получение **прозрачного и очевидного** представления всех **деталей** проекта;
- выявление потенциально **узких** мест;
- выявление скрытых факторов, прежде всего **затрат**;
- **согласование тактического и стратегического видения между всеми участниками проекта.**

### Divide et Empera

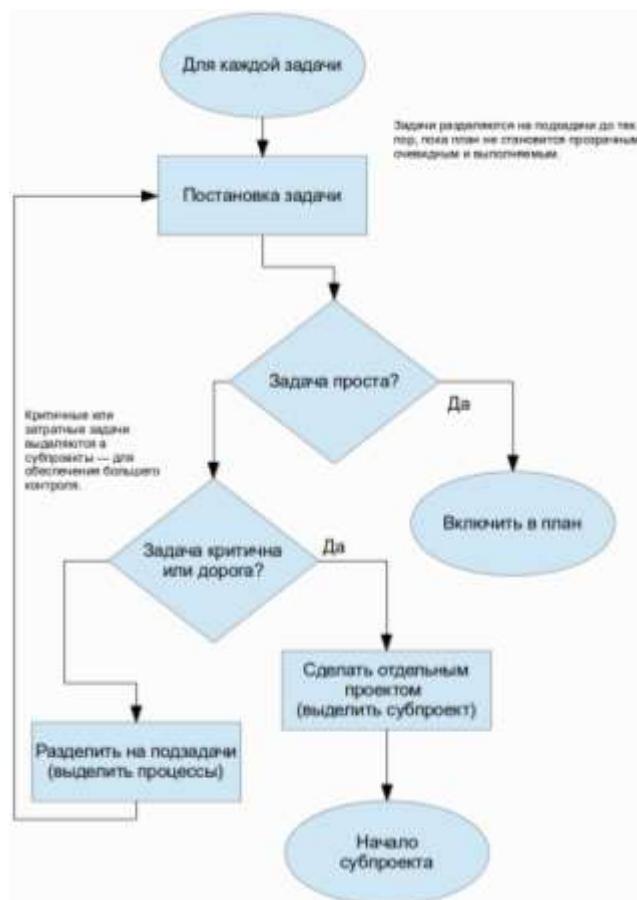
«Разделяй и властвуй», завещали нам древние правители. И по сей день эта методика управления проектами оказывается одной из самых эффективных.

Методику разделения задач на подзадачи можно применять как для разработки рабочего плана, так и для анализа других аспектов и ситуаций.

Для каждого аналитического элемента, для каждого фактора и подзадачи формализуются три вопроса: цель, объекты и методы. Оценивается масштаб, важность и сложность задачи. Если задача является сложной, масштабной или критичной — она делится на подзадачи или выделяется в отдельный проект.

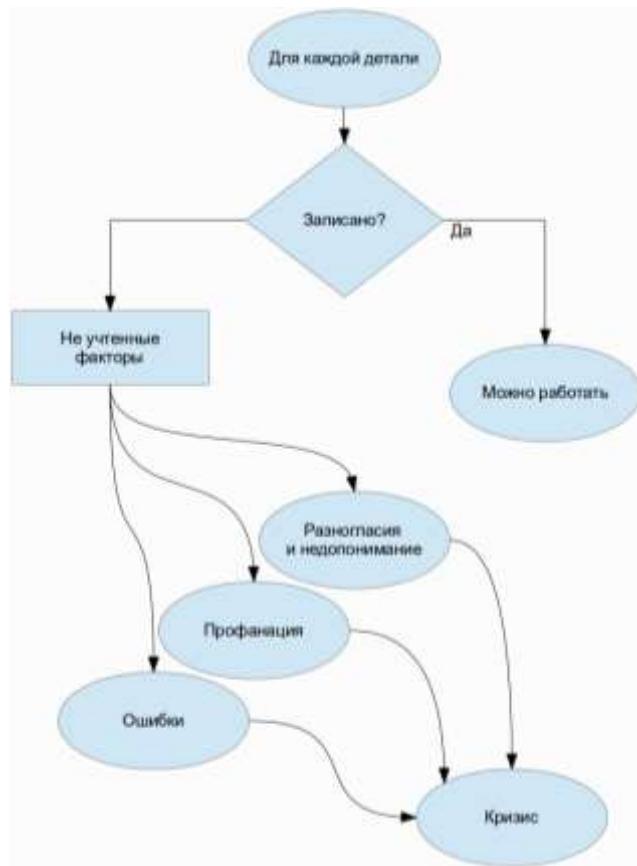
Простой является та задача, которая содержит очевидно мало деталей. Отсюда легко получить обратный тезис — любая задача, не признанная простой, должна быть признана сложной, и подвергнутся разделению.

Совершенно正常но, когда в результате разработки плана действий содержание проекта пересматривается.



## Логгирование

Записывать имеет смысл все и всегда. В текстовом редакторе, в Evernote или в специализированном ПО — не важно. Главное — записывать, и регулярно осведомляться о том, что с записями осведомлены все ключевые участники проекта.



## Оценка множественных и неявных факторов

При оценке рисков, а также выборе стратегических и тактических решений нужно оценивать неизвестные, неявные или сложные (многокомпонентные) факторы.

В рамках низкобюджетного стартапа сложно, и зачастую просто бессмысленно проводить экспертизы. Для принятия решений есть достаточно эффективные низкозатратные методики.

Одним из таких приемов является применение **оценок-весов**.

Составляется таблица. По горизонтали — варианты решения, включая гипотетические. По вертикали — факторы. В ячейках проставляются субъективные оценки — степень влияния факторов на эффективность рассматриваемого решения.

Использовать результаты можно по-разному, суммируя баллы в различных направлениях и по разному принципу, находить среднее и т. д. Таким образом можно оценивать как методологии, так и варианты выбора в самых различных ситуациях.

Еще этот способ является одним из средств разрешения разногласий между участниками проекта. Спорные варианты решения проблемного вопроса скрупулезно детализируются, из полученных

тезисов и вопросов составляется анкета, и каждый из участников заполняет свой вариант. Лидирующий вариант выявить несложно, и участники скорее всего согласятся с целесообразностью его выбора.

### Время, как основной фактор

Называя что-то «дорогим», программист подразумевает затраты ресурсов машины или сети. Аналогично, менеджер ИТ-проекта имеет в виду время. Буквально — время является основным измерением, для которого можно применять оценки вида «дорого» или «приемлемо».

Если ваш проект требует софта на десятки тысяч долларов, или это игровой проект, где нужно оплачивать тысячи работ дизайнёров — это не столь страшно. В рамках анализа инвестиционного проекта эти затраты могут быть оценены, и адекватно сопоставлены с прибылью.

**Но самой дорогой тратой, и одной из самых серьезных неприятностей, которые могут случиться с проектом, будет затягивание времени.**

Везде и всегда имеет менеджеру проекта смысл искать пути и способы экономии времени. Для этого послужит и **оценка приоритетов**, и более глубокий анализ с целью поиска узких мест и фактической минимизации трудозатрат, **планирование и контроль** работ на разных уровнях.

Хорошими способами управления и оптимизацией временных затрат является **поэтапность**, и возможно более скорое открытие проекта в меньшем масштабе, с прицелом на дальнейшее развитие. На самом деле, практически в любом плане часть действий можно перенести из подготовительных в боевые, или под другим предлогом — просто отложить. Чтобы сэкономить самое дорогое — время.

### Два в одном

Подготовительные мероприятия и проект в рабочем режиме — это два разных проекта!

Естественно, работать с ними имеет смысл по отдельности. Естественно, сначала имеет смысл спроектировать второй, и только потом — первый. В подавляющем большинстве стартапов совершаются лишние действия, лишние траты. А их не так уж сложно уменьшить — если сначала сосредоточиться на вопросе «что мы хотим иметь», и только потом — «как мы это хотим получить».

Пока проекта нет — все действия до рентабельности относятся к первой части. Когда проект работает — все действия относятся ко второй. Естественно, при этом работают разные механизмы, совершенно разная специфика, поэтому и работать с этими этапами нужно по-разному.

**Маркетинг и бизнес-модель**, безусловно, относятся ко **второй части**. Туда же можно отнести большую часть итеративных затрат (в том числе **реклама и техническое обеспечение**).

**Пакет действий, необходимый от нуля до открытия проекта — в первую часть.**

Кстати, легко отличить опытного ИТ-предпринимателя от дилетанта. Эксперты в банках и инвестиционных фондах часто пользуются в том числе и этим методом.

Опытный как правило не допускает пробелов в представлении, и имеет четкое видение уже работающего проекта. Описание бизнес-модели у него никак не может ограничиваться одним-двумя предложениями, потому что ему уже известны масса деталей его и конкурентных проектов, а также специфика отрасли.

Неопытные всегда акцентируют внимание на действиях, связанных с открытием проекта, но имеют пространное представление о дальнейшем развитии проекта и нормальном режиме его работы. Например, вот некоторые из различий в предметной сфере, показывающие разницу между проектами-этапами:

Подготовительные работы	Основной процесс
Покупка ПО или программирование	Поддержка технического обеспечения
Дизайн	Бизнес-модель, план доходов и план расходов
Исследования, первое планирование маркетинга и рекламы	Маркетинг и выполнение плана рекламы
Определение состава специалистов	Фонд Оплаты Труда

Оба процесса имеют итеративные признаки и общие объективные черты, но в то же время имеют массу различий.

Эксперта, мнение которого возможно будет учитывать инвестор, будет интересовать следующее.

- Раздельные оценки затрат для двух этапов
- Раздельные временные оценки (время до получения прототипа, альфы, беты, открытие, время до рентабельности)
- Оценка бизнес-модели и ее детальных параметров
- Оценка затрат на маркетинг и рекламу до открытия проекта и в процессе

Календарный план-график обычно составляется по кварталам.

В процессе бизнес-планирования имеет смысл попытаться спрогнозировать развитие проекта хотя бы на 1-2 года вперед.

### Риски

Риски есть везде и всегда, и главная задача менеджера следить за этими рисками, минимизировать их насколько возможно. А риски и проблемы могут встречаться такие:

- Непопадание в область целевой аудитории
- Некорректное определение целевой аудитории
- Недостаточная активность аудитории
- Неудачная бизнес-модель
- Срыв сроков в разработке
- Невовлеченность участников проекта
- Проекты с бюджетом менее \$1000
- Дизайн
- Непопадание в тренд

- Спалить тему
- «Юноша бледный, со взором горящим»
- Изобретение велосипедов

Остановимся подробнее на каждом из них, чтобы у будущих менеджеров было чёткое понимание проблем.

#### *Неподание в область интересов целевой аудитории*

Для того, чтобы как можно вернее определиться с тем, будет ли проект интересен конкретной группе людей, которые будут платить или кликать на баннеры, достаточно провести небольшое исследование.

Во-первых, как можно более точно определить целевую аудиторию. Как правило, это одна или несколько групп людей, которых отличает определенный возраст, пол, география проживания, социальный статус, культурные особенности, правовые детали и главное — область интересов.

Понять, что определение целевой аудитории неудачно, очень просто — когда оно содержит слово «все». Увидеть удачное определение целевой аудитории также несложно — как правило, в нем присутствуют конкретные данные, и оно содержит весьма подробные описания нескольких групп.

Хороший материал по этому вопросу можно прочесть в книгах Филиппа Котлера.

После того, как целевая аудитория определена, становится следующая суперинтересная задача. Найти этих людей. Мы живем в веке высокоразвитых социальных сетей, поэтому найти сообщества-форумы по интересам проблем не составит.

Обратите внимание также на то, что реальные группы, подходящие под определенные вами требования, могут и не быть найдены в тематических сообществах. Вывод о размере одной из групп людей, которых заинтересует ваш проект, можно сделать, оценивая проекты, возможно далекие по тематике, но совпадающие по целевой аудитории.

Подсчитав количество участников в реально найденных вами группах, можете получить представление и о прогнозируемом размере будущей аудитории вашего проекта.

#### *Некорректное определение целевой аудитории*

Выявляется посредством интервью среди представителей ЦА (целевой аудитории). Подробности есть ниже.

Простой способ застраховаться от этой ошибки — собрать достаточное количество информации, изучать мнения, интервьюировать ключевых участников. Учесть полученные мнения! Не только себя слушать, но и критикующих. Принимать их позицию не обязательно, но выслушать и зафиксировать информацию — в ваших же интересах.

И здесь, и далее по материалу одним из основополагающих является принцип, популяризованный Суворовым «Тяжело в учении, легко в бою». Ну, или вороном из мультика «Крылья, Ноги и Хвосты» — «Лучше день потерять, потом за пять минут долететь».

Понимаю, что лень. Понимаю, что в лом. Понимаю, что местами сложно. Но по факту наблюдений за многочисленными стартапами — потом будет еще сложнее. И еще ленивее. И с огромной степенью вероятности — «уже поздно».

### *Недостаточная активность аудитории*

«Нас 3 миллиона», и при этом на сайте не более трех-пяти сотен активных участников, которые пользуются и платят деньги. Частый случай, хорошо знакомый многим из читателей.

Над лояльностью аудитории следует работать, как над отдельным проектом. В помощь — средства «социализации», джентельменский набор, создание групп в социальных сетях и тем в близких форумах, обязательно несколько сильных копирайтеров или вовлеченных участников.

Важный момент — удобная обратная связь. Стартапер должен часто выступать с инициативой, и расспрашивать участников о сервисе, о том, как они его используют.

Повышению лояльности способствуют конкурсы, встречи и развлечения, партизанская работа по развитию социальных групп. Также можно просто выкладывать фотографии с встреч и мероприятий. Работает, проверено.

Имеет смысл подружиться с наиболее активными участниками, и регулярно советоваться с ними. Спросите у них, почему они платят деньги? То, что вы услышите, скорее всего будет далеким от целей, которые вы рассматривали и прорабатывали на этапе планирования проекта, но будет являться отличным материалом для дальнейшего развития.

### *Неудачная бизнес-модель*

Самый популярный диагноз. Удивительно, но взрослые, и совершенно серьезные люди инвестируют стартапы деньгами и собственным временем, не посвятив достаточного внимания бизнес-модели.

Есть буквально тысячи случаев, когда интернет-проект раскручен, аудитория есть, а денег нет.

Что такое бизнес-модель? Это буквально метод, которым проект получает деньги. Часто называют термином «Монетизация». При проработке бизнес-модели, данные, полученные в результате маркетингового исследования, с учетом конверсии, можно применить для расчета прибыли. Должны быть ответы на два вопроса: «как» и «сколько». Причем план прибыли может быть относительным, но обязан быть календарным.

В общем случае, удачными оказываются бизнес-модели:

- просчитанные;
- популярные, или коррелирующие с популярными методами «монетизации».

Просчитать очень просто. В электронных таблицах рассчитайте количество чистых денег, полученное от одного довольного клиента, умножьте на прогнозируемое количество посетителей (не стоит в расчетах брать конверсию выше 1%, ой не стоит...), и рассчитайте календарный план график. Какая выручка ожидается в месяц, квартал, год?

Сложным схемам имеет смысл предпочесть традиционные:

- абонентская плата;
- непосредственная выручка от продаж товаров/услуг;
- фримиум; [Суть бизнес-модели **ФРИМИУМ** (*free* - свободный и *premium* - дорогой) - в том, что доступ к базовым функциям потребитель получает **бесплатно**, а к более продвинутым за дополнительную абонентскую плату]

- продажа рекламы;
- другие простые схемы (google в помощь).

Сложные схемы работают тогда, когда они внедряются в уже работающий проект, в процессе оптимизации базовой бизнес-модели. Не знаю ни одного интернет-проекта, который смог выжить со сложной схемой монетизации со старта. Знаю десятки проектов с замудренными схемами, которые не окупили хостинг.

#### *Срыв сроков в разработке*

Во-первых, отдайте приоритет прототипированию. Подробности есть ниже.

Во-вторых, до реального планирования разработки получите мнения по план-графику от нескольких аутсорсеров. Умножьте надвое полученные сроки и стоимость — лучше действовать с запасом, чем обнаруживать проблемы уже после того, как они произошли.

Народная примета — все действительно успешные стартапы в разработке пребывали не более трех месяцев. Давайте не будем обсуждать причины, это просто примета.

В-третьих, имеет смысл сделать календарный план-график с периодами не более месяца, и отслеживать его выполнение. Если времени прошло на два периода, а релиза все нет — очевидно, имеет смысл в разработке что-то радикально изменять.

Высокодетальные технические задания, что удивительно, панацеей в данном случае не являются. Очень часто встречаются солидные с виду документы, описанные в которых программы не работают. Описания попросту не самодостаточны, но изобилуют ссылками на методики и банальные аспекты.

Сделайте лаконичное описание бизнес-логики, и опишите наиболее важные аспекты интерфейса. Не стоит рисовать формы, и схемы с десятками стрелочек, если вы не являетесь опытным разработчиком. Это будет пустой тратой времени.

Получайте консультации по техническим вопросам не от одного, а от нескольких профессионалов. Причем степень профессионализма интервьюируемого стоит оценивать, изучая отзывы и репутацию в тематических сообществах, а не только мнения родственников и далеких от темы знакомых.

Если вы себя считаете прошаренным, воспользуйтесь советом одного из древних мыслителей, не помню кто это сказал: «Сомневайся!». Просто обсудите технические аспекты реализации с коллегами, которые умнее вас. Даже если это сложно признавать — найдите таких, и побеседуйте. Как правило, большинство профессионалов открыты для контактов, и не чуждаются консультирования коллег в частном порядке. На всех форумах есть личка.

#### *Невовлеченность участников проекта*

Многие стартаперы ищут инвестиции только ради того, чтобы их осваивать. Фактическая судьба стартапа им при этом по боку. Выявить таких возможно в результате интервью, некоторыми из следующих вопросов:

- они готовы продавать идею и проект на любых стадиях;
- они легкодвигают бюджет проекта под возможности инвестора;

- они не готовы подписывать инвестиционные соглашения, обмениваться личными данными и так далее.

Конечно, перечень вопросов не полон, и люди меняются, и изменяют свое мнение и позицию — это совершенно нормально.

Но определение позиций и некоторая диагностика вполне поможет избавиться от развития событий, где вы оказываетесь наедине с «разбитым корытом».

Чем меньше команда — тем лучше. Порой, имеет смысл купить идею и лояльность ее автора за небольшую сумму, и ответственно плотно заниматься реализацией в составе малой диверсионной группы — где бойцы воюют плечом к плечу, и нет страха за то, что товарищ может подвести.

Практика показывает, что наиболее лояльны и вовлечены именно наемные сотрудники. Удивительно, но так.

#### *Проекты с бюджетом менее \$1000*

Если вы достоверно лично тратите **не менее половины своего времени** на проект — тогда может выстрелить.

Если нет — скорее попадете в огромную статистику неудачных экспериментов, которых весьма много в портфолио каждого фрилансера.

Есть такое понятие, как «уровень вхождения на рынок». Сотни тысяч микробюджетных стартапов, молящихся на десяток лидеров, этот самый уровень вхождения, на самом деле подняли.

Есть расхожее мнение о том, что уровень вхождения на рынок в ИТ-стартапах невысок. Это действительно так, если говорить о неудачных проектах. Их до безумия много.

Хотя бы по диагонали почитайте литературу по управлению проектами. Ее немало, и почти все книжки хороши.

Целесообразно ведь учиться на чужих ошибках, а не на своих. Как вы считаете?

#### *Дизайн*

На вкус и на цвет товарищей нет. Подавляющее большинство неопытных стартаперов, которые воспринимают проекты дизайном, вкладывают в него больше денег и внимания, чем следовало бы. При этом уповают на опыт Стива Джобса.

Да, Стив очень внимательно относился к дизайну. Вот только он при этом оперировал бюджетом в сотни миллионов долларов, и занимался оптимизацией уже работающих проектов. Не эксперименты на ранней стадии, а менее 1% бюджета раскрученных проектов.

В результате создают дизайны, в которых очень много «мяса». Зачастую они получаются трудноверстаемыми, потому как дизайнеры послушно внимаюят заказчику, и рисуют то, что их просят. Естественно, такие макеты не учитывают развития функциональности, и главное — они имеют свойство вдохновлять только их авторов. Они яркие и пестрые, поэтому естественно отвлекают пользователей от того, на что им следовало бы обращать внимание. Конечно, результат предсказуем — интерфейсы просто «не работают».

Еще одна проблема — навязчивый дизайн, который нравится одним, и просто не нравится другим.

Застраховаться просто — минимизировать работы по дизайну, сосредоточив внимание лишь на основных деталях (модульная сетка, палитра, знаки, стиль, аскетичность), сделать полнофункциональный прототип, предусматривающий все, что следует предусмотреть, и отдать дизайнеру работающую программу.

В этой стратегии скрывается еще одно явление.

Дело в том, что дизайнеры, как творческие личности — народ весьма мнительный и даже обидчивый. Подавляющее большинство дизайнеров просьбы переделать что-то воспринимают едва ли не как личное оскорбление.

Простой способ, который поможет избежать проблем с дизайном, и с необходимостью сменой дизайнера — отдать ему на одизайнивание не схемы и наброски, а работающую программу. Вдохновение будет на высоте, проверено многократно!

#### *Непадание в тренд*

IT-отрасль изумительно молода. Она сверхдинамично развивается, но по сути ей нет и 30 лет в мире, 15 лет в нашей стране.

Тренды в IT-отрасли очень коротки. Буквально — 3-6 месяцев. Сравните с 3-5 годами в рекламе (отрасли около 80 лет), 100 годами в строительстве (ой старая...). Тут все меняется очень быстро.

Быстро же распространяются темы. e-Сороки на хвостах в течении недели разносят по всем интересующимся каждую новую идею. Мы изнутри часто наблюдаем это — когда десятки авторов одной и той же «новой» идеи одновременно начинают искать ресурсы и аутсорсеров для своего «универсального» проекта.

Есть популярная мода — повторять известные западные стартапы. В большинстве случаев в моду попадают только суперпопулярные проекты, которые открывались несколько лет назад. Таким образом, повторяя их, авторы заведомо оказываются в ретроспективе, выпадают из тренда.

Доступный метод для удачного копирования — соотнести разработанную бизнес-модель с современными трендами. При этом, очень прошу вас — убедиться в актуальности трендов, которые вы рассматриваете. Хабр поможет!

Серьезнейшая проблема — затягивание времени. От идеи до открытия зачастую проходит неприлично много времени. Это имеет серьезное влияние на успех проекта в различных ракурсах. Не будем перечислять их, так как в наличии присутствует достаточное количество замечательной литературы по этим вопросам. Замечу лишь коротко, что это очень плохо.

Один из способов избежать этой катастрофы — планировать подготовительные работы к открытию проекта таким образом, чтобы завершить их в пределах средней продолжительности тренда. А именно — не более 3-6 месяцев.

На памяти автора и многих его коллег десятки проектов, работа над которыми длилась больше и существенно больше. Ни один из проектов без существенных изменений не выстрелил. Опять эта примета...

#### *Спасти тему*

Самый популярный страх среди стартаперов! И, честное слово, оправданный.

Спалить тему легко и просто.

- При поиске инвесторов. Есть сотни псевдоинвесторов, основной целью которых является коллекционирование тем. Нарваться на таких очень легко.
- При поиске исполнителей.
- Во взаимоотношениях с партнерами и исполнителями. Особенно — когда это одни и те же люди. Вы хотите, чтобы вашу идею реализовывали за идею.
- При недостаточно динамичном развитии.

Что интересно, и застраховаться не сложно.

- Сделайте развитие динамичным. Открыть проект и начать его работу... после того, как вы это сделаете в первый раз, обнаружите, что 70% действий были не по теме.
- Вам инвестиции нужны или проект, прибыль приносящий? Честное слово, это разные вещи. В поисках инвестиций легко потерять проект.
- Ограничьте круг вовлеченных. Для получения информации и обратной связи совершенно не обязательно рассказывать о проекте. У проекта есть детали, и вы можете исследовать их по отдельности.

Первая заповедь японского самурая **«Никому не говори о своих намерениях»**. Подумайте об этом. Банзай!

*«Юноша бледный, со взором горящим»*

Ну умница ведь! Такая замечательная идея. Молодец!

А с чего вы взяли, что он еще и сможет успешно выполнить роль директора малого предприятия?

Если, тьфу-тьфу-тьфу, вам надо будет себе хирургическую операцию сделать — вы доверитесь юноше? С блестящими глазами и великолепными идеями?

Даже если вы и есть тот самый юноша, внемлите разуму! Найдите менеджера. Опытного, который достоверно умеет проекты развивать.

Как минимум — так называемого ментора.

Секрет: не стоит искать профильного ментора. Как правило, на самом деле им наплевать на результаты. Не знаю почему так, просто констатирую итоги наблюдений.

Найдите успешного предпринимателя, добившегося успеха в различных направлениях, возможно далеких от информационных технологий. Дело в том, что принципы успешного бизнеса во всех отраслях одни и те же. А около ИТ, благодаря популярности тренда, собралось очень много дилетантов и профанов. Даже из деревни Сколково или подобных. Причем большинство из них сложно определить... Предприниматель, действующий в более традиционных сферах, даст вам намного более полезные консультации, чем условный средний айти-менеджер.

Автор проекта будет отличным консультантом. Но не стоит возлагать на него административную работу, если у него нет соответствующего опыта.

Ну почему в других отраслях все прекрасно понимают абсурдность обратного?

## *Изобретение велосипедов*

Речь о разработке.

Давайте сравним с автопромом. Если удачные решения по организации производства, когда основой является сборочный конвейер, к нему — наложенная логистика, обеспечивающая производство комплектующими.

Если бы автоконцерны с нуля занимались производством стали, прокатом листов, разработкой электроники и прочими низкоуровневыми задачами — мы с вами об этих автоконцернах ничего не узнали. Удачная стратегия — сфокусироваться на основных аспектах, и для их реализации активно искать и подбирать опробованные решения.

В аналогии с программированием, речь идет о том, что продолжительность и стоимость разработки можно существенно сокращать:

- используя популярные библиотеки и фреймворки;
- изучая технологии схожих решений (на гитхабе и битбакете есть почти все);
- используя стабильные, проверенные технологии, игнорируя модные фишки;
- как от огня бежать от тех, кто с благоговением в речи своей употребляет фразу «свой фреймворк».

Для того, чтобы добиться успеха — необходимо сужать подконтрольную область. Что вы хотите развивать? Ваш проект, или их «свой фреймворк»? Последнее, поверьте, дорого и просто бессмысленно — есть доступные аналоги, куда уже вложены тысячи человекочасов, у которых есть огромные группы, где можно получить поддержку. Никакой самописный фреймворк не сможет конкурировать с, например: семейством решений jQuery, продуктами Adobe, базовыми библиотеками C++, Yii, CodeIgniter, Drupal и сотнями других. На тезис «наш фреймворк поможет вам» имеет смысл переспросить — а кому он уже помог? И оценивать не дизайн сайтов, а динамику развития проектов. Так отсеиваются все самодеятельные предложения.

Прежде чем писать какие-то модули, например, эквайринга, имеет поискать существующие. Их можно использовать как есть, либо с доработкой, либо использовать как основу для проектирования и разработки нового решения.

Писать все «с нуля» — еще страшнее, чем «свой фреймворк». Гарантируется затягивание сроков и «непредсказуемое», фактически спонтанное увеличение бюджета.

Сюда же можно отнести преждевременную оптимизацию. Опытные программисты знают о так называемом принципе вреда преждевременной оптимизации. Оптимизация не бессмыслена, когда она применяется к уже работающей программе. Основной метод оптимизации заключается в поиске узких мест, например методом юнит-тестирования, и последующей доработке, именно по объективно найденному.

Многие новички начинают оптимизировать еще до того, как они получили что-то работоспособное. В 100% случаев это не приносит пользы — времени затрачивается много, а с узкими местами они не угадывают. Это глупость — гадать, вместо того чтобы исследовать и принимать решения на основании объективной, реальной, а не воображаемой информации.

Универсальные методы, которые помогают разобраться с рисками и проблемами

### *Стратегия*

Следующее явление хорошо знакомо опытным дизайнерам. Скрупулезно проработанный и сбалансированный легко портится одним-двумя неверными действиями.

Самую хорошую идею легко портят так называемые «узкие места». Как правило, на этапе проектирования они не заметны, и выявляются только постфактум.

Для того, чтобы застраховать проект от неудачи, имеет смысл сделать следующее.

1. Сузить объектную область. Выбросить все лишнее, оставив лишь наиболее важные детали.
2. Выполнить проектирование! На самом деле, степень проработки имеет вторичное значение. Изначально важен сам факт того, что идея, стратегия, тактика и детали хранятся не только в вашем воображении, а на бумаге.

Представьте, что вы хотите построить себе дом. Работать будете без чертежа? Правильно — хоты бы какой-нибудь, но вы сделаете и чертеж, и смету.

Совершенно непонятно, почему стартаперы игнорируют проектирование. Совершенно не обязательно для этого нанимать РБК-консалтинг. Начните с записей в текстовом редакторе или в звэрноте.

Сделайте два документа: «Все» и «Основное». Таким образом, вы сможете фиксировать все идеи, улучшая качество дальнейшего анализа, и отделять зерна от плевел, выбирая действительно важные факты.

Понять, является ли выбранный аспект важным, просто. Сделайте документ «Основное» в формате таблицы из двух столбцов. В левый пишите факторы. В правый — то, на что эти факторы влияют. Объективно, и с пояснением, как именно влияют. Если в ячейке в правом столбце пусто, или написано неубедительно — это потенциальный кандидат на откладывание «на потом».

Структурируйте и упрощайте план действий до тех пор, пока он не станет изумительно простым, и вы сможете проставить относительные даты около каждого из пунктов — таким образом получив план-график реализации проекта.

Отслеживание реализации план-графика и будет являться вашей основной целью на ближайшее время.

### *Анализ*

Аналитика считается уделом ботаников.

Ну и хорошо! Пока ваши конкуренты размышляют об уделах и о ботаниках — вы обезопасите свои деньги, силы и время, проведя самодостаточную рекогносцировку (разведку для получения сведений о противнике, производимую лично командиром, т.е. в нашем случае, менеджером) перед началом боевых действий.

Достаточный объем затрат на анализ определить просто. Они должны быть меньше, чем затраты на открытие проекта.

Совет первый: записывайте все подряд. И рисуйте. Логическое (абстрактное) мышление человека устроено таким образом, что работает очень быстро, но способно обрабатывать лишь 3-7 фактов единовременно.

Записывая и зарисовывая даже очевидное — вы достигаете двух целей. Во-первых, вы фиксируете данные для последующих выводов. Записанное — не забудется. Во-вторых, вы освобождаете собственное абстрактное мышление для более полезных, чем тривиальные, мысли.

Важный момент — в записанном виде мысли выглядят более наглядно, чем в фантазиях и на словах. Записывать имеет смысл все. Кстати, все успешные бизнесмены именно так и делают. Если сильно лень писать — используйте диктофон.

Потом можно перечитывать, и еще раз обдумывать, принимая полезные решения. Ваши результаты будут гениальными.

### *Интервью*

Для сбора данных, кроме чтения информации в интернете, также полезно проводить интервью.

У вас есть определенное мнение по конкретным вопросам. Четкое и убедительное. Но для того, чтобы уменьшить количество ошибок — это мнение должно быть объективным.

Как доказал Эйнштейн, субъективно все. Поэтому, для того, чтобы уменьшить количество ошибок, имеет смысл мнения коллекционировать. Решение принимать все равно вам, но! Чем больше мнений учтено, даже откровенно «неправильных» — тем больше точек опоры вашего решения. Тем релевантнее и успешнее ваше решение.

Интервьюировать можно как коллег, так и гипотетических пользователей. Для обеспечения релевантности результатов включите в состав интервьюируемых ярких, и средних представителей.

Воспользуйтесь известными копирайтерскими технологиями:

1. Составьте бриф. Чем подробнее, тем лучше. Только по завершению работы над брифом не забудьте выделить наиболее важные аспекты, и включить их в короткую версию. Сделайте две версии брифа — короткую, и полную, для заинтересовавшихся. Вопросы в брифе должны быть открытыми. Масса рекомендаций на этот счет есть в литературе для рекламистов и маркетологов.
2. Зайдите в сервис опросов.
3. Проводите личные опросы и беседы, самостоятельно и старательно беспристрастно заполняя бриф по итогам. Отмечайте все идеи и мысли, которые вы не предусмотрели.

При этом идею палить совершенно не обязательно. Вы можете создавать обезличенные опросы, разделяя их и формулируя вопросы таким образом, чтобы изучить мнения, но при этом не намекать на тему.

Подскажу хороший способ обеспечения безопасности от несанкционированного распространения идеи — сделать вид, что у вас другая тема. Прощупывайте нечно, лишь параметрически похожее на то, что вас интересует. Параметры и фиксируйте — в сумме получится достаточно яркая и самодостаточная картинка.

Полученную информацию и используйте для пересмотра идеи и отдельных ее параметров.

### *Прототип*

Пробничек стоит существенно дешевле флакончика. Эффект от пробничка такой же, но риски меньше.

Суть в том, чтобы прототип был полнофункциональным. Выберите наиболее важные функции, продумайте простой и понятный для заведомо широкой (читай- неподготовленной) аудитории интерфейс, и реализуйте его с малым бюджетом. Запустите мини-проект, и внимательно исследуйте все его характеристики — от статистики до единичных продаж. Обязательно — вам нужны отзывы пользователей!

Что интересно, проекты-чемпионы рождаются именно из прототипов. Зачастую они даже несущественно изменяются в процессе развития. Яркий пример — Gmail.

После того, как от прототипа получены какие-то результаты, через 3-6 месяцев после запуска — имеет смысл задуматься о его развитии. Но на этот раз вы будете иметь уже объективную информацию о самых разных аспектах проекта. И ваши управленческие решения будут на порядок удачнее.

Прелесть прототипа в том, что он позволяет опробовать идеи и стратегии даже не то, чтобы в миниатюре, а с минимальными вложениями. Нет смысла делить с кем-то и отдавать большие кусочки будущего пирога за скромнейшее участие на раннем этапе. В подавляющем большинстве случаев достаточно минимального объема инвестиций. Сделайте работающий вариант проекта с минимальным набором самых важных задумок, и запустите в работу.

Еще о прототипировании:

- суперскоростная по сути методика полнофункционального прототипирования позволит оперативно застолбить место под солнышком, и развиваться далее;
- ускоряя разработку и минимизируя затраты, вы имеете больше шансов попасть в удачный тренд, и меньше вероятности — открываться одновременно с множеством конкурентов;
- принципиально — дать проекту жизнь. Вместо длительных и безуспешных поисков крупных сумм, сопряженных с огромным риском спалить идею — открыть проект с заведомо скромными затратами, и развивать его релевантно, основываясь на реальных объективных данных, а не на мечтах и предположениях.

### *Непрофильное прототипирование*

Весьма условная формулировка.

Оценить успешность проекта возможно, совершив один или несколько экспериментов в отвлеченном формате. Целью экспериментов может быть:

- оценка объема целевой аудитории;
- оценка релевантности интересов целевой аудитории;
- осторожная оценка релевантности идеи стартапа.

В этом поможет копирайтер, или можно осуществить самостоятельно. Говорить с людьми по теме, публикуя статьи и вопросы в сообществах и на форумах, и делать выводы.

Прежде чем инвестировать деньги, личное время и усилия в большой проект, может быть имеет смысл сделать маленький, но отвлеченный. Не аналогичный, возможно даже не по аналогичной тематике, но работающий с той же целевой аудиторией, с той же бизнес-моделью — можно опробовать совершенно различные аспекты, и при этом цена экспериментов будет весьма скромной.

Результаты экспериментов можно использовать в качестве исходных данных при анализе и проектировании перспективы основной задумки.

#### *Отзывы*

Плохие отзывы — один из самых ценных инструментов хорошего директора. Только недовольный клиент совершенно искренне расскажет, что, как и почему ему не понравилось.

У интернет-проектов есть такая особенность, что пользователи его пассивны. Им не нужно говорить вслух, куда-то ходить или что-то делать, чтобы использовать предоставленное. Они могут пользоваться им абсолютно молча. И, даже если им что-то не нравится — они, совершенно цинично, могут просто игнорировать предложенное, или игнорировать недостатки, и вы никогда не узнаете об этих проблемах.

Недовольные клиенты, напротив — на детали не скупятся.

Специфика этого бизнеса в том, что слабые стороны зачастую создают больше проблем, чем сильные решения — продвигают проект вперед. Из-за «узких горлышек» возникает отток посетителей, снижение лояльности, проблемы в реализации интерфейсных сценариев и другие неприятности.

Именно злой, недовольный клиент поможет вам узнать об этих проблемах. Долой розовые очки! Обучите сотрудников вашей технической поддержки, занимайтесь консультированием и разбором отзывов самостоятельно, принимайте много эффективных мер, инвестируйте в получение отзывов, какими бы они не были. И анализируйте каждый.

Отрицательные отзывы — бесценны. Недовольный клиент может дать вам понять, как получить сотни и тысячи довольных. Это не мое открытие.

Самая злая ошибка стартаперов — это глухота. У успешных предпринимателей уши как правило открыты, и они вкладывают деньги и развиваются проекты по получению отзывов пользователей.

#### *Сильные решения*

В соответствии с принципом Парето, лишь малая часть действий приводит к достижениям потенциально большей части результата. Это работает, это физика, и это факт.

Сделайте таблицу. В первом столбце, подробно, тезисно — все решения, все функции, фишки, примочки, особенности интерфейса и дизайна вашего проекта. Во втором столбце напишите, на что каждый тезис влияет.

Потом, только потом — в третьем столбце баллами оцените степень влияния тезиса на результат. Баллы должны отмечать относительную важность, то есть насколько конкретный пункт важнее соседнего. Не стоит мудрствовать, ограничьте шкалу 5 или 10 баллами. Простота будет играть на ваш счет, в данном случае. Отсортируйте таблицу по третьему столбцу. В теории, только первые 20% имеют существенное

значение.

При планировании, выделите их в отдельный документ. Это то, без чего проект не сможет жить и развиваться. Именно этим аспектам стоит уделить наибольшее внимание.

Из оставшейся части выберите те пункты, которые прямо или косвенно влияют на вовлеченность и лояльность аудитории, а также те, которые вдохновляют авторов или ключевых пользователей проекта. Кнопки для шаринга и вход через соцсети без форм для регистрации туда попали?

Сформируйте из них пакет «джентельменский» — то, что просто должно быть. Таким образом, вы сможете определить полезные концепции развития вашего проекта, и отсеять напускное.

### Команда менеджмента проекта

Условно можно определить четыре вида **Команд (групп)**, классифицированных по содержанию их работы, которые наиболее часто формируются в явном или неявном виде в практической деятельности предприятий.

1. **Команды, которые создают что-нибудь новое для организации или делают работу, ранее не осуществлявшуюся.**

Команды проекта (проектные Команды) полностью попадают в эту группу. Они носят временный характер, который определяется сущностью проекта как временной специфической организационной формой достижения целей и решения уникальных задач.

2. **Команды (группы), которые имеют дело с проблемами, целями и задачами на предприятии через анализ, контроль и рекомендации.**

Команды по аудиту и контроллингу, группы оценки качества.

3. **Команды (группы), которые не являются специальными, а составляют постоянную часть организационного развития и осуществляют процесс производства и выполнения повторяющихся работ.**

Производственные команды (группы), команды продаж и обслуживающие команды (бригады, группы).

4. **Команды многоисполнительской управлеченческой природы.**

Эти команды обычно формируются на высших уровнях управления предприятием и имеют форму исполнительных комитетов, управлеченческих команд или топ-менеджмента предприятия.

В организационной структуре больших проектов и в их менеджменте можно выделить по крайней мере *три типа проектных команд*.

1. **Команда проекта (КП)** — организационная структура, создаваемая на период осуществления всего проекта либо одной из фаз его жизненного цикла. Задачей руководства команды проекта является выработка политики и утверждение стратегии проекта для достижения его целей. В команду входят лица, представляющие интересы различных участников проекта.
2. **Команда управления проектом (КУП)** - организационная структура, включающая тех членов КП, которые непосредственно вовлечены в управление проектом, в том числе — представителей отдельных участников проекта и технический персонал. В относительно небольших проектах КУП может включать в себя практически всех членов КП. Задачей КУП

является исполнение всех управленческих функций и работ в проекте по ходу его осуществления.

3. **Команда менеджмента проекта (КМП)** — организационная структура, возглавляемая управляющим (главным менеджером) проекта и создаваемая на период осуществления всего проекта или его фазы. В команду менеджмента проекта входят физические лица, непосредственно осуществляющие менеджерские и другие функции управления проектом. Главными задачами команды менеджмента проекта являются осуществление политики и стратегии проекта, реализация стратегических решений и осуществление тактического (ситуационного) менеджмента.

#### Соотношение между различными командами в проекте

Требует пояснения вопрос о том, когда и зачем нужно выделять несколько типов команд в проекте, стоит ли не усложнять ситуацию.

Мировая практика показывает, что разделение команд целесообразно в проектах, в которых необходимо четкое фиксирование позиций различных его участников (прав, полномочий, ответственности, доли участия и долей в прибыли и пр.). В частности, выделение нескольких проектных команд целесообразно для больших, смешанных, средне- и долгосрочных проектов или же когда количество участников проекта достаточно велико, а их интересы противоречивы.

Главный критерий эффективности деятельности команд в проекте - его успех. Если главный менеджер проекта считает, что дифференциация команд снижает риски и способствует успеху проекта, то в этом случае именно он берет на себя всю ответственность за успешное достижение целей проекта. Однако он должен четко оговорить условия его осуществления, обеспечить формальное описание, разделение и закрепление компетенций различного типа проектных команд. В современной культуре менеджмента проектов (как «западной», так и «восточной») этот факт является осознанной необходимостью успешного осуществления любого проекта.

Так как Команды в проекте различаются своими целями, задачами, компетенцией и мерой ответственности за результаты осуществления проекта, то их позиция, место и роль в проекте и по отношению к проекту определяются целями входящих в них лиц и представителей участников проекта, степенью участия команды в процессах проекта и ее ответственностью.

КП организуется, как правило, под долгосрочные проекты с большим числом участников, которые, может быть, не участвуют непосредственно в управлении процессами проекта, но определяют политику и стратегию проекта, исходя из собственных интересов.

КУП также организуется в рамках достаточно больших проектов или когда проект («контрольный пакет») в основном принадлежит исполняющей (или родительской) организации. В этом случае отдельные управленческие функции или осуществление некоторых процессов проекта могут быть поручены техническому персоналу или функциональным подразделениям организации (например, часть функций управления стоимости проекта или коммуникаций, связанных с информационной инфраструктурой организации-исполнителя).

Особенностью КМП является то, что она одновременно занимает внешнюю (субъект управления) и внутреннюю (изменяющийся по ходу проекта элемент) позицию по отношению к проекту (как объекту управления) и к процессам его осуществления.

Дело также в том, что КМП - это совокупность управленческих ролей, которые могут выполнять как несколько человек, так и один профессионально грамотный главный менеджер проекта. К этой совокупности относятся такие роли, как «руководитель», «администратор», «тренер», «лидер», «менеджер проекта», «управляющий проекта». В каждом конкретном случае распределение ролевых управленческих функций между физическими лицами - участниками проекта, их полнота и содержание носят уникальный характер, зависящий от многих факторов (культуры исполняющей организации, класса, вида и типа проекта, имеющихся ресурсных возможностей и т. п.).

Соотношения между этими типами команд зависят от проекта. Для больших проектов наличие трех типов команд достаточно очевидно. А в малых проектах КП и КУП могут «умещаться» в КМП.

#### *Интерпретация системных свойств применительно к КМП.*

Применительно к КМП свойства системы можно интерпретировать следующим образом:

##### *Целесообразность*

Под целесообразностью понимается общая характеристика поведения сложных динамических систем (в случае КМП – организационной и социальной), описывающая ориентацию системы на достижение целей и получение определенных результатов.

Целью самоорганизующейся системы является модель «желаемого будущего», а в рамках КМП – достижение запланированных целей и получение ожидаемых результатов проекта, как результатов сознательной деятельности всей команды.

С точки зрения определения целей ТСО команды сложностью является большое количество персональных характеристик членов Команды, которые имеют разные оценочные шкалы, не имеют однозначных трактовок и проявление которых зависит от конкретной ситуации по проекту. Также немаловажную роль играют динамика изменений окружающей среды проекта, неопределенность и неоднозначность информации о внешней и внутренней среде проекта, «человеческий» фактор и т.п.

##### *Иерархическое строение.*

В КМП иерархия уровней управления отсутствует, поэтому деятельность выстраивается на связях координации и партнерства. В этом сущность менеджерской команды и ее принципиальное отличие от таких типов проектных команд, как команда управления проекта и команда проекта. Поэтому в рамках КМП используется иерархия целей и задач проекта и ответственности членов КМП. Каждый зависит от каждого, и эффективная работа всей команды является совокупностью вкладов каждого ее члена. Если сравнить КМП с тепловозом, то неисправность или отсутствие тех или иных узлов не позволяет ему выполнять работу, для которой он предназначен. В данном случае вопрос о том, какой узел важнее, не возникает.

##### *Адаптация*

Функционирование любой самоорганизующейся системы обусловлено ее отношениями с внешней средой и реакциями приспособления к изменениям в ней. Адаптивная система должна выполнять свои функции, наиболее эффективным путем в зависимости от состояния окружающей среды. Уникальным свойством самоорганизующейся системы является изменение (корректировка) ее структуры и функций, адекватных изменениям внешней среды и наличие памяти.

Адаптация КМП происходит за счет изменения взаимосвязей между ее элементами и корректировки ее управленческих функций с целью выполнения проекта наиболее экономичным путем.

#### *Память*

Память, как способность к воспроизведению прошлого опыта в рамках деятельности Команды, позволяет работать быстрее и эффективнее. Накопление информации в процессе осуществления проекта, т.е. формирование опыта, позволяет предсказывать ход проекта системы в его непрерывно изменяющемся контексте. Как следствие, в ряде случаев решения на основе предсказания хода проекта, базирующегося на основе прошлого опыта, оказываются эффективнее, нежели решения, принятые без его учета.

Однако такой путь далеко не всегда является наиболее эффективным. В ряде случаев, использование памяти, как совокупности стереотипов подходов и деятельности, приводит к принятию типовых, простых и неверных решений в нетипичных случаях.

#### *Разнообразие состояний*

Разнообразие состояний КМП обуславливается многочисленностью ее элементов, имеющих разную природу (человеческую, социальную, техническую и проч.) и наличием различных как измеряемых, так и неизмеряемых явных и неявных связей между ними.

Для того, чтобы использовать ТСО надо создать КМП, обладающую или способной создать в себе еще большее разнообразие, чем разнообразие решаемых проектных задач. Иначе, совокупный потенциал (профессиональный, человеческий, трудовой) КМП должен быть большим, нежели необходимый для осуществления данного проекта. В условиях недостаточного совокупного потенциала КМП или отсутствия ее целостности ТСО просто не будет работать.

Целостность КМП, как системы, проявляется в возникновении новых интегративных качеств, не свойственных образующим ее компонентам, т.е. свойства КМП не являются суммой свойств ее элементов. Такое проявление целостности называется синергией КМП.

#### *Технология самоорганизации КМП*

##### *Метатехнология и технология самоорганизации*

В реальном проекте технология самоорганизации создается самой командой, возможно, с помощью профессионалов по данному вопросу. Сам процесс ее разработки является мощным интегрирующим фактором в деятельности КМП.

Она разрабатывается на стадии ЖЦ команды «Нормализация деятельности (normalizing)» и корректируется в последующем исходя из реальных условий. Поэтому говорить о единой «технологии самоорганизации» для всех типов и видов проекта некорректно. Однако, возможно говорить об уровне метатехнологии для достаточно большого числа проектов, характеризующихся сходными параметрами.

На базе такой метатехнологии самоорганизации команды, например, для IT-проектов, создается своя «технология самоорганизации команды конкретного проекта», учитывающая ресурсные возможности, особенности проекта, организационную культуру компании, уровень профессионализма членов команды, стандарты (международные, национальные, корпоративные) по менеджменту и управлению проектами и многое другое.

Метатехнологию можно сформулировать для достаточно широкого спектра проектов. Она является базой (организационной моделью и моделью деятельности) для создания адекватной конкретному проекту и конкретному набору членов команды ТСО.

Иначе, готовых рецептов нет. Большинство попыток свести управление современным проектом к работе по операциям и к управлению в технических системах не увенчались успехом. Таким образом, метатехнология является базой и совокупностью инструментов для построения ТСО команды под конкретный проект.

#### *Область применимости*

На начальной стадии проекта (*start-up*) используется практика проведения стартовых семинаров, тренингов или интеграционных процедур для создания и организации работы команды (*Team Building*). На этой стадии понятие «самоорганизации» не применимо и не имеет смысла. На стадии расформирования и/или реорганизации команды сам термин «самоорганизация» выглядит странно. Поэтому «технология самоорганизации» применима только к процессу развития команды (*Team Development*).

Начинается ее разработка на стадии «Нормализация деятельности (*normalizing*)» жизненного цикла КМП. На этой стадии члены команды приходят к взаимному согласию в результате переговоров и принятия компромиссов и разрабатывают нормы и правила, на основании которых будет построена их дальнейшая работа.

Сама технология «работает» на стадии «Исполнение планов по выполнению проекта (*performing*)», т.е. уже после того, как мотивация членов команды определена и ориентирована на успешное выполнение проекта, процесс осуществления проекта стабилизируется, эффективность работы команды возрастает, каждый член команды знает свою роль и т.п.

Следует также учесть, что при фазовых переходах в проекте (переход от одной фазы или стадии жизненного цикла проекта к другой) всегда необходимо проводить корректировку деятельности КМП с учетом изменившейся проектной ситуации. Как следствие, требуется также провести и ревизию некоторых элементов используемой технологии самоорганизации. Недооценка потребностей в изменениях может привести к неадекватности используемой технологии самоорганизации и изменившейся КМП.

#### *Условия применимости*

Самоорганизация возможна при определенных условиях:

- сведение «искусства управления» к выполнению операций;
- ограничения на самостоятельность действий члена КМП на уровне постановки задачи; иначе: «что делать?» - задается в рамках интегрированного контекста проекта и целей проекта;
- свободный выбор инструментов для решения вопроса «как делать?» в рамках ограничений на ресурсы, временные параметры и требования к результатам проекта
- согласование и координация промежуточных результатов - по проекту, его фазы (стадии), по задаче.

Иными словами, в рамках ТСО КМП:

- уровень и границы осуществления стратегии проекта задаются и координируются в рамках командных действий;
- осуществление тактических действий предполагает принятие самостоятельных решений и свободный выбор инструментов в рамках ограничений на решаемую задачу.

Процесс. Статичность процесса определяется управленческой культурой команды, включающей систему ценностей, ментальность и образ командных действий для данной совокупности индивидуумов, образующих команду, и целями (задачами) проекта.

Изменяющийся интегрированный контекст проекта и ход проекта определяет динамику изменений самой команды. Сам процесс самоорганизации является динамическим. На входе – КМП, которая должна быть уже построена, т.е. проведен процесс *Team Building*, интегрированный контекст проекта и метатехнология ТСО. На выходе – успешное завершение проекта и изменившаяся КМП.

Технология. Определяется самим проектом, совокупностью индивидуумов – членов Команды, совокупностью управленческих ролей КМП и их весом для конкретного проекта и/или его жизненной фазы, профессиональным и человеческим совокупным потенциалом членов команды и, в ряде случаев, других участников проекта. Следует учесть, что совокупный профессиональный и личностный потенциал членов КМП должен превышать требующийся для осуществления ТСО (требование избыточности системы).

Вид проектной команды. Представление о самоорганизации применимо к управленческому звену проекта (команде или группе менеджмента проекта) или когда вся команда проекта не превышает 10-12 чел.

Требования к членам команды. Для каждого члена КМП, сознательно участвующим в процессах самоорганизации командной и личной деятельности по проекту, необходимыми начальными условиями являются:

- адекватность культуре команды менеджмента конкретного проекта;
- понимание каждым своей роли, как элемента целого под названием «команда проекта»;
- знание и принятие выработанных в результате согласования между членами КМП норм, правил и процедур совместной работы в рамках конкретного проекта.

Для каждого члена КМП в самом процессе исполнения проекта при использовании технологии самоорганизации необходимыми условиями являются:

- соблюдение выработанных норм и правил (формально оформленных и неформальных);
- следование принятым процедурам;
- постоянный мониторинг прогресса проекта с целью корректировки норм, правил и процедур.

#### *Запуск процесса самоорганизации*

Сама по себе самоорганизация не происходит. Естественным образом происходит только повышение энтропии, развитие беспорядка, хаоса и развал работы. Поэтому процесс надо «запустить» и сознательно «силовым» способом поддерживать систему регуляторов этого процесса (регулировать границы процесса самоорганизации КМП).

Для запуска нужно иметь:

- Сформированную КМП, прошедшую этап start-up и Team Building.
- Заданные форматы и формы повторяемых (циклических) действий.
- Заданные правила: их наличие, принятие членами КМП и соблюдение.
- Действия: обязательные периодические и регулярные.
- Коммуникации: формальные, неформальные, личностные.
- Информация: интегрированный информационный контекст проекта.

#### *Инструменты менеджмента проектов для ТСО КМП*

Возникающие трудности при выборе инструментов всегда связаны с уникальностью проекта (по определению). Уникальность проекта всегда требует создания уникальной команды проекта, т.к. вне проекта «команда проекта» не существует. Поэтому совокупность инструментов можно в той или иной степени определить, однако их набор и взаимосвязи в обязательном порядке формируются под конкретный проект в рамках групповой работы конкретных членов КП.

Для того, чтобы поддержать необходимую эффективность деятельности команды, как системы, и избежать возможной подмены целей проекта и команды на групповые или индивидуальные цели, используется ряд инструментов для организации деятельности команды. В частности:

- партисипативный менеджмент проекта;
- согласованные взаимные ожидания и требования, как членов команды, так и других участников проекта;
- определенные границы самостоятельности каждого члена команды в рамках собственной ответственности в принятии решений о сроках, распределении и использовании ресурсов, результатах по проекту;
- групповая позитивная синергия;
- адекватная и справедливая система мотивации членов КМП;
- сбалансированная обратная связь;
- и другие.

В качестве регулирующих процесс самоорганизации инструментов можно использовать:

- корпоративные стандарты, правила и нормы;
- процедуры;
- развивающие мероприятия по корректировке адекватности профессионализма управляющего проекта и членов команды целям и задачам проекта;
- систему регулярных событий, формирующих единый информационный контекст проекта – рабочие совещания по проекту, отчеты по прогрессу и т.п.;
- другие, определяющиеся конкретным проектом.

#### **Системы управления проектами**

**Системы управления проектами** (Project Management system, PMS) — компьютерные программы, предназначенная для организации выполнения проекта (работа с требованиями, планами, задачами, сборками, рисками, ошибками; учет бюджета и трудозатрат; отчетность по различным управленческим аспектам и т.п.)

### **Непрофессиональные PMS:**

- Microsoft Project Standard,
- Suretrak (Primavera),
- OpenProj,
- TimeLine.

### **Профессиональные PMS:**

- Microsoft Project Professional,
- Primavera Project Planner Professional
- Artemis Project Views (Artemis International),
- Open Plan (Welcom Corp.),
- Primavera P4 (Primavera),
- IBM Rational Portfolio Manager,
- Opus Magnum Enterprise Management,
- 1С-Рарус: Управление проектами

**Распределенное управление проектами** (distributed project management, DPM) - предполагает не только территориальную удаленность друг от друга мест выполнения работ по проектам и его частям, но и распределенность команды проекта и, как следствие, распределенное принятие решений при общности задач и целей управления.

**DPM** ориентировано на обеспечение совместной деятельности участников проекта компьютерными средствами, где ключевой функцией становится менеджмент взаимодействия между исполнителями (interaction management).

**Радикальное отличие DPM от PM** в смене объекта управления. В изменившихся условиях приходится управлять не потоками данных (документооборот), как прежде, а согласованной корпоративной деятельностью соучастников процесса.

Система DPM служит для поддержки деятельности проектных команд, состоящих из работников знаний, средствами современных сетевых технологий.

Переход к DPM предполагает отказ от долговременного планирования: ставится некоторая главная стратегическая цель, а движение к ней осуществляется путем согласованного действия исполнителей, в том числе методом «проб и ошибок».

### ***Резюме***

- При использовании системного подхода для построения ТСО КМП всегда следует учитывать неопределенности в проекте, которые связаны с влиянием человеческого фактора. Поэтому типовые подходы можно определить на уровне метатехнологии для групп проектов, обладающих сходными характеристиками.
- Для конкретного проекта всегда требуется построение уникальной ТСО на базе метатехнологии самоорганизации для определенного типа проектов с учетом особенностей конкретного проекта, «зрелости» компании-исполнителя, профессионального потенциала и особенностей индивидуумов, входящих в КМП.

- Целесообразность использования ТСО определяется в каждом конкретном случае и зависит от масштаба проекта (чем выше объемы, тем большая эффективность), его длительностью (чем продолжительней проект, тем большая экономия высокопрофессионального управленческого ресурса) и профессионализма индивидуумов (чем выше профессионализм и мастерство, тем больший эффект).
- Использование ТСО возможно только при определенных условиях: на стадии Team Development, при наличии достаточного для получения кумулятивного эффекта профессионального и человеческого потенциала членов КМП, установленных и выполняемых правилах и процедурах совместной и индивидуальной работы членов КМП.
- Использование ТСО возможно в такой КМП, совокупный потенциал членов, который превышает необходимый совокупный профессиональный потенциал, требуемый для осуществления конкретного проекта (требование избыточности системы).

### Оценка деятельности работы КМП в IT-проекте

**Ключевые показатели эффективности** (англ. *Key Performance Indicators, KPI*) — показатели деятельности подразделения (предприятия), которые помогают организации в достижении стратегических и тактических (операционных) целей. Использование ключевых показателей эффективности даёт организации возможность оценить своё состояние и помочь в оценке реализации стратегии.

KPI позволяют производить контроль деловой активности сотрудников, подразделений и компании в целом. Для термина «key performance indicators (KPI)» зачастую используется русский перевод «ключевые показатели эффективности» (КПЭ), однако это не совсем верно.

С переводом по смыслу слов key (ключевой, характеризующий степень достижения какой-либо цели, существенный для работы одной из областей деятельности компании) и indicator (индикатор, показатель) проблем не возникает, но слово performance невозможно однозначно трактовать, хотя технически, это «производительность, КПД». Правильную формулировку можно найти в стандарте ISO 9000:2008. Он разделяет слово performance на два термина: результативность и эффективность. По стандарту, результативность — это степень достижения запланированных результатов (способность компании ориентироваться на результат), а эффективность — соотношение между достигнутыми результатами и затраченными ресурсами (способность компании к реализации своих целей и планов с заданным качественным уровнем, выраженным определёнными требованиями — временем, затратами, степенью достижения цели). Слово performance объединяет в себе и результативность, и эффективность. Таким образом, правильным переводом термина KPI будет «ключевой показатель результата деятельности», так как результат деятельности содержит в себе и степень достижения, и затраты на получение результата.

КПЭ — это инструмент измерения поставленных целей. Если показатель, который вы придумали, не связан с целью, то есть не образуется исходя из её содержания, тогда нельзя использовать данный KPI. Технологии постановки, пересмотра и контроля целей и задач легли в основу концепции, которая стала основой современного управления и называется «Управление по целям».

**Управление по целям** — метод управленческой деятельности, предусматривающий:

- предвидение возможных результатов деятельности
- планирование путей их достижения

Основоположником «Управления по целям» является Питер Друкер ([нем.](#) Peter Ferdinand Drucker (1909—2005)). Именно он превратил управление — непопулярную и не уважаемую в 50-е годы XX века специальность в научную дисциплину. Питер Друкер также является основоположником системы оценки достижения результатов — целей через ключевые показатели эффективности. Согласно Друкеру, начальники должны избегать «ловушек времени», когда они вовлечены в процесс решения текущих ежедневных задач, поскольку это приводит к тому, что они начинают забывать выполнять задачи, направленные на достижение результатов (целей). Современным воплощением управления по целям, является «Система КПЭ», которая включает в себя множество управлеченческих концепций, которые появились за последние 20—30 лет и дополняют классическое «Управление по целям».

По мнению Питера Друкера, лишь немногие области менеджмента имеют такое большое влияние на организацию, как оценка деятельности подразделений и компании в целом. Однако оценка, подчёркивает Друкер, сегодня одна из самых слабо проработанных областей управления. Так в результате опроса, проведённого в США, стало ясно, что 60 % руководителей высшего уровня недовольны своими системами оценки результатов деятельности. По отечественным оценкам количество российских менеджеров, ещё больше, более 80 %. Это недовольство выражается в отсутствии связи между планами, исполнением, результатом и мотивацией.

KPI и мотивация персонала стали неразрывными понятиями, так как с помощью данных показателей (KPI) можно создать совершенную и эффективную систему мотивации и стимулирования сотрудников компании.

В зависимости от стратегии компании различают разные KPI. В основном их применяют для определения результативности работы административно-управленческого персонала. Например, в стратегической цели «увеличить средний доход на клиента с 10 рублей до 15 рублей на 2008 год» ключевым показателем эффективности является «средний доход на клиента». **KPI это не Ключевые факторы успеха.** В примере выше, ключевыми факторами успеха будет что-либо, что необходимо, чтобы достичь указанной цели, например, организация производства нового продукта.

Ключевые показатели эффективности можно разделить на:

- Запаздывающие — отражают результаты деятельности по истечении периода
- Опережающие — дают возможность управлять ситуацией в пределах отчётного периода с целью достижения заданных результатов по его истечении

К запаздывающим относятся финансовые показатели. Финансовые показатели демонстрируют связь с желаниями собственника и возможностями компании генерировать денежные потоки, однако в силу своего запаздывающего характера не могут описывать текущую эффективность подразделений и компании в целом.

Оперативные (опережающие) показатели рассказывают о текущей деятельности подразделений и компании в целом, параллельно и косвенно отвечая на вопросы о том, какие денежные потоки могут быть в будущем, а также каково качество процессов и продукции, степень удовлетворённости заказчиков.

Ключевые показатели эффективности являются частью системы сбалансированных показателей (Balanced Scorecard), в которой устанавливаются причинно-следственные связи между целями и показателями для того, чтобы видеть закономерности и взаимные факторы влияния в бизнесе — зависимости одних показателей (результатов деятельности) от других.

## Разработка KPI

Разработку KPI рекомендуется провести в ряд этапов:

### 1. Предпроектные работы:

- Получение одобрения и поддержки высших руководителей.
- Инициирование и планирование проекта.
- Создание проектной группы.
- Проведение предпроектного исследования.

### 2. Разработка методологии системы KPI:

- Оптимизация организационной структуры.
- Разработка методической модели.
- Разработка процесса управления компанией на основе KPI
- Разработка системы нормативно-методической документации (регламентация).

### 3. Разработка информационной системы KPI:

- Разработка ТЗ для настройки (программирования) информационной системы.
- Настройка (программирование) информационной системы.
- Обучение пользователей.
- Проведение опытной эксплуатации.

### 4. Завершение проекта.

Ввод системы KPI (методологии и информационной системы) в промышленную эксплуатацию.

При разработке методологии KPI важно акцентировать внимание на:

1. Изменения корпоративной культуры и организация процессов
2. Разработке целостной стратегии развития KPI
3. Разъяснении персоналу достоинств KPI
4. Идентификации общекорпоративных КФУ
5. Выборе решающих KPI для всей организации
6. Разработке структуры отчетности для всех уровней
7. Координации применения решающих KPI
8. Уточнении KPI для поддержания их актуальности.

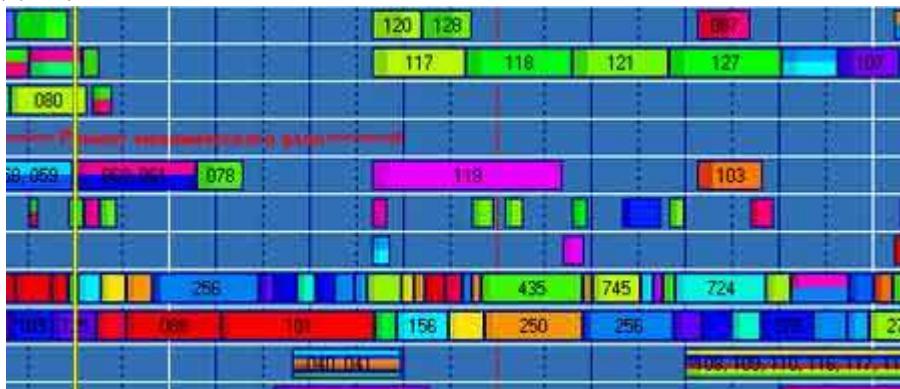
## Правила и принципы внедрения KPI

- **Правило «10/80/10»** — Каплан и Нортон рекомендовали использовать не более 20 KPI. Хоуп и Фрейзер предлагают использовать не более 10. Самой лучшей рекомендацией из существующей практики является правило «10/80/10». Это означает, что организация должна иметь около 10 ключевых показателей результативности, до 80 производственных показателей и 10 ключевых показателей эффективности. Для подразделений Панов

рекомендует использовать не более 10—15 KPI, в противном случае менеджеры будут перегружены планированием, а руководство компании — «разбором полётов» по исполнению KPI, которые не сильно влияют на результативность, как подразделения, так и компании.

- **Принцип управляемости и контролируемости** — Подразделению, ответственному за определённый показатель, должны быть выделены ресурсы на его управление, а результат может быть проконтролирован.
- **Принцип партнерства** — Успешное решение задачи повышения производительности требует установления эффективного партнерства между всеми заинтересованными лицами: совместная разработка стратегии внедрения системы, необходимость добиться понимания того факта, что требуются перемены.
- **Принцип перенесения усилий на главные направления** — Повышение производительности требует расширения полномочий сотрудников организации, особенно тех, кто работает непосредственно на «передовой линии»: помочь сотрудникам нуждающимся в повышении квалификации, обеспечение проведения тренингов, передачу ответственности на разработку собственных КПЭ, эффективное действие коммуникаций (горизонтальной и вертикальной)
- **Принцип интеграции процессов оценки показателей, отчетности и повышения производительности** — Очень важно, чтобы менеджеры создали такую интегрированную схему оценки показателей и отчетности, которая стимулировала бы конкретные ответственные действия. Необходимо регулярно проводить отчетные совещания, по срокам, в зависимости от сложности решаемого вопроса.
- **Принцип согласования производственных показателей со стратегией** — Показатели производственной деятельности лишены всякого смысла до тех пор, пока они остаются не привязанными к текущим критическим факторам успеха (КФУ), составляющим сбалансированную систему показателей (ССП), и стратегическим целям организации.

#### Диаграммы Ганта как оценка КМП



**Генри Лоренс Гантт** (Henry Gantt, 1861—1919) — американский инженер-механик, исследователь принципов организации производства и систем человек-машина, создатель метода наглядного упорядочения работ - **диаграммы Ганта** (Gantt Charts).

Идея Ганта состояла в том, что главным ресурсом планирования является время, а основой принятия управленческих решений - сравнение запланированного и фактического состояния работ.

На диаграммах Ганнта по горизонтали обычно показывают интервалы времени, а по вертикали - работы, операции, оборудование.

Горизонтальные отрезки отражают длительность выполнения работ. Выбрав по горизонтальной оси текущий момент времени и получив оперативную информацию о ходе производства, можно сопоставить фактическое состояние дел и планировавшееся.

## Роли в команде и их функции

### Для чего нужно разделение ролей

Каждый ИТ-специалист идет по пути наименьшего сопротивления, как правило, пытаясь сбросить часть работы на коллегу. Для избегания возможных конфликтов в команде нужно четко разграничить участок работ каждого участника, в том числе заказчика. Для целенаправленного выполнения проекта должен быть выполнен ряд работ, различных как по своему назначению, так и по квалификационным требованиям, предъявляемым к *разработчикам*. Иными словами, в ходе развития проекта командой *разработчиков* выполняются те или иные функции.

### Распределение функций между исполнителями

Функции, выполняемые разработчиками, — понятие неформализованное. В разных проектах оно может обретать свое содержание. Заметим, что в рамках деятельности менеджера любого проекта необходимо организовать *распределение функций* проекта между исполнителями. В результате ее выполнения члены команды, выполняющей проект, начинают играть соответствующие роли. Обычно роль объединяет родственные функции. Также принято обозначать роли их главными функциями. Продолжая только что приведенную иллюстрацию функций, выполняемых *разработчиками* проекта, укажем на следующие роли: кодировщик — действующее лицо, главной функцией которого является кодирование, аналитик — тот, кто занимается анализом требований. Подобную характеристику можно дать и *тестировщику*. Что же касается функции отладки, то в реальных проектах она обычно подразделяется на несколько видов: отладка компонентов, которой занимаются *разработчики* компонентов (например, те же кодировщики) и комплексная отладка, которая может поручаться специально выделенным сотрудникам или рассматриваться в качестве задачи группы *разработчиков*. Часто выполняются такие работы, как отладка спецификаций, декомпозиция проекта и др. Иными словами, функция отладки обычно не рассматривается как образующая роль, а распределяется по нескольким ролям в соответствии с принятой стратегией развития проекта. Роли назначаются на начальной стадии жизненного цикла проекта.

Не следует путать функции, которые предписано выполнять *разработчику* как исполнителю определенной роли, с поручениями в проекте. Поручения — это разовые или систематические задания, из которых обычно и складываются действия, необходимые для выполнения той или иной функции. Если для функции определен регламент выполнения поручений, т. е. последовательность выполнения составляющих ее поручений не требует дополнительных разъяснений для исполнителя, то такая функция называется технологической. В случае нетехнологической функции сотруднику, который выполняет соответствующую роль, приходится самому выстраивать нужную последовательность. Иными словами, технология здесь уступает место ремеслу.

При разработке любых проектов естественно стремление к повышению их технологичности, к установлению регламента для как можно большего числа функций. И одним из способов достижения этого для менеджера является использование работников с нужной квалификацией,

для которых поручаемые им роли оказываются технологичными, т.е. состоящими только из *технологических функций*. К сожалению, реальность такова, что менеджеру приходится работать в условиях ограниченных возможностей в подборе кадров, а потому уровень технологичности выполнения проекта снижается по сравнению с идеальной ситуацией. Таким образом, в рамках любого проекта возникает задача повышения квалификации сотрудников. Для различных схем ведения проектов эта задача решается по-разному. Крайнюю точку зрения на проблему соответствия квалификации работников требованиям проекта отражает идея *экстремального программирования*, когда используется неформальная организация группы исполнителей проекта без четкого распределения ролей, а значит, и обязательств сотрудников. В этом случае провозглашается принцип, когда каждый в группе делает то, что он умеет делать лучше всего. И хотя все функции, которые должны выполняться, остаются, создается впечатление, что в группе исполнителей проекта исчезают роли. В результате возможны пробелы в разработке, в частности при анализе и декомпозиции проектируемой системы. Чтобы этого избежать, *разработчики* должны понимать, какую абстрактную роль они исполняют в каждый конкретный момент, выполнение каких проектных функций необходимо сейчас, как связаны между собой работы, как должны распределяться ресурсы. Словом, они должны обращать внимание на выполнение распределенных по группе менеджерских функций. И даже в этом случае схему *экстремального программирования* можно рекомендовать лишь для слаженных групп исполнителей с высоким уровнем коллективной ответственности.

Функции, выполняемые *разработчиками* в проекте, подразделяются на:

- Организационные
- Производственные

Первые создают условия для выполнения проектных заданий, вторые непосредственно связаны с этими заданиями. Часто неудачи проекта возникают из-за того, что менеджер не учитывает важность выполнения *организационных функций*. Так, обычно проектное задание фиксирует лишь то, что нужно предъявлять *заказчику* в качестве результатов. С точки зрения результатов просто не требуется знать, например, как организована передача проектных материалов между *разработчиками*, какая процедура отчетности предусматривается, но игнорирование задач реализации информационных потоков в проекте может привести к хаосу, что в конечном итоге отразится и на результатах.

Понятно, что как состав, так и значимость ролей *разработчиков* и тех, кто с ними связан, различаются в зависимости от выполняемого проекта, от коллектива исполнителей, принятой технологии, от других факторов. Неоднозначность выбора ролей приводит к тому, что их список часто становится непомерно велик. Чрезмерное увеличение числа есть следствие отождествления роли и функции и, соответственно, игнорирования понятия родственности функций. В то же время, если ролей выбрано недостаточно, есть опасность, что не все нужные в проекте функции будут охвачены планированием и управлением.

Также при выборе состава нужно учитывать направление того или иного человека. В случае крупных проектов необходимо нанимать узкопрофильным специалистов, так как они лучше знают свою сферу. Если у проекта небольшой бюджет, то можно сэкономить, наняв многопрофильных специалистов. К слову, есть такой класс разработчиков, как *full-stack разработчики* — это еще одна

попытка «работодателей» получить задешево, то, что никогда дешевым быть не могло. Это такой же психологический прием, как и «вы же профессионал! Вы же профессионал?»

Цепочку распределений ролей можно описать в следующем виде:

- спонсор (куратор) проекта (это сотрудник (как правило, руководитель высшего звена) организации, реализующей проект, который курирует проект со стороны организации (владельца проекта), обеспечивает общий контроль и поддержку проекта) назначает менеджера проекта и обеспечивает ему необходимую поддержку.
- менеджер проекта выбирает команду управления проектом, среди которых есть командный лидер (team leader).
- командный лидер назначает разработчиков.

#### [Подход компании Microsoft к распределению ролей](#)

Как конкретный разработчик может получать одновременно несколько ролей, так и роль может быть распределена между несколькими исполнителями. Когда менеджеру в конкретных условиях руководства коллективом придется распределять роли, он неизбежно столкнется с тем, что эта задача зависит и от специфики проекта, и от контингента исполнителей. В связи с этим уместно упомянуть об одном из ее решений, которое предлагается специалистами Microsoft в качестве универсального подхода.

Предлагается образовывать небольшие мобильные коллективы как атомарные производственные единицы с общей ответственностью за выполняемые задания — так называемые проектные группы. Такие группы строятся как многопрофильные команды, члены которых распределяют между собой ответственность и дополняют *области компетентности* друг друга. Группа состоит не более чем из 10 человек. Все они считаются обладающими сходным уровнем профессиональной подготовки, хотя и в разных областях индивидуальной специализации. Провозглашается равноправие членов группы и коллективная ответственность за выполняемые задания: проектная группа — команда равных. Все это позволяет сохранять внутри группы неформальные отношения. Вместо понятия роли для группы в целом определяются *ролевые кластеры*, которые заполняются точно так же, как происходит распределение ролей. В то время как за успех проекта ответственна вся команда, каждый из ее *ролевых кластеров*, определяемых моделью, ассоциирован с одной из проектных целей и работает над ее достижением. В данной модели именно эти цели задают роли *разработчиков*, которые определяются кластерами. В терминологии используется понятие *области компетенции*, или *области функциональной специализации* (functional area), обозначающее ту или иную роль, которую выполняет кластер группы в проекте. Принципиальное отличие распределения исполнителей по *ролевым кластерам* от распределения ролей заключается лишь в том, что ответственность за это несет не *менеджер проекта*, а сама группа. Менеджер проекта выдает задания и контролирует их выполнение лишь в целом для группы, не вмешиваясь в ее работу.

Определено шесть **ролевых кластеров**, которые соответствующим образом структурируют проектные функции разработчиков:

- **Управление продуктом (product management).** Ключевая цель кластера — обеспечивать удовлетворение интересов заказчика. Для ее достижения кластер должен содержать следующие области компетенции:

- планирование продукта;
  - планирование доходов;
  - представление интересов заказчика;
  - маркетинг.
- **Управление программой (program management).** Задача — обеспечить реализацию решения в рамках ограничений проекта, что может рассматриваться как удовлетворение требований к бюджету проекта и к его результату. Области компетенции кластера:
  - управление проектом;
  - выработка архитектуры решения;
  - контроль производственного процесса;
  - административные службы.
- **Разработка (development).** Первостепенной задачей кластера является построение решения в соответствии со спецификацией. Области компетенции кластера:
  - технологическое консультирование;
  - проектирование и осуществление реализации;
  - разработка приложений;
  - разработка инфраструктуры.
- **Тестирование (test).** Задача кластера — одобрение выпуска продукта только после того, как все дефекты выявлены и устранены. Области компетенции кластера:
  - разработка тестов;
  - отчетность о тестах;
  - планирование тестов.
- **Удовлетворение потребителя (user experience).** Цель кластера — повышение эффективности использования продукта. Области компетенции кластера:
  - общедоступность (возможности работы для людей с недостатками зрения, слуха и др.);
  - интернационализация (эксплуатация в иноязычных средах);
  - обеспечение технической поддержки;
  - обучение пользователей;
  - удобство эксплуатации (эргономика);
  - графический дизайн.
- **Управление выпуском (release management).** Задача кластера — беспрепятственное внедрение и сопровождение продукта. Области компетенции кластера:
  - инфраструктура (infrastructure);
  - сопровождение (support);
  - бизнес-процессы (operations);
  - управление выпуском готового продукта (commercial release management).

Основные роли, встречающиеся на проекте и их обязанности

Существуют следующие роли на IT-проектах:

- **Заказчик (Customer)** — отвечает за:
  - своевременный просмотр спецификаций и других присыпаемых документов (с целью утвердить документ, дать комментарии, исправить неточности и т.п.);
  - внесение замечаний, дефектов, пожеланий в багтрекинговую систему;

- своевременный просмотр каждого выпуска и предоставление комментариев.
- **Планировщик ресурсов (Planner):**
  - выдвигает и координирует требования к проектам в организации, осуществляющей данную разработку;
  - развивает и направляет план выполнения проекта с точки зрения организации.
- **Менеджер проекта (Project Manager) — отвечает за:**
  - проектная документация;
  - составление плана проекта;
  - согласование сроков;
  - анализ возможных рисков;
  - участие в подборе и утверждении проектной команды;
  - разбивка продукта на компоненты и раздача их исполнителям;
  - определение требуемых ресурсов и рабочей среды, их распределение внутри команды;
  - постановка рабочего процесса в команде (разработка, тестирование, работа с требованиями);
  - определение приоритетности задач;
  - организация работы команды вокруг требуемой задачи;
  - отслеживание состояния проекта, хода выполнения задач;
  - отслеживание должной приоритетности выполнения задач;
  - отслеживание нагрузки задачами и прогресса по задачам каждого разработчика;
  - отслеживание сроков выполнения задач;
  - удерживание команды в рабочем состоянии, мотивация команды;
  - создание прозрачной среды общения между всеми участниками процесса;
  - отслеживание удовлетворенности проектом со стороны команды;
  - решение всевозможных конфликтных ситуаций внутри команды и в связке заказчик-команда;
  - общение с заказчиком, управление его ожиданиями;
  - предоставление заказчику отчетности о ходе выполнения задач и проекта в целом;
  - презентация заказчику готовых решений, демоверсий, прототипов;
  - интервьюирование новых членов команды.
- **Руководитель команды (Team Leader)**

Руководитель команды — это нечто среднее между проектным менеджером и квалифицированным разработчиком. Командный лидер обязан перевести бизнес-задачу в понятную техническую для разработчиков и сказать не только то, что нужно сделать, но и зачем это нужно.

На проектах есть две роли: менеджерская — PM, и техническая — System Architect. Командный лидер отчасти выполняет обе роли, но акцент его обязанностей направлен на менеджмент (акцент на техническую часть — это tech lead).

*Под управленческую роль TL попадают такие обязанности, как:*

- менеджмент;
- распределение и делегирование задач:

- всевозможные оценки и составление рабочего графика;
- контроль состояния проекта;
- проведение митингов;
- коммуникации с заказчиком, руководством и всеми членами команды (разработчиками, архитекторами, тестировщиками, менеджерами).

Под техническую роль TL попадают:

- участие в написании технической документации;
  - выбор технологий для проекта;
  - разработка архитектуры;
  - обзор и анализ кода (code review);
  - контроль и наставление молодых разработчиков;
  - проведение технических собеседований;
  - грамотное вовлечение новых членов команды в рабочий процесс;
  - ответственность за техническую часть проекта.
- **Системный аналитик (Technical Leader)** — отвечает за:
    - координацию и контроль деятельности по дизайну, архитектуре и кодированию;
    - поддержку контроля версий;
    - настройку скрипта для авто-билдера и своевременную сборку версий.
  - **Архитектор (Architect)** — отвечает за:
    - проектирование архитектуры системы;
    - согласование развития работ, связанных с проектом.

Архитектор — это человек, который решает, как в конечном итоге будет выглядеть информационная система организации в целом и в деталях. Основная цель архитектора в компании заключается в том, чтобы обеспечить решение задач бизнеса при помощи информационных технологий. Причем, он должен не только сформировать решение, но и контролировать правильность его реализации.

Внутри профессии существуют специализации: *функциональная* и *техническая*. В первом случае архитектор в большей степени отвечает за общение с бизнесом и по результатам контактов определяет конструкцию системы, которая нужна заказчику. Во втором ИТ-архитектор в основном общается с разработчиками и конструирует систему изнутри.

Не всякой компании нужен ИТ-архитектор. На небольших предприятиях или там, где информационные проекты не слишком масштабны, функции ИТ-архитектора может выполнять опытный менеджер проекта, разработчик или иной технический специалист в сфере ИТ.

Иметь собственного ИТ-архитектора необходимо, в первую очередь, крупным компаниям с развитой функциональностью унаследованных систем, разветвленной региональной оргструктурой и имеющим согласованные руководством планы развития ИТ.

- **Эксперт предметной области (Domain Expert)** — отвечает за:
  - изучение сферы приложения;
  - поддержку направленности проекта на решение задач данной области.

- **Разработчик (Developer):**
    - разработку качественного кода;
    - проведение модульного тестирования;
    - поддержку контроля версий;
    - написание пользовательской документации, относящейся к инсталляции и администрированию.
- Это широкое понятие, которое может подразделяться на специальные роли (например, разработчик классов). В зависимости от сложности проекта команда может включать различное число разработчиков.
- **Бизнес аналитик (Business Analyst)** отвечает за:
    - выяснение и анализ всех требований заказчика;
    - фиксирование всех требований заказчика (в багтрекинговой системе и в функциональных спецификациях), прослеживание всех изменений в требованиях;
    - написание и поддержка спецификаций.
  - **Разработчик информационной поддержки (Information Developer):**
    - создает документацию, сопровождающую продукт, когда выпускается версия. Включаемые в нее инсталляционные материалы, равно как ссылочные и учебные, а также материалы помощи предоставляются на бумажных и машинных носителях.
    - Для сложных проектов возможно распределение этих задач между несколькими разработчиками информационной поддержки.
  - **Специалист по пользовательскому интерфейсу (Human Factors Engineer):**
    - отвечает за удобство применения системы;
    - работает с заказчиком, чтобы удостовериться, что пользовательский интерфейс удовлетворяет требованиям.
  - **QA менеджер (Quality Assurance manager) —** отвечает за:
    - организацию и контроль процесса тестирования в проекте;
    - планирование тестирования;
    - участие в адаптации процесса разработки под проект, анализ его качества;
    - анализ результатов тестирования и качества продукта;
    - участие в управлении требованиями;
    - участие в настройке багтрекинговой системы, полное прослеживание багов;
    - контроль готовности нового выпуска для QA.
  - **QA аналитик (QA Analyst) —** отвечает за:
    - подготовку тест дизайна;
    - написание тест кейс спецификаций;
    - проведение тестирования;
    - регистрацию багов;
    - прослеживание и проверку багов;
    - написание документации пользователя.

#### Возможность совмещения ролей

Роли	Характеристика совмещения ролей
Менеджер и архитектор	Желательно

Менеджер и руководитель команды	Противоречиво
Руководитель команды и архитектор	Возможно
Руководитель команды и проектировщик подсистемы	Нежелательно
Менеджер и разработчик	Не допускается
Для различных разработчиков	Эффективно с ограничениями
Создание документации (все сотрудники)	Успешно распределяется
Специалист по интерфейсу и менеджер	Разумно
Эксперт предметной области и менеджер	Зачастую разумно
Специалист по интерфейсу и эксперт предметной области	Редко бывает эффективно
Эксперт предметной области и разработчик	Бывает полезно
Специалист по интерфейсу и разработчик	Часто полезно
Библиотекарь и один из разработчиков	Допустимо
Тестировщики и другие члены команды	Перекрестно
Эксперт предметной области, тестировщик	Оправданно

#### Распределение ролей посредством RACI-матрицы

Модель RACI — средство для выявления активностей и распределения их по ролям и зонам ответственности. Использование матрицы RACI позволяет избежать непонимания в том, кого необходимо привлечь к проекту, а также кто и что должен делать.

RACI — сокращение от основных ролей участников проекта:

- **Responsible (Исполнитель):** Тот кому назначена эта роль отвечает за выполнение работы и достижение целей проекта. На каждом этапе может быть несколько исполнителей.
- **Accountable (Ответственный):** Исполнитель этой роли отвечает за качество и результаты процесса. Обладатель этой роли обеспечивается полномочиями для обратной связи с исполнителями. На каждом этапе может быть только один ответственный.
- **Consulted (Консультант, Эксперт):** Тот кому назначена эта роль привлекается, как носитель уникальных знаний или информации. Часто в этой роли выступают эксперты в предметной области.
- **Informed (Информируемый):** Это лицо, которого необходимо держать в курсе о ходе и результатах процесса, чаще всего в одностороннем порядке, т.к. у него нет полномочий напрямую влиять на ход проекта.

Иногда в эту модель добавляются и другие роли, например, S — supported (Оказывающий поддержку).

	Director Service Management	Service Level Manager	Problem Manager	Security Manager	Procurement Manager
Activity 1	A	C	I	I	C
Activity 2	A	R	C	C	C
Activity 3	I	A	R	I	C
Activity 4	I	A	R	I	
Activity 5	I	I	A	C	I

Для того, чтобы понимать, по какому принципу такая табличка должна рисоваться, а также как ее использовать на практике (в реальных проектах), рекомендуется уделить должное внимание следующему порядку действий при построении матрицы:

- Определяется список необходимых активностей/процессов в поставленной задаче (проекте)
- Определяется и указывается функциональные роли (людей, которые заинтересованы или которых тем или иным образом касается данная задача)
- Собирается митинг и назначаются RACI коды (собственно — буквы) конкретным ролям, непосредственно разграничитываются ответственности
- Определяются несоответствия (например, слишком много ответственных либо отсутствие таковых)
- Описывается таблица и собираются отзывы
- Контролируется выполнение назначенных ролей

По функциональным ролям, анализировать можем, отвечая на такие вопросы:

- *Много «A»* — правильно ли распределены обязанности? Есть ли в наличии «узкие места»?
- *Много «R»* — не многовато ли ответственности для одной роли?
- *Отсутствие пустых ячеек в таблице* — действительно ли эта роль должна быть вовлечена в такое количество задач?

Также, проводится анализ по выполняемым активностям:

- *Более одного «A»* — только одна роль должна быть подотчетной
- *Отсутствие «A»* — необходимо найти подотчетного

- Более одного «R» или отсутствие такового – кто-то должен быть ответственный, однако нетребуется, чтобы ответственность была широко распределена – есть риск того, что задача не будет выполнена
- Много «C» — стоит ли консультироваться с многими ролями и будет ли это эффективно?
- Отсутствие «C» и «I» — правильно ли установлены коммуникации?

### Пример использования RACI-матрицы

Допустим, есть авиакомпания, которая на своем сайте собирается внедрить систему online check-in. Глобальные активности, необходимы к выполнению в контексте задачи, будут приблизительно следующие: сбор требований к системе; дизайн решения; непосредственная разработка решения (development); внедрение; собственно – стадия “production”; оптимизация решения.

Далее – определяется список функциональных ролей, в данной задачи возможны такие варианты: внутренний сервис провайдер (IT отдел авиакомпании) или же внешний сервис провайдер в случае отсутствия первого; ISP – компания предоставляющая хостинг для сайта авиакомпании; бизнес подразделение авиакомпании (представляющее интересы заказчика); финансовое подразделение (бухгалтерия); сервис менеджер (в зависимости от размеров организации, может входить во внутренний IT отдел); команда разработчиков (в зависимости от размеров организации, может входить во внутренний IT отдел).

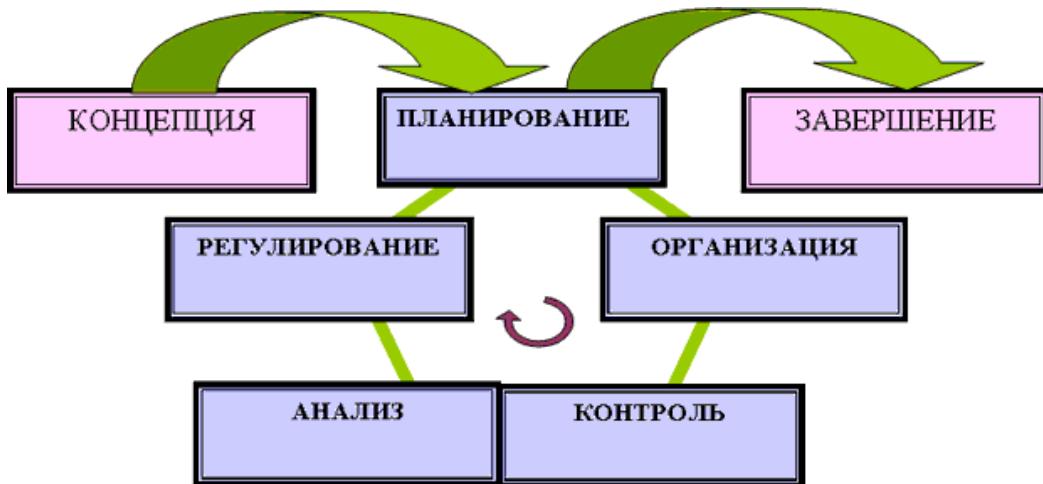
Попробуем расставить RACI коды соответственно ролям и выполняемым ими активностям (ясно, что данный процесс проходит при участии всех сторон).

	IT Отдел	ISP	Бизнес подразделение	Бухгалтерия	Сервис менеджер	Разработчики
Требования	R	I	C	C	A	I
Дизайн	C	C	I	I	AR	C
Разработка	C	I	I	I	C	AR
Внедрение	AR	R	C	I	C	C
Production	AR	R	I	I	I	I
Оптимизация	R	C	C	C	A	R

### Управление IT-проектами

#### Контроль качества

Управление качеством проекта осуществляется на протяжении всего жизненного цикла проекта. На рисунке представлены стадии управления качеством проекта.



Стадия "Концепция". На этой стадии определяется политика и стратегия для обеспечения качества разрабатываемого продукта, удовлетворяющего ожидаемым запросам потребителя. "Концепция" имеет следующие разделы:

- Политика и стратегия качества;
- Общие требования и принципы обеспечения качества;
- Стандарты, нормы и правила;
- Интеграция функций обеспечения качества;
- Требования к системе управления качеством.

Стадия планирования. На стадии планирования качества определяются стандарты, которые следует использовать, чтобы содержание проекта оправдывало ожидания участников проекта. Планирование качества включает как идентификацию этих стандартов, так и поиск путей их реализации. Ниже перечислены основные задачи стадии планирования:

- определение показателей оценки качества;
- определение технических спецификаций;
- описание процедур управления качеством;
- составление списка объектов контроля;
- выбор методов и средств оценки качества;
- описание связей с другими процессами;
- разработка плана управления качеством.

Стадия организации. Стадии организации контроля качества предполагает создание необходимых и достаточных организационных, технических, финансовых и др. условий для обеспечения выполнения требований к качеству проекта и продукции проекта и возможностей их удовлетворения.

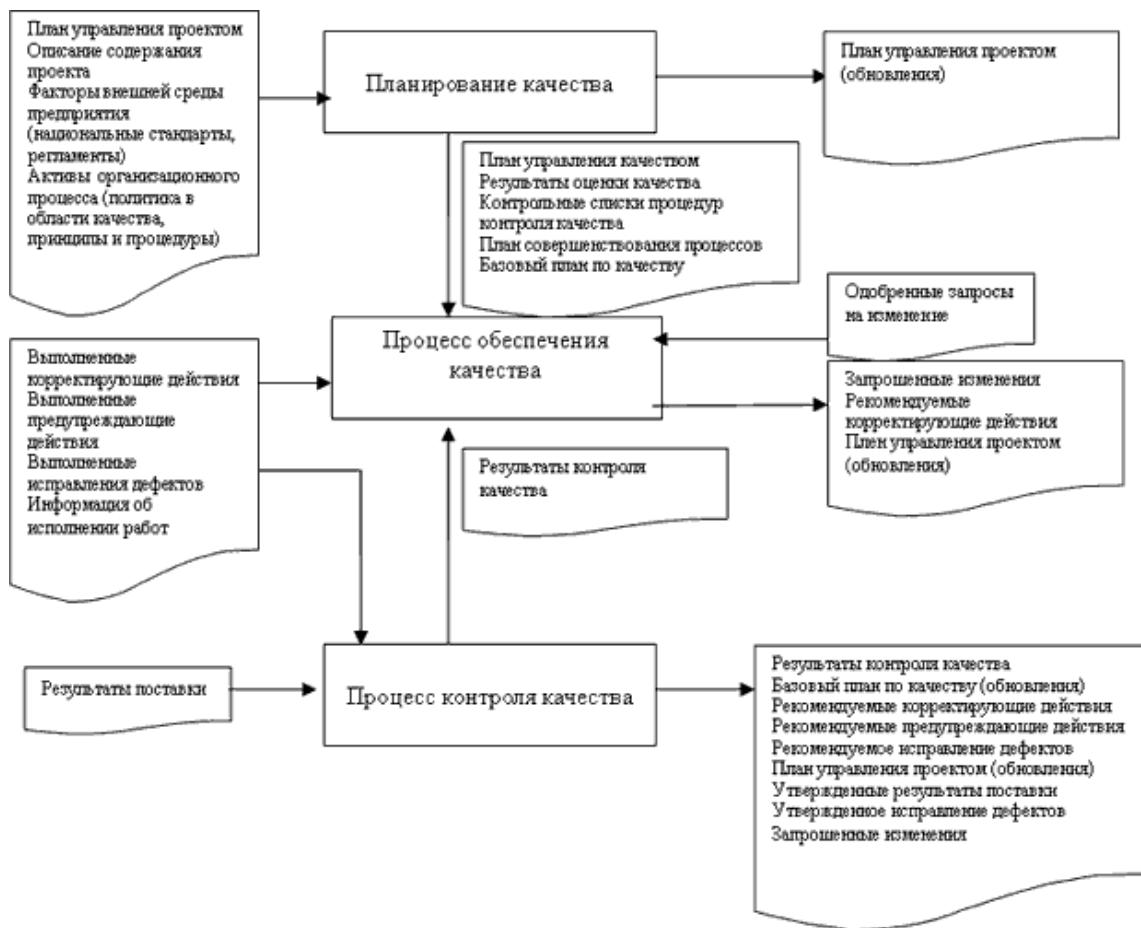
Стадия контроля. Контроль качества заключается в определении соответствия результатов проекта стандартам качества и причин нарушения такого соответствия.

Стадии регулирования и анализа. Стадия осуществления контроля качества предполагает регулярную проверку хода реализации проекта в целях установления фактического соответствия определенным ранее требованиям.

- Сравнение фактических результатов проекта с требованиями.
- Анализ прогресса качества в проекте на протяжении его жизненного цикла.
- Формирование списка отклонений.
- Корректирующие действия.
- Документирование изменений.

Стадия завершения. На стадии завершения выполняются сводная оценка качества результатов проекта, завершающая приемка, составление списка претензий по качеству, разрешение конфликтов и споров, оформление документации, анализ опыта и полученных уроков по управлению качеством.

Основными процессами обеспечения качества проекта являются планирование качества, его обеспечение и контроль. Связь этих процессов, их входы и выходы представлены на рисунке.



**Планирование качества** - процесс определения того, какие из стандартов качества относятся к данному проекту и как их удовлетворить.

Планирование качества осуществляется как часть планирования проекта и выполняется совместно руководителем проекта, архитектором проекта и ответственным за качество проекта. В план управления качеством включаются работы, выполнение которых обеспечивает качество результатов проекта. Одной из главных составляющих плана управления качеством IT-проектов,

является план проведения тестирования. Пример формы плана тестирования представлен на рисунке.

№ Сценария	Сценарий	Предпосылки	Плановая дата	Место проведения	Ответственные	Участники
<p><b>Комментарий к форме:</b></p> <p>1) № Сценария: Уникальный идентификатор сценария тестирования. 2) Сценарий: Название сценария тестирования. 3) Предпосылки: Перечень предварительных условий, которые должны быть выполнены перед тем, как приступить к прохождению сценария. Если таким условием является необходимость предварительного прохождения других сценариев, приводятся номера этих сценариев. 4) Плановая дата: Плановая дата проведения тестирования в формате ДД.ММ.ГГГГ. 5) Место проведения: Предполагаемое место проведения тестирования (указывается полный адрес, и, по возможности, номер комнаты). 6) Ответственные: Перечень сотрудников, ответственных за подготовку и проведение тестирования (в формате И.О.Фамилия). 7) Участники: Перечень участников тестирования (в формате И.О.Фамилия). В тестировании в обязательном порядке должны участвовать координаторы соответствующих функций и процессов.</p> <p>Сценарии в таблице упорядочены по плановой дате, далее - по номеру.</p>						

План по качеству должен определять, как в проекте будет обеспечено качество выполнения работ с позиции организационной структуры, ресурсов, методического обеспечения. На стадии планирования качества рекомендуется разработать документы, регламентирующие действия по контролю качества управления проектом (форму отчетности по выполнению проекта, анкеты мониторинга проекта) и процедуры управления качеством, например контроль качества результатов проекта, контроль качества документов проекта, утверждение документов проекта, подготовка и проведение контроля проекта. Для контроля качества документов проекта в плане по качеству следует определить список лиц, согласующих и утверждающих каждый документ проекта, сроки и форму их согласования. Пример плана согласования документа приведен на рисунке.

План согласования документа		
Список лиц, участвующих в согласовании:		
№	ФИО	Должность
1		
2		
План согласования:		
<ul style="list-style-type: none"><li>- До 18:00 XX.XX.2008, лицам, участвующим в согласовании пакета документов, необходимо представить замечания в группу управления качества на электронный адрес: _____</li><li>- Способ согласования: по электронной почте.</li><li>- XX.XX.2008 будет организовано совещание для обсуждения замечаний.</li><li>- При отсутствии замечаний до указанного срока документ будет считаться согласованным и будет передан на утверждение.</li></ul>		

На IT-проектах вводится множество специфических терминов, поэтому в план контроля качества проекта необходимо включать разработку и согласование глоссария проекта. Глоссарий проекта представляет собой структурированный список всех терминов и определений проекта, а также используемых аббревиатур с кратким описанием их смысла. Как правило, Руководитель группы управления качеством отвечает за пополнение и работу с Глоссарием проекта на основании поступающих

документов. Проверка глоссариев документов проводится в рамках времени, отведенного на общий контроль качества документов. Пример формы Глоссария представлен на рисунке.

<b>Термин/ Определение*</b>	<b>Английское название</b>	<b>Сокращение</b>	<b>Объяснение*</b>	<b>Область*</b>	<b>Документ*</b>
<p>Комментарий к форме глоссария:</p> <p>* пункты, обязательные для заполнения</p> <p><b>Термин/Определение</b> - используемый в документе термин или определение</p> <p><b>Сокращение</b> - принятое сокращение или аббревиатура</p> <p><b>Объяснение</b> - краткое объяснение смысла термина или определения</p> <p><b>Область</b> - указание, к какой области деятельности проекта относится данный термин:</p> <ul style="list-style-type: none"> <li>Техническая архитектура</li> <li>Обучение</li> <li>Поддержка</li> <li>Приложение</li> <li>Проектная терминология (в том числе методологическая)</li> </ul> <p><b>Документ</b> - из Глоссария какого документа поступил данный термин/определение</p>					

**Планирование качества** - процесс определения того, какие из стандартов качества относятся к данному проекту и как их удовлетворить.

Планирование качества начинается с определения целей качества проекта, политик и стандартов, относящихся к содержанию проекта. Потом определяются действия и обязанности членов команды, выполнение которых необходимо для достижения целей и соблюдения стандартов. Результат планирования качества представляется в форме планов обеспечения качества и процессов управления, обеспечивающих выполнение этих планов, и достигается путем синхронизации с основными (планирование содержания, расписания, стоимости) и вспомогательными (планирование рисков, команды) процессами планирования.

Заказчик устанавливает стандарты качества. Стандарты могут быть международными, национальными или корпоративными. После того как стандарты качества установлены, нужно определить задачи, решение которых обеспечит соответствие стандартам, далее закрепляется ответственность за выполнение намеченных работ и сроки их исполнения.

#### Входная информация процесса планирования качества

**Факторы внешней среды предприятия** - правила, стандарты и предписания, свойственные определенным областям приложения.

**Активы организационного процесса** - политика в области качества, принятая на предприятии, процедуры и предписания, базы данных и накопленные знания из предыдущих проектов.

**План управления проектом** обеспечивает интеграцию процесса планирования качества с другими процессами планирования.

**Описание содержания проекта** является ключевым входом для планирования качества, так как оно содержит описание главных целей проекта, а также критериев приемки и пороговых величин значения стоимости, времени или ресурсов.

#### Планирование качества: инструменты и методы

**Иерархическая структура работ** (она же структурная декомпозиция работ) — это разбиение проекта на более мелкие и измеримые части. **ИСР** описывает все результаты или работы, которые должны быть получены или выполнены для завершения проекта.

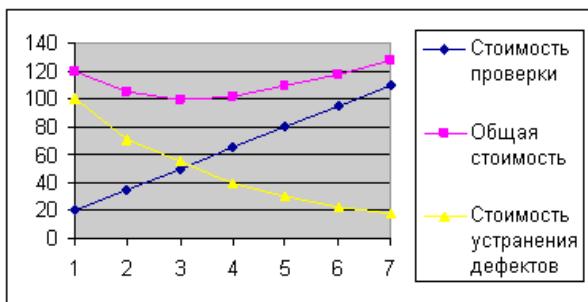
Для планирования качества проекта рекомендуется использовать нижеследующие методы:

**Программа обеспечения качеством** - план действий, обеспечивающий соответствие фактического качества запланированному.

**Анализ выгод и затрат.** При планировании качества необходимо принимать во внимание соотношение прибыли и затрат. Цель метода - выдержать необходимое соотношение между доходами и затратами в проекте. Обеспечение качества проекта, несомненно, приводит к дополнительным расходам, поэтому для каждого предложенного метода обеспечения качества необходимо анализировать коэффициент рентабельности.

Основная выгода от затрат на профилактику дефектов заключается в уменьшении числа доработок, что в свою очередь ведёт к уменьшению затрат.

На рисунке представлен выбор оптимальной пропорции затрат на профилактику дефектов и устранение дефектов.



**Бенчмаркинг** включает в себя сопоставление проекта с другими проектами с целью выработать идеи для повышения качества исполнения проекта.

**Планирование экспериментов** - статистический метод, позволяющий определить факторы, которые оказывают влияние на определенные

переменные величины продукта или процесса.

**Стоимость качества** - совокупная стоимость всех действий, направленных на повышение качества, а также на предупреждение факторов, способных вызвать снижение качества.

#### *Планирование качества: выходы*

**План управления качеством** - описание того, каким образом команда управления проектом будет осуществлять политику в области качества.

Мероприятия по обеспечению качества должны быть разработаны в самом начале проекта и проводиться на основе независимых экспертных оценок.

**Контрольные списки процедур контроля качества** - структурированный документ, который используется для подтверждения выполнения всех намеченных операций. Такие списки позволяют убедиться в правильной последовательности действий в часто выполняемых задачах.

**Базовый план по качеству** содержит требования к качеству данного проекта и служит основой для оценки по исполнению требований качества.

**Обновления в плане управления проектом.** Обновление плана происходит вследствие добавления к нему вспомогательного плана управления качеством.

## Процесс обеспечения качества

Обеспечение качества - процесс выполнения плановых систематических операций по качеству, которые обеспечивают выполнение всех предусмотренных процессов, необходимых для того, чтобы проект соответствовал установленным требованиям по качеству.

Функцию обеспечения качества может выполнять команда проекта, руководство исполняющей организации, заказчик или спонсор, другие участники проекта. Для контроля качества проекта проводятся аудиторские проверки, целью которых является выяснение, удовлетворяет ли качество проекта стандартам, установленным в плане обеспечения качества.

### *Входы процесса обеспечения качества*

- **План управления качеством**, содержащий описание, как будет осуществляться обеспечение качества в рамках проекта.
- **Результаты оценки качества.**
- **План улучшения процесса.**
- **Информация об исполнении работ** – это информация (о состоянии результатов поставки, о необходимых корректирующих действиях, а также отчеты об исполнении), которая используется при проведении аудита, экспертной оценке качества и анализе процессов.
- **Одобренные запросы на изменение.**
- **Контрольные списки качества.** Контрольный список представляет собой страницу с инструкциями для проверяющего лица. Пункты списка должны быть достаточно значимыми, поскольку, если контрольный список будет перегружен, его не будут использовать.
- **Результаты контроля качества** – результат выполнения операций по контролю качества. Данные о результатах контроля передаются исполняющей организации для использования в процессе обеспечения качества, для повторной оценки и анализа стандартов качества.

### *Процесс обеспечения качества: инструменты и методы*

**Инструменты и методы планирования качества** могут использоваться для операций по обеспечению качества.

**Аудит качества** - независимая экспертная оценка, определяющая, насколько операции проекта соответствуют правилам и процессам. Целью аудита качества является выявление неэффективных и экономически не оправданных правил и процессов. Количество и сроки плановых проектных аудитов могут определяться основными этапами проекта или ключевыми событиями. Внеплановые аудиты проводятся по запросам Заказчика, руководителей департаментов и отделов.

Схема проведения внутреннего аудита качества проекта может выглядеть следующим образом:

- анализ исправления замечаний предыдущей проверки;
- проведение проверки проекта в соответствии с контрольными списками;
- оформление отчета о контроле качества;
- информирование команды проекта о появлении новых отчетных документов.

**Анализ процесса** предусматривает выполнение действий, описанных в плане улучшения процесса и направленных на выявление организационных и технических моментов, которые нуждаются в улучшении.

### *Процесс обеспечения качества: выходы*

**Запрошенные изменения** имеют целью проведение специальных мероприятий по повышению эффективности правил, процедур и процессов в исполняющей организации.

**Рекомендованные корректирующие действия.** **Корректирующее действие** - это рекомендованное к немедленному исполнению действие, выработанное в результате мероприятий по обеспечению качества (аудита или анализа процессов).

**Обновления в активах организационного процесса.** Обновленные стандарты качества используются в дальнейшем процессе контроля качества.

**Обновления в плане управления проектом.** План подлежит обновлению согласно изменениям, в плане управления качеством, выработанным в результате процесса обеспечения качества. Запрошенные изменения в план управления проектом и во вспомогательные планы (добавления, изменения, удаления) подвергаются экспертной оценке и вносятся в соответствующие планы в процессе общего управления изменениями.

### *Процесс контроля качества*

**Контроль качества** - процесс, который включает отслеживание промежуточных результатов проекта, определение их соответствия принятым стандартам и разработку действий для устранения причин, вызывающих отклонения от стандарта.

**Цель контроля качества** – определение соответствия результатов поставки определенным требованиям.

Управление качеством должно производиться на всех этапах выполнения проекта. Количественная оценка контроля качества осуществляется на основе статистического анализа и теории вероятности.

### *Процесс контроля качества: входы*

- **План управления качеством.**
- **Результаты оценки качества.**
- **Контрольные списки процедур контроля качества.**
- **Активы организационного процесса.**
- **Информация об исполнении работ** включает техническое измерение исполнения, состояние завершенности результатов поставки проекта и исполнение необходимых корректирующих действий.
- **Одобренные запросы на изменение** могут содержать такие изменения, как исправленные методы работы и исправленное расписание.
- **Результаты поставки.**

### *Процесс контроля качества: инструменты и методы*

Следует отметить, что большинство инструментов качества, являются средствами визуализации и организации знаний, которые систематическим образом облегчает понимание и конечную диагностику проблем.

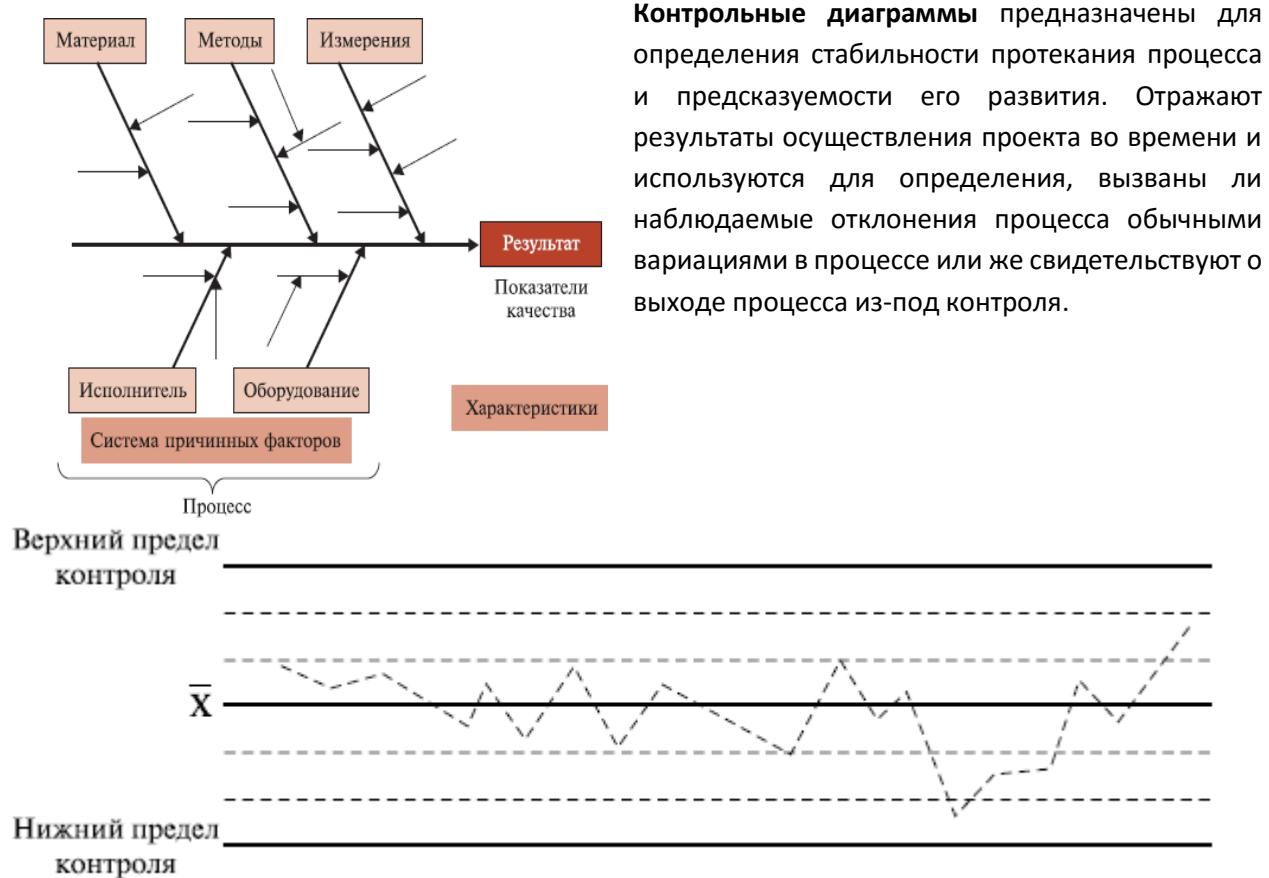
Для осуществления контроля качества используют следующие методы и средства:

**Диаграмма причинно-следственных связей** помогает отразить возможные причины, влияющие на качество продукта или процесса в проекте. Такая диаграмма, которую также называют диаграммой

Искавы или диаграммой рыбьего скелета, иллюстрирует связь различных факторов с возможными проблемами или эффектами.

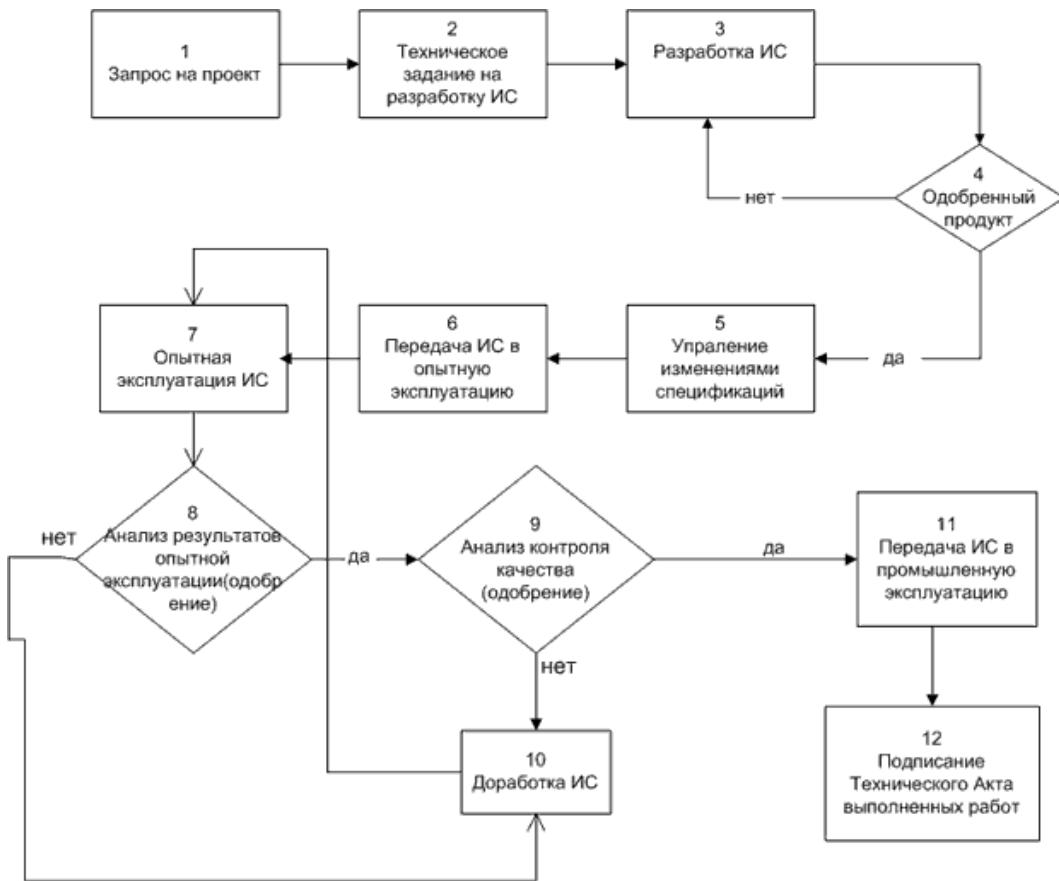
Диаграмма позволяет в простой и доступной форме систематизировать все потенциальные причины рассматриваемых проблем, выделить самые существенные и провести поуровневый поиск первопричины.

Проблема обозначается основной стрелкой. Факторы, которые усугубляют проблему, отражают стрелками, покосившимися к основной вправо, а те, которые нейтрализуют проблему — с наклоном влево. К стрелкам факторов могут быть добавлены стрелки, влияющих на них факторов второго порядка и т. д.



При помощи контрольной диаграммы можно определять, как внесенные изменения повлияли на улучшение процесса, - это осуществляется посредством постоянного мониторинга выходных данных процесса во времени. Контрольные диаграммы могут использоваться для отображения жизненного цикла как проекта, так и продукта. Например, применение контрольных диаграмм в проекте позволяет определить, насколько отклонения по стоимости и отклонения по срокам выходят за рамки допустимых пределов (скажем, +/-10 процентов). Контрольные диаграммы можно использовать для наблюдения за любыми выходными переменными. Хотя контрольные графики чаще всего нужны для отслеживания повторяющихся операций, они также могут применяться для наблюдения за колебаниями издержек и исполнением расписания, за объемом и частотой изменения содержания проекта, за ошибками в документах проекта или другими

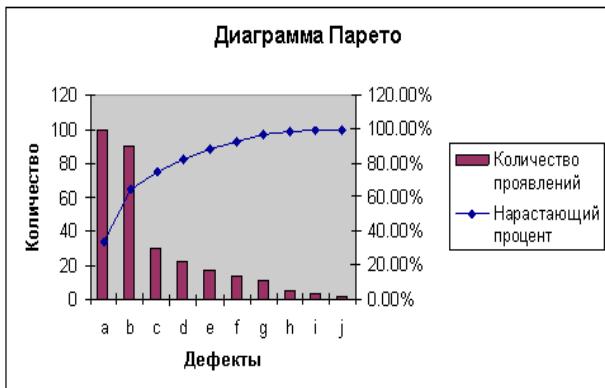
результатами управления. Это позволяет определить, насколько действенным является процесс управления проектом.



**Диаграммы зависимостей** помогают анализировать причины возникновения проблем. Диаграмма зависимостей представляет собой графическое отображение процесса. Существует множество различных стилей представления этих диаграмм, но все они отображают операции, точки принятия решений и порядок обработки данных. Диаграммы зависимостей дают представление о том, как различные элементы системы взаимодействуют между собой. Такая диаграмма зависимостей может оказать помощь команде проекта в прогнозировании, где и какие могут возникнуть проблемы с качеством, - и, следовательно, в разработке мер по их предотвращению.

**Диаграмма Парето** представляет собой особый тип гистограммы, упорядоченной по частоте возникновения.

Диаграмма Парето — это инструмент, позволяющий распределить усилия для разрешения проблем и выявить основные причины, с которых нужно начинать действовать. Метод анализа Парето заключается в классификации проблем качества на немногочисленные, но существенно важные и многочисленные, но несущественные. Он позволяет распределить усилия и установить основные факторы, с которых нужно начинать действовать с целью преодоления возникающих проблем.



При использовании диаграммы Парето для выявления результатов деятельности и причин наиболее распространенным методом является ABC-анализ.

Сущность **ABC-анализа** в данном контексте заключается в определении трех групп, имеющих три уровня важности для управления качеством:

- группа А — наиболее важные, существенные проблемы, причины, дефекты (80%).
- группа В — причины, которые в сумме имеют не более 20%;
- группа С — самые многочисленные, но при этом наименее значимые причины и проблемы.

ABC-анализ позволяет обоснованно определять приоритеты работ по управлению качеством проекта.

Диаграммы Парето логически связаны с Законом Парето, который гласит, что относительно малое число причин обычно приводит к большинству проблем или дефектов. Этот закон также известен как принцип 80/20, согласно которому 80 процентов проблем создается 20-ю процентами причин.

**Схема прогноза** отображает историю и модель изменений. Она представляет собой линейный график, отображающий точки ввода данных, расположенные на графике в порядке их возникновения. Схема прогноза дает представление о трендах процесса во времени, колебаниях во времени, а также о позитивных и негативных изменениях процесса во времени

**Статистические выборки** - это часть контролируемой продукции, позволяющей сделать вывод обо всей продукции данного вида в проекте.

**Инспекция** включает такие процессы, как тестирование, предпринятое с целью определения соответствия результатов проекта принятым требованиям и стандартам. Различают тестирование как отдельных процессов, так и их совокупности (интеграционное тестирование).

**Проверка исправления дефектов** - это действие, предпринимаемое отделом контроля качества, чтобы удостовериться, что дефекты продукта исправлены и сам продукт полностью соответствует Техническому заданию и спецификации.

#### *Выходы процесса контроля качества*

- Результаты контроля качества** представляют собой результаты мероприятий по контролю качества.
- Добавление в базовый план по качеству (обновления).**
- Рекомендованные корректирующие действия** - определенные мероприятия, проведение которых вызвано результатами операций по контролю качества.
- Рекомендованные предупреждающие действия** - специальные мероприятия по предупреждению возникновения условий, при которых процессы проекта могут выйти за пределы установленных параметров.
- Обновления в плане управления проектом.**

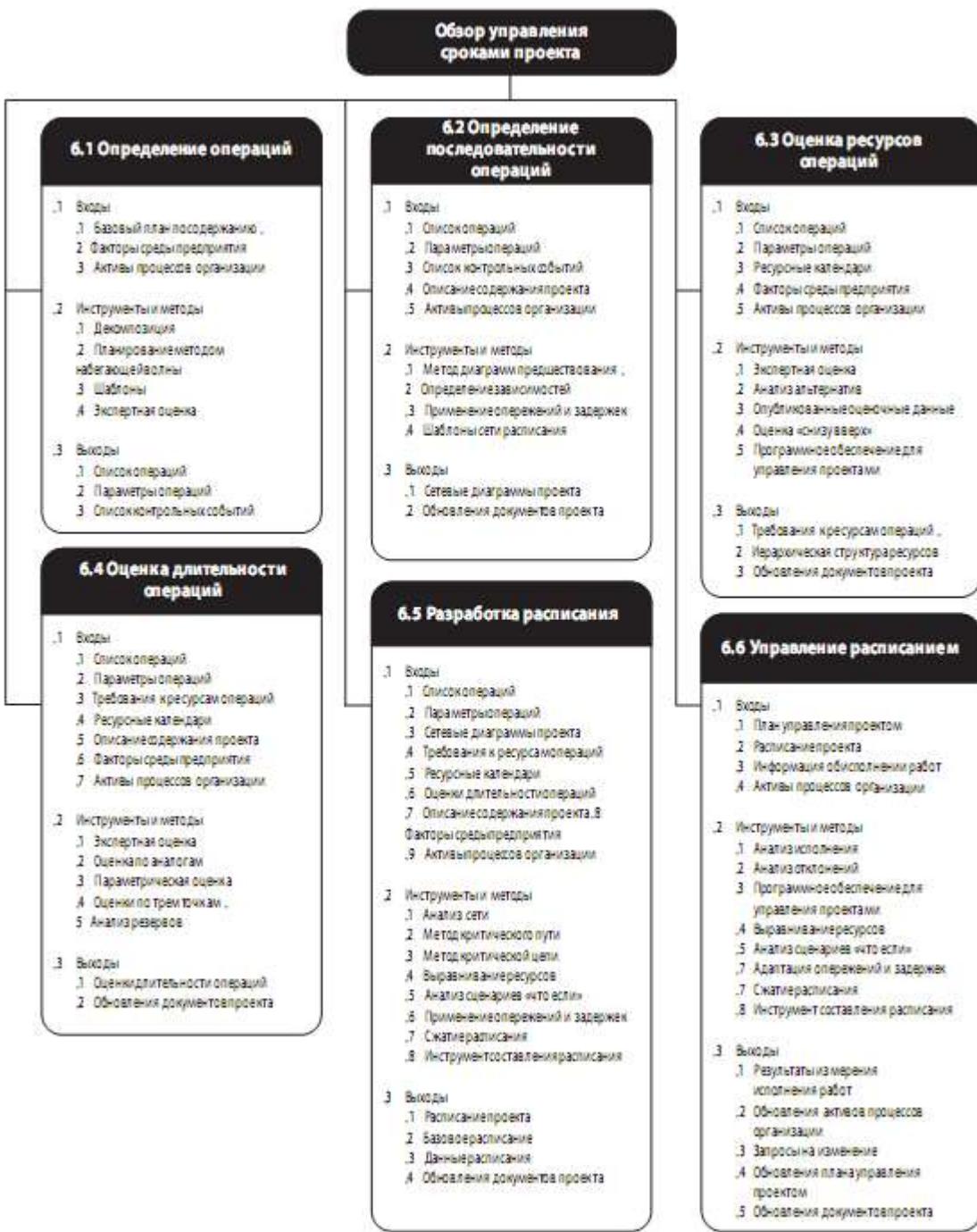
- **Обновления в активах организационного процесса**, содержащие заполненные контрольные списки и документацию о накопленных знаниях.
- **Утвержденные результаты поставки.**
- **Результатом процесса контроля качества является утвержденные результаты поставки.**

## Управление сроками проекта

Управление сроками проекта состоит из:

1. **Определение операций** – процесс определения конкретных операций, которые необходимо выполнить для получения результатов проекта.
2. **Определение последовательности операций** – процесс выявления и документирования зависимостей между операциями проекта.
3. **Оценка ресурсов операций** – процесс оценки типов и количества материалов, человеческих ресурсов, оборудования или поставок, необходимых для выполнения каждой операции.
4. **Оценка длительности операций** – процесс приблизительного определения количества рабочих периодов, требуемых для завершения отдельных операций при предполагаемых ресурсах.
5. **Разработка расписания** – процесс анализа последовательностей операций, их длительности, потребности в ресурсах и временных ограничений для создания расписания проекта.
6. **Управление расписанием** – процесс мониторинга статуса проекта для корректировки его исполнения и внесения изменений в базовое расписание.

## Общая схема управления сроками проекта



### Определение операций

**Определение операций** – процесс определения конкретных операций, которые необходимо выполнить для получения результатов проекта. В процессе разработки Иерархической Структуры Работ (ИСР) определяются результаты самого нижнего уровня – операции, которые описывают работу, необходимую для выполнения пакета работ. Операции предоставляют основу

для оценки, планирования, исполнения, мониторинга и контроля работ по проекту. Подразумевается, что определение и планирование операций расписания в данном процессе проводятся таким образом, который обеспечивает достижение целей проекта



#### Определение операций: входы

##### 1. Базовый план по содержанию

Результаты, ограничения и допущения проекта документируются в базовом плане по содержанию и детально рассматриваются при определении операций.

##### 2. Факторы среды предприятия

Факторы среды предприятия, которые могут оказывать влияние на процесс определения операций, включают в себя информационную систему управления проектами, не ограничиваясь только ей.

##### 3. Активы процессов организации

Активы процессов организации, которые могут оказывать влияние на процесс определения операций, включают в себя, среди прочего:

- существующие формальные и неформальные, связанные с планированием, правила, процедуры и руководящие указания, такие как методология составления расписания, которые учитываются при определении операций;
- базу накопленных знаний, содержащую историческую информацию относительно списков операций, использованных в предыдущих подобных проектах.

#### Определение операций: инструменты и методы

##### 1. Декомпозиция

##### 2. Планирование методом набегающей волны

Планирование методом набегающей волны представляет собой вид планирования способом последовательной разработки, при котором работа, которая должна быть выполнена в ближайшей перспективе, планируется в деталях на низшем уровне ИСР, а работа в отдаленном будущем планируется на более высоком уровне ИСР

##### 3. Шаблоны

В качестве шаблона для нового проекта зачастую можно использовать стандартный перечень операций из предыдущего проекта или его часть.

##### 4. Экспертная оценка

#### Определение операций: выходы

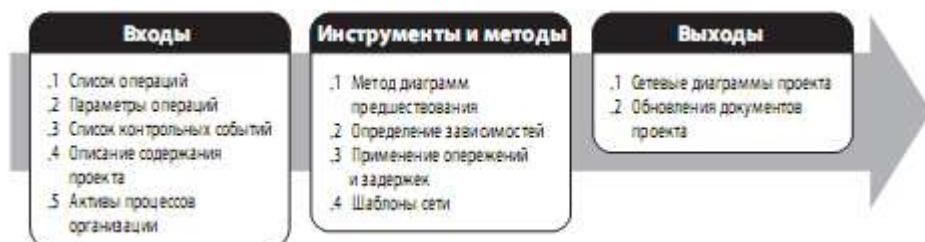
- Список операций** - исчерпывающий перечень, включающий все операции расписания, предусмотренные для данного проекта. В список операций входят идентификатор операции и описание содержания работ по каждой операции, подробное настолько, чтобы члены команды проекта понимали, какие работы необходимо провести.

2. **Параметры операции** - расширяют ее описание путем определения ряда элементов, связанных с каждой операцией. Элементы каждой операции формируются с течением времени. На первоначальных стадиях проекта они могут включать в себя идентификатор операции, идентификатор ИСР и название операции, а в конце формирования – коды и описание операции, перечни предшествующих и последующих операций, логические взаимосвязи, опережения и задержки, требования к ресурсам, статусные даты, ограничения и допущения.
3. **Список контрольных событий** – список важных моментов или событий проекта. Список контрольных событий определяет все контрольные события, указывая при этом, является ли контрольное событие обязательным (например, необходимым согласно контракту) или необязательным (например, основывающимся на исторической информации).

#### *Определение последовательности операций*

**Определение последовательности операций** – процесс определения и документирования взаимосвязей между операциями проекта. Определение последовательности операций осуществляется с помощью логических взаимосвязей.

Каждая операция и контрольное событие, кроме первых и последних, связаны по крайней мере с одной предшествующей и одной последующей операцией. Иногда бывает необходимо использовать время опережения или задержки между операциями для поддержания реалистичного и достижимого расписания проекта. Определение последовательности может быть выполнено с помощью программ управления проектами или с помощью автоматических или ручных методов



#### *Определение последовательности операций: входы*

- Список операций
- Параметры операций
- Список контрольных событий
- Описание содержания проекта
- Активы процессов организаций

#### *Определение последовательности операций: инструменты и методы*

- Метод диаграмм предшествования
- Определение зависимостей
- Применение опережений и задержек
- Шаблоны сети

## Определение последовательности операций: выходы

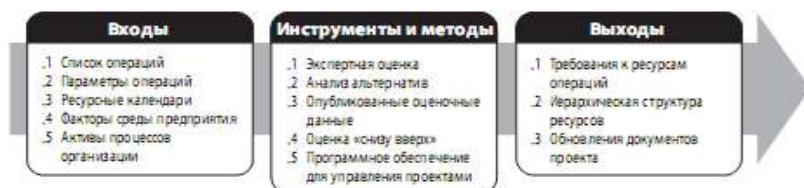
- Сетевые диаграммы проекта
- Обновленные версии документов проекта

## Оценка ресурсов операций

**Оценка ресурсов операции** – это процесс **оценки** типа и количества материалов, человеческих ресурсов, оборудования или поставок, необходимых для выполнения **каждой** операции.

Команда проекта в сфере строительства должна быть знакома с местными строительными нормами и правилами. Это знание может быть получено у местных представителей. Однако в том случае, когда местная рабочая сила не имеет опыта применения нетрадиционных или специализированных строительных технологий, наилучшим способом получения знаний о местных строительных нормах и правилах будет приглашение консультанта.

Команда проекта в области автомобилестроения должна быть знакома с передовыми методами автоматизированной сборки. Для приобретения требуемых знаний можно воспользоваться услугами приглашенного консультанта, отправить проектировщика на семинар по вопросам робототехники или включить в команду проекта представителя производственного сектора.



## Оценка ресурсов операций: входы

- Список операций
- Параметры операций
- Ресурсные календари
- Факторы среды предприятия
- Активы процессов организации

## Оценка ресурсов операций: инструменты и методы

- Экспертная оценка
- Анализ альтернатив

У многих запланированных операций имеются альтернативные методы их реализации. К ним относится использование различных уровней способностей или навыков ресурсов, машин различных габаритов или типов, различных инструментов (ручных или автоматических), а также принятие решений «производить или покупать» в отношении ресурсов.

- Публикуемые оценочные данные

Некоторые компании регулярно публикуют данные о производительности и единичные расценки ресурсов по широкому спектру рабочих профессий, материальных средств и оборудования по различным странам и регионам отдельных стран.

- Оценка «снизу-вверх»
- Программы управления проектами

### Оценка ресурсов операций: выходы

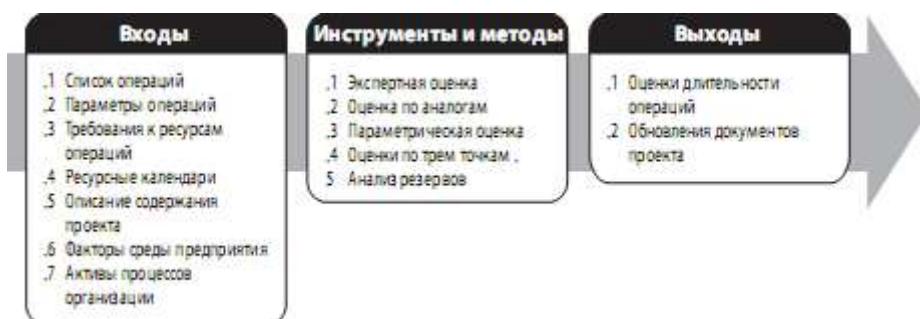
- Требования к ресурсам операций
- Иерархическая структура ресурсов
- Обновленные версии документов проекта

### Оценка длительности операций

**Оценка длительности операции** – процесс приблизительного определения количество рабочих периодов, требуемых для выполнения отдельных операций при предполагаемых ресурсах.

При оценке длительности операции используется информация о содержании работ операции, требуемых типах ресурсов, оценках количества ресурсов, а также ресурсных календарях. Входы для оценки длительности операций исходят от одного или нескольких членов команды проекта, в наибольшей степени знакомых с характером работ определенной операции.

Оценка длительности постепенно уточняется, и процесс учитывает качество и доступность данных на входе. Например, по мере выполнения инженерно-конструкторских работ по проекту данные становятся более детальными и определенными, при этом повышается точность оценок длительности. Таким образом, можно считать, что с течением времени оценка длительности постепенно становится более точной, а ее надежность повышается



### Оценка длительности операции: входы

- Список операций
- Параметры операций
- Требования к ресурсам операций
- Ресурсные календари
- Описание содержания проекта
- Факторы среды предприятия
- Активы процессов организации

### Оценка длительности операций: инструменты и методы

- Экспертная оценка
- Оценка по аналогам
- Параметрическая оценка
- Оценки по трем точкам
- Анализ резервов

### Оценка длительности операций: выходы

- Оценки длительности операций

Оценки длительности не включают в себя какие-либо задержки. Оценки длительности операций могут включать и диапазон возможных значений.

- Обновленные версии документов проекта

#### *Разработка расписания*

**Разработка расписания** – процесс анализа последовательностей операций, их длительности, требований к ресурсам и временных ограничений для создания расписания проекта. Ввод операций, длительностей и ресурсов в инструмент составления расписания генерирует расписание с запланированными датами завершения операций проекта.

Разработка приемлемого расписания проекта зачастую является итеративным процессом. Он определяет запланированные даты **старта** и **финиша** операций и контрольных событий проекта.

**Разработка расписания** может потребовать проведения анализа и проверки оценок длительности и ресурсов для создания утвержденного расписания проекта, способного служить в качестве базового плана, по которому будет проходить отслеживание исполнения. Пересмотр расписания и поддержание его реалистичности продолжается на всем протяжении проекта по мере выполнения работ, изменения плана управления проектом и выявления характера событий риска



#### *Разработка расписания: входы*

- Список операций
- Параметры операций
- Сетевые диаграммы проекта
- Требования к ресурсам операций
- Ресурсные календари
- Оценки длительности операции
- Описание содержания проекта
- Факторы среды предприятия
- Активы процессов организации

#### *Разработка расписания: инструменты и методы*

- Анализ сети
- Метод критического пути
- Метод критической цепи
- Выравнивание ресурсов

- Анализ сценариев «что если»
- Применение опережений и задержек
- Сжатие расписания
- Инструмент составления расписания

#### [Разработка расписания: выходы](#)

- Расписание проекта
- Базовое расписание
- Данные расписания
- Обновленные версии документов проекта
- Требования к ресурсам операций
- Параметры операций
- Календарь
- Реестр рисков

**Диаграммы контрольных событий** - данные диаграммы показывают только запланированные даты начала или завершения получения основных результатов и ключевые внешние события

**Ленточные диаграммы** – те диаграммы, в которых полоски представляют операции, показывают даты начала и завершения операций и их ожидаемые длительности. Ленточные диаграммы сравнительно легко читаются и часто используются для представления информации высшему руководству организаций.

**Сетевые диаграммы проекта**– содержат информацию о датах операций, обычно показывают как логику сети проекта, так и операции критического пути проекта.

#### [Управление расписанием](#)

**Управление расписанием** представляет собой процесс мониторинга статуса проекта для оценки его исполнения и управления изменениями базового расписания.

Управление расписанием связано с:

- определением текущего состояния расписания проекта;
- влиянием на факторы, вызывающие изменения расписания;
- определением фактов изменения расписания проекта;
- управлением фактическими изменениями по мере их возникновения.

Управление расписанием является элементом процесса осуществления общего управления изменениями



#### Управление расписанием: входы

- План управления проектом
- Расписание проекта
- Информация об исполнении работ
- Активы процессов организации

#### Управление расписанием: инструменты и методы

- Анализ исполнения
- Анализ отклонений
- Программы управления проектами
- Выравнивание ресурсов
- Анализ сценариев «что если»
- Адаптация опережений и задержек
- Сжатие расписания
- Инструмент составления расписания

#### Управление расписанием: выходы

- Результаты измерения исполнения работ
- Обновленные активы процессов организации
- Запросы на изменение
- Обновленный план управления проектом
- Обновленные версии документов проекта

## Совместная работа в проектировании

### О чём мы говорим

Сегодня мы немного отдалимся от технологий и уделим свое внимание собственно процессу разработки. Речь пойдет о причинах перехода от индивидуальной работы к коллективной. Далеко не все так просто в организации этого процесса и получении от него конкретных результатов. Посмотрим, как это все организуется и к чему нужно стремиться при организации.

### Причины перехода от отдельных специалистов к коллективам

#### Усложнение технологий.

Одна из наиболее очевидных причин. Если вы беретесь за создание какого-нибудь продукта, то скорее всего ваших знаний в определенной области окажется недостаточно. Его совершенство только в одном критерии не избавит от краха в другом. И как следствие, ваш продукт будет сильно уступать вашим конкурентам.

Вообще говоря, в свое время я убедился в том, что в современной науке и технике почти не осталось примитивных технологий.

#### Стремление как можно быстрее выйти на рынок.

Еще одна важная побудительная сила, которая вынуждает проектировщиков сплачиваться в коллективы, состоит в стремлении как можно быстрее вывести новый проект (новый продукт) на рынок. Согласно одному эмпирическому правилу, компания, которая первой вывела на рынок продукт нового типа, вполне может рассчитывать на долговременное обладание долей рынка в 40%, тогда как остальная часть будет неизменно распределяться между многочисленными, не столь проворными конкурентами. Кроме того, опередившая всех компания может собирать всю прибыль в тот период времени, пока конкуренты не могут предоставить свои аналоги. Встречаются и такие случаи, когда компания, впервые представившая на рынке определенный продукт, так и продолжает доминировать на нем. Эти законы рынка известны всем, поэтому соблюдение планов разработки проектов всегда находится в центре внимания руководства компаний. Коллективное проектирование становится необходимым, если оно позволяет ускорить поставку нового продукта в конкурентной среде. Причины усиления конкурентной борьбы в части сроков выхода новых продуктов состоят в следующем. Системы связи и передачи данных охватили весь мир, а рынки стали глобальными, поэтому зародившиеся где бы то ни было привлекательные идеи в наше время распространяются очень быстро.

### Затраты

“Чем больше рук, тем легче становится работа”. Так бывает часто.

“Но чем больше рук, тем больше становится работы”. Так бывает всегда.

На каждого участника трудового процесса ложится меньшая нагрузка, поэтому продолжительность выполнения всей работы сокращается. Но возможности распределения задач проектирования по исполнителям ограничены, причем весьма невелико количество таких задач, которые могут быть сведены к более мелким в достаточной степени. Поэтому в связи с организацией совместной работы возникают дополнительные издержки.

Стоимость разделения на подзадачи. Само разделение задач проектирования на подзадачи представляет собой достаточно серьезную дополнительную проблему. Дело в том, что при этом должны быть четко и точно определены интерфейсы между подзадачами, а это связано с выполнением большого объема работы, требующей внимательного контроля. По мере выполнения проектных работ приходится непрерывно пересматривать интерфейсы, определяющие взаимодействие подзадач, независимо от того, насколько точно они были определены с самого начала. Не исключено и возникновение потенциальных нестыковок. В ходе проектирования неизбежно возникают несогласованности в определениях и конфликты в интерпретациях; все эти расхождения должны быть своевременно устранены. В целях упрощения подготовки изделия к выпуску обязательно следует провести стандартизацию общих элементов по всем компонентам. Кроме того, необходимо добиться определенной общности стиля проектирования во всем коллективе. А в конечном итоге отдельные части проекта должны быть собраны в единое целое, что становится окончательной проверкой правильности интерфейсов. В наши дни нельзя больше руководствоваться принципом, применявшимся при сборке судна на старинных верфях: "Отрезай по чертежу, подгоняй по месту"

Стоимость изучения и обучения. Если в проектировании совместно участвуют п человек, то все они должны стремиться быстрее всего достичь поставленных целей, выполнить предъявленные требования, соблюсти ограничения и добиться оптимизации функции полезности. Вся группа проектировщиков должна иметь общее представление обо всех этих аспектах проектирования, т.е. хорошо представлять себе, что должно быть получено в конечном итоге. В качестве первого приближения можно утверждать, что если работа по проектированию, выполняемая одним человеком, состоит из двух частей (изучение  $l$  и разработка  $d$ ), то общий объем работы  $work$  с привлечением  $n$  человек больше не измеряется формулой  $work = l + d$ , а составляет по меньшей мере  $work = n * l + d$ . Более того, определенная часть сотрудников, обладающая лучшим представлением о будущем проекте и большим объемом знаний, должна заниматься обучением других и в связи с этим не принимать участия в самом проектировании. Таким образом, остается лишь надеяться на то, что благодаря преимуществам специализации хотя бы некоторые из этих затрат окупятся.

Расходы на обеспечение связи и передачи данных в процессе проектирования. На протяжении всего процесса проектирования совместно работающие проектировщики должны быть уверены в том, что создаваемые ими отдельные части в конечном итоге будут успешно состыкованы. Для этого необходимо организовать между проектировщиками обмен структурированными данными

Управление изменениями. Должен быть введен в действие такой механизм управления изменениями, с помощью которого каждый проектировщик мог бы вносить только те изменения, которые, во-первых, затрагивают лишь порученные ему части проекта или, во-вторых, согласованы с проектировщиками частей проекта, затрагиваемых предлагаемыми изменениями. В действительности значительную долю затрат на проектирование составляют изменения и переделки, поэтому стоимость управления изменениями становится достаточно заметной. Но если формальный процесс управления изменениями не организован, то приходится нести гораздо большие издержки.

## **Достижение концептуальной целостности – основная цель**

Одним из наиболее важных внешних признаков качественного проекта является целостность и единообразие его концепций. Согласованность во всем конкретного проекта может не только вызывать восторг, но и способствовать удобству в изучении и простоте в использовании.

Даже если намеченное к выпуску изделие является очень крупным, технически сложным или предназначенным для создания в кратчайшие сроки, не допускается ни малейшего отступления от требования, чтобы это произведение конструкторской мысли многих участников разработки выглядело как концептуально целостное с точки зрения каждого пользователя. Безусловно, такая целостность обычно становится естественным следствием проектирования, осуществляемого одним человеком, но для ее достижения при коллективном проектировании приходится решать нетривиальные управленческие задачи, требующие большого внимания.

Предпосылками создания поистине инновационных, а не клонированных продуктов являются физическая изолированность коллектива, небольшое количество сотрудников, полное сосредоточение на полученном задании и единоличное руководство со стороны выдающегося человека. В качестве примеров можно указать разработку самолета Spitfire автономным коллективом под руководством Джо Митчелла (Joe Mitchell), которая проводилась в Херсли-Хаус, величественном здании в Хэмпшире, Великобритания; работы секретного подразделения Skunk Works компании Lockheed, возглавляемые Келли Джонсоном (Kelly Johnson), в результате которых были созданы самолет-шпион U-2 и истребитель F-117, выполненный по технологии Stealth; деятельность секретной лаборатории IBM в г. Бока-Ратон, шт. Флорида, позволившая IBM успешного догнать Apple в области разработки персонального компьютера.

## **Способы достижения целостности**

- 1) Единый архитектор, который владеет всеми технологиями, использующиеся в проекте
- 2) Создание единообразного пользовательского интерфейса, каждый элемент которого дополняет остальные, а не работает произвольно.

## **Условия успешной совместной работы**

### **Определение потребностей и пожеланий со стороны заинтересованных лиц.**

Если исходить из того, что определение требований к проекту является наиболее сложной из всех задач проектирования, то в этой области увеличение количества привлеченных людей действительно позволяет добиться лучших результатов. Это невозможно отрицать! Коллектив, пусть даже небольшой, намного лучше по сравнению с отдельным человеком справляется с задачей выявления скрытых потребностей и с изучением существующей системы, подлежащей замене. Как правило, с привлечением различных людей удается рассмотреть гораздо больше разных вопросов, а сами эти вопросы становятся разнообразнее. С увеличением количества затрагиваемых тем удается лучше понять особенности разрабатываемого проекта. В коллективе должна царить атмосфера сотрудничества, гарантирующая, что каждый член коллектива получит полную возможность найти ответы на все интересующие его вопросы, касающиеся будущего продукта.

**Постановка целей.** При любой организации процесса проектирования разработчики начинают свою работу с общения с людьми, заинтересованными в успехе создаваемого проекта. При этом

основные рассматриваемые темы касаются целей и ограничений, в рамках которых разрабатывается проект. В ходе этого взаимодействия одной из самых сложных задач является обнаружение таких подразумеваемых целей и ограничений, о существовании которых потенциальные пользователи часто просто забывают. Фактически по итогам проводимых собеседований и на основе анализа того, что и как было сказано и что осталось невысказанным, проектировщик впервые оценивает функцию полезности создаваемого проекта. Одна из наиболее важных задач в этой фазе состоит в наблюдении за тем, как пользователь выполняет свою работу сегодня, в каких обстоятельствах и с помощью каких инструментальных средств. Часто бывает полезным проведение видеосъемки в ходе этих наблюдений и многократный просмотр полученных материалов обследования.

Успешному проведению этой фазы весьма способствует привлечение достаточно большого количества сотрудников. Чем больше людей,

- тем больше вопросов они задают,
- замечают различные нюансы, пусть даже о них ничего не было сказано,
- вырабатывают независимые и даже, возможно, противоречивые мнения по поводу того, что было ими услышано,
- наблюдают за разными аспектами организации труда в текущих условиях,
- продуктивно обсуждают видеоматериалы обследования.

### **Способы организации самого процесса проектирования**

#### **1. Мозговой штурм**

О организации мозгового штурма нас подробно рассказывал Антон

#### **2. Конкуренция**

В качестве альтернативного подхода при проведении фазы концептуального исследования можно попытаться задать стимулы к проявлению творческих способностей отдельных проектировщиков путем проведения конкурсов на лучшие решения по проекту.

Такие конкурсы становятся особенно удачными, если цели и ограничения известны, однозначно сформулированы и доведены до всех участников, а ненужные ограничения тщательно исключены.

#### **3. Спонтанно возникающие конкурсы между проектами — борьба за то, какой из продуктов останется на рынке.**

Нередко происходит так: коллектив проектировщиков, допустим, Б, настолько успешно разрабатывает свой проект, что перекрывает часть целей вывода на рынок своего продукта, которыми руководствуется коллектив проектировщиков А. В этом случае возникает соревнование между проектами особого рода — борьба за окончательный выход продукта на рынок. Порой это приносит очень большие плоды.

### **Контролирование процесса. Обзор проекта**

Той фазой проектирования, в которой совместная работа становится наиболее важной и даже необходимой, является обзор проекта. В обзоре должны участвовать специалисты по многим направлениям: другие проектировщики, пользователи и (или) представляющие их лица, конструкторы, покупатели, изготовители, специалисты по сопровождению, эксперты в области

надежности и безопасности, а также экологи. Специалист по каждому направлению должен выполнять обзор документации проекта отдельно от других, с учетом того, что тщательный обзор требует времени, размышлений, а также, возможно, изучения справочных материалов, архивов и других проектов. Благодаря этому каждый участник обзора вырабатывает собственное, уникальное мнение; в каждом случае выдвигаются разные вопросы и обнаруживаются различные недостатки. Но результаты обзора должны быть обязательно сведены воедино с участием всех представителей группы.

Использование графических представлений. Рисовать или как-то фиксировать то, что у вас происходит с проектированием. Наблюдать за процессом таким образом.

#### Что нужно помнить

Специалисты в области программирования вложили значительные усилия в создание инструментальных средств организации совместной работы на основе компьютеризации для собственной дисциплины и других дисциплин. Но вызывает разочарование то, что лишь немногие из выдвинутых ими идей и созданных инструментальных средств вошли в категорию применяемых повседневно.

Причина такого положения дел, по-видимому, состоит в том, что многие специалисты в области программного обеспечения, взявшие на себя задачу создания инструментальных средств проектирования, не смогли учсть некоторые важные особенности коллективного проектирования, осуществляемого в действительности.

- Реальные проекты всегда намного сложнее, чем можно было бы себе представить. Это особенно справедливо, поскольку самое обучение в области проектирования начинается с освоения примеров из учебников, которые в силу своего назначения чрезмерно упрощены. Реальное проектирование осуществляется с учетом необходимости достижения более сложных целей, в ходе него приходится учитывать более сложные ограничения, а конечный продукт оценивается по более сложным критериям качества. Фактически по мере развития процесса проектирования приходится соприкасаться со все более и более мелкими деталями, количество которых порой кажется бесконечным.
- В реальном коллективном проектировании всегда должен применяться процесс управления изменениями в проекте, чтобы левая рука не разрушала то, что сделано правой.
- Насколько углубленным не было бы сотрудничество, оно не может быть настолько глубоким, чтобы полностью устранить потребность в общении для бегства от "безысходности работы и одиночества размышлений".

#### Два – магическое число

В создании выдающихся творений человеческого ума нередко участвовали также двое.

И действительно, два считается магическим числом, когда речь идет об организации совместной работы. На эту тему уже было сказано очень много; достаточно отметить такое блестящее изобретение человечества, как семья (создаваемая двумя людьми).

Исходные показатели производительности двух совместно работающих проектировщиков, как правило, ниже, чем у двух проектировщиков, работающих отдельно, но относительное количество ошибок значительно меньше. Считается, что примерно 40% трудозатрат во многих проектах связано

с исправлениями, поэтому суммарная производительность проектировщиков, работающих в паре, становится выше, а сами продукты характеризуются большей надежностью.

Два человека обменяются идеями быстро и неформально, для чего не требуется брать слово для выступления или выяснить, кто из партнеров главный. Любой из этих двух сотрудников может высказывать свои замечания в любой момент. В процессе проектирования фактически происходят мгновенно сменяющиеся сеансы выдвижения предложений, обзора и критического анализа, высказывания встречных предложений, синтеза и принятия решения. Разработка идеи, как правило, происходит в виде единого потока обсуждения, в ходе которого не приходится задумываться над тем, что у кого-то из участников дискуссии могут быть иные мысли по сравнению с другими, как обычно происходит в дискуссиях с участием нескольких человек. Следы от двух вместе взятых карандашей можно провести по одному и тому же листу бумаги, не пересекая их и не сталкивая сами карандаши.

### [Дистанционное сотрудничество](#)

Масштабы современных проектов, осуществляемых крупными международными предприятиями с правительственной поддержкой, таковы, что требуют разделения труда между несколькими странами, что связано с созданием территориально разрозненных подразделений.

Рассмотрим в качестве примера проект создания самолета Airbus 380. Не только производство, но и разработка этого самолета были разделены между Францией, Германией, Великобританией и Испанией. Джекфри Джапп (Jeffrey Jupp), который в свое время занимал должность технического директора подразделения Airbus UK в Англии, рассказывал, как проектировались крылья самолета Airbus в одном городе, с учетом того, что они должны состыковываться с фюзеляжем, разрабатываемым в другом городе, и успешно поднимать его в воздух.

- Использовались все существующие возможности передачи данных.
- Каждый день самолет компании совершал полет из одного города в другой и обратно для обеспечения оперативного обмена материалами между подразделениями.

К сожалению, при разработке самолета Airbus 380 не удалось избежать связанных с дистанционным сотрудничеством потенциальных опасностей, которые еще больше усугублялись тем, что в проектировании участвовали представители разных стран. Во французском и британском коллективах проектировщиков использовалась пятая версия программного обеспечения CATIA CAD, в германском и испанском — четвёртая версия того же ПО. И вот, отчасти из-за различий между этими версиями, жгуты кабелей, спроектированные одним коллективом, имели больший диаметр. По этой и другим причинам произошли задержки начальных поставок, составляющие приблизительно 22 месяца.

Ещё один пример – разработка компьютеров семейства IBM System/360 в которой изначально принимали участие специалисты из стран: США, Великобритании и Германии, к которым затем присоединились Франция, Швеция и Нидерланды.

Связь между подразделениями из разных стран поддерживалась не только благодаря тому, что совершались тысячи телефонных звонков и передавались многочисленные документы, но и с помощью многих личных встреч. Неразрешенные конфликты и сложности улаживались на ежегодных двухнедельных встречах с участием всех групп, причем каждый раз удавалось устраниТЬ

до двухсот рассогласований. При этом ощутимы были пространственные барьеры между подразделениями и не менее важные временные и языковые барьеры.

### Качественные интерфейсы

Одной из самых сложных задач является определение качественных интерфейсов между компонентами, проектируемыми отдельно расположенным подразделениями. При этом на простом определении работы не заканчивается, приходится вносить изменения, управлять ими и доводить сведения об этом до всех заинтересованных лиц. Еще одной важной частью создания архитектуры системы становится не просто определение интерфейсов, но и организация применения руководителями заранее установленного механизма, предназначенного для устранения разногласий во мнениях или вкусах. Безусловно, какой бы ни была организация работы, решения всегда должен принимать уполномоченный на это руководитель. Труд руководителей оплачивается дорого, но если они по праву занимают свое место, то получаемая от них отдача чрезвычайно велика! От качества интерфейсов во многом зависит то, насколько велико будет количество ошибок, допущенных при проектировании. В литературе встречаются такие оценки, что стоимость исправления ошибок и переделки, пусть даже это касается лишь небольшой части проекта, может приближаться к половине затрат на проектирование. Но хуже всего то, что ошибки, обусловленные применением неточно определенных или произвольно трактуемых интерфейсов, обычно обнаруживаются слишком поздно, во время интеграции на уровне системы. Наиболее затруднительные в поиске, наиболее дорогостоящие в исправлении, эти ошибки способны сорвать график разработки всей системы.

Кроме того, качественные интерфейсы позволяют добиться подлинного удовлетворения от работы. Проектирование является увлекательным занятием, а улаживание разногласий с коллегами — обычно нет. Те, кто занимаются проектированием, чувствуют, как продвигается работа, а те, кому приходится улаживать разногласия в части трактовки интерфейса, жалеют о том, что время проходит даром. Если интерфейсы имеют высокое качество, то каждый из проектировщиков, участвующий в работе, ощущает радость от успеха, порученного ему дела и получает привилегию утвердить своей подписью завершение определенного этапа работы. Кроме того, применение качественных интерфейсов способствует упрощению процедуры передачи ответственности за выполняемую работу по мере того, как малые компоненты объединяются друг с другом в ходе создания все более значимых и крупных подсистем.

### Средства дистанционного сотрудничества

Проходит одно десятилетие за другим, но по-прежнему избавиться от необходимости в пространственных перемещениях проектировщиков не выходит. Возможно, когда-нибудь в результате создания все более удобных и приближенных к жизни технологий связи действительно удастся постепенно снизить необходимость в проведении личных встреч. Но человеческое общение характеризуется наличием бесконечного количества нюансов, поэтому необходимость пребывания время от времени в одном помещении сотрудников, занимающихся совместным проектированием, никогда не исчезнет.

### Документ

Наиболее продуктивным средством дистанционного сотрудничества является документ, совместно используемый на основе технологий передачи данных по сети или электронной почтой. Документ

содержит текст и изображения, передающие буквальный смысл, обеспечивает критический анализ и стимулирует взаимодействие.

### *Телефон*

Как средство дистанционного взаимодействия, на втором месте после документа стоит телефон, который иногда позволяет гораздо быстрее добиться согласования, чем электронная почта. В частности, многие пользователи электронной почты ощутили на себе последствия неправильного толкования их непродуманных писем, в условиях, когда нельзя дать устные пояснения и моментально устранить недоразумения. Мгновенная передача сообщений — не самая лучшая замена телефонной связи.

### *Видеоконференции*

Непосредственно после широкого распространения средств проведения видеоконференций началась шумиха по поводу того, что эти средства “изменят всю картину” в области дистанционного сотрудничества, но фактически уровень их практического применения возрастал не так быстро, как предполагалось первоначально.

В наши дни достигнуты приемлемые скорости передачи видеоданных. Но возросло ли благодаря техническим усовершенствованиям удобство в работе?

- Видеосвязь хорошо подходит для проведения сеансов взаимодействия двух собеседников, но если одна половина комитета проводит сеанс связи с другой, то невозможно уследить за всеми сразу.
- Желательно, чтобы можно было видеть одновременно и докладчика, и демонстрируемый им документ или слайд, а не пассивно воспринимать то, как видеокамера попеременно показывает то одно, то другое. В ходе участия в видеоконференции может возникнуть желание, например, чтобы содержимое какой-то таблицы было раскрыто более подробно. Любому из участников может потребоваться сделать личные заметки или внести замечание, которое могли бы увидеть все. В действительности при проведении видеоконференции нельзя обойтись без виртуальной доски, предоставляемой обеим сторонам в совместное использование.
- При проведении видеоконференций изображение обычно кажется плоским. Безусловно, из-за этого редко возникают недоразумения, касающиеся раскрытия темы самой видеоконференции, зато участники никак не могут забыть о том, что не беседуют со своими коллегами как при личной встрече.

В определенных обстоятельствах проведение видеоконференции оказывается более продуктивным по сравнению с телефонной связью, несмотря на все еще существующие технические ограничения.

Наиболее продуктивно использование различных средств дистанционного сотрудничества.

### *Крайняя важность личных встреч*

Чтобы понять, насколько важны личные встречи, достаточно подумать о том, как мы сами разговариваем по телефону. Не правда ли, что ощущается меньший комфорт и даже обнаруживается недостаточная продуктивность общения, если разговор ведется с незнакомцем, а

не с хорошо знакомым человеком. Обычно люди готовы даже совершить небольшое путешествие, чтобы избавиться от необходимости использовать такие средства связи, как оборудование для проведения видеоконференций, телефон, электронную и обычную почту, для решения следующих задач:

- Заключить сложную коммерческую сделку.
- Найти скидку на приобретаемую услугу.
- Уволить своего помощника с административной должности.

Безусловно, при решении некоторых подобных задач предпочтительным является использование электронной почты или телефона, а не личная встреча (для чего требуется согласовывать наиболее подходящее время), тогда как для достижения других целей многие не откажутся преодолеть некоторое расстояние. Известные примеры дистанционного сотрудничества, связанные с достижением наибольшего успеха, базировались на долгой истории личного знакомства, но даже и в этих случаях требовалось проведение дополнительных личных встреч в ходе продолжающегося дистанционного сотрудничества. Если же участники совместной работы, осуществляющейся в разрозненных подразделениях, мало знакомы друг с другом, то следует хоть иногда преодолевать разделяющие их пространственные барьеры, поскольку это оправдано с точки зрения затрат времени и денег.

Иногда не отдавая себе в этом отчета, высоко оценивают значение личных встреч и сами люди. Именно поэтому, несмотря на наличие столь мощной технологии проведения видеоконференций, самолеты по-прежнему заполнены людьми, отправляющимися в командировку для решения деловых вопросов.

## [Стратегии и история развития крупных IT-компаний](#)

### [Apple Inc.](#)

Оборот на 2015 год: \$233.715 миллиардов

Слоган: Think Different.

На самом деле про эту компанию легко можно написать несколько увесистых книг. Без преувеличения можно сказать, что **Apple** – одна из самых ярких технологических компаний, что пачками появлялись в 70-х годах прошлого века.

История «фруктовой компании» содержит в себе немало фольклорных мотивов. Многие, простые внешне события пересказываются на самые разные лады, обрастают мифами и домыслами, превращаясь в красивые легенды.

В качестве одной из легенд можно привести историю надкусанного яблока.

Логотип компании Apple не случайно относится к числу самых известных. Причин тому много — как громкая слава компании, так и узнаваемость самого логотипа. Есть такая старая (правда, не выдерживающая серьезной критики) теория, согласно которой логотип должен быть не просто запоминающимся, но и таким, чтоб любой пользователь мог в любой момент изобразить его на бумаге. Замечательными примерами могут служить автомобильные логотипы: Mercedes, Volkswagen, Opel... Не менее удачный пример и надкусанное яблоко от Apple.

Логотип этот несколько моложе самой компании. Дело в том, что в начале создатели хотели обыграть известную любому школьнику легенду про яблоко, что свалилось на голову Ньютона и позволило ему открыть закон всемирного тяготения. Идея в целом оригинальная, вот только выбранный логотип был явно громоздким, не слишком запоминающимся.



Первый логотип Apple. 1976 год

Логотип в виде надкусенного яблока разработал для компании представитель рекламного агентства Regis McKenna (Regis McKenna Advertising Agency). Согласно легенде (а именно с этого момента и начинаются легенды и домыслы), арт-директор агентства Роб Янов (Rob Janoff) накупил в ближайшем супермаркете яблок и начал экспериментировать, надрезая их, расставляя стройными рядами и вообще, изощряясь по-всякому. Но в итоге, разработка оказалось очень простой. Почему яблоко надкусено?

Есть две основные теории: первая правдоподобнее, она гласит, что так яблоко становится более «настоящим» и не напоминает какой-либо другой фрукт; вторая же основывается на похожести английских слов «byte» («байт») и «bite» («укус»). Был также случай, когда некий священник узрел в «надкусенности» явный намек на искушение Адама и Евы.

Говорят, также, что Джобс, уставший ждать логотип от Роба, наблюдая за его работой просто надкусил одно из яблок и сказал, что если в ближайшее время тот ничего не придумает, то пусть берет за основу это. Но эту версию можно поставить под сомнение, поскольку сам Роб ни о чем подобном никогда не упоминал.

Первое яблоко было радужным. Это стало поводом для появления еще одной теории, согласно которой, в надкусенном яблоке кроется глубинный смысл. Мол это прямой намек на самоубийство Алана Тьюринга — ученого, что так много сделал для информатики и вычислительной техники. Он был геем и, как рассказывает история, покончил с собой съев отравленное яблоко, не выдержав гонений общества.

Но теория «гомо-яблока» не выдерживает критики: дело в том, что радуга стала официальным логотипом сексуальных меньшинств несколько позже — впервые геи официально использовали

радугу в 1979 году, через три года после появления «яблочного» логотипа. Вероятнее всего, радуга была взята (а Джобс настаивал на ее использовании, согласно воспоминаниям Роба) как символ толерантности и взаимопонимания — именно этот смысл она и носила изначально. Ее часто использовали хиппи, к которым некогда относился и сам Джобс. Возможно также, что тем самым подчеркивался факт того, что компьютеры Apple способны работать с цветом, что в те годы было в новинку. Более того, гораздо правдоподобней кажется теория, согласно которой Apple в 1998 году отказалась от радужной окраски яблока именно в силу того, что радуга стала восприниматься как неизменная атрибутика секс-меньшинств, что могло навести на не самые приятные сравнения для компании, которая к тому времени активно формировала новый имидж.

Интересен тот факт, что Джобса отговаривали от использования радуги. Исключительно по причине того, что стоимость распечатки документов (а также наклеек на дискеты, руководств пользователя и т.п.) с таким количеством цветов в те годы была слишком высокой. Зато выглядела такая продукция не в пример лучше, чем у конкурентов с их монохромными распечатками. С первых же дней своего существования Apple привлекала потребителей вниманием к деталям.

Но история наша была бы неполной, если не рассказать того, что дизайнер Роб Янов ничего не получил за свою работу, даже благодарности. Джобсу удалось настолько втереться в доверие к Реджису МакКенне, что тот практически безвозмездно помогал молодой компании, предоставив возможность пользоваться услугами своих работников.

Причем замечание о фольклорных мотивах относится не только к самой **Apple**, но и к продукции ею производимой. Что и говорить, самый первый созданный ею компьютер — Apple 1, с которого все и начиналось в 1976 году, сейчас является настоящей коллекционной ценностью, оцениваемой в сотни тысяч фунтов стерлингов. Многих сейчас удивляет такое отношение к этому уродцу в деревянном ящике. Однако, именно этот компьютер заложил основу индустрии как таковой, именно с оглядкой на него делались впоследствии многочисленные решения от других производителей.

Рождение компании произошло от союза технического гения Стефана (Стива) Возняка (Stephen Woźniak) и маркетологического гения Стива Джобса (Steve Jobs). Познакомились они благодаря Биллу Фернандесу (Bill Fernandez), их общему знакомому, который решил свести будущих партнеров друг с другом, не сомневаясь в том, что они найдут общий язык. 26-летний Возняк тогда учился в университете Беркли, а более молодой Джобс уже закончил школу и пока еще не знал куда приткнуться. Они быстро нашли общий язык — Фернандес оказался прав.

Создавая компьютер, который впоследствии получит имя Apple 1, Возняк даже и не задумывался о какой-либо коммерции. Он просто любил технику и мечтал создать компьютер, с которым сможет работать «любая кухарка». Это был не первый персональный компьютер, но это был первый персональный компьютер, что имел монитор и клавиатуру, а не кучу лампочек и переключателей, как в случае с появившимся ранее Altair 8080. К тому же, Apple 1 продавался в виде готового устройства, а не набора «собери сам», как в случае с другими моделями.



В те годы Возняк все еще работал в **Hewlett-Packard** и по условиям контракта все его разработки принадлежали работодателям, которые имели право делать с ним все, что заблагорассудится. Но, к счастью, персональный компьютер совершенно не заинтересовал руководство компании.

Первый заказ на 50 000 долларов пришел от магазина **Byte Shop**, который считается ныне как самый первый компьютерный магазин в истории. Его уже давно не существует, однако, название его вошло в историю, которая, правда, обходит тот факт, насколько выгодным была данная сделка для владельца магазина, которого звали Пол Террелл (Paul Terrell). Продажная стоимость Apple 1 была 666 долларов 66 центов (создателям устройство обходилось в 500 долларов). Столь странная цена возникла из любви Возняка к повторяющимся цифрам. К 500 просто приплюсовали еще треть, как наценку за работу.



Официально **Apple Computers** была основана 1 апреля 1976 года. На самом деле основателей было трое. К знаменитой парочке присоединился их знакомый Рон Вейн (Ronald Wayne), работавший вместе с Джобсом в Atari. Ему принадлежало тогда 10% акций **Apple** и занимался он юридическими и бумажными вопросами. Впрочем, проработал он в компании совсем недолго — как только

основные юридические вопросы были решены, Вейн покинул **Apple** и отказался от своей доли (которая уже через десять лет измерялась сотнями миллионов долларов).

Модель Apple II (1977 год) оказалась настоящим шедевром. Это был первый компьютер, что выпускался компанией серийно. На её основе создавались впоследствии многие другие модели персональных компьютеров. К примеру, многие разработки ПК в СССР базировались именно на Apple II. Компьютер в корпусе, формы которого были позаимствованы у популярного в те годы кассетного магнитофона, пришелся по душе многим пользователям тех лет. Он прекрасно смотрелся и в офисе, и в университете. Модель продавалась до начала 90-х годов, лишь с незначительными изменениями в конструкции.



Одна из модификаций Apple II

А вот проект Apple III (1980 год) оказался провальным. Джобс настоял на избавлении от вентиляторов, справедливо полагая, что они будут мешать своим шумом. Вот только в результате получилось крайне ненадежное, склонное к перегреву устройство. К тому же, спешка, в которой оно собиралось, привела к резкому падению качества. Все это и привело к провалу на рынке.

Пока Apple II приносила деньги, компания занималась новыми проектами — **Lisa** и **Macintosh**, главной особенностью которых было наличие графического интерфейса, благополучно позаимствованного у **Xerox** (позже, таким же путем появится и **MS Windows**). Первый проект, названный в честь дочери Джобса, канул в Лету, второму, что был назван в честь сорта недорогих, но популярных яблок, суждено было стать новой легендой. В продаже **Mac** появился в 1984 году, чему предшествовал знаменитый рекламный ролик «1984», который до сих пор считается примером того, какой должна быть реклама. Первый **Macintosh** не был самым удачным с технической точки зрения компьютером. Скорее наоборот, красивый внешне, он являл собой настоящее скопище проблем, на устранение которых ушло немало времени. Однако, высокое уважение, которым пользовалась **Apple**, позволило ей превзойти все невзгоды. А постепенное превращение «Мака» в настоящий рабочий инструмент, сделало этот компьютер одним из самых популярных. Для него создали программные пакеты такие компании, как **Aldus** (программа

настольной верстки **Page Maker**) и **Microsoft** (электронные таблицы **Excel**). Аналоги данного ПО на РС появились лишь по прошествии некоторого времени.

В 1985 году **Apple** покидает Стив Джобс. Покидает, чтобы вновь вернуться по прошествии десятилетия и возродить свое, впавшее к тому времени в уныние, детище. Впрочем, слово уныние тут не совсем уместно. Сам Джобс отметил впоследствии, как он был удивлен и обрадован тому, что вернувшись в **Apple** увидел, что в ней по прежнему работает множество действительно интересных людей и талантливейших инженеров, которые в свое время принимали участие в создании множества интересных проектов. Сейчас принято считать, что Джобс вернулся в компанию, которая умирала. Но это неверно. Да, дела у **Apple** шли не самым лучшим образом, однако, тогдашнему CEO Джилу Амелио (Gil Amelio) удалось добиться очень многого, у компании не было долгов, она приносila хоть и небольшую, но прибыль.

За время отсутствия Джобса, компания создала такие решения, как ноутбуки **PowerBook** и карманные компьютеры **Newton**. Первые исключительно популярны и в наши дни под именем **MacBook Pro**, второй реинкарнировался в 2010 году как **iPad**.

Возвращение Джобса привело к созданию таких продуктов, как **iBook** (ныне **MacBook**), **iMac** возродив идею простого компьютера для всех, а также **iPod**, которому суждено было перевернуть все устои музыкальной индустрии. Но самым ярким событием стало появление в 2006 году **iPhone**, что сыграл такую огромную роль в индустрии мобильных телефонов. Этот смартфон послужил основой для появления многих других сенсорных устройств и привел новых производителей на рынок, где уже не осталось места таким ветеранам, как **Nokia**, **Motorola** и **Sony Ericsson**. Окончательно ясно кто теперь начал задавать тон на рынке смартфонов, стало после того, как финская компания, осознав, что она уже не в состоянии создавать конкурентоспособные устройства, начала множественные судебные разбирательства.

Кстати, поскольку компания уже не занималась исключительно компьютерами, в 2007 году её название было изменено с **Apple Computers** на **Apple Inc.**.

Одним из важных достижений компании после возвращения Джобса стал её переход на новейшую **MacOS X**, что пришла на смену неплохой, но окончательно устаревшей **MacOS Classic**. Тут Стиву удалось обойти Жана-Луи Гассе (Jean-Louis Gassée), бывшего сотрудника **Apple**, который в свое время ушел и основал компанию **Be Inc.**, что занималась операционной системой **BeOS**.

Феномен именуемый «*Cult of Mac*» — это настоящая загадка для многих людей, которые считают поклонников **Apple** просто сектой. **Apple** любят и ненавидят. Причем, зачастую бывает так, что ненависть эта выглядит куда более неприятной, нежели фанатизм любого из поклонников «яблок». Многих удивляет то, что устройства **Apple** стоят дороже при более слабых характеристиках, забывая при этом, что дело не только в гигагерцах. Не всем хочется сидеть за безликим устройством лишь потому, что его характеристики пестрят красивыми цифрами.

Однако можно смело сказать, что **Apple** была и остается одной из самых популярных компаний в мире.

Вероятно, мы просто подчиняемся магии стиля, фирменным знаком которого является минимализм. Если речь идет о материале, то он добротный и дорогой; а если о цвете, то это белый,

черный, красный или стальной. В минималистских вещах просто не бывает случайностей — это аксиома.

Глаз потребителя попадает в ловушку, приятно скользя по контуру такой простой, но совершенной формы. Мы бессознательно проглатываем логотип, ведь глаз всегда быстрее нашей мысли. Да, Apple — не для бедных, и для тех, кто ценит форму и содержание. «Возьми, если сможешь быть настолько безупречным, как и я». Без сомнения, это вызов человеку с деньгами и развитым эстетическим чутьем. Достаточно 1 раз попасть под обаяние «яблочного» стиля, — и Вы пропали надолго и навсегда.

Философия превосходства:

- Продукцию мегабренда отличают не только оригинальный дизайн и высочайшее качество сборки, но и беспрецедентная слаженность каждого устройства. Начиная от самой маленькой микросхемки и заканчивая самым глобальным программным обеспечением, все элементы мобильной и компьютерной техники целиком и полностью являются разработкой корпорации Apple.
- В компании очень много внимания уделяется объему рекламы и позиционированию на рынке среди конкурентов, несмотря на мировую известность.
- Простота в запоминании рекламных слоганов, наряду с до боли знакомым логотипом в форме надкусенного яблока сумели подействовать на всех нас лучше, чем длительное расхваливание продукции других конкурентов.

Эксклюзивная рыночная стратегия:

- Поскольку Mac есть профессиональным выбором ИТ-специалистов, то большинство программистов, менеджеров проектов и системных администраторов советуют переходить на Apple и его массовый продукт. В результате, темп роста продаж компьютеров Mac стал в 3 раза большим, чем PC.
- Оба продукта, iPod и iPhone, являются дополнением к компьютерам Mac. Следовательно, довольные своими iPod и iPhone люди в итоге покупают Macintosh. Как просто: «продавать небольшое замечательное решение – означает познакомиться с чем-то большим»!
- В какой-то мере, продукция Apple считается данью моде и частью современной массовой культуры.

Наконец, элементарному качеству продукта, маркетинговой стратегии и развитому сообществу вокруг этой компании просто-напросто нет равных, разве не так?

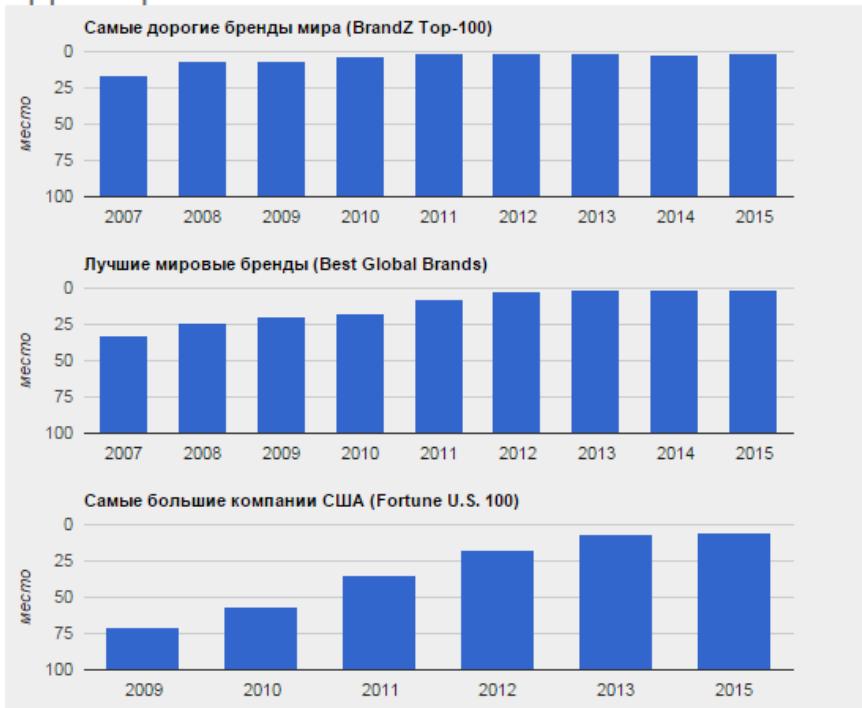
На самом деле, это лишь крошечный обзор интереснейших событий, что происходили во второй половине 20-го века, когда зарождалась индустрия, которой суждено было в корне изменить жизнь человека. Компьютер — одно из самых гениальных творений Цивилизации. А история вычислительной техники все еще только начинается. Сейчас трудно предположить, каким будет будущее Apple, но никто не станет сомневаться в том, что название этой компании и имена её основателей навсегда вписаны в историю компьютерной техники.

Стивен Пол Джобс (Steven Paul Jobs) скончался 5 октября 2011 года. Причиной его смерти стала раковая опухоль.

Многие говорят, что компания не достигла бы успеха без Стива Джобса. Но если посмотреть на компанию сейчас, то без Джобса они пришли к созданию Ipad mini, IOS 7 и IOS 8, Iphone, начиная с пятого и другим новым девайсам и технологиям. Возможно Джобс дал необходимый толчок компании по возвращению, но сейчас она не собирается терять лидирующие позиции.



## Apple в рейтингах



Соответственно на 2015 год: 1-ое место среди самых дорогих брендов мира, 1-ое место среди лучших мировых брендов, 5-ое место среди больших компаний США.

Samsung Electronics

Оборот на 2015 год: \$189.5 миллиардов

Слоган: Digitally yours.

Samsung Group — один из крупнейших конгломератов в мире бизнеса, на его родине, в Южной Корее для таких фирм используют слово «чобол» («chaebol»). Чобол — это крупная финансово-промышленная группа, преимущественно принадлежащая одной семье и связанная с правительственными кругами.

Ведущим подразделением корпорация Samsung по праву является Samsung Electronics, всемирно известный производитель LCD панелей, DVD плееров, мобильных телефонов, модулей памяти, используемых в компьютерах, телефонах, плеерах. Корпорации Samsung также принадлежат Samsung Life Insurance, Samsung SDS, Samsung Securities, Samsung C&T Corporation. До 2000-го года в состав Samsung также входило подразделение Samsung Motors, сейчас принадлежащее Renault.

Samsung Group была основана в городе Тэгу, в Корее, 1 марта 1938 года. Ее основатель предприниматель Бёнг Чхуль Ли (Byung-Chull Lee) (1910-1987), чей стартовый капитал составлял всего 30 000 вон (2000 долларов), назвал фирму «Samsung» (Samsung Trading Co), в переводе с корейского — «три звезды», на первых логотипах компании эти три звезды присутствуют в разных вариациях. Одна из наиболее правдоподобных версий о происхождении названия рассказывает, что у предпринимателя было три сына. (Судя по дальнейшему развитию никто из троих сыновей не оказался дураком, что, собственно, и отличает корейскую сказку от русской народной.) В пользу этой версии говорит также и то, что компания, в духе многих азиатских фирм, осталась семейным делом, передавая и умножая капитал среди круга родственников (и делая родственником того, кто сумел войти в бизнес, выделиться: внутриклановые браки — одна из традиций бизнеса в Азии). Предприниматель, по некоторым данным так и не получивший учченую степень, стал одним из известнейших и наиболее уважаемых людей в Корее, его именем назван корейский аналог Нобелевской премии — Но-Ам Prize, учрежденный компанией Samsung и присуждаемый за выдающиеся достижения в области науки и техники.

Компания пережила свое второе рождение в 1951-м году. После войны и грабительских действий со стороны враждующих сторон бизнес был полностью уничтожен, но уничтожить предпринимательский дух невозможно и, начав с нуля, Бёнг Чхуль Ли возродил фирму, достигнув еще большего благосостояния всего за год. Чем только не занимался предприниматель, в сферу его интересов входили: производство сахара, шерсти, других товаров широкого потребления, розничная торговля, страхование, радиовещание, издательский бизнес, торговля ценными бумагами. В 1960-е годы Samsung ждал невиданный успех. С целью возрождения экономики Кореи проводилась политика развития крупных национальных компаний, государство субсидировало, поддерживало и всячески помогало избранным корпорациям, фактически создав им тепличные условия, устранив конкуренцию и наделив широкими полномочиями. Создателю Samsung удалось сблизиться с правительственныеими кругами, что обеспечило корпорации неограниченные возможности для роста и расширения.

В 1970-х Samsung выходит на рынок полупроводников, увидев перспективы этого направления и быстрое развитие отрасли. Создается Samsung Electronics Co. Ltd, компания, включившая в себя несколько более мелких отделений Samsung Group, занимавшихся электроникой (Samsung Electron Devices, Samsung Electro-Mechanics, Samsung Corning, Samsung Semiconductor and Telecommunications).

В 1969-м году подразделение Samsung — Samsung-Sanyo выпускает первую партию черно-белых телевизоров. 5 лет спустя компания начинает заниматься выпуском холодильников и стиральных машин. Еще через 5 лет — выпуском микроволновых печей и кондиционеров. В 1978-м открывается представительство компании в США. Сумев стать первыми в Корее (Samsung занимает пятую часть всего экспорта Кореи), Samsung начинает путь по завоеванию мирового лидерства. В 1980-

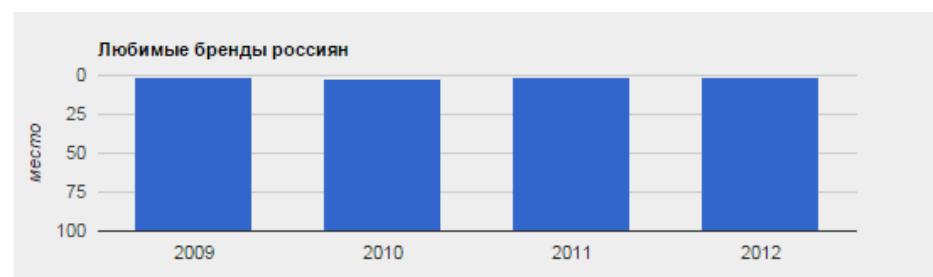
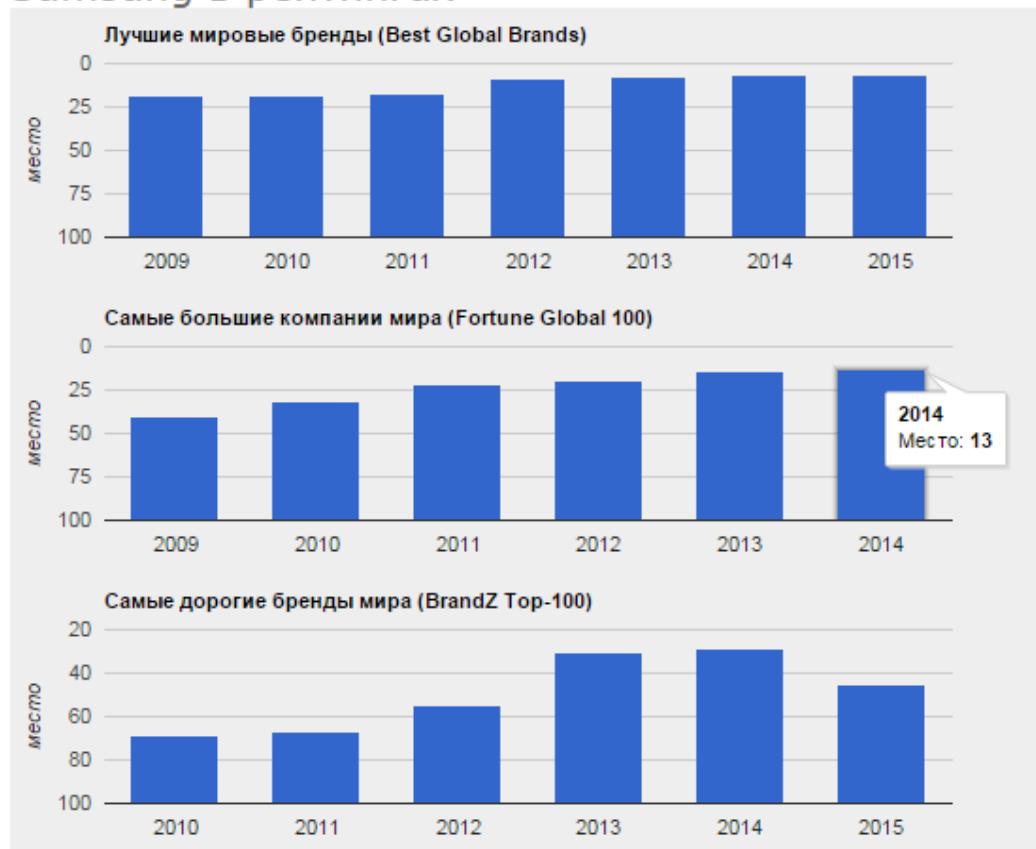
x Samsung выходит на рынок персональных компьютеров. В 1991-м году разрабатывается первый мобильный телефон Samsung, а в 1999-м — первый смартфон. В 1992-м году компания начинает разработку своей первой микросхемы памяти DRAM, тогда речь шла о емкости — 64 Мб, сейчас выпускаются чипы емкостью 64 Гб. В 1998-м начинается массовый выпуск цифровых телевизоров, разработанных в исследовательском центре компании. Год за годом Samsung расширяет свое присутствие на мировом рынке, удерживая первенство по продажам сотовых телефонов и телевизоров.

В 1993-м году, в год 55-летия компании, появился обновленный логотип Samsung — наклоненный эллипс синего цвета, с надписью внутри. Новый логотип удачно отображал выход фирмы на международную арену, своеобразную заявку на мировое лидерство. Визуально кажется, что слово Samsung находится внутри орбиты небесного тела, бесспорно корпорация является своего рода вселенной, но в то же время эта вселенная открыта миру, достаточно посмотреть на буквы «S» и «G» — они соприкасаются с внешним пространством. Одной из изюминок логотипа является написание букв «A» без черточки, много раз повторенный впоследствии, этот прием все же остался знаком Samsung.

Сегодня передовое подразделение Samsung Group — Samsung Electronics стал мировым лидером в области производства электроники и бытовой техники. Samsung Heavy Industries — подразделение занимающееся кораблестроением — второе в мире. Возглавляет корпорацию Ли Кун Хи (Lee Kun Hee), сын основателя. Именно ему Samsung обязан своими успехами в современном мире, приняв обязанности руководителя конгломерата после смерти отца в 1987-м Ли Кун Хи отказался от идеи массового производства товаров невысокого, так называемого бюджетного качества и сосредоточил усилия компании на производстве высококачественных продуктов, инновационных и опережающих рыночные тенденции. Бренд Samsung немало выиграл от такого решения, ведь те, кто считал продукцию фирмы недостаточно качественной, в последние годы открывают для себя мир бытовой техники и электроники исключительной по сочетанию «цена-качество», а если прибавить сюда и высокий уровень сервисного обслуживания компании, то альтернативы продуктам фирмы практически не остается.



## Samsung в рейтингах



Соответственно на 2015, 2014 и 2012 года: 45-ое место среди самых дорогих брендов мира, 7-ое место среди лучших мировых брендов, 13-ое место среди больших компаний мира, 1-ое место среди любимых брендов россиян.

HP

Оборот на 2015 год: \$233.715 миллиардов

Слоган: Invent

История этой компании берет свое начало в 1934 году. Именно тогда познакомились два выпускника Стэнфордского Университета — Билл Хьюлетт (Bill Hewlett) и Дэйв Паккард (David Packard). Оба с дипломами инженеров-электронщиков. Идея основать компанию возникла

спонтанно. И была довольно быстро реализована. Уже 1 января 1939 года в мире стало на одну компанию больше — появилась Hewlett-Packard или, в простонародье, HP.

Компания, которая ассоциируется у большинства современных потребителей лишь с компьютерной техникой и периферией, занималась за годы своего существования всевозможной электроникой, подчас к компьютерам никакого отношения не имеющей. Впрочем тут нет ничего удивительного — в те годы о компьютерах никто еще даже и не мечтал.

Название, возникло из простого объединения фамилий друзей. Никто никого не обидел: выбор, чей фамилии быть первой, был произведен посредством незатейливого голосования — друзья просто подбросили монетку. Так что, упали монетка другой стороной, знать бы нам компании Packard-Hewlett (или PH — согласитесь, звучит не очень).

Уже первая продукция компании (прибор для тестирования звуковой аппаратуры — аудиосциллятора HP Model 200A) оказалась весьма удачной. На тот момент в компании по-прежнему работало лишь два человека — ее основатели. Первым заказчиком была студия Disney, которая приобрела сразу 8 моделей. В пользу Model 200A говорили значительно более низкая цена, нежели у конкурентов, что никоим образом не сказывалось на качестве.

Как и многие известные ныне производители, друзья-инженеры начинали буквально «на коленке». Первые производственные мощности компании находились в гараже. Кстати, в 2004 году гараж этот был восстановлен, превратившись в своеобразный музей.

Со временем у компании появился простой, но запоминающийся логотип, незначительно менявшийся время от времени. Слоган, появившийся позднее, так же прост — лишь одно слово «Invent» (анг. — изобретать, придумывать, сочинять).

Начало 40-х годов пришлось на войну. Флоту, сильно страдавшему от немецких подводных лодок, требовалась специальная аппаратура так или иначе связанная с радарами. И тут HP оказалась на высоте. Компании не только удалось создать генераторы сигналов и подавители радарного излучения, но и закрепиться в этой отрасли, доказав свое превосходство. На тот момент количество сотрудников исчислялась уже десятками (если быть точным, то 45 человек), а доходы вплотную приблизились к миллиону.

К 60-м годам компания стремительно разрастается. Людей в ней работают уже тысячи, а доходы исчисляются уже сотнями миллионов. HP — это уже не только один из игроков на рынке электроники. Это уже солидная иуважаемая компания, разработки которой уважают и знают практически во всем мире.

1966 год ознаменовался приходом компании на рынок ЭВМ. Был создан HP 2116A. Этот компьютер до сих пор остается самым крупным механическим устройством, созданным компанией. Стоимость этого чуда составляло от 25.000 до 50.000 долларов США, в зависимости от комплектации. Изначально HP 2116A создавался для внутренних нужд компании, но позже был расширен для массового производства.

К концу же 60-х компания представляет новое вычислительное устройство — настольный калькулятор HP 9100A. Устройство обладало расширенным набором функций, позволяло использовать принтер и накопитель на магнитных карточках. По тем временам просто шедевр.

Стоил он кстати дороже, чем современный «навороченный» домашний компьютер — 4900 долларов США. Но прославилось устройство не только этим. В его рекламе впервые был применен термин «персональный компьютер».

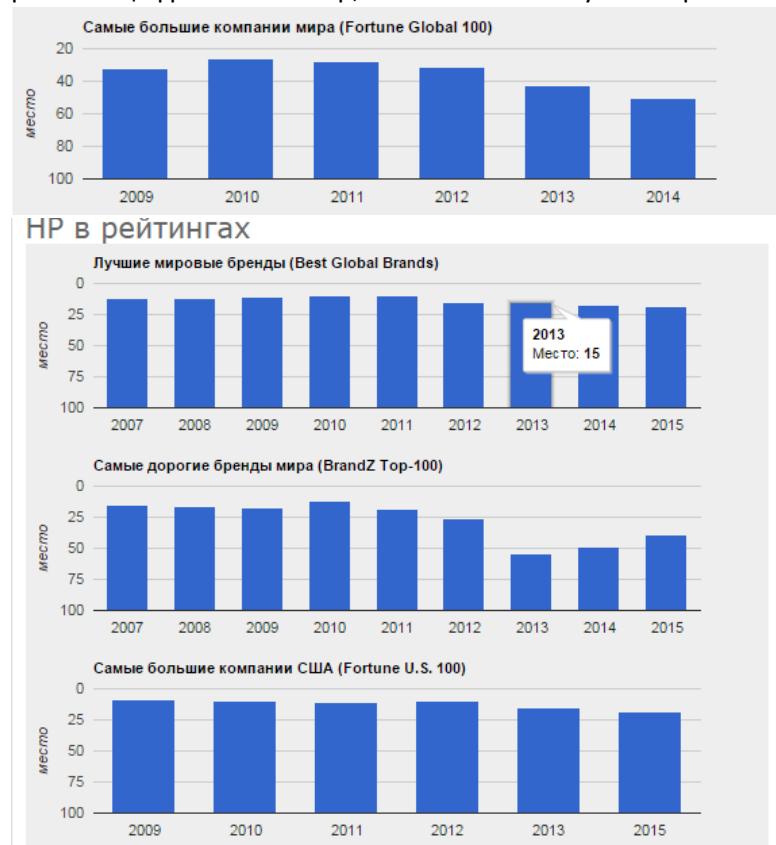
В 1972 году, компания представляет свой первый научный калькулятор, способный умещаться в карман — модель HP-35. Новинка стала легендарной. Более того, она входит в двадцатку продуктов, которые смогли изменить мир. По прошествии 35 лет, компания, отдавая дань прошлому, представила новую модель с таким же названием, но уже с использованием современных технологий. Так что, в какой-то мере, HP-35 жив до сих пор.

70-е годы вообще вошли в историю компании, как годы становления HP на компьютерном рынке. Компания не только представляет миниатюрные устройства, но и осуществляет попытки выхода на рынки настольных компьютеров, предназначенных, как для дома, так и для офиса.

В результате, в начале 80-х компания уже признанных игрок на рынке вычислительных систем. Кем и остается по сей день.

В 2001 году произошло одно важное событие в истории компании — слияние с другим крупным производителем компьютеров, компанией Compaq. HP получила некоторые технологии, которые позволили ей активнее развиваться в серверном направлении. Покупка Compaq обошлась компании в 25 миллионов долларов.

На сегодняшний день, HP это один из крупнейших производителей серверной техники, принтеров, сканеров, настольных компьютеров и ноутбуков. Хорошо зарекомендовала себя компания на рынке цифровых камер, КПК и коммуникаторов. Ну и конечно же калькуляторов.



Соответственно на 2015 и 2014 года: 39-ое место среди самых дорогих брендов мира, 18-ое место среди лучших мировых брендов, 19-ое место среди больших компаний США, 50-ое место среди больших компаний мира.

## IBM

Оборот на 2015 год: \$92.79 миллиардов

Слоган: Think!

Это не просто известная компания, это нечто большее. Начав задолго до компьютерной эры, она не только оказала влияние на современные компьютеры — IBM по сути создала их современный облик. Есть компании, история которых умещается в несколько абзацев. По истории IBM можно писать книги.

История началась в далеком 1911 году, 15 июня. Именно тогда стало известно об объединении сразу трех американских компаний: Computing Scale Company of America, International Time Recording Company и TMC (Tabulating Machine Company). Объединенное предприятие получило название CTR (Computing Tabulating Recording).

Чем они только не занимались первое время: под брендом CTR выпускалась самая разнообразная электротехника. Выпускались мясорезки и часы. Вообще, ассортимент компании был очень широк. Но особую статью занимали так называемые табуляционные машины (табуляторы), разработанные в 1889 году Германом Холлеритом (Herman Hollerith). Эти устройства использовались для переписи населения уже с 1890 года. Холерит даже основал компанию по их выпуску — уже названную выше ТМС. При желании, эти устройства можно назвать одними из первых компьютеров.

Постепенно, табуляторы становятся одним из главных направлений деятельности CTR. В 1924 году компания была переименована в привычную нам IBM — International Business Machines. Слово «International» в названии появилось благодаря выходу и на канадский рынок.

Без особого вреда для себя компания пережила Великую депрессию. А вступление США во Вторую мировую войну, принесло IBM многомиллионные заказы. Первое время компания производила оружие, а затем, в 1943 году, появляется и первый компьютер компании — знаменитый Marc I. Это был первый программируемый компьютер, произведенный в США. Этот электромеханический монстр, весил 20 тонн, занимал площадь в несколько десятков квадратных метров и производил адский шум во время работы. Но он полностью оправдал все возложенные на него надежды.

Так IBM начала заниматься компьютерами.

В 1952 году IBM выпускает свой первый компьютер на радиолампах — IBM 701. О мясорезках уже никто и не вспоминал. На компьютерах IBM впервые был применен язык программирования высокого уровня — FORTRAN, применяемый и поныне в научных исследованиях. В 1959 году, представлены первые компьютеры компании, построенные на полупроводниках. IBM первая начинает массовый выпуск универсальных компьютеров. Это были машины семейства System/360. В мир компьютеров приходит стандартизация. Отныне не требовалось переписывать программное обеспечение под каждую отдельно взятую модель.

Вклад IBM в развитие мира компьютеров огромен. Это и накопители на жестких магнитных дисках (проще говоря жесткие диски), впервые представленные 13 сентября 1956 года. И накопители на

гибких магнитных дисках (дискеты), разработанные в 1971 году. И язык структурированных запросов SQL, применяемый во всех реляционных базах данных. А самое главное — персональный компьютер IBM PC, представленный в 1981 году. Этот компьютер положил начало новой эпохи — эпохи архитектуры РС. Устройство, над которым в свое время посмеялись инженеры из Apple, полностью изменило представление о том, каким должен быть компьютер. Впрочем, саму IBM производство дешевых компьютеров мало интересовало. Она активно продавала лицензии и уже в 1986 году перестала лидировать на рынке персональных компьютеров. В дальнейшем (2004 год) IBM вообще ушла с этого рынка, как десктопов, так и ноутбуков, продав свой бизнес китайской компании Lenovo. Вообще, IBM с легкостью расставалась с производствами, приносившими мало прибыли. Так, жесткие диски компании стали достоянием истории, а активы приобрела Hitachi. Отказалась IBM и от бизнеса по производству принтеров — ныне этим занимается Lexmark.

На сегодняшний день, компания занимается производством суперкомпьютеров (ее модели занимают первые места в рейтинге Top 500 Supercomputers), высокопроизводительными серверами, процессорами и программным обеспечением. IBM входит в десятку крупнейших производителей различного электротехнического и электронного оборудования по всему миру. Очень важную часть работы компании составляют научные разработки и консалтинг.

Первое время IBM использовала громоздкий и невнятный круглый логотип. Затем, в 1947 году от него отказались. На смену пришел новый, практически не изменившийся и в наши дни. Иногда корпорацию IBM называют Big Blue. Происхождение этого прозвища уходит корнями в 20-е столетие и не совсем понятно. Версий три. Первая, будучи самой правдоподобной, предполагает, что причина этого кроется в том, что первые компьютеры компании имели голубую окраску. Вторая версия утверждает, что причина кроется в древней рекламе компьютеров, которые с легкостью побеждали гиганта, одетого во все голубое. Ну и последняя, самая простая, версия гласит, что причина в голубых буквах на логотипе.

Одним из самых известных слоганов компании является простое «Think!» (Более полный вариант: «I think, therefore IBM. Think»). Он был разработан еще 1930 году генеральным директором Томасом Уотсоном (Thomas Watson). Подразделение IBM Business consulting использует в рекламе «Need some new thinking?». Известен также и слоган «Business on Demand».



International Time Recording Company (1888)



Computing Scale Company of America (1891)



Tabulating Machine Company (1911)



1924

IBM

1947

IBM

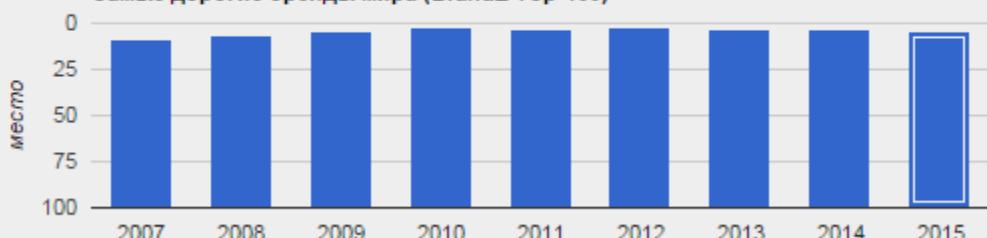
1956

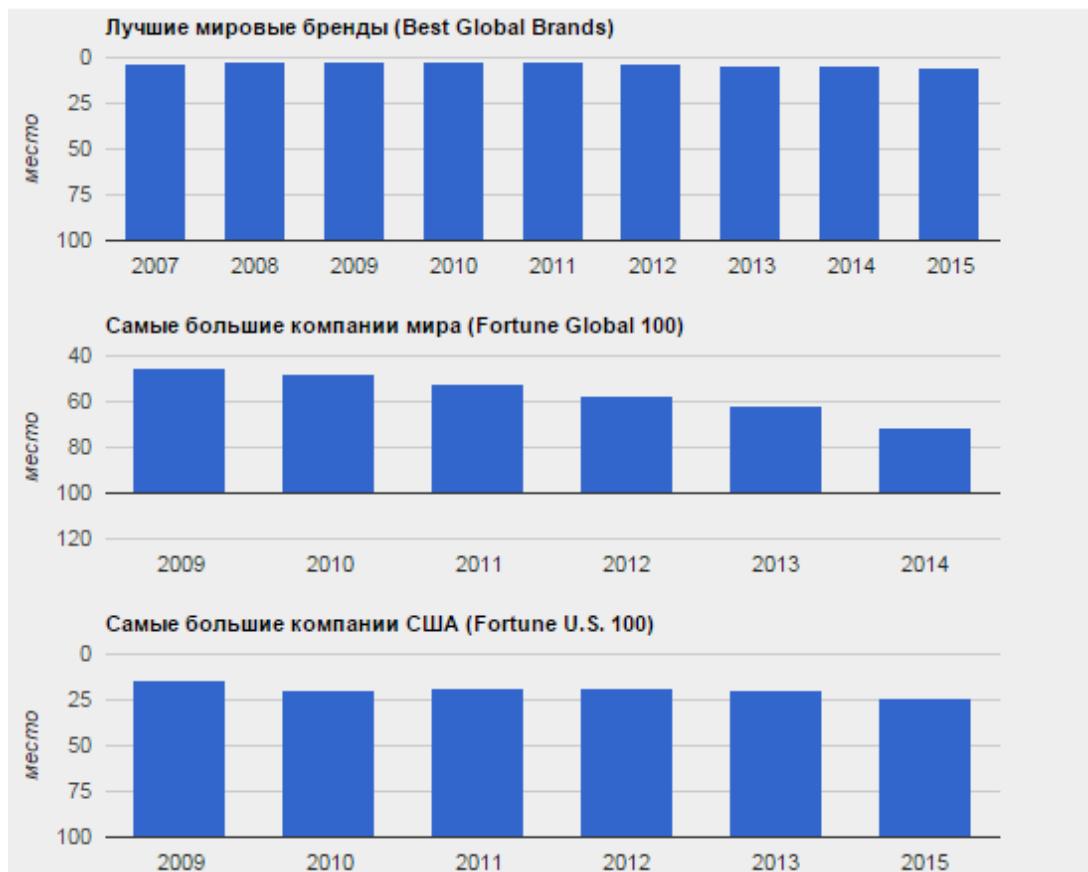
IBM

1972

## IBM в рейтингах

Самые дорогие бренды мира (BrandZ Top-100)





Соответственно на 2015 и 2014 года: 4-ое место среди самых дорогих брендов мира, 5-ое место среди лучших мировых брендов, 24-ое место среди больших компаний США, 71-ое место среди больших компаний мира.

### [Amazon](#)

Оборот на 2015 год: \$88.99 миллиардов.

История этого крупнейшего интернет-магазина началась 16 июля 1995 года. Создал его бизнесмен Джек Бэзос (Jeffrey Preston «Jeff» Bezos). Изначально продавались исключительно книги, позже, к 1998 году ассортимент был расширен — сначала в него добавили аудио-, а после и видеопродукцию. На сегодняшний день [Amazon.com](#) предлагает своим посетителям уже несколько десятков различных категорий товаров, включая также электронику, одежду, спортивные товары и даже продукты питания. Книги, одежда и диски доставляются по всему миру, в то время, как все остальное — только по территории США (или той страны, где находится представительство компании — Германия, Франция, Япония и других). Часто компания заключает договора на получение на некоторое время эксклюзивного права продаж того или иного товара.

Название **Amazon** было выбрано по двум причинам. Во-первых, основатель желал, чтобы его компания находилась как можно ближе к началу телефонного справочника или любого другого списка по алфавиту. А во-вторых, упоминая Амазонку — самую крупную по полноводности и длине реку — Джек как бы намекал на то, что его компания также должна стать самой крупной в своей отрасли. И ведь получилось же.

На самом деле изначально компания называлась **Cadabra, Inc.**. Вот только название это было крайне неудачным, поскольку при телефонном разговоре клиентам часто слышалось нечто иное, а именно «*cadaver*» («кадавр, труп»). А ведь именно телефонные звонки приносили основную часть заказов первое время.

Уже в течении первых пары месяцев было принято большое количество заказов из практических всех штатов США. Через полгода каждый день продавалось уже по несколько десятков тысяч книг по всему миру, так что молодая компания едва успевала обрабатывать все поступавшие заказы. При этом, затраты на рекламу были минимальны. А в 1998 году открылось и первое зарубежное представительство в Германии. **Amazon.com** сравнительно легко пережил «крах доткомов», когда в начале 2000 года обанкротились сотни компаний, бизнес которых строился исключительно на интернете.

Рассказ про **Amazon** будет не полным, если не упомянуть **Kindle**. Это сервис по распространению электронных книг, поддерживающийся также одноименным устройством для чтения. «Букридер» **Kindle** интересен тем, что при достаточно низкой стоимости отличается массой возможностей. Компания не получает доход от его продаж — для этого существуют книги, а также периодика, что распространяется через сервис. И нужно отметить, что вот как раз стоимость контента весьма высока. Май 2011 года ознаменовался тем, что число продаваемых электронных книг превысило продажи книг бумажных, причем и в мягких, и в твердых переплетах.

Естественно, что такой гигант, как **Amazon.com** не мог не привлечь к себе и негативное внимание. На компанию несколько раз подавали в суд. Однажды её обвинили в том, что она вводит своих клиентов в заблуждение, называя себя «крупнейшим в мире интернет-магазином», другой раз последовало обвинение от **Walmart**, считавшего, что **Amazon** переманивает у него сотрудников. В обоих случаях для героя нашего повествования все закончилось благополучно.

За годы своего существования **Amazon.com** поглотила немало других компаний и интернет ресурсов. К примеру, приобретенный в 1998 году ресурс **Bookpages.co.uk** занимавшийся онлайн-торговлей книг лег в основу **Amazon UK** — представительства в Великобритании.

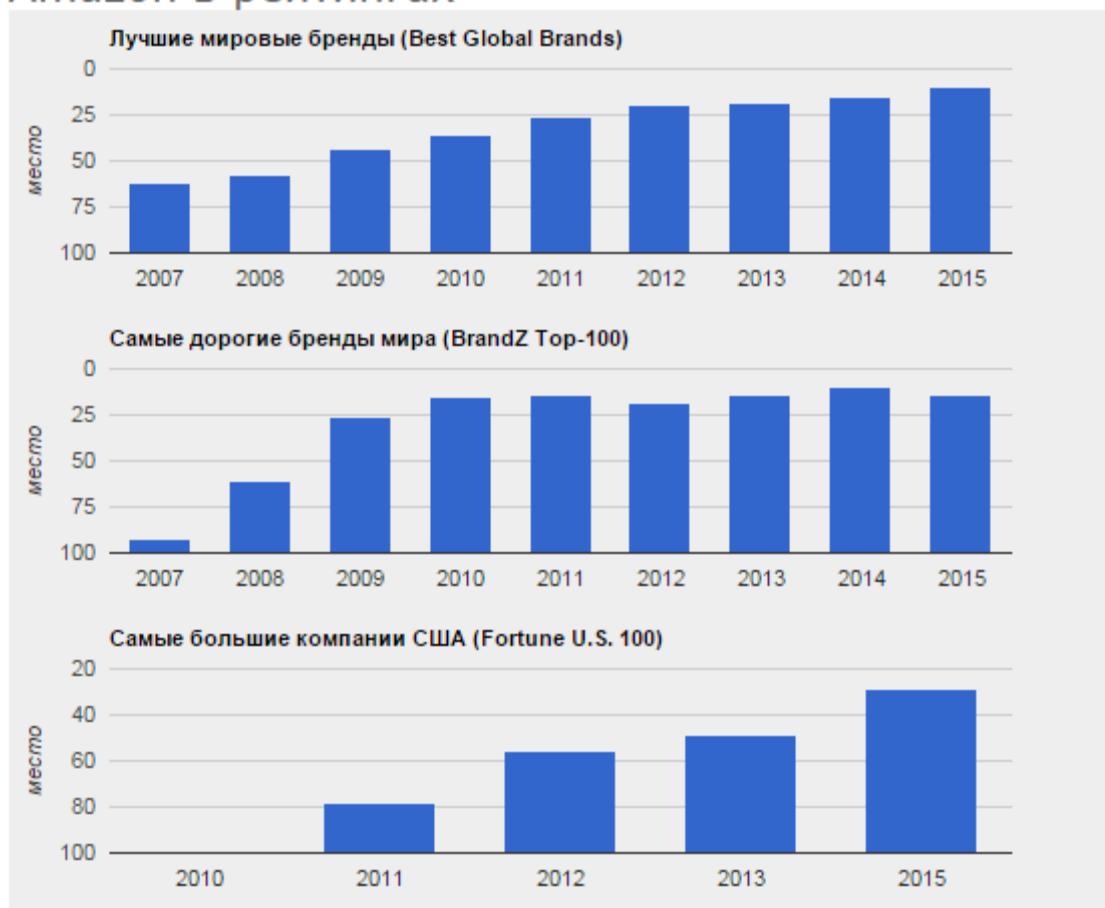
Однако **Amazon** — это не только интернет-магазин. **Amazon Web Services (AWS)** — инфраструктура платформ облачных веб-сервисов, представленная компанией **Amazon** в начале 2006 года. В данной инфраструктуре представлено много сервисов для предоставления различных услуг.

На сайте предоставлен широкий спектр фундаментальных сервисов облачной инфраструктуры: вычисления (виртуальные серверы, контейнеры, развертывание веб-приложений методом 1-click, управляемые событиями вычислительные функции и проч.), хранение данных и доставка контента (объектное хранилище, хранилище файловых систем, архивное хранилище, блочное хранилище и проч.), база данных (реляционные БД, миграция баз данных, NoSQL, кэширование, хранилище данных), сетевые решения (изолированные облачные ресурсы, прямые подключения, балансировка нагрузки, DNS).

Штаб-квартира **Amazon** находится в Сиэтле, штат Вашингтон (Seattle, Washington, U.S.A). Число работников по всему миру уже перевалило за 20 тысяч человек.



## Amazon в рейтингах



Соответственно на 2015 год: 10-ое место среди лучших мировых брендов, 29-ое место среди больших компаний США, 14-ое место среди дорогих брендов мира.

### Microsoft

Оборот на 2015 год: \$88.99 миллиардов.

Слоган: Where do you want to go today? (1994 — 2002 годы). Your potential. Our passion. (нынешний слоган)

Всемирно известный производитель программного обеспечения (ПО) для настольных компьютеров, мобильных устройств, кластеров, серверов и игровых консолей. Помимо

программного обеспечения, компания занимается также производством самих игровых консолей, компьютерных манипуляторов и аудиоплееров.

История могучей ныне корпорации началась 14 апреля 1975 года. Ее основателями были такие известные теперь личности (тогда они были никому неизвестными студентами), как Билл Гейтс (Bill Gates) и Пол Аллен (Paul Allen). Согласно официальной легенде, они решили разработали интерпретатор языка программирования BASIC для персонального компьютера Altair 8800, производства компании **Micro Instrumentation and Telemetry Systems (MITS)**. Студентам даже удалось заключить соглашение с производителем Altair, о поставках интерпретатора вместе с устройством. Идея продавать программное обеспечение вызвала резкую критику и непонимание в определенных кругах — программ тогда писалось мало, и они свободно распространялись между программистами. Тогда же и был отмечен первый случай пиратства. Однако время подтвердило правоту Била Гейтса — уже к концу 1975 года, доходы компании исчислялись тысячами, что очень даже неплохо для новичков.

Но настоящий успех пришел к Гейтсу и его компании вместе с представителями фирмы **IBM**, которые искали того, кто создаст операционную систему для компьютера IBM PC. Произошло это 12 августа 1981 года. Появление MS DOS (Disk Operation System) и определило окончательную судьбу **Microsoft**. Позже, 20 ноября 1985 года, появляется первая графическая оболочка к DOS, получившая название Windows. Начиналась новая эра.

Название компании образовалось из двух слов: «**MICRO**computer **SOFT**ware». Первое время оно писалось через дефис: **Micro-soft**. Компания несколько раз видоизменяла логотип, не меняя при этом его сути; менялся лишь шрифт, которым было написано слово «Microsoft». На сегодняшний день такое название уже не соответствует истине — ПО создается не только для микрокомпьютеров, но и высокопроизводительных серверов и тому подобного.

История **Microsoft** — это история головокружительного успеха и постоянного развития. Корпорация неоднократно допускала ошибки и просчеты, но всегда быстро исправляла их, завоевывая, в итоге, все новые и новые для себя области и устанавливая новые рекорды. И если первый год в **Microsoft** работало лишь трое человек (включая основателей), то к 2008 году их число составило более 80 тысяч, в офисах разбросанных по всему миру.

Список программного обеспечения, выпускаемого **Microsoft** огромен. Это операционные системы (MS-DOS, Windows), офисные пакеты (MS Office), Серверные приложения (SQL Server...), средства разработки (Visual Studio, Visual FoxPro...), игры (Age of Empires, Microsoft Flight Simulator...) и многое другое. Заслуженным уважением пользуются клавиатуры и мыши, выпускаемые под брендом Microsoft. Исключительно популярна игровая консоль **Xbox**. Похвального слова заслуживает и превосходный сервис поддержки, предоставляемый компанией. Нельзя не упомянуть и поисковую систему **Bing**, созданную для конкуренции с **Google**.

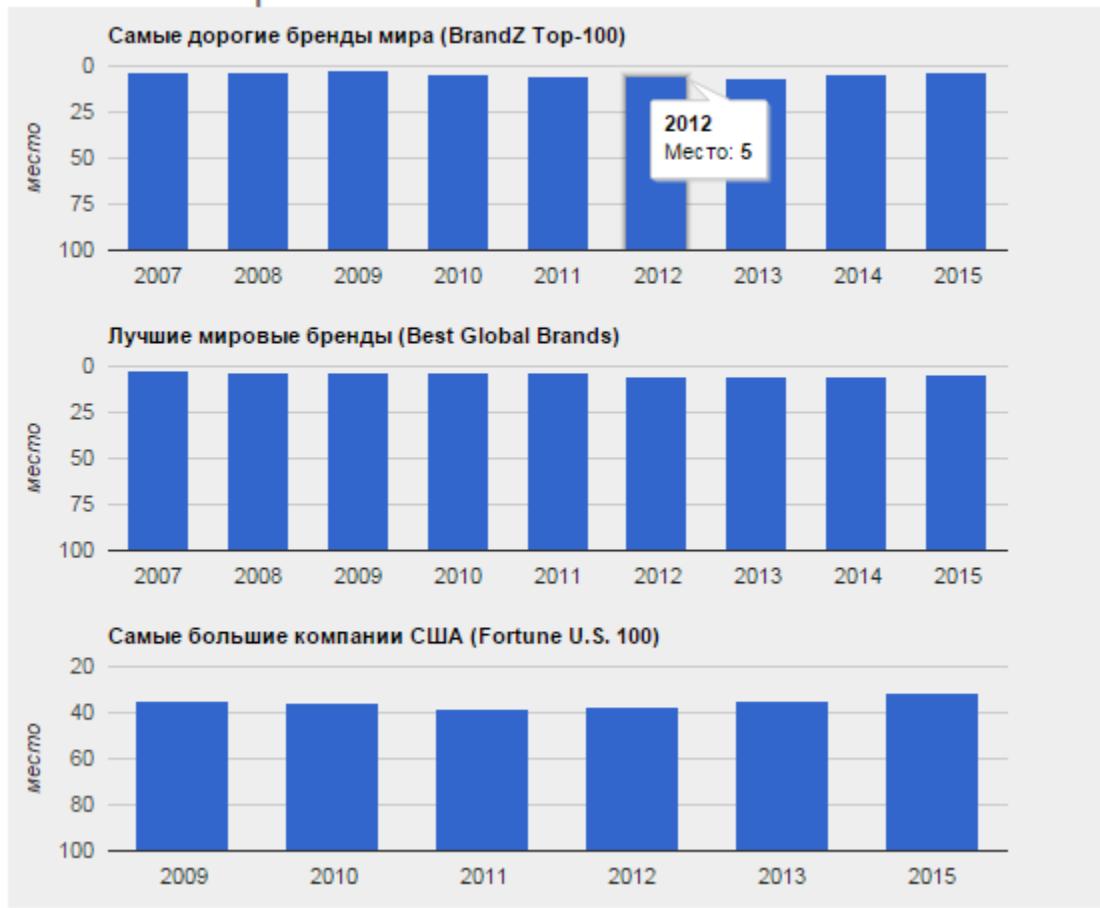
Но не следует думать, что все так гладко. Прославилась **Microsoft**, прежде всего, как монополист, навязывающий свои решения. Политика корпорации исключительно агрессивна. Многие из продуктов компании стали столь популярными вовсе не потому что они самые лучшие, а благодаря их агрессивному продвижению на рынок, чего не могли себе позволить конкуренты. Критикуют компанию и за «закрытость» выпускаемого ей программного обеспечения.

Монополия **Microsoft** на рынке программного обеспечения не дает покоя многим. Корпорацию неоднократно привлекали к ответственности. Некоторые из судебных разбирательств **Microsoft** проигрывала, но особого вреда ей это не принесло. Главная причина такого положения дел — отсутствие серьезных конкурентов. Операционные системы и офисные пакеты, способные бросить вызов продуктам компании, появились лишь недавно, и пока не сумели нанести ей хоть сколь-нибудь серьезный урон.

Штаб-квартира **Microsoft** находится в Редмонде, штат Вашингтон, США (Redmond, Washington, USA).



## Microsoft в рейтингах



Соответственно на 2015 год: 4-ое место среди лучших мировых брендов, 31-ое место среди больших компаний США, 3-е место среди дорогих брендов мира.

[Sony](#)

Оборот на 2015 год: \$72.34 миллиардов.

Слоган: make.belive.

У истоков многих всемирно известных компаний стояли два человека, один из которых был талантливым инженером, другой хорошо ориентировался в мире бизнеса. Не стала исключением и **Sony**.

Случилось это в 1946 году, в Японии, которая только-только начала возрождаться после поражения и потрясений Второй мировой войны. В частично разрушенном торговом центре Nihonbashi, чудом устоявшем после бомбардировок Токио, молодой инженер Масару Ибука (Masaru Ibuka) открыл мастерскую по ремонту различного электрооборудования и электроники. Через некоторое время он со своим старым другом Акио Морита (Akio Morita) основал в том же помещении офис для новой компании, которая получила громкое название **Tokyo Telecommunications Research Institute**, которое порой сокращали до **Totsuko**. Годом позже они переедут уже в некое подобие того, что уже можно назвать головным офисом. Первой их разработкой стала приставка для радиоприемников,

которая расширяла возможности устройства, позволяя ему принимать зарубежные программы. Эта продукция пользовалась не слишком большим спросом, но позволила удержаться на плаву, сколотив некое подобие начального капитала. Причем подчас приходилось брать плату не деньгами, а различными продуктами, что было обычным явлением для обнищавшей страны. В дальнейшем появляется и более прибыльная продукция.

Но настоящий успех пришел в сентябре 1949 года, когда был создан первый в Японии магнитофон. Довольно уродливая массивная коробка, использовавшая бабины диаметром 25 см, получила название Type G.



Магнитофон Type G (изображение с официального сайта)

Друзья всегда осознавали всю важность создания не только качественной продукции, но и красивого бренда, что было просто необходимо для выхода на мировой рынок. Так в 1950 году родился бренд **Sony** – производное от латинского «sonus» («звук»). Слово получилось простым, легко запоминающимся и уникальным. В 1955 был официально утвержден новый логотип и представлена первая продукция под новым брендом — транзисторный радиоприемник TR-55. Успех этого приемника и определил успех бренда. Следующей моделью стал первый миниатюрный приемник TR-63, цена которого была обратно пропорциональна размерам. Коммерческого успеха он не имел. К тому времени комплектующие производимые **Totsuko** начинают закупать другие японские производители.

В 1958 году компания официально меняет имя на **Sony Corporation**, используемое и поныне.

В дальнейшем основное внимание уделялось двум вещам — инновационным разработкам и красивым брендам. Компании принадлежит великое множество торговых марок. Среди них есть и всемирно известные (**Trinitron, Vaio, PlayStation, Walkman, Bravia, Cyber-shot, Clie**), так и таких, которые известны только специалистам.

Вторая половина XX-века ознаменовалась рассветом **Sony**. Этакий «золотой период». Компания успешно осваивает новые сегменты рынка. А иные сама и создает. Появляются многие уникальные устройства и разработки, аналоги которых конкуренты смогут создать еще не скоро.

В своей книге «Just for fun», создатель операционной системы **Linux**, Линус Торвальдс (Linus Torvalds) прочил **Sony** великое будущее. По его мнению, корпорация должна была стать для мира электроники приблизительно тем же, чем является **Microsoft** для мира программного обеспечения. Оно и не удивительно — в те годы, когда писалась книга (90-е прошлого века), **Sony** действительно развивалась стремительными темпами. Только за один 1990 год было представлено более 500 инновационных разработок! Бренд **Sony** стал мегабрендом — многие потребители часто приобретали электронику ориентируясь только на него, даже не обращая внимания на продукцию конкурентов. Но...

На сегодняшний же день дела у **Sony** обстоят уже не так замечательно как раньше. Сказалась слишком сложная структура, не позволяющая адекватно и быстро реагировать на новые веяния рынка, да и уверенность в собственной непоколебимости. Негативную роль сыграла и политика навязывания собственных стандартов. Компания, всегда считавшаяся одной из самых инновационных, внезапно перестала успевать реагировать на технические веяния рынка. В результате, ведущие позиции по многим направлениям были потеряны — портативные плееры (ныне правит бал **Apple**), телевизоры (**Samsung**), игровые приставки (**Nintendo**). Неудачным оказался альянс со шведским **Ericsson**, — бренд **Sony-Ericsson** не смог оказать должного влияния на рынок (**Nokia**, **Samsung**, **LG**, **HTC**, **Apple**). Главным же конкурентом неожиданно оказался южнокорейский конгломерат **Samsung**, обошедший японцев по многим направлениям.

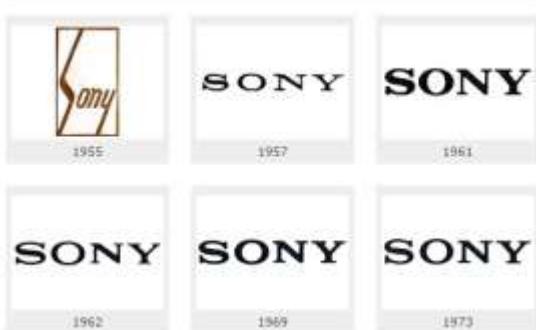
Самое главное, что не учли в **Sony** — это факт того, что современных пользователей интересует уже не «громкий» бренд, но высокая функциональность, пусть даже в некоторый ущерб качеству. Людей, согласных выплачивать большие суммы только за красивый лейбл стало меньше. **SonyStyle** потерял свою былую привлекательность, хотя и не потускнел окончательно. Да и в профессиональной технике **Sony** играет заметную роль. Но предсказанию Торвальдса сбыться было не суждено.

Штаб-квартира компании находится в Токио, Япония (Tokio, Japan). **Sony Group** — сложнейшая структура со множеством подразделений и дочерних компаний. Контролирующей компанией является **Sony Corporation**. Основная сфера производства — электроника, но заметную роль играет компания и в масс-медиа, занимаясь телерадиовещанием и производством кинофильмов.

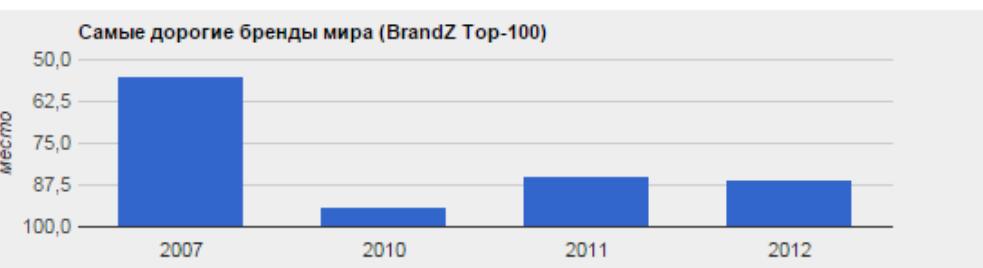
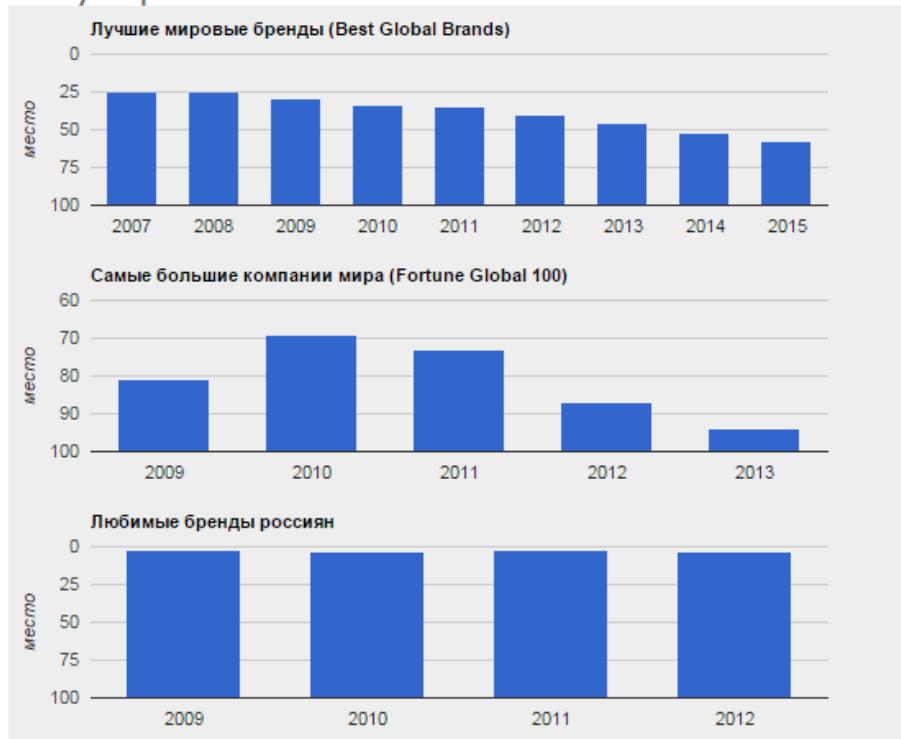
#### Интересные факты:

В 1946 году основной доход молодой компании приносила подушка с электроподогревом, которая продавалась под брендом **Ginza Heating Company**. Причина появления этого бренда анекдотична — будучи совершенно неуверенными в качестве этой продукции, друзья решили использовать другое название, дабы в случае провала не навлечь проблемы на основное, разрушив репутацию компании, которая только-только начала вставать на ноги. К чести их следует отметить, что подушки эти оказались очень даже ничего.

В настоящее время, как сообщается на официальном сайте Sony, убыточное ПК-подразделение Vaio выделено в новую компанию, принадлежащую фонду Japan Industrial Partners (совместное предприятие Bain & Co. and Mizuho Securities Co). Sony получает в этой фирме 5-процентную долю, а также перевела в нее своих сотрудников.



## Sony в рейтингах



Соответственно на 2015 год и 2012 года: 58-ое место среди лучших мировых брендов, 94-ое место среди больших компаний мира, 3-е место среди любимых брендов россиян, 86-ое среди дорогих брендов мира.

Google

Оборот на 2015 год: \$66.00 миллиардов.

Компания, входящая в состав холдинга Alphabet Inc., владеющая популярнейшим поисковиком. **Google** обрабатывает десятки миллиардов запросов в месяц, владея абсолютным большинством поискового рынка. Ближайшие конкуренты (**Yahoo!** и **Bing**) отстают от него на порядок.

Именно **Google** первым приходит на ум, когда разговор заходит про знаменитые бренды. Совсем молодая компания, сумевшая за очень короткий период взлететь столь высоко. А ведь все базировалось на одной простой, но весьма интересной идее.

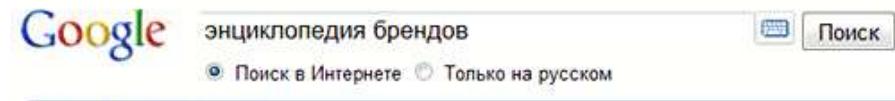
Аспиранты Стэнфордского университета Лари Пейдж (Larry Page) и Сергей Брин (Sergey Brin) справедливо рассудили, что чем интереснее интернет-ресурс, тем больше сторонних ссылок будет на него вести. Именно таким сайтам и было решено уделять больше внимания при поиске. Создание **Google** началось не вдруг — такой проект требовал немалых вложений. Однако найти инвесторов оказалось непростым делом, поскольку многие ожидали увидеть нечто особенное, а им показывали «еще один поисковик». Но Лари и Сергею хватило упорства и харизмы, они верили в будущее придуманной ими технологии контекстной рекламы. Деньги были найдены.

В 1998 году, наскребя около миллиона долларов США, друзья зарегистрировали новую компанию, назвав ее в честь числа, которое выглядит как единица и сто нулей — гугол. Кстати, в обозримой Вселенной нет ничего, что могло бы ей измеряться. Так что в какой-то степени **Google** замахивается на бесконечность. Изначально поисковая машина тестировалась, будучи расположена на домене **google.Stanford.edu**. Позже, будучи в стадии бета-тестирования, она обрабатывала порядка 10 тысяч запросов в день. В общих чертах повторялась история **Yahoo!**.

Взлет начался в 1999 году. Количество запросов исчисляется уже сотнями тысяч в день. Начинают заключаться контракты с гигантами интернет-бизнеса. Первым был канувший ныне в Лету **Netscape**, а затем **Yahoo!**. Все это значительными темпами увеличивало популярность нового поисковика. Уже к 2001 году **Google** стал приносить прибыль.

История **Google** интересна не сама по себе, а своим существованием. Она — новое воплощение «американской мечты», прорыва «в духе времени». Пример того, что еще не все возможности использованы, всегда есть место для шага вперед, открывающего новые горизонты.

Веб [Картинки](#) [Видео](#) [Карты](#) [Новости](#) [Переводчик](#) [Gmail](#) [ещё ▾](#)



Веб [Показать настройки...](#)

[BrandReport - энциклопедия брендов, знаменитые бренды](#)

Энциклопедия брендов, новости компаний, история известных брендов, знаменитые бренды, история появления брендов.  
[www.brandreport.ru/](http://www.brandreport.ru/) - Сохраненная копия

Представить себе современный интернет без **Google** просто не получается. Недаром столь популярным стал жаргонный термин «гуглить». Более того, подчас невольно задаешься вопросом «а чем мы искали и как вообще находили что либо раньше?». И не можешь вспомнить ответ. Хотя **Google** и не идеален, подчас приходится прибегать и к услугам других поисковиков. Впрочем,

это справедливо для совсем уж экзотических запросов. К примеру, в азиатском регионе позиции **Google** слабы и лучше прибегать к услугам **Baidu** (если не знаете китайского, то тут на помощь придет **Google Translator**).

**Google** никогда не торопится. Компания внедряет множество сервисов, но развивает их при этом очень медленно. Но есть в них нечто, что вызывает привыкание. Попользовавшись некоторое время, начинаешь понимать, что уже не можешь обходится без **Gmail**, а **Google Docs** так удобны, что постепенно начинаешь отказываться от установленных на компьютере текстовых редакторов. А чего стоит только одна лишь синхронизация закладок в **Google Chrome**! И так во всем — понемногу компания захватывает все большие и большие территории, становясь для них чуть ли не стандартом.

Появившийся в конце 2008 года **Google Android**, хотя и вызывал изначально лишь скептические ухмылки, ныне метит в конкуренты таким монстрам, как **iPhone OS**, **BlackBerry OS** и **Symbian**. Более того, по количеству производителей, выпускающих устройства на ее базе, ОС **Android** уже лидирует.

Прекрасно понимая, что на одной только рекламе им не существовать вечно, руководство **Google** постоянно ищет новые интересные идеи. Памятую о собственных начинаниях, Сергей и Лари охотно меценатствуют, вкладывая деньги во многие любопытные проекты. Так, благодаря их участию появилась компания по производству спортивных электромобилей **Tesla Motors**. Сотрудничество с **HTC** принесло плоды в виде **Nexus One** — первого смартфона под брендом **Google**, работающего, естественно, под **Android**. Нельзя не упомянуть про постоянные научные и медицинские исследования, а также космические программы, субсидируемые щедрой рукой **Google**. И так далее и тому подобное.

Говоря про **Google** нельзя не упомянуть и еще одну очень важную деталь. Этот бренд вызывает у большинства людей позитивные ассоциации. К примеру, от **IBM** веет чем то анахроническим, присыпаным пылью (хотя на самом деле это одна из самых инновационных компаний); **Microsoft** многие основательно не любят, хотя и сами не могут толком объяснить почему. А вот **Google** воспринимают благосклонно. Повсеместно люди с восторгом, будто очевидцы, рассказывают не раз слышанные истории о том, какие замечательные условия в офисе компании, про великолепные обеды для сотрудников (обхажанные известным дизайнером Артемием Лебедевым), про ненормированные рабочие дни... Судебные же нападки со стороны различных личностей, слишком ревностно относящихся к авторским правам, единогласно осуждаются большинством. Все это помогает в деле укрепления светлого образа среди общественных масс. А это очень и очень важно. Компания может смотреть в будущее с оптимизмом.

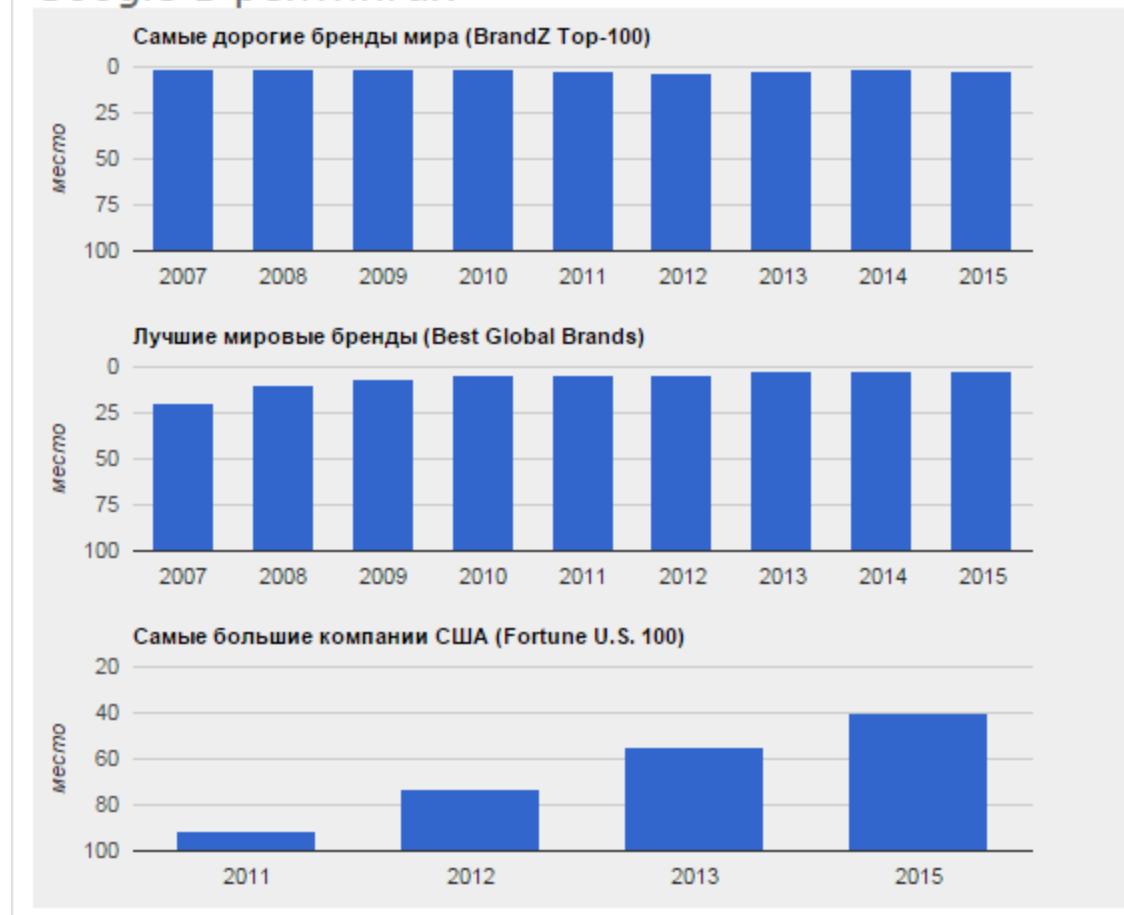
Август 2011 ознаменовался довольно неожиданным приобретением — была поглощена компания **Motorola Mobility** (бывшее подразделение **Motorola**). Благодаря этому **Google** получил производственные мощности для экспериментов со смартфонами. Впрочем, в итоге бренд **Motorola** достался китайскому производителю электроники **Lenovo**.

Частью холдинга **Alphabet Inc.** компания **Google** стала в августе 2015 года. Сделано это было для упрощения управления структурой, которая к тому времени очень сильно разрослась.

Штаб-квартира **Google Inc.** находится в Маунтин-Вью, Калифорния, США (Mountain View, CA, USA). Представительства находятся в различных странах по всему миру.



## Google в рейтингах



Соответственно на 2015 год: 2-ое место среди лучших мировых брендов, 40-ое место среди больших компаний США, 2-ое среди дорогих брендов мира.

Panasonic

Оборот на 2015 год: \$64 миллиардов.

Сегодня под брендом Panasonic выпускаются самые разные товары электроники и бытовой техники – от тостеров до фотоаппаратов (под брендом Lumix). Компания уже давно вошла в десятку лидеров мирового производства электротехники и сдавать свои позиции ни в коем случае не намерена. Однако такой успех был далеко не всегда.

Если спустимся к истокам, то обнаружим, что основателем и идейным вдохновителем компании стал японский бизнесмен Коносукэ Мацуцита. В 1917 году 24-летний Коносукэ, проработав более 7 лет на компанию Osaka Light, принимает решение начать свое собственное предпринимательство. Для него, выходца из разорившейся семьи, это был единственный выход покончить, в конце концов, с бедностью.

В небольшой арендаемой квартире Коносукэ вместе с женой начинают производство штепсельных вилок. Они трудятся по 18 часов в день, самостоятельно реализуя создаваемый ими товар – Коносукэ лично ходил от двери к двери, предлагая свою продукцию. А потому не удивительно, что молодые люди с готовностью берутся за совершенно любые заказы. Один из таких заказов и спас однажды их положение – перед Коносукэ была поставлена задача изготовить 1000 деталей для электровентиляторов. Качественная и своевременно выполненная работа принесла предпринимателю первую серьезную прибыль, которую он тут же направил на нужное дело. Так и была основана компания Matsushita Electric, которая впоследствии и превратится в хорошо известный нам Panasonic. 1 октября 2008 года компания была официально переименована в Ltd. Panasonic, взяв название одной из своих самых известных торговых марок.



Электрический утюг, 1927 год

Особенностью Коносукэ Мацуциты являлось то, что трудолюбивый японец никогда не останавливался на достигнутом, всегда продолжая искать дополнительные возможности и инновационные пути решения тех или других проблем. Так, прочитав однажды в популярном журнале про электрические фары для велосипедов, предприниматель загорелся идеей их выпускать. И хоть рынок поначалу и отнесся скептически к этой идее, Коносукэ все-равно запустил в продажу первую партию товара. Реакция не заставила себя долго ждать – велосипедный фонарь

приобрел настолько большую популярность, что люди приобретали его даже для освещения помещений. Коносукэ Мацуцита тут же среагировал на это — и вскоре была выпущена первая в Японии домашняя электрическая настольная лампа, которая мгновенно вытеснила лампы керосиновые, которые использовались до этого. Вместе с электрической лампой в мир пришел и новый бренд – National.

Популярность продукции Matsushita Electric была настолько велика, что уже к концу 20-х годов в распоряжении Коносукэ находилось восемь заводов, на которых тысячи рабочих трудились над производством самой разной продукции – электролампочек, фонарей, велосипедных фар, электронагревателей, утюгов и не только.



Первый цветной телевизор K21-10, 1960 год

В 1932 году Коносукэ Мацуцита официально сформулировал цель компании — «Наше дело было поручено нам обществом. Вот почему мы должны направлять и развивать нашу компанию так, чтобы помогать тем самым развиваться и обществу, способствуя улучшению жизни людей». Видимо, именно этот подход и позволили фабрикам Мацуциты столь удачно пережить и Великую Депрессию, и годы Второй Мировой Войны. Одной из причин, почему компания выжила там, где другие корпорации прекращали свое существование, стала забота Коносукэ о работниках. «Бизнес – это люди», говорил он. И именно этот слоган не раз помогал ему оставаться на плаву даже в самые нелегкие времена.

Так, когда в 30-х годах Японию захлестнул экономический кризис, и другие компании были вынуждены сокращать количество своих сотрудников, Коносукэ Мацуцита не уволил ни одного из рабочих. Он нашел совершенно другой выход из ситуации. Сократив вдвое рабочий день каждого из них, он предложил работникам самостоятельно заняться реализацией накопившейся на складах компании продукции. За восемь месяцев благодарные работяги распродали все залежи товара, и компания только выиграла.



Первый CD-плеер SL-P10, 1982 год

Востребованность Matsushita Electric росла с каждым днем. Более того – в начале 50-х годов ее основатель, наконец, принял решение о выводе компании на мировую арену. Первоначально предполагалось и за пределами Японских островов продавать ее продукцию все под той же маркой – National. Но, как оказалось, такой бренд в США уже был зарегистрирован ранее. Необходимо было придумать нечто совершенно новое. А поскольку компания на тот момент предполагала делать ставку на аудиопродукцию, то и название родилось само по себе – Panasonic (это слово с древнегреческого переводится как «весь звук»). Позднее под этой маркой стала выпускаться практически вся продукция компании – от велосипедов до микроволновок. В 50-х годах Matsushita Electric завоевала США, Мексику и Канаду, а в 1962 году дошла и до Европы. Бренд Panasonic стал настолько узнаваем, что даже электроника National, предназначенная для внутреннего японского рынка, также была переименована.

Что касается логотипа Panasonic, то он появился в 1955 году и за все время своего существования менялся три раза. Поначалу слова “Pana” и “Sonic” писались с заглавных букв – PanaSonic. В 1966-м они были вписаны в круг, который, в свою очередь, находился внутри большой буквы “N”. И только в 1971-м была создана уже знакомая нам эмблема – простая надпись “Panasonic” без дополнительных элементов.



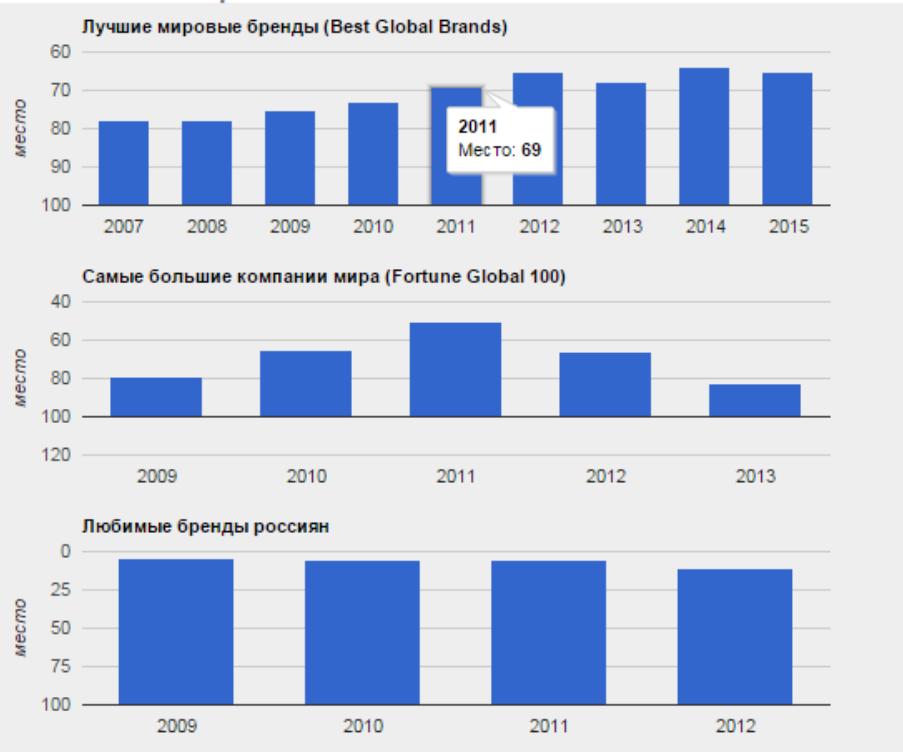
Первый DVD-плеер DVD-A300, 1996 год

В 1959 году Коносукэ Мацусита принял решение покинуть пост главы корпорации и отправиться на покой. В результате его отставки влияние Panasonic на рынке электротоваров заметно ослабло. Спрос на теле- и аудиоаппаратуру в 60-х годах заметно вырос, и конкуренцию японской марке составили также американские и европейские производители. Panasonic проигрывал по сравнению с ними — несмотря на высокое качественно техники, компания все еще пользовалась застаревшей системой сбыта продукции. После долгих уговоров Коносукэ Мацусита вновь возглавляет корпорацию. Его новаторский подход и талант управленца быстро вывели Panasonic из кризиса и вновь превратили в лидера по продажам электроники.

Активное развитие Panasonic продолжилось до середины 70-х годов, пока 80-летний Коносукэ не оставил своё детище уже окончательно. Сегодня компания все еще занимает увереные позиции на рынке электротоваров, продолжая двигаться в направлении, заложенном еще ее создателем. В последние годы благодаря Panasonic мир увидели первые пишущие DVD-плееры, SD-карты флэш-памяти и не только. Активной является работа компании в сфере создания новых технологий.



## Panasonic в рейтингах



Соответственно на 2015, 2013 и 2012 года: 65-ое место среди лучших мировых брендов, 83-е место среди больших компаний мира, 11-ое среди любимых брендов россиян.

## Epam

### **2014 год**

Международная компания в сфере инвестиционного менеджмента Emerald Asset Management вручила награду E-3 Awards Аркадию Добкину, президенту EPAM Systems, и Энтони Дж. Конте, финансовому директору компании, за «предпринимательский дух и преданность непревзойденному качеству».

### **2013 год**

EPAM заняла второе место в списке Forbes «20 самых быстрорастущих технологических звезд Америки».

Everest Group назвала EPAM звездой сферы ИТ-услуг для инвестиционного бизнеса в исследовании 2013 ITO in Capital Markets Service Provider Landscape and PEAK Report.

Штат специалистов в области разработки ПО в EPAM достиг 9000 человек.

EPAM заняла 6 место в списке Forbes «25 самых быстрорастущих технологических компаний Америки». Всего в оценке Forbes участвовало 2100 публичных компаний.

EPAM стала обладателем премии Duke's Choice Awards на конференции JavaOne Russia 2013. Данная награда вручается за инновационные проекты и вклад в развитие технологий Java.

EPAM продолжает активно сотрудничать с вузами страны. В первой половине 2013 года EPAM открыла новые лаборатории в ряде университетов – БГУ и БГУИР, БГЭУ, ГрГУ им. Янки Купалы, МГЛУ.

На пятой ежегодной конференции Oracle Commerce для партнеров в регионе EMEA EPAM получила награду Oracle «Top Business Impact Partner, EMEA – 2012».

Компания EPAM присоединилась к программе Windows Azure Circle. В данную программу входят партнеры Microsoft, которые оказывают услуги по планированию развертывания приложений компаниям, заинтересованным в переходе на использование облачных технологий на базе Windows Azure.

В рамках национального профессионального конкурса «Бренд года» в номинации «Бренд-персона» Аркадий Добкин, президент и председатель совета директоров компании EPAM Systems, был объявлен победителем в категории «Бизнес».

Компания EPAM Systems стала стратегическим партнером компании TriZetto по разработке ПО. TriZetto является ведущим поставщиком ИТ-решений для здравоохранения.

### **2012 год**

EPAM провела первичное размещение акций (IPO) на Нью-Йоркской фондовой бирже.

В рамках Профессионального конкурса «Бренд года-2011», EPAM была признана «Лучшим работодателем».

EPAM открыла совместную учебно-научную лабораторию по подготовке SAP-специалистов на базе БГЭУ. Благодаря такому сотрудничеству, БГЭУ получил возможность стать участником программы SAP University Alliances.

В рамках Партнерской программы Microsoft, EPAM получила компетенции уровня Gold в областях Microsoft Mobility, Business Intelligence и Microsoft SharePoint.

EPAM провела I Business Intelligence Open House, первую в Беларуси международную конференцию, посвященную Business Intelligence.

EPAM получила статус Золотого Партнера в рамках программы Oracle PartnerNetwork (OPN).

EPAM заключила соглашение о приобретении ведущей компании в области программных решений и IT-консалтинга Thoughtcorp (Торонто, Канада), расширив свое присутствие в Северной Америке.

EPAM снова признана «Лучшей IT-компанией глазами сотрудников» по итогам конкурса, организованного проектом Dev.by.

EPAM вошла в список 250 самых инновационных компаний Америки по данным рейтинга InformationWeek 500.

EPAM объявила о приобретении ведущей компании в области решений для электронной коммерции и цифровых медиа Empathy Lab, расширив свою компетенцию в разработке цифровых маркетинговых стратегий.

Численность IT-специалистов в штате EPAM Systems превысила отметку в 8000 человек.

## **2011 год**

EPAM в шестой раз подряд занимает лидирующие позиции среди всех ИТ-компаний в Центральной и Восточной Европе в рейтинге «100 лучших аутсорсинговых компаний мира» («2010 Global Outsourcing 100»), ежегодно составляемом Международной ассоциацией профессионалов аутсорсинга (International Association of Outsourcing Professionals — IAOP).

По итогам первого в Беларуси профессионального конкурса «Золотой Байт 2010» EPAM Systems признана лучшим экспортером Парка высоких технологий.

EPAM названа «Лучшей ИТ-компанией Беларуси глазами сотрудников». Компания получила максимальную оценку своих сотрудников по всем параметрам в конкурсе, организованном проектом Dev.by при поддержке Ассоциации «Инфопарк».

Новый центр разработки EPAM Systems в Krakowе (Польша) органично дополнил развитую сеть существующих европейских центров разработки и центров поддержки клиентов в России, Венгрии, Беларуси, Украине, Германии, Великобритании, Швеции и Швейцарии.

Штат специалистов в области разработки ПО в EPAM превысил 7000 человек.

## **2010 год**

Годовой оборот компании превысил \$200 миллионов.

Everest Group называет EPAM признанным лидером в Центральной и Восточной Европе, которого «должны знать» все компании Fortune 1000, в своем отчете «Поставщики из стран с развивающейся экономикой: отличные возможности для диверсификации рисков» («Emerging Markets Suppliers: A valuable Lever for Risks Diversification»).

Список клиентов EPAM Systems из финансового сектора пополнился одним из крупнейших мировых инвестиционных банков Barclays Capital.

EPAM Systems стала привилегированным поставщиком ИТ-услуг для Группы компаний Bosch.

EPAM значительно углубила отраслевую специализацию в сфере бизнес информации, объявив о приобретении компании Instant Information, Inc., лидера в области разработки программных решений и предоставления «облачных сервисов» для анализа информации и управления данными.

Компания получила наивысший партнерский статус в рамках программы SAP PartnerEdge — SAP Gold Channel Partner, который предоставляется компаниям за наличие компетенций в сфере развития решений на основе технологий SAP, высокий уровень квалификации специалистов, а также за успехи в области продвижения программных продуктов SAP.

EPAM Systems получила статус официального поставщика решений ATG (ATG Solutions Provider) в Европейском регионе и на Ближнем Востоке.

Система для управления проектами EPAM Project Management Center(EPAM PMC) компании EPAM Systems отмечена среди лучших продуктов для поддержки реализации проектов по методологии Scrum.

Центр тестирования и контроля качества, специализированное подразделение компании EPAM Systems, награжден в номинации «ИТ-подразделение года» в рамках ежегодного международного конкурса в области бизнеса The International Business Awards 2010.

EPAM успешно завершила проект по построению информационной системы управления в компании velcom.

По итогам года EPAM Systems признана «Лучшим экспортёром 2010» среди ИТ-компаний Беларуси.

## **2009 год**

На фоне очередного мирового финансового кризиса EPAM удалось успешно направить усилия на модернизацию и оптимизацию процессов в компании. Завоевав доверие и упрочив взаимоотношения с ключевыми заказчиками, EPAM Systems смогла предложить еще более выгодные решения и уникальные преимущества, чем когда-либо ранее.

Приверженность компании высокому качеству поставляемых услуг и решений в чрезвычайно сложных условиях кризиса вывела EPAM на лидирующие позиции среди всех поставщиков ИТ-услуг в Центральной и Восточной Европе:

EPAM, единственная из ИТ-компаний с центрами разработки в Центральной и Восточной Европе, второй год подряд включена в список 10 лучших мировых поставщиков ИТ-услуг («Top 10

Best Performers: IT services»), ключевой категории международного рейтинга «2009 Global Services 100». Кроме того, EPAM заняла второе место в десятке лидеров в категории «Аутсорсинг разработки программных продуктов» («Top 10 Best Performers: Outsourced Product Development»)

EPAM отмечена в числе мировых лидеров индустрии аутсорсинга в рамках независимого рейтинга «100 лучших аутсорсинговых компаний мира» («2009 Global Outsourcing 100»). Компания подтвердила свои лидирующие позиции, в очередной раз, заняв самое высокое место среди ИТ-компаний из Центральной и Восточной Европы

Уже в третий раз с 2006 года, EPAM является первой и единственной среди ИТ-компаний с центрами разработки в Центральной и Восточной Европе в рейтинге «50 наиболее эффективно управляемых аутсорсинг-компаний 2009» (2009 Top 50 Best Managed Global Outsourcing Vendors), ежегодно составляемом консалтинговой группой Brown-Wilson Group

EPAM названа компанией №1 в области разработки заказного программного обеспечения по рейтингу книги «Справочник об аутсорсинге» (the Black Book of Outsourcing)

## **2008 год**

EPAM привлекла инвестиции в \$50 миллионов для дальнейшего роста компании.

Существенно расширился спектр услуг для клиентов из финансового сектора благодаря приобретению компании B2BITS Corp., ведущего поставщика решений и консалтинговых услуг для фондового рынка. На основе B2BITS в EPAM создан специализированный финансовый Центр компетенции. Это приобретение позволило компании заполучить в качестве клиента один из крупнейших мировых инвестиционных банков.

Deloitte признала соответствие процессов в EPAM Systems мировым стандартам информационной безопасности и надежности услуг SAS 70 Type II. По сути, это означает признание EPAM достойным партнером для реализации ИТ-проектов, способным обеспечить сохранность конфиденциальных данных клиентов и высокое качество услуг в области разработки программного обеспечения, внедрения бизнес-приложений и консалтинга.

EPAM продолжает расширять свое географическое присутствие в Европе и открывает представительство в Стокгольме (Швеция).

Аудиторская компания Ernst&Young назвала Аркадия Добкина «Предпринимателем года» штата Нью-Джерси в области ИТ-консалтинга.

По итогам работы в Беларуси EPAM Systems названа «Лучшим экспортером года в сфере науки, образования и информационных технологий 2008», кроме того, компания признана «Лучшим предпринимателем года Республики Беларусь 2008» в сфере создания новых рабочих мест.

Число сотрудников EPAM Systems увеличилось до 5000.

## **2007 год**

Годовой оборот компании достиг \$100 миллионов. Многие независимые аналитические агентства признали EPAM Systems безусловным лидером в области разработки ПО в Центральной и Восточной Европе.

EPAM значительно расширяется, добавив к списку своих центров разработки 18 совместных с ведущими вузами учебных лаборатории в России, Беларуси и Украине, делая ставку на количественный и профессиональный рост своих существующих и потенциальных сотрудников. Открылись новые офисы в Бресте (Беларусь), в украинских Харькове, Виннице и Львове и в российском Ижевске.

В августе 2007 г. EPAM Systems первой в России и странах СНГ сертифицировала по CMMI4 сразу несколько центров разработки программного обеспечения.

Компания EPAM Systems выступила консультантом адаптации ERP-системы SAP для «Кока-Кола Бевриджиз Белоруссия».

Штат специалистов EPAM Systems перешагнул отметку в 3500 человек, в том числе в течение года к компании присоединились свыше 500 новых сотрудников в Минске.

## **2006 год**

EPAM Systems для обеспечения дальнейшего уверенного роста и повышения конкурентоспособности привлекла внешние инвестиции компании Siguler Guff.

Авторитетное международное издание Consulting Magazine назвало Аркадия Добкина одним из 25-ти лучших консультантов года.

EPAM Systems становится резидентом №1 в созданном в Беларуси Парке высоких технологий.

В августе 2006 года впервые состоялась EPAM Software Engineering Conference — первая глобальная конференция разработчиков EPAM Systems. В мероприятии приняло участие более 300 сотрудников компании из пяти стран, в которых находятся центры разработки компании.

Осенью 2006 года EPAM объявила о слиянии с компанией VDI, одним из лидеров в области разработки заказного программного обеспечения и внедрении сложных корпоративных решений в России, укрепив свои позиции на рынке России и других стран СНГ. Объединенная компания стала крупнейшим поставщиком услуг в области разработки программного обеспечения и решений в Центральной и Восточной Европе.

EPAM Systems стала Глобальным партнером Hyperion — мирового лидера в области программного обеспечения для управления эффективностью бизнеса.

Появился центр поддержки клиентов во Франкфурте-на-Майне (Германия) и новые центры разработки в России (на базе офисов VDI).

Штат сотрудников EPAM Systems превысил 2000 квалифицированных специалистов в области разработки ПО. Из них за год только в Минске число сотрудников увеличилось на 400 человек, за что компания удостоилась звания «Лучший предприниматель 2006 года г. Минска» в сфере создания новых рабочих мест.

## **2005 год**

EPAM впервые признана компанией № 1 в категории «Пять ведущих аутсорсинг-компаний в Центральной и Восточной Европе», а также включена в список «Десяти лидеров в области

разработки специализированных приложений в ежегодном международном рейтинге лучших поставщиков услуг аутсорсинга «Offshore 100».

EPAM Systems стала второй в списке самых быстрорастущих софтверных компаний с годовыми доходами от 30 до 100 млн. долларов во всемирном рейтинге крупнейших компаний-разработчиков программного обеспечения Software 500.

Компания EPAM Systems успешно завершила работы по созданию для авиакомпании «Сибирь» ([www.s7.ru](http://www.s7.ru)) первой в России системы онлайновой продажи авиабилетов.

Открывается представительство в Лондоне (Великобритания) и новые центры разработки в С.-Петербурге (Россия) и Киеве (Украина).

#### **2004 год**

Стратегическое слияние с Fathom Technology, венгерским разработчиком ПО, открывает для EPAM Systems европейский рынок и укрепляет позиции компании как ключевого игрока в верхнем эшелоне поставщиков услуг в области глобального аутсорсинга. EPAM Systems становится крупнейшим поставщиком программных услуг и решений в Центральной и Восточной Европе.

В Будапеште (Венгрия) создается европейская штаб-квартира компании.

Авторитетное издание Software Magazine включает EPAM Systems в ежегодный рейтинг 500 ведущих софтверных компаний мира.

Компания EPAM Systems, уделяя большое внимание качественной подготовке молодых специалистов, впервые выступила спонсором команды БГУ на чемпионате мира по программированию.

При содействии EPAM открылась первая совместная учебная лаборатория на базе Белорусского государственного университета информатики и радиоэлектроники.

Появился офис в Саратове (Россия).

Штат сотрудников EPAM достигает 1000 высококлассных специалистов в области разработки ПО.

#### **2002-2003 год**

EPAM значительно расширила свою клиентскую базу, упрочила отношения с некоторыми ключевыми клиентами, которые со временем переросли в стратегическое долгосрочное партнерство.

Аналитики в области ИТ начинают замечать успехи компании. EPAM названа одним из лидеров предпринимательского роста на американском рынке по версии журнала Inc. EPAM Systems впервые включена в список самых быстрорастущих технологических компаний Technology Fast 500, составленного компанией Deloitte & Touché.

В феврале 2002 года EPAM Systems была сертифицирована на соответствие требованиям международного стандарта ISO 9001:2000.

В сентябре 2003 компания первой в Европе получила сертификат соответствия четвертому уровню зрелости модели SEI CMMI.

Открылся внутренний тренинг-центр. Появился офис в Могилеве (Беларусь).

#### **1999-2001 год**

EPAM испытала на себе бурный рост и крах пузыря «доткомов», выстояв в сложной ситуации после потери своего крупнейшего заказчика. События тех лет послужили ценным опытом, придали уверенности и решительности для дальнейшего развития компании.

EPAM Systems вышла на формирующийся рынок eCommerce-решений. Компания занялась разработкой сложных интерактивных персонализированных порталов и мобильных приложений для компаний из различных отраслей (автобизнес, телекоммуникации, ИТ, здравоохранение и т.д.) в Северной Америке, Европе и Японии.

В 2001 году EPAM стала официальным партнером корпорации Microsoft.

В Беларусь открылись новые офисы в Гродно и Гомеле. Появился отдел обеспечения качества.

#### **1997-1998 год**

Решение, разработанное для Colgate-Palmolive, было замечено одним из руководителей компании SAP, лидера на рынке корпоративных приложений. В результате чего EPAM получила пробный проект на разработку ПО для SAP. Успешное выполнение этого проекта и последующее заключение партнерства с SAP ознаменовало выход EPAM Systems на новый профессиональный уровень — аутсорсинг разработки программных продуктов для высокотехнологичных компаний. С этого времени, EPAM активно принимает участие в сотнях проектов, как для ведущих мировых поставщиков программного обеспечения, так и для многочисленных развивающихся технологических вендоров и стартапов.

В этот период появляется первый офис компании в Москве.

#### **1995-1996 год**

Спустя несколько лет напряженной работы, накопив определенную долю опыта, EPAM Systems подписала контракт со своим первым крупным заказчиком — Bally of Switzerland, одним из ведущих мировых производителей модной одежды. EPAM разработала для американского подразделения компании решение по автоматизации деятельности торгового персонала и управлению товарными складами.

Примерно в это же время, вторым значимым клиентом EPAM Systems стала компания Colgate-Palmolive, транснациональный производитель потребительских товаров. Специалисты EPAM создали для них гибкую автоматизированную систему поддержки продаж, эффективность работы которой в дальнейшем стала поводом для ее внедрения в десятках представительств Colgate-Palmolive по всему миру.

Успех этих проектов во многом предопределил будущее направление развития EPAM Systems в качестве поставщика современных информационных технологий и эффективных решений для ведущих мировых предприятий и крупнейших корпораций.

## **1993 год**

EPAM Systems была основана двумя одноклассниками — Аркадием Добкиным и Леонидом Лознером. В начале 90-х Аркадий Добкин переехал в США, где в 1993 году в городе Принстон, штат Нью-Джерси, создал компанию EPAM. Леонид Лознер же, оставшись в Беларуси, занялся развитием офиса в Минске. Таким образом, EPAM с первого дня своего существования стала международной компанией, разрабатывающей программное обеспечение для небольших американских заказчиков, которые на тот момент не могли себе позволить услуги более крупных поставщиков ПО. Первоначально даже само название «EPAM» было акронимом фразы «Effective Programming for America», однако, со временем, выходя на новые рынки, оно трансформировалось в современное — EPAM Systems.

Для решения управлеченческих задач в EPAM Systems используется программное обеспечение собственной разработки, интегрирующее в себе лучшие практики мировой ИТ-индустрии:

- EPAM Project Management Center (PMC) – для управления проектами в распределенной среде. Система покрывает все этапы разработки программного обеспечения, создавая комфортные условия для отчетности для команды и отслеживания хода проекта со стороны менеджеров проекта, заказчиков и других лиц, вовлеченных в процесс. EPAM PMC занимает особое место в ИТ-инфраструктуре EPAM. Эта современный, мощный и надежный инструмент управления распределенными проектами получил высокие оценки как клиентов EPAM Systems, так и признание со стороны экспертного сообщества. EPAM PMC предоставляет проектной команде, менеджменту компании и представителям заказчика следующие ресурсы и сервисы:
  - Общий репозитарий документации;
  - Планирование проектов;
  - Формирование команды;
  - Управление требованиями и изменениями;
  - Управление конфигурациями;
  - Управление дефектами;
  - Отчет об отработанном времени;
  - Доступ команды и заказчика;
  - Задания – формирование, импорт, уведомления, отчеты, учет;
  - Автоматическое уведомление об изменениях;
  - KPI проектов.
- EPAM Utilization and Project Staffing Analyzer (UPSA) – для ресурсного планирования и анализа проектного портфолио. Эта система, созданная EPAM на базе web-технологий и доказавшая свою эффективность в ходе реализации сотен ИТ-проектов, обеспечивает все необходимые возможности для успешного управления человеческим капиталом в проектно-ориентированной организации. Инструмент позволяет сделать максимально быстрым и эффективным процесс отбора специалистов на проект по заданным критериям (например, знание определенных технологий, опыт участия в аналогичных проектах, местоположение, доступность и т.п.) и обеспечить оптимальное распределение ресурсов компании.

- EPAM Human Resource Management – Applicants Workbench (EPAM HRM –AW) — для эффективного управления персоналом. Система содержит базу данных всех кандидатов, обращавшихся в компанию, включая студентов, которые в данный момент обучаются в тренинг-центре или студенческих лабораториях EPAM. При этом в системе содержится не только резюме, но и результаты пройденных тестов, предварительные результаты собеседований и другая важная информация о кандидатах. Это дает возможность в случае необходимости оперативного привлечения в компанию необходимых специалистов.
- EPAM Key Performance Indicators System (EPAM KPI) – для анализа и принятия оперативных управленческих решений. Система позволяет в удобном для анализа виде получить доступ к статистике, отслеживать эффективность процессов по различным показателям и оперативно вносить необходимые корректировки и принимать меры для достижения запланированных результатов.
- EPAM Cost Tracking Center (EPAM CTC) – для контроля над затратами. Система обеспечивает такие функции, как планирование, утверждение командировок и поддержка процесса отчетности по ним, а также формирование и одобрение заявок на закупки товаров или услуг для внутренних нужд компании и предоставление отчетов по произведенным затратам.
- EPAM Hardware Utilization Analyzer (EPAM HUA) – для эффективного использования технологических ресурсов компании. Аналогично UPSA, система EPAM HUA позволяет в любой момент проверить загрузку серверов и убедиться в том, что каждый проект обеспечен необходимыми программно-аппаратными мощностями, оперативно выявить несоответствия и решить проблему.

EPAM Scrum Tool - для поддержки полного цикла разработки программного обеспечения по Agile-методологии

И другие внутренние системы, позволяющие сделать процессы реализации проектов для заказчиков максимально эффективными.

Кроме Ерат, в Беларусь активно развиваются следующие компании:

Десять белорусских IT-компаний вошли в 2015 году в список Software 500, опубликованный одним из самых влиятельных изданий мировой индустрии высоких технологий — журналом Software Magazine.

Как сообщается на сайте ПВТ, компания IHS (резидент ПВТ ИП "АйЭйЧЭс Глобал") заняла 64-е место среди крупнейших компаний — разработчиков ПО.

Bell Integrator (ООО "Бэлл Интегратор") заняла 228-е место; Ericpol Telecom (ИООО "ЭРИКПОЛЬ БРЕСТ") — 296-е; IBA (резидентами ПВТ являются ИП "АйБиЭй АЙТи Парк" и УП "ИВА-Гомель-Парк") — 301-е; Itransition (ЗАО "Итранзишэн") — 374-е; SoftClub (резидент ПВТ "СОФТКЛУБ — Центр разработки") — 417-е; Coherent Solution (резидент ПВТ "Иссофт Солюшенз") — 418-е; Artezio (УП "Артезио") — 436-е; Intetics (резидент ПВТ "Интетикс Бел") — 444-е место.

## Глава 3. Архитектура компьютера и мобильных устройств

### Архитектура компьютера

В современных персональных компьютерах, как правило, используется **принцип открытой архитектуры**. Он заключается в том, что все устройства компьютера взаимодействуют и соединяются между собой стандартным или известным образом и любой производитель, руководствуясь ими, может начать производство какого-либо устройства.

Преимущества открытой архитектуры:

- возможность выбора необходимой конфигурации компьютера
- возможность расширять и модернизировать компьютерную систему

Закрытую архитектуру тоже используют, например компания Apple. Они не разглашают спецификацию своих устройств и производят их сами.

Архитектура современных компьютеров основана на магистрально-модульном принципе. Компьютер состоит из разрозненных частей – модулей. Модулем ПК будем называть любое относительно самостоятельное устройство компьютера (процессор, оперативная память, контроллер, дисплей, принтер, сканер и т.д.) Для того чтобы компьютер работал как единый механизм, необходимо осуществлять обмен данными между различными устройствами, за это отвечает **системная (магистральная) шина**. Она как правило располагается на системной плате - это основная электронная плата в компьютере.

Устройства могут соединяться с системной шиной напрямую, некоторые устройства соединяются через адаптеры, контроллеры.

Системная шина включает в себя:

- **Адресную шину** - По этой шине передается адрес требуемой ячейки памяти или устройства, с которым будет происходить обмен информацией.
- **Шину управления** - Регулирует процесс передачи данных.
- **Шину данных** - По этой шине данные передаются между различными устройствами. Разрядность шины данных определяется разрядностью процессора.

### Процессор

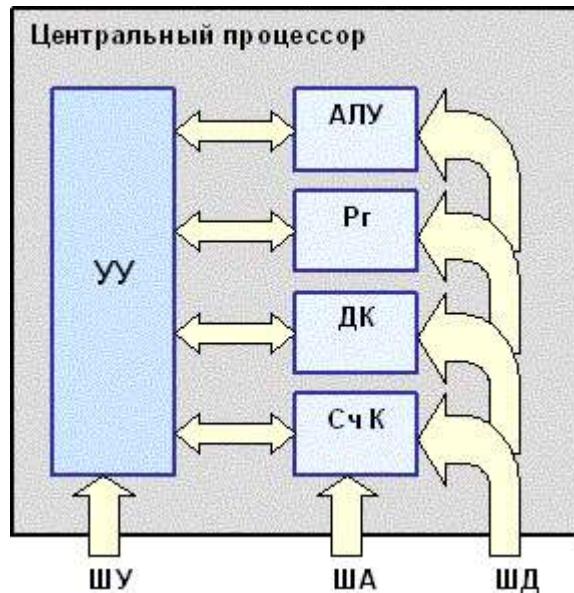
Соединяется процессор с магистралью напрямую и находится на системной плате вместе с ней.

Сам процессор состоит из десятка миллионов транзисторов, а сейчас и больше, при помощи которых собраны отдельный логические схемы, находящиеся в специальном кремниевом корпусе. Именно из-за кристалла кремния очень часто процессор называют «камнем».

Состоит процессор из:

- 1) устройства управления (УУ), которое управляет работой процессора с помощью электрических сигналов;
- 2) арифметико-логического устройства (АЛУ), производящего операции над данными;
- 3) регистров (Рг) для внутреннего хранения в процессоре этих данных и результатов операций над ними;
- 4) дешифратора команд (ДК) – устройство для расшифровки команд;
- 5) счетчика команд (СЧК) – регистр для хранения адреса очередной команды.

Функциональная схема ЦП



Важной характеристикой процессора является его **производительность** — количество элементарных операций, выполняемых им за одну секунду — которая и определяет **быстродействие компьютера** в целом.

Производительность процессора зависит от тактовой частоты и разрядности.

**Тактовая частота** – количество тактов в секунду, где такт – интервал времени между началами двух соседних тактовых импульсов. Эти импульсы, поступают от кварцевого тактового генератора, который при включении ПК начинает вибрировать с постоянной частотой (100 МГц, 200-400 МГц и выше). Эти колебания задают темп работы всей системной платы. Поднять тактовую частоту можно просто поменяв модель процессора на более новую, где тактовый генератор будет мощнее, но иногда можно изменить настройки bios персонального компьютера, указав тактовую частоту выше установленной производителем. Но не все процессоры могут изменять тактовую частоту. Слишком высокая тактовая частота может привести к перегреву компьютера.



Для современных компьютеров тактовая частота измеряется единицами гигагерц ( $1 \text{ ГГц} = 10^9 \text{ Гц}$ ). Разрядность процессора в зависимости от типа – 8, 16, 32, 64 разрядные. **Разрядность** – число битов, одновременно обрабатываемых процессором за один такт (машинное слово). Зависит от разрядностей регистров и шины данных.

Для расширения возможностей ПК и повышения функциональных характеристик микропроцессора дополнительно может поставляться **математический сопроцессор**, служащий для расширения

набора команд МП. Например, математический сопроцессор IBM-совместимых ПК расширяет возможности МП для вычислений с плавающей точкой.

### Память

Различают два вида памяти - **внутреннюю и внешнюю** (по отношению к системной плате).

Внутренняя память компьютера является быстродействующей, но имеет ограниченный объем. Работа с внешней памятью требует гораздо большего времени, но она позволяет хранить практически неограниченное количество информации.

Внутренняя память состоит из нескольких частей: оперативной (ОЗУ), постоянной (ПЗУ) и кэш-памяти.

Скорость работы компьютера существенным образом зависит от быстродействия оперативной памяти. Поэтому, постоянно ведутся поиски элементов для оперативной памяти, затрачивающих меньше времени на операции чтения-записи. Но вместе с быстродействием возрастает стоимость элементов памяти, поэтому наращивание быстродействующей оперативной памяти нужной емкости не всегда выгодно экономически.

Под оперативной памятью понимают **оперативное запоминающее устройство** (ОЗУ) – RAM – RandomAccessMemory – это запоминающее устройство, непосредственно связанное с процессором и предназначено для записи, считывания и временного хранения выполняемых программ и данных

Оперативная память обеспечивает хранение информации лишь в течение сеанса работы с ПК – после выключения компьютера из сети данные, хранимые в ОЗУ, теряются безвозвратно, то есть ОЗУ – энергозависимое устройство.

**Процессор** – устройство, которое обеспечивает общее управление компьютером и осуществляет обработку информации по программе, загруженной в оперативное запоминающее устройство (ОЗУ).

Для выполнения программы, хранящейся в ОЗУ, процессор:

1. считывает из ОЗУ очередную команду программы;
2. расшифровывает команду;
3. выполняет действия, указанные в этой команде.

В современных компьютерах скорость обработки информации процессором уже так высока, что современные ОЗУ не справляются с функцией посредника между ЦП и внешней памятью. Поэтому было добавлено еще одно устройство – кэш-память – служащее посредником между ОЗУ и ЦП. Современные процессоры имеют встроенную кэш-память.

Если увеличить объём кэш памяти, данные будут передаваться быстрее между регистрами и внутренней памятью. Кэш-память играет буфера между ограниченными, но очень быстрыми регистрами процессора и сравнительно медленной, но гораздо более вместительной внутренней памятью компьютера. Кэш-память работает примерно со скоростью самого процессора, поэтому, когда процессор обращается к данным в кэше, процессору не приходится ждать этих данных.

Кэш память устроена так, что при попытке прочитать данные из внутренней памяти сначала аппаратным образом проверяется, нет ли нужных данных в кэше. Если эти данные в кэше, они быстро извлекаются и используются процессором. Однако в противном случае эти данныечитываются из внутренней памяти и в момент передачи процессору также помещаются в кэш (на случай, если они понадобятся позже). Важное отличие между обращениями к данным в кэше или обращением к данным в внутренней памяти — это время, необходимое для получения данных.

Кэш центрального процессора включать в себя несколько уровней. Существует четыре уровня кэш-памяти. Как правило, кэш-память следующего уровня больше по размеру и медленнее по скорости доступа и передаче данных предыдущего уровня кэш-памяти.

Самым быстрым является кэш первого уровня — L1 cache (level 1 cache). По сути, она является неотъемлемой частью процессора, поскольку расположена на одном с ним кристалле и входит в состав функциональных блоков. В современных процессорах обычно L1 разделен на два кэша — кэш команд (инструкций) и кэш данных. Большинство процессоров без L1 не могут функционировать. L1 работает на частоте процессора, и, в общем случае, обращение к нему может производиться каждый такт. Зачастую является возможным выполнять несколько операций чтения/записи одновременно.

Вторым по быстродействию является кэш второго уровня — L2 cache, который обычно, как и L1, расположен на одном кристалле с процессором. В ранних версиях процессоров L2 реализован в виде отдельного набора микросхем памяти на материнской плате. Объём L2 от 128 кбайт до 1–12 Мбайт. В современных многоядерных процессорах кэш второго уровня, находясь на том же кристалле, является памятью раздельного пользования — при общем объёме кэша в n Мбайт на каждое ядро приходится по n/c Мбайта, где c — количество ядер процессора.

Кэш третьего уровня наименее быстродействующий, но он может быть очень большим — более 24 Мбайт. L3 медленнее предыдущих кэшей, но всё равно значительно быстрее, чем оперативная память. В многопроцессорных системах находится в общем пользовании и предназначен для синхронизации данных различных L2.

Существует четвёртый уровень кэша, применение которого оправдано только для многопроцессорных высокопроизводительных серверов и мейнфреймов. Обычно он реализован отдельной микросхемой.

**Постоянное запоминающее устройство (ПЗУ)** — предназначено для хранения оперативной информации, обеспечивающей запуск компьютера.

Блок ПЗУ состоит из двух частей:

- **BIOS (Basic Input/Output System)** — базовая система ввода и вывода. В ней хранится постоянная информация, заложенная на заводе-изготовителе, обеспечивающая запуск ПК.
- **CMOS** — переменная часть ПЗУ, где хранится информация о конфигурации ПК (перечень устройств, входящих в комплект ПК и их характеристики).

После выключения питания компьютера, информация в ПЗУ сохраняется, за счет энергии от специальных автономных батарей. Таким образом, ПЗУ является энергонезависимой памятью.

## Периферийные устройства

Периферийные устройства - вспомогательное оборудование, которое не является необходимым для работы компьютера их можно разделить на три типа:

- устройства ввода данных,
- устройства вывода данных,
- устройства хранения данных.

Примеры периферийных устройств: мониторы, принтеры, мышки, накопители на жёстких или гибких магнитных дисках.

Такие устройства подключаются к компьютеру с помощью специальных адаптеров или контроллеров, которые связывают их с системной шиной. Эти контроллеры (или адаптеры) могут находиться как на системной плате, так и вне её. (например, контроллеры клавиатуры, аудио контроллеры, контроллеры сети могут находиться на системной плате). Любое периферийное устройство нуждается в специальных программах (для управления каждым устройством - своим). Такие программы называются **драйверами** (от английского drive - приводить в движение, управлять).

Раньше работой устройств ввода-вывода руководил центральный процессор, что занимало немало времени. Архитектура современных компьютеров предусматривает наличие каналов прямого доступа к оперативной памяти для обмена данными с устройствами ввода-вывода без участия центрального процессора, а также передачу большинства функций управления периферийными устройствами специализированным процессорам, разгружающим центральный процессор и повышающим его производительность.

Схема аппаратурной части компьютера:



### Операционные и вычислительные системы

Операционной системой называют комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

#### Структура операционной системы

Ядро – центральная часть операционной системы (ОС), обеспечивающая приложениям доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации.

В состав операционной системы входит специальная программа — командный процессор, которая запрашивает у пользователя команды и выполняет их. Пользователь может дать, например, команду выполнения какой-либо операции над файлами (копирование, удаление, переименование, команду вывода документа на печать и т. д.). Операционная система должна эти команды выполнить.

Для упрощения работы пользователя в состав современных операционных систем, и в частности в состав Windows, входят программные модули, создающие графический пользовательский интерфейс. В операционных системах с графическим интерфейсом пользователь может вводить команды с помощью мыши, тогда как в режиме командной строки необходимо вводить команды с клавиатуры.

В состав операционной системы входят драйверы устройств — специальные программы, которые обеспечивают управление работой устройств и согласование информационного обмена с другими устройствами. Любому устройству соответствует свой драйвер.

Операционная система содержит также сервисные программы, или утилиты. Такие программы позволяют обслуживать диски (проверять, сжимать, дефрагментировать и т. д.), выполнять операции с файлами (архивировать и т. д.), работать в компьютерных сетях и т. д.

Для удобства пользователя в операционной системе обычно имеется и справочная система. Она предназначена для оперативного получения необходимой информации о функционировании как операционной системы в целом, так и о работе ее отдельных модулей.

### [Запуск компьютера](#)

При поступлении сигнала о запуске процессор обращается к специально выделенной ячейке памяти. В ОЗУ в этот момент ничего нет, если бы там была какая-либо программа, то она начала бы выполняться. Для того чтобы компьютер мог начать работу необходимо наличие ПЗУ, где записываются программы на BIOS. После включения компьютера процессор начинает считывать и выполнять микрокоманды, которые хранятся в микросхеме BIOS. Прежде всего начинает выполняться программа тестирования POST, которая проверяет работоспособность основных устройств компьютера. В случае неисправности выдаются определенные звуковые сигналы, а после инициализации видеoadаптера процесс тестирования отображается на экране монитора. Затем BIOS начитает поиск программы-загрузчика операционной системы. Программа-загрузчик помещается в ОЗУ и начинается процесс загрузки файлов операционной системы.

### [Загрузка операционной системы](#)

Файлы операционной системы хранятся во внешней, долговременной памяти. Однако программы могут выполняться, только если они находятся в ОЗУ, поэтому файлы ОС необходимо загрузить в оперативную память. Диск, на котором находятся файлы операционной системы и с которого происходит загрузка, называют системным. Если системные диски в компьютере отсутствуют, на экране монитора появляется сообщение "Nonsystemdisk" и компьютер «зависает», т. е. загрузка операционной системы прекращается и компьютер остается неработоспособным. После окончания загрузки операционной системы управление передается командному процессору. В случае использования интерфейса командной строки на экране появляется приглашение системы для ввода команд, в противном случае загружается графический интерфейс операционной системы. В случае загрузки графического интерфейса операционной системы команды могут вводиться с помощью мыши.

### [Наиболее известные операционные системы](#)

#### [\*Windows\*](#)

На сегодняшний день операционные системы семейства Windows наиболее популярны, которые являются коммерческим продуктом корпорации Microsoft.

Свою «родословную» Windows начинают от операционной системы DOS и первоначально представляли собой надстраиваемые над ней оболочки (Windows запускался из-под DOS), увеличивающие возможности DOS и облегчающие неподготовленному пользователю работу с компьютером. Уже более поздние версии (начиная с Windows NT) представляли собой полноценные операционные системы.

Преимуществом Windows считается дружественный для пользователя интерфейс. Из недостатков отмечают большую уязвимость к вирусам.

### *Unix-подобные ОС*

Операционная система UNIX оказала большое влияние на развитие мира операционных систем, заложив основы работы современных ОС. Изначально UNIX был системой для разработки ПО. В основном в UNIX работали программисты (да и вообще в 70-е годы мало кто другой работал с вычислительными машинами).

UNIX развивался на нескольких фундаментальных идеях. Например, одна небольшая задача должна решаться одной небольшой программой, а сложные задачи должны быть решаемы комбинацией простых программ.

В UNIX большое внимание уделено распределению ресурсов компьютера между пользователями. Эта система является мультитерминальной (каждый пользователь работает с компьютером с помощью своего терминала).

Несмотря на то, что Unix-подобные системы уступают по популярности Windows, они работают на больших типах компьютеров.

### *Linux*

ОС Linux представляет собой множество Unix-подобных операционных систем (дистрибутивов), которые чаще всего являются свободно распространяемыми.

Linux – очень устойчива. С ней почти никогда не случаются «глюки», и она очень редко виснет. Разве только иногда может зависнуть какое-нибудь приложение, но это сразу разрешается «двумя кликами». В Linux очень мало вирусов и все они известны. В результате, если пользователь мало-мальски знаком с командной строкой, то угрозы они не представляют.

Linux является основной операционной системой для суперкомпьютеров. В рейтинге самых мощных суперкомпьютеров мира Топ-500 за июнь 2014 года зафиксирован рекорд: 485 из 500 суперкомпьютеров работают на операционной системе Linux. Это 97%.

### *MAC OS*

Это операционная система также создавалась на основе ядра UNIX. Является продукт компании Apple для ее же компьютеров Macintosh. Считается надежной и удобной.

*Основные преимущества:*

Отсутствие проблемы поиска драйверов под новые устройства. При установке операционной системы Mac с DVD диска происходит автоматическая установка драйверов.

Стабильность работы. Программные сбои происходят намного реже так, как весь софт разрабатывается специально для компьютеров фирмы Apple, одной и очень большой командой профессиональных разработчиков.

Небольшое количество вредоносных программ и отсутствие целых их классов. Систему не получится повредить без помощи пользователя.

### *Вычислительные системы*

ВС – это взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации. Основными ресурсами

вычислительной системы являются процессоры, области оперативной памяти, наборы данных, периферийные устройства, программы.

*Структура вычислительной системы:*

- техническое обеспечение (hardware): процессор, память, монитор, дисковые устройства и т.д., объединенные магистральным соединением, называемым шиной;
- программное обеспечение: прикладное и системное.

К прикладному программному обеспечению относятся разнообразные пользовательские программы, игры, текстовые процессоры и т. п. К системному – программы, способствующие функционированию и разработке прикладных программ. Операционная система является фундаментальным компонентом системного программного обеспечения.

В зависимости от ряда признаков различают следующие вычислительные системы:

- по количеству программ, одновременно находящихся в оперативной памяти: однопрограммные и многопрограммные вычислительные системы;
- числу пользователей, которые одновременно могут использовать ресурсы вычислительной системы: индивидуального и коллективного пользования;
- организации и обработки заданий: вычислительные системы с пакетной обработкой и с разделением времени;
- числу процессоров: однопроцессорные вычислительные системы, многопроцессорные и многомашинные.

## Мобильные операционные системы

*В общих чертах*

Дизайн мобильных ОС прошел эволюцию от ОС для настольных ПК через встраиваемые ОС до тех продуктов, которые мы видим в смартфонах сейчас. В течение этого процесса архитектура ОС менялась от сложной к простой и остановилась где-то на середине. Сама же эволюция приводилась в движение технологическими достижениями в аппаратной и программной области, а также в интернет сервисах.

С точки зрения моделей потребления, все представители мобильных ОС сегодняшнего дня (такие как Apple iOS, Google Android, Microsoft Windows) имеют больше сходных черт, нежели различий:

- Все они имеют документированные SDK с прописанными API, что позволяет разработчикам создавать приложения под эти ОС;
- Все они имеют онлайн каталоги приложений, где разработчики публикуют свои приложения и откуда пользователи их скачивают;
- В каждой реализована многозадачность и поддержка 3D-графики, широко используются датчики и сенсорные экраны;
- Во всех системах большое внимание уделено гладкости и отзывчивости во взаимодействии с пользователем;
- Использование интернет далеко ушло от статических страниц, HTML5 становится платформой по умолчанию для веб-приложений;

- Все ОС поддерживают мобильные системы платежей;
- Все системы сфокусированы на оптимизации энергопотребления.

Общность нынешних мобильных ОС обусловлены глобальностью технологических трендов в аппаратной и программных областях, а также в коммуникациях. Проанализируем теперь ОС нового поколения с точки зрения критериев, приведенных выше.

#### *Управление энергопотреблением*

Энергоэффективность всегда была головной болью для разработчиков мобильных ОС. Прожорливость приложений постоянно растет, и прогресс в аккумуляторных технологиях за ней хронически не успевает. Вот почему важность управления питанием все время возрастает, и для решения этой проблемы необходимо применять поистине глобальный подход.

За последнее десятилетие значительных успехов в области экономии энергии достигли мобильные процессоры. Современные модели поддерживают технологии динамического изменения напряжения и частоты, таких как Enhanced Intel SpeedStep. Со стороны ОС управлением режимами работы процессора занимаются специальные компоненты ядра, такие, например, как cufs freq в Linux. В настоящее время мы наблюдаем процесс перемещения передового фронта борьбы за энергоэффективность от процессоров (где уже сделано немало) к другим системам мобильных устройств. Например, внедрение динамического управления графическим процессором (подобного тому, что применяется в ЦПУ) позволяет в некоторых случаях экономить до 50% энергии. Внимания заслуживают также системы ввода-вывода; повышение их интеллектуальности, способности самостоятельно выбирать оптимальный режим работы также положительно скажется на потреблении.

Корректность работы приложений с точки зрения энергопотребления остается ахиллесовой пятой обоих описанных подходов к сбережению. Недавние исследования показали, что бесплатные приложения Android потребляют 75% энергии впустую, показывая рекламу в свернутом режиме и не отдавая блокировку. То же самое справедливо и для Windows 8, где даже одно приложение, написанное неверно с точки зрения энергоэффективности, не позволит всей системе уйти в подключенный ждущий режим. В настоящее время не существует четкого понимания, как бороться с такого рода «кривыми» приложениями.

Немного конкретики: как показывает HTC One (M8), мобильная операционная система Microsoft Windows Phone позволяет сохранить заряд батареи дольше, чем система Google Android. Это продемонстрировала, по крайней мере, последняя модель HTC с Windows Phone. Две версии HTC One (M8) продаются в США у мобильного оператора Verizon Wireless. Обе версии одинаковы технически, но у них разные операционные системы. На веб-сайте Slashgear опубликованы результаты теста, показывающие, что смартфон с Windows Phone после использования в течение дня, двух часов и 41 минуты имеет еще 81 процент заряда батареи. Версия с Android имеет лишь 68 процентов после такого же периода работы.

#### *Открытость*

Другой важной отличительной чертой мобильной ОС является ее открытость. Под открытостью мы понимаем меру свободы в использовании, распространении, настройки и усовершенствовании ОС для своих нужд.

Еще совсем недавно большинство телефонов имели внутри себя закрытое ПО, куда не имели доступ сторонние разработчики; пользователям же приходилось довольствоваться встроенным инструментарием. В процессе эволюции появились смартфоны с операционными системами, допускающими установку стороннего ПО, которое взаимодействовало с ОС посредством API; разработчикам были предоставлены соответствующие инструменты программирования (SDK). Хорошим примером ОС подобного рода является Apple iOS. Большую свободу для всей экосистемы предоставляют ОС с открытым кодом, как, например, Android; преимущества открытого кода может почувствовать даже конечный пользователь, не имеющий отношения к программированию – они, например, в количестве производителей, использующих эту ОС и, в конечном счете, количестве моделей.

#### *Поддержка облачных технологий*

Облачные технологии находят все более широкое распространение в мобильных ОС; в большинстве своем, приложения, их использующие, представляют собой веб-сайты, открывающиеся в браузере или веб-приложения. Очень часто приложения созданы на HTML5, поэтому реализация данной технологии в мобильной ОС находится в центре внимания ее разработчиков.

Подводя итог всему сказанному, еще раз отметим, что несмотря на некоторые различия в подходах, все мобильные ОС развиваются в одном направлении, в какой-то степени сближаясь друг с другом. Думается, что эта тенденция сохранится и в дальнейшем, что идет только на руку как пользователям, так и разработчикам приложений.

#### **Статистика**

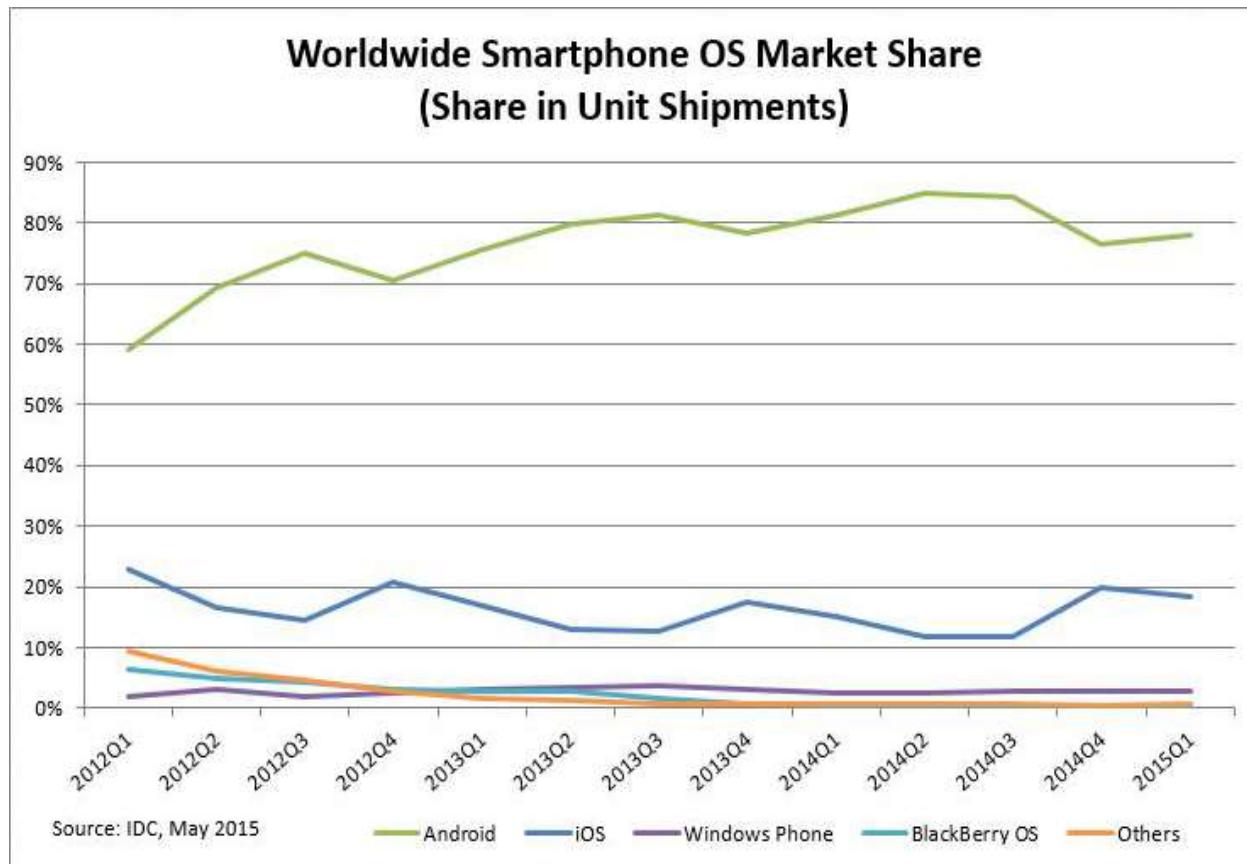
Американская компания IDC (International Data Corporation), проводящая маркетинговые исследования по всему миру, недавно обнародовала данные по самым популярным операционным системам для смартфонов в 2015 году. На 99 процентах смартфонов, проданных в первом квартале нынешнего года, была установлена одна из трех лучших на сегодня мобильных операционных систем: Android, iOS и Windows Phone. Доли рынка между ними распределились следующим образом:

3 место. Windows Phone - мобильная операционная система, разработанная американской компанией Microsoft. Она установлена на 2,7% смартфонов, проданных в 2015 году, показав небольшой рост по сравнению с первым кварталом 2014 года, когда на ее долю приходилось 2,5% проданных смартфонов. Из-за небольшой доли рынка число приложений для Windows Phone заметно уступает количеству приложений для Android и iOS, тем не менее магазин приложений Windows Phone Store может удовлетворить практически любые потребности, т.к. количество приложений в нем превышает 340 тысяч.

2 место. iOS - операционная система для смартфонов, электронных планшетов и носимых проигрывателей, разработанная американской компанией Apple. Доля iOS на рынке мобильных операционных систем (18,3%) в точности отражает долю рынка айфонов на рынке смартфонов, т.к. iOS устанавливается только на айфонах (смартфонах от компании Apple), в то время как Android и Windows Phone используются различными производителями смартфонов. Доля рынка iOS за год увеличилась: в первом квартале 2014 года она составляла 15,2%. Количество приложений для iOS в магазине приложений App Store превышает миллион.

1 место. Android - операционная система для смартфонов и множества других устройств. Изначально разрабатывалась калифорнийской компанией Android Inc., которую затем купил американский поисковый гигант Google. Доля Android на рынке ОС составляет 78%, немного снизившись по сравнению с прошлогодними 81,2%. Количество приложений для Андроид в магазине приложений Google Play превышает 1,43 млн.

График изменения долей рынка между Android, iOS, Windows Phone, Blackberry (эта операционная система три года назад имела долю 6,35, а сейчас лишь 0,3%) и прочими ОС за последние 4 года:



### Обзор некоторых мобильных операционных систем

Далее рассмотрим некоторые популярные на данный момент мобильные операционные системы более детально.

#### *Android*

Android — операционная система для смартфонов, планшетных компьютеров, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, нетбуков, смартбуков, очков Google, телевизоров и других устройств. В будущем планируется поддержка автомобилей и бытовых роботов. Основана на ядре Linux и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android Native

Development Kit позволяет портировать (но не отлаживать) библиотеки и компоненты приложений, написанные на Си и других языках.

23 сентября 2008 года официально вышла первая версия операционной системы, а также первый полноценный пакет разработчика SDK 1.0, Release 1. С момента выхода первой версии платформы произошло несколько обновлений системы. Эти обновления, как правило, касаются исправления обнаруженных ошибок и добавления новой функциональности в систему.

Android доступен для различных аппаратных платформ, таких как ARM, MIPS, x86.

Существует сообщество энтузиастов, разрабатывающее открытые варианты прошивок Android — CyanogenMod, MIUI, AOKP (Android Open Kang Project) Paranoid Android и другие. Модифицированные версии Android создаются для дополнения операционной системы новыми настройками, опциями, функциями или для улучшения качества работы устройств; удаления из Android-устройства сервисов Google для исключения возможности передачи идентификационной информации на серверы компании, например, информацию о перемещении пользователя в реальном времени; более оперативного и частого (по сравнению с производителями самих аппаратов) предоставления новых версий Android.

Некоторые интересные факты:

- Несмотря на изначальный запрет на установку программ из «непроверенных источников» (например, с карты памяти), это ограничение отключается штатными средствами в настройках аппарата, что позволяет устанавливать программы на телефоны и планшеты без интернет-подключения (например, пользователям, не имеющим Wi-Fi-точки доступа и не желающим тратить деньги на мобильный интернет, который обычно стоит дорого), а также позволяет всем желающим бесплатно писать приложения для Android и тестировать на своем аппарате. Кроме того, возможность установки программ из «непроверенных источников» способствует пиратству на платформе Android.
- Существуют альтернативные Google Play магазины приложений: Amazon Appstore, Opera Mobile Store, Yandex.Store, GetUpp!, Mobogenie, F-Droid, 1Mobile Market.
- Кодовое имя каждой версии операционной системы Android, начиная с версии 1.5, представляет собой название какого-либо десерта. Первые буквы наименований в порядке версий соответствуют буквам латинского алфавита: 1.5 Cupcake («кекс»), 1.6 Donut («пончик»), 2.0/2.1 Eclair («эклер» или «глазурь»), 2.2 Froyo (сокращение от «замороженный йогурт»), 2.3 Gingerbread («имбирный пряник»), 3.0 Honeycomb («медовые соты»), 4.0 Ice Cream Sandwich («брюкет мороженого»), 4.1/4.2/4.3 Jelly Bean («желейная конфета»), 4.4 KitKat (в честь одноименного бренда шоколадных батончиков[104]), 5.0/5.1 Lollipop («леденец на палочке»), 6.0 Marshmallow («маршмэллоу»).
- В версиях Android 2.3 и выше есть пасхальное яйцо. Чтобы его запустить, нужно зайти в «Настройки», потом зайти в «Об устройстве», найти функцию «Версия Android», и быстро нажимать на нее несколько раз (чаще всего 4 раза). На экране в версии 2.3 появится рисунок; 4.0 — один Android, который увеличивается, а затем появляется много летающих; 4.1, 4.2 и 4.3 — летающие конфеты; в 4.4 — вращающаяся буква «K», затем надпись «Android» в стиле батончика KitKat, затем — появляются плитки в стиле Windows с логотипами предыдущих версий Android; в 5.0 и 5.1 сначала появляется леденец с надписью «Lollipop», затем открывается мини-игра в стиле «Flappy Bird».

Как ни странно, но лишь смартфоны на Android позволяют действительно гибко настроить систему под собственные вкусовые предпочтения, а заодно жестко ограничить полномочия для сторонних приложений. Особенно ярко это проявляется на т.н. кастомных прошивках с предварительной процедурой открытия Root-доступа к возможностям системы. По гибкости настройки параметров, Android-смартфоны можно сравнить лишь с традиционными ПК под управлением Linux, или Windows. Изменить можно практически всё, что душе угодно. А можно оставить стандартный внешний вид системы и ничего не менять.

#### *iOS*

iOS (до 24 июня 2010 года — iPhone OS) — операционная система для смартфонов, электронных планшетов и носимых проигрывателей, разрабатываемая и выпускаемая американской компанией Apple. Была выпущена в 2007 году; первоначально — для iPhone и iPod touch, позже — для таких устройств, как iPad и Apple TV. В отличие от Windows Phone([Microsoft](#)) и Android ([Google](#)), выпускается только для устройств, производимых фирмой Apple.

В iOS используется ядро XNU, основанное на микроядре Mach и содержащее программный код, разработанный компанией Apple, а также код из ОС NeXTSTEP и FreeBSD. Ядро iOS почти идентично ядру настольной операционной системы Apple OS X. Начиная с самой первой версии, iOS работает только на планшетных компьютерах и смартфонах с процессорами архитектуры ARM.

Операционная система iPhone OS была представлена 9 января 2007 года совместно с мобильным телефоном iPhone лично Стивом Джобсом на выставке-конференции Macworld Conference & Expo и выпущена в июне того же года. Apple не предполагала отдельного названия для операционной системы, поэтому первоначальный слоган звучал так: «iPhone работает на OS X».

Другие приложения могут быть разработаны с помощью Xcode для Mac и iPhone, iPod Touch и iPad, Codea для iPad, и опубликованы в App Store — онлайн-магазине, который поставляется с самим iPhone/iPod touch/iPad, начиная с версии iPhone OS 2.0, и является крупнейшим магазином мобильных приложений.

Ключевым отличием Apple iOS от конкурентов является гораздо более упрощенная схема взаимодействия пользователя с операционной системой. Если быть более точным, с самой iOS владелец iPhone практически не работает, а лишь запускает нужные ему приложения с основного экрана, да просматривает уведомления с панелей уведомлений. Такой принцип работы можно назвать одним из самых простых и лаконичных, но только в том случае, если вы готовы внимательно изучать сами приложения и мириться с отсутствием единых хранилищ данных. К примеру, у iPhone нет общедоступной файловой системы для файлов, зато у каждого приложения есть свои виртуальные папки, в которых они хранят свою информацию. По этой причине вкладка настроек iPhone просто обязательна к подробному изучению, ведь там находятся сотни различных пунктов корректировки тоностей работы самой ОС. Добавим к этому тесную интеграцию с фирменными магазинами приложений, музыки и видео, и на выходе получается действительно своеобразный гаджет. Из плюсов высочайшая скорость работы приложений и их широчайший ассортимент. Минус: высокая стоимость iPhone при весьма средних характеристиках.

#### *Windows Phone*

Windows Phone — мобильная операционная система, разработанная Microsoft, вышла 11 октября 2010 года. 21 октября начались поставки первых устройств на базе новой платформы.

Работа над масштабным обновлением Windows Mobile могла начаться еще в 2004 под рабочим названием «Photon», но процесс двигался медленно, и в результате проект был закрыт. В 2008 году Microsoft переформировала команду Windows Mobile и начала разработку новой мобильной операционной системы. Выход продукта под названием Windows Phone был анонсирован на 2009 год, но в связи с несколькими отсрочками Microsoft решила разработать Windows Mobile 6.5 в качестве промежуточной версии. Причиной тому стала несовместимость новой операционной системы с приложениями Windows Mobile. Старший продакт-менеджер Windows Mobile Ларри Либерман также объяснил это стремлением Microsoft по-новому взглянуть на рынок мобильных телефонов, учитывая как интересы конечных пользователей, так и корпоративных сетей.

Новая операционная система Windows 10 для мобильных устройств получила название «Windows 10 Mobile», вместо Windows Phone 10.

Для устройств на Windows Phone предусмотрен интернет-магазин программ и игр Windows Phone Store (ранее Windows Phone Marketplace), доступный в 60 странах. Покупка или установка этих приложений возможна через раздел Marketplace на телефоне или через браузер. В конце июня 2012 года Microsoft официально заявила, что в магазине количество приложений перевалило за 100 тысяч. На июнь 2015 количество приложений в Windows Phone Store составляет 380 тысяч.

В целом система отличается максимально простым и лаконичным оформлением интерфейса, а также ориентирована на установку игр и приложений с единого магазина Marketplace. Но она также обладает свободным доступом к файлам и данным, а заодно позволяет разместить на домашнем экране не просто ярлыки, но и полезные информационные уведомления, или ссылки на сайты/вкладки приложений.

Однако сейчас у Windows Phone большие неприятности. Несмотря на приобретение Nokia за 7,2 млрд долларов, Microsoft не может ничего заработать на своих телефонах и объявила о списании средств на 7,6 млрд долларов. Сатья Наделла, глава Microsoft, в последнее время активно увольнял руководителей, которые перешли из финской компании, а теперь объявил о сокращении 7,8 тыс. работников — в основном телефонного подразделения.

Он хочет прекратить массовое производство линейки телефонов: по его мнению, в мобильной сфере есть вещи, которые Microsoft удаются хорошо. На них и нужно сосредоточиться — как сейчас, так и в будущем. Windows Phone по-прежнему важен для компании, но в иной роли, нежели раньше.

Microsoft упустила рынок мобильных устройств: слабые продажи Windows Phone, отсутствие интереса со стороны операторов и глобальных телефонных брендов — HTC, Samsung, Sony и LG. Если учесть еще и нежелание разработчиков создавать приложения для системы, то убедить кого-либо приобрести товар становится действительно трудно.

Гендиректор признался, что основная ошибка Microsoft, допущенная в прошлом, — это централизация ПК в списке устройств пользователя. Все это время в Редмонде акцентировали свое внимание на улучшении компьютерных продуктов, не придавая большого значения развитию рынка мобильных устройств. За что Microsoft и поплатилась, заняв за последнее время ничтожно малую часть мирового мобильного рынка — доля компании в сфере производства и реализации смартфонов составляет всего лишь 3%.

Теперь рассмотрим менее популярные мобильные операционные системы.

### *BlackBerry OS*

BlackBerry OS — операционная система с основным набором приложений для смартфонов и коммуникаторов, выпускаемых компанией Research In Motion Limited (RIM).

Особенности:

- Реализация шторки быстрых переключателей - эталон для всех ОС. Пункты в этой шторке можно настроить - выбрать только те, которые вам нужны (а выбрать есть из чего), и отсортировать их порядок. Каждая кнопка разделена на две области: нажатие на область иконки вкл./выкл. функцию, нажатие на область с названием - проваливаемся в настройки этой функции.
- BlackBerry, помимо стандартных способов блокировки, можно обезопасить с помощью "Picture Password". Это уникальный, безопасный и в то же время гениальный способ. На экране появится картинка с множеством цифр, и нужно переместить одну заранее выбранную цифру в определённое вами место экрана.
- Удобна с qwerty-клавиатурой. Первое преимущество - это сквозной поиск по всему содержимому без дополнительного вызова программы поиска: начинаете печатать, и на экран тут же начинают выводиться результаты поиска. Второе преимущество - это использование сокращений для упрощения работы - их более двухсот, и можно создавать собственные. Вот несколько типовых примеров: "C" (create) - в BlackBerry Hub создает новое письмо; "R" (replay) - ответить на сообщение; "F" (forward) - переслать сообщение.

### *Firefox OS*

Firefox OS (кодовое имя Boot to Gecko, B2G) — свободная операционная система, предназначенная для смартфонов и планшетных компьютеров. Разработку ведет Mozilla Foundation на базе свободного веб-движка Gecko.

На создание этого проекта разработчиков подтолкнуло появление движка для обработки PDF средствами HTML5 и JavaScript.

26 июля 2011 года представитель Mozilla Foundation сообщил о начале работ над операционной системой, основанной на движке Gecko, используемом в браузере Mozilla Firefox.

Особенности:

- Открытый исходный код и аппаратная платформа.
- Малая требовательность к мощности процессора.
- Быстрое выполнение несложных приложений.
- Поддержка HTML5.
- Бренд и сообщество Mozilla (по мнению представителей Telefónica).

Интересная цитата: «ОС построенная на браузере? Когда сами разработчики говорили, что труднее всего было научить браузер звонить.»

Firefox OS поддерживает только приложения, написанные на языке веб-программирования: CSS, Javascript и HTML5. Это значит, что с переносом программ на разные платформы вопросов не будет. Но создание "тяжелых" приложений и игр, которым нужны большие мощности аппаратной части

смартфонов, под большим вопросом. Ориентация Mozilla на бюджетный сегмент пока спасает компанию: покупая смартфон за \$200, никто не ждет, что на нем пойдет GTA.

### *Sailfish OS*

Sailfish OS — операционная система с закрытым исходным кодом, включающая в себя некоторые компоненты с открытым исходным кодом. Sailfish OS развивается с 2012 года финской компанией Jolla и предназначена для портативных устройств на основе Linux.

Особенности Sailfish OS:

- Sailfish OS планируется устанавливать на различные по типу устройства.
- Многие приложения для Google Android смогут работать под Sailfish OS. 16 сентября 2013 года в официальном пресс-релизе сообщается о полной аппаратной и программной совместимости с Android.
- SDK основано на QtCreator и обладает всеми его инструментами: редактор кода, дизайнер, отладчик.
- Приложения создаются на QML с использованием QtQuick 1.1 и компонентов Jolla.
- Графический интерфейс позволяет управлять приложениями из единого экрана, не требуя отдельно открывать каждое. Такой принцип получил название «covers».

### *Ubuntu touch*

Ubuntu Touch — мобильная платформа, разработанная компанией Canonical Ltd. Для смартфонов и планшетов. Ubuntu Touch призвана обеспечить удобство работы, как в UbuntuDesktop Edition.

ОС была анонсирована 2 января 2013 года и официально публично показана на выставке Consumer Electronics Show 8—11 января 2013 года.

Ubuntu Phone базируется на настольной Desktop версии Ubuntu с заменой стандартной графической оболочки на мобильную версию Unity.

9 февраля 2015 BQ выпустила BQ Aquaris E4.5 Ubuntu Edition, первый смартфон с предустановленной Ubuntu Touch. С апреля 2015 доступен только в европейском союзе.

1 июня 2015 было анонсировано, что BQ Aquaris E5 Ubuntu Edition будет выпущен 9 июня 2015.

### *Классическое противостояние — Android vs iOS*

Секрет успеха операционной системы Android кроется в ее открытости. По сути на любой современный мобильный гаджет можно установить данную операционную систему и в Интернете можно найти множество видео роликов, где представлены устройства Apple под управлением операционной системы от Google. При этом нет устройств Android, которые бы управлялись операционной системой iOS.

Успех мобильной операционной системы от Google лежит на плечах гораздо более простых технических и дешевых устройств. **Huawei**, **ZTE**, **LG** и та же **Samsung** выпускают множество более доступных смартфонов. Пусть их продажи уступают флагманам в сравнении лицом к лицу, но по крупицам десятки дешевых «гуглофонов» и формируют цифры продаж и доли рынка для всей системы.

Для того чтобы скинуть приложения или любые другие файлы на Android не нужно разбираться в программных продуктах - достаточно иметь стандартный USB-провод, карту памяти в аппарате и трезвую голову. Большинство устройств даже без драйверов определяются на современных компьютерах как простые накопители. Если рассматривать объективно ту же самую задачу на iOS, всё будет гораздо сложнее: необходимо использовать софт iTunes, который нужно настраивать и регистрировать через Интернет, что занимает значительное время у среднего пользователя.

Также большинство всевозможных приложений для iOS являются платными, Android напротив предлагает нам много бесплатных приложений.

Однако многие пользователи отмечают, что игры на iOS намного более разнообразны и развиты нежели на Android. К тому же на яблочных устройствах многие новинки выходят намного раньше. Да и количество приложений в App Store гораздо больше, нежели в Android Market.

Так как iOS - это полностью закрытая система, тяжело поддающаяся модификациям, приложения здесь, как правило, в среднем более стабильны и не могут содержать вирусов, в то время как вирусы на Android - обычная ситуация.

И, конечно же, iOS — операционная система с удобным интерфейсом. Система не только хорошо работает, но и выглядит безупречно. По крайней мере до iOS 8.

Если посмотреть на смартфоны успешных людей, нетрудно заметить, что 80% используют iPhone. Не потому, что продукция Apple считается элитной — как раз клиенты делают ее такой. Устройства под iOS крайне удобны в использовании, что и делает их такими популярными.

### [Несколько ОС в одном смартфоне — реальность?](#)

Устройства линейки Google Nexus могут похвастаться не только быстрым получением обновлений ПО, но и являются гаджетами, на которые можно одновременно установить несколько ОС. При помощи MultiROM эти устройства также могут иметь несколько прошивок, вроде CyanogenMod или MIUI.

Также стало известно, что совсем недавно корпорация Microsoft создала патент «Multi-OS boot via mobile device», который в свою очередь описывает возможность загрузки сразу нескольких операционных систем на одном смартфоне. Однако, если рассмотреть данный патент более детально, то становится ясно, что речь в этом патенте идет о более широком спектре функционала, связанного с загрузкой мобильного устройства.

Таким образом, смартфоны все больше приближаются к возможностям персонального компьютера.

## [Энергосбережение](#)

### [Введение](#)

Энергосберегающие функции ПК появились не за одну ночь — у них долгая история эволюции, в которой в свое время участвовали и бытовые компьютеры, и отдельные контроллеры внешних устройств, и мобильные компьютеры, и первые ПК. Собственно, сейчас эта технология продолжает развиваться и в новейших системах Windows, встроена в BIOS/EFI, поддерживается практически всеми устройствами ПК: от процессора до сетевой платы, от кулера до USB интерфейса.

Способы программного управления электропитанием компонентов компьютера - **ACPI (Advanced Configuration and Power Interface)** – это стандарт (спецификация), определяющий способы программного управления электропитанием компонентов компьютера с помощью встроенных средств ОС (операционной системы). Другими словами, данная технология предназначена для управления состоянием персонального компьютера и энергопотреблением его компонентов.

Кроме управления электропитанием данный стандарт позволяет выполнять конфигурацию устройств Plug and Play.

Управление электропитанием и конфигурирование устройств Plug and Play осуществляется на уровне операционной системы (предшественник спецификации ACPI стандарт APM реализован на уровне BIOS), то есть ОС практически полностью управляет энергопотреблением и конфигурированием устройств ПК.

Спецификация ACPI требует поддержки со стороны, как материнской платы, так и подключаемых устройств.

Для технологии ACPI определяют несколько состояний и подсостояний системы (компьютера): глобальные состояния системы, состояния ЦП (центрального процессора) и состояния устройств.

*Глобальных состояний системы различают четыре:*

1. **G0 (S0)** – нормальное функционирование системы;
2. **G1 (S1, S2, S3, S4)** – режимы уменьшенного энергопотребления, о которых мы поговорим чуть ниже.
3. **G2 (S5)** – программное выключение. В данном состоянии компьютер выключен, но блок питания находится под напряжением.
4. **G3** – состояние в котором питание полностью отключено от блока питания (БП).

#### [Режимы уменьшенного энергопотребления](#)

1. **S1 (Power On Suspend, POS, Doze)** – режим энергосбережения, при котором отключается монитор, винчестер, но на центральный процессор и ОЗУ (модули оперативной памяти) питание подается, снижается частота системной шины. Процессорные кэши сброшены, процессоры не выполняют инструкции, отключен генератор тактовой частоты ЦП.
2. **S2 (Standby, Standby Mode)** – режим уменьшенного энергопотребления. При данном режиме происходит отключение монитора, винчестера. От ЦП отключается напряжение питания. Останавливаются все тактовые генераторы (продолжают работать только те тактовые генераторы, которые необходимы для работы оперативной памяти). Питание подается только на системную память (в ней хранится информация о состоянии системы).
3. **S3 (Suspend to RAM, STR, Suspend)** – ждущий режим. При данном режиме энергосбережения питание подается только на оперативную память (в ней хранится информация о состоянии системы). Все другие компоненты ПК отключены.
4. **S4 (Suspend to Disk, STD, Suspend to Hard Drive, S4-Hibernation)** – глубокий сон. При данном режиме энергосбережения текущее состояние системы записывается на винчестер, после чего следует отключение питания всех компонентов ПК.

Для стандарта ACPI определяют несколько *состояний процессора*:

1. **C0** – процессор работает в номинальном режиме.
2. **C1 (Halt)** – состояние уменьшенного энергопотребления. Работа процессора приостановлена, но он может незамедлительно вернуться в рабочее состояние.
3. **C2 (Stop-Clock)** – работа процессора приостановлена. Но регистры и кэш остаются в рабочем состоянии. Процессор может немедленно приступить к обработке заданий.
4. **C3 (Sleep)** – режим сна. Процессор в спящем режиме не обновляет кэш.

Для технологии ACPI также определяют четыре **состояния устройств**:

1. **D0** – устройство работает в номинальном режиме.
2. **D1** – режим уменьшенного энергопотребления (устройство использует меньше энергии чем состояние D0).
3. **D2** – режим уменьшенного энергопотребления (устройство использует меньше энергии чем состояние D1).
4. **D3** – устройство выключено.

**Интерфейс автоматического управления конфигурацией и питанием (ACPI)** - это промышленный стандарт, который определяет функции управления питанием и другие сведения о конфигурации компьютера. Некоторые предыдущие версии BIOS не поддерживают интерфейс ACPI, поэтому компьютеры не могут успешно переходить в дополнительные режимы питания, например в ждущий или спящий режим.

Интерфейс автоматического управления конфигурацией и питанием (ACPI) пришел на смену уже устаревшего APM (Advanced Power Management).

Чтобы вникнуть в тонкости реализации технологии, нужно почитать руководства, заглянуть в BIOS, прогуляться в Панель управления («Электропитание»). Последний пункт выполнить проще всего, установив галочки во всех рекомендуемых окошках. И тогда... Собственно, а что будет тогда? ПК – не ноутбук, которому нужно беречь аккумулятор ради многочасовой работы. Дело в том, что именно для ПК энергосбережение практически не несет никаких положительных факторов. Не берем в расчет режим гибернации – это несколько другой процесс, более важный. Скорее, при некотором размышлении, можно сообразить, что «зеленые функции» ПК несут больше вреда. Не пытайтесь возмущаться, это так. Представьте себе лампочку в ванной или коридоре, которую включают и выключают по сто раз за день. Именно при очередном включении она и перегорает, отработав свой ресурс и не выдержав множество циклов максимальной нагрузки – спираль-то светится далеко не от радости, а от мощного тока, который запускает кучу всяких процессов – физических (износ, старение), химических (взаимодействие с инертными газами, но далеко не чистыми) и т.д. На долговечность работы лампы влияют влажность, температура, пыль и страшно подумать, что еще. И после этого нам говорят, что периодическое выключение подсистем ПК оказывается на нем положительным образом. Возможно, если программы будут выключать винчестер каждые 20 минут, то при этом можно будет сэкономить пару десятков ватт, но первое же включение обнулит экономию – запуск систем всегда происходит на всплеске используемой мощности. Но страшно даже не это: каждое включение, как и в лампе накаливания, разогревает микросхемы, проводники, двигатели и т.д. По закону физики нагретое тело расширяется, а потом сужается, и получается волна микродвижения в платах, причем неоднородная. Со временем это приведет к обязательному разрыву связей в схеме – если долго расшатывать здоровый зуб, то даже он выпадет. Итак, эфемерная мода на «зеленое» чуть-чуть экономит деньги, которые придется

потратить на новый компьютер гораздо раньше планируемого срока. Причем – более мощный. Такое вот оно – энергосбережение.

### [Энергосбережение мобильных устройств](#)

На сегодняшний день, мобильные телефоны, как правило, имеют большие экраны, и пользователи получают преимущество в этом, используя многочисленные услуги, такие как GPS, Wi-Fi и другие приложения, которые соответственно быстрее разряжают телефон.

Все, конечно же, заметили, что во время телефонных переговоров или использования Интернета, телефон разряжается гораздо быстрее. Или это всего лишь вопрос дополнительного энергопотребления динамиком, микрофоном или экраном? Терминал в зависимости от потребностей и текущего использования может быть в различных режимах и состояниях.

### [Автономность и энергосбережение операционной системы iOS](#)

Дебютировавшая относительно недавно операционная система iOS 9 принесла с собой множество нововведений. Особую роль среди них занимает режим пониженного энергопотребления. Технология способна добавить три часа к стандартному времени использования iPhone. Сама «операционка» при этом обеспечивает увеличение времени автономной работы смартфонов на один час.

Активация режима энергосбережения приводит к отключению ряда функций мобильного устройства, работу которых пользователь зачастую не замечает. Так, ограничиваются визуальные эффекты iOS, приостанавливается автоматическое обновление приложений. Кроме того, активируется «механизм троттлинга», который снижает процессорную мощность, что также положительно сказывается на автономности. Это позволяет оптимизировать энергопотребление и продлить время работы гаджета.

### [Как работает режим энергосбережения](#)

При включении режима пониженного потребления iPhone отключает некоторые функции операционной системы до тех пор, пока пользователь не зарядит устройство. Среди параметров, на которые оказывает влияние энергосбережение: проверка почты; фоновое обновление приложений; автоматические загрузки; некоторые визуальные эффекты; время автоблокировки – 30 секунд; производительность процессора снижается на 40%.

Как было отмечено выше, все это направлено на достижение максимальной автономности iPhone. Режим пониженного потребления способен обеспечить увеличение времени работы до трех часов.

### [Как использовать режим энергосбережения](#)

Стоит иметь в виду, что режим пониженного энергопотребления не включается автоматически. Для этого пользователь должен перейти в настройки гаджета, пролистать до раздела «Аккумулятор», зайти в него и перевести соответствующий тумблер в положение «Вкл». iPhone также предложит активировать эту функцию при снижении уровня заряда до критических 20% и 10%. Энергосбережение автоматически отключится во время зарядки iPhone при достижении 80%.

Режим работы энергосбережения определяется двумя составляющими. Во-первых, зеленый значок батареи в строке состояния окрашивается в желтый цвет. Во-вторых, даже если у пользователя отключено отображение заряда в процентах, оно включается автоматически для улучшения наглядности.

### *Способы продлить автономность в iOS 9*

В iOS 9 есть и другие способы увеличить время автономной работы аккумулятора. Обновленный раздел «Аккумулятор» позволяет следить за тем, какие приложения расходовали больше всего энергии за последние 24 часа и 5 дней. Также в Центре уведомлений доступен виджет, позволяющий следить за сопряженными устройствами.

Помимо нового энергосберегающего режима, в iOS 9 доступны и другие технологии. Была переработана система взаимодействия с приложениями для большей эффективности и меньшего энергопотребления. Теперь, если положить смартфон экраном вниз, уведомления не будут отображаться и тем самым можно снизить расход батареи.

Ресурс AppleInsider.com отмечает:

*Благодаря окружающему освещению и сенсорам приближения, ваш iPhone знает о том, что [в данный момент] он лежит на столе лицевой стороной вниз и способен предотвратить включение экрана в таком положении даже в том случае, когда вам пришло уведомление.*

Данный метод является дополнением к новому режиму Low Power операционной системы iOS 9. Ожидается, что он позволит продлить работу устройства от запаса энергии в его батарее еще на три часа за счет ограничения работающих в фоновом режиме процессов — уведомлений и тому подобного.

Многие пользователи уже обратили внимание на то, что использование сенсоров приближения — знакомый им подход. Менее чем за две недели до презентации Apple Google представила подобный подход к сокращению расхода энергии в Android M.

Тогда Google сообщила о новой функции, позволяющей Android M «задремать». Операционная система будет снижать свою активность на основании данных сенсора о том, что телефон в настоящее время не используется, поскольку, предположительно, лежит на столе, прикроватной тумбочке или в кармане.

В этом случае Android M переходит в режим экономии энергии и сокращает фоновую активность девайса. Приоритетные уведомления и входящие звонки все равно будут поступать, но остальные процессы перейдут в замедленное или даже в спящее состояние.

### *Автономность и энергосбережение операционной системы Android*

Новые функции Android 5 Lollipop предназначены для улучшения работы аккумулятора и повышения уровня безопасности. Так, в режиме энергосбережения устройство будет работать гораздо дольше. Удобно вручную активировать режим при полном заряде батареи или изменить параметры так, чтобы при уровне заряда в 15% этот режим активировался сам. В первом случае мой XPERIA Z2 проработал примерно на 3.5 часа дольше, во втором — почти на 30 минут. Хорошие результаты ОС от Google продемонстрировала благодаря ограничению фоновых процессов, производительности процессора и яркости дисплея. Кроме того, Android информирует пользователя о том, сколько времени осталось до полного разряда аккумулятора, или — при подключении к сети питания — о том, сколько осталось до полной зарядки. Автоматически активирующегося режима энергосбережения у iOS 8, увы, нет, зато появилось новое меню использования батареи, где удобно отслеживать активность приложений и отключать наиболее ресурсоемкие из них. При желании можно отдельно ограничить наиболее энергозатратные

фоновые процессы. Однако в том, что касается настройки энергосбережения, особенно следует отметить ОС Windows Phone 8.1, предлагающую наиболее обширные возможности конфигурирования. Энергоэффективность за счет использования черного фона в большинстве меню в Windows Phone, если вариация на базе системы Microsoft позволяет говорить по смартфону 22 часа, то с Android можно рассчитывать максимум на 20 часов. Есть разница и в режиме ожидания, 528 часов против 496 часов.

### Автономность

Google снабдила Android 5 полезным режимом экономии электроэнергии, слегка уменьшающим производительность и яркость дисплея вашего гаджета.

В системном меню мониторинга работы аккумулятора операционная система iOS отображает статистику энергопотребления отдельных приложений и позволяет ограничить различные фоновые процессы. В этом аспекте система Windows Phone в более продвинутая: здесь предлагаются не только автоматические, но и ручные настройки.

Galaxy S5. В смартфоне присутствует целых два режима энергосбережения. Без них, при нормальном использовании устройства я получил полтора дня работы от батареи.

Первый режим, который позволит сэкономить заряд аккумулятора – Энергосбережение. Он ограничивает частоту процессора, фоновые процессы, а также переводит дисплей в монохромный режим, если нужно. Этот мод лишь немного улучшает автономность устройства. Но с ч/б экраном смартфон выглядит как ретро-девайс – идеальное решение для хипсетров.

Гораздо интереснее второй режим энергосбережения.

Максимальное энергосбережение нужно, когда заряд аккумулятора почти на исходе, а зарядить его нет возможности. Как и в предыдущем случае, смартфон переключается в монохромный режим, отключаются ненужные сетевые модули, а интерфейс становится максимально простым. В этом режиме доступны звонки, отправка СМС, веб-браузер, а также одно из шести приложений. Производитель немного ограничил настройки, но при необходимости можно включить Wi-Fi, Bluetooth и GPS, настроить яркость.

### *Технологии энергосбережения Android*

Можно долго рассуждать на тему преимуществ Android – гибкость платформы, тысячи самых разных моделей устройств на любой вкус, обилие первоклассного и зачастую бесплатного софта. Но на протяжении шести лет жизни системы с зеленым роботом существует и один неизменный недостаток – короткое время жизни на одном заряде батареи. Почему же в этой жизненно важной области не наблюдается практически никакого прогресса? А может, он все-таки есть, просто мы его не видим?

Характеристики смартфонов прошлых лет сейчас вызывают улыбку – ну как можно доверять устройству с батареей 1390 мАч (Motorola XT720) или, того хуже, 1230 мАч (HTC Desire HD)? Именно маленькая емкость батарей и высокое энергопотребление первых версий Android надолго закрепили за платформой имидж устройств для тех, кто постоянно озирается в поисках розетки.

Год от года технологии внутренней компоновки устройств неуклонно взрослели – уменьшались техпроцессы чипов, уплотнялась схемотехника печатных плат, SD и SIM обрастили “nano-” и “micro-

“ приставками в названиях, попутно экономя место. Все это вело к тому, что в смартфонах освобождалось место под более крупные батареи.

На радость покупателям планка емкости год от года росла все выше – пропустив все промежуточные этапы, сейчас можно говорить о стандарте в 2600-3000 мАч, ниже которого большинство А-брендов старается не опускаться. Получается, что емкость батарей флагманских устройств с момента рождения платформы выросла более чем в два раза! Но выросло ли в два раза время работы устройства? Отнюдь.

Большинство производителей решили, что разумнее пустить выжатый запас энергии на кормление дополнительных пикселей экрана, анимированных “красивостей” фирменных оболочек и охочих до батареи новых стандартов мобильной связи(4G). Один день работы от аккумулятора по-прежнему остается стандартом де-факто для всех производителей. Ночевка андроид-смартфона на зарядке – его естественное состояние.

Конечно, существуют исключения из правил – смартфоны с гигантскими батареями на 4000 мАч и больше. Такой аппарат запросто может продержаться два дня и даже больше, в зависимости от того, как его нагружает владелец. Цена, которую приходится платить за большую емкость, – толщина устройства. Все аппараты с такими аккумуляторами имеют толщину не менее 12 мм, что по нынешним временам высокого спроса на тонкость чрезвычайно много. И вес у них совсем не маленький.

Но говорить, что индустрия не движется в сторону более элегантного решения задачи, нельзя. Сразу несколько топовых вендоров разработали собственные схемы сохранения драгоценной энергии. Одним из первенцев философии тотальной экономии стала технология Stamina во флагманских аппаратах Sony. Позднее режимы экстремального энергосбережения появились у Samsung и HTC.

Не отстают от производителей и разработчики самой платформы. Создатели Android называют кардинальное улучшение тех или иных слабых мест системы проектами. Первым был Project Butter, подаривший андроидам долгожданную плавность интерфейсов. Вторым – Project Svelte, задачей которого было добиться уверенной работы “шоколадной” версии даже с 512 МБ оперативной памяти и победить таким образом царящий беспорядок самых разных итераций Android на рынке. Теперь настал черед хирургического вмешательства в главную ахиллесову пяту – энергопотребление системы.

Project Volta даст устройствам на базе Android L (5.0) массу новых технологий сбережения электричества. Разработчикам приложений будут даны новые инструменты, которые помогут понять, как и почему программа влияет на время жизни батареи (Battery Historian). Появятся и дополнительные средства контроля, гарантирующие, что определенные процессы не запускаются, когда заряд аккумулятора на исходе (Job Scheduler API).

К примеру, весьма охочее до электроэнергии обновление приложений в Play Store будет работать только при наличии определенного уровня заряда или подключении к розетке. Также система не разбудит девайс для выполнения связанного с подключением к сети процесса, если не найдет эту самую сеть.

	Sony	Samsung	HTC
Название	Stamina	Режим максимального энергосбережения / Ultra Power Saving mode	Критический режим энергосбережения / Extreme Power Saving mode
Особенности	Уменьшение яркости экрана; запрет использования фоновой передачи данных приложениями; отключение Wi-Fi и Bluetooth; создание списка приложений с доступом к сети передачи данных	Режим работы дисплея "оттенки серого"; запрет использования фоновой передачи данных приложениями; рассылка push-уведомлений с централизованного сервера; ограничение числа доступных приложений; отключение Wi-Fi и Bluetooth	Уменьшение яркости экрана; запрет использования фоновой передачи данных приложениями; отключение виброотдачи; снижение рабочей частоты процессора; ограничение числа доступных приложений; отключение Wi-Fi и Bluetooth; выключение шагомера

Кроме того, в Android L появится собственный режим экстремального энергосбережения. При достижении выбранного пользователем (от 20% до 5%) остатка запаса энергии, он будет снижать до минимума яркость дисплея, частоту процессора, отключать анимацию оболочки и передачу данных через мобильную сеть и Wi-Fi, а также другие не нужные работающим приложениям интерфейсы.

Выполненные нашими зарубежными коллегами тесты показывают, что уже сейчас, на стадии бета-тестирования (Android L Developer Preview), Project Volta дает заметные невооруженным глазом результаты. Лобовое сравнение Nexus 5 на Kit Kat и Android L по фиксированному сценарию дает 36% преимущества во времени работы последнему. И все это – с отключенным режимом экстремального энергосбережения и неоптимизированными сторонними приложениями.

Как видите, о проблеме энергосбережения Android никто не забыл – ей озабочены и разработчики платформы, и производители смартфонов. Рано или поздно, но мы снова вернемся к привычкам прошедшей “кнопочной” эпохи – заряжать телефон раз в несколько дней и спокойно выходить из дома, даже если индикатор батареи показывает жалкую дюжину процентов заряда.

А как вы относитесь к тому, что о заряде батареи необходимо беспокоиться каждый вечер? В котором часу ваш смартфон обычно превращается в безжизненный кирпич? Чем вы доводите его до такого состояния?

Современные смартфоны на базе Android достаточно требовательны в отношении электроэнергии. Очевидными пожирателями аккумуляторов являются игры, мультимедиа, GPS. Однако не для всех пользователей является очевидным то, что постоянный доступ в интернет посредством сетей

передачи данных 3G в большинстве случаев является тоже одной из причин быстрого разряда батареи.

### *Монстр 3G*

Смартфоны нового поколения, такие как трубки на базе ОС Android, являются мощными и производительными вычислительными устройствами. Например, даже бюджетные аппараты на базе ОС Android обладают 500-600 мегагерцовным процессором и могут воспроизводить трёхмерную графику в компьютерных играх.

Такая производительность требует и большого расхода электроэнергии батареи. Но расход заряда аккумулятора в случае ресурсоёмких мобильных приложений - игр и мультимедиа - это слишком очевидная проблема.

Другое дело, когда на смартфоне пользователь не запускает ни игр, ни мультимедийных программ, ни GPS, а заряд аккумулятора всё равно быстро заканчивается. На форумах можно прочитать много претензий к смартфонам на базе Android, которым в пример ставятся "айфоны", что работают без подзарядки подольше собратьев-гуглофонов.

Но не стоит спешить с обвинениями самой системы "Андроид" в излишней прожорливости в сравнении с iOS. Как любит говорить известный автогонщик Фернандо Алонсо, когда инженерам соперников удаётся сделать более быструю машину, чем у него: "У соперников, конечно, работают гении, но не волшебники". В Apple тоже работают хоть и гении, но не волшебники.

В силу того, что Apple выпускает одну единственную модель, ей, конечно же, удается уделить энергосбережению "айфона" большое внимание, вплоть до разных хитростей в работе аппаратной и программной частей. Например, iOS не может похвастать честной многозадачностью в отличие от "Андроидов".

Однако реальная причина прожорливости гуглофонов лежит не в их операционных системах. Главный пожиратель аккумулятора смартфона - это интернет, особенно используемый ныне для мобильного доступа к Сети стандарт передачи данных 3G.

Главный пожиратель аккумулятора смартфона - это интернет, особенно используемый ныне для мобильного доступа к Сети стандарт передачи данных 3G.

Если быть педантом, то следует сказать, что 3G - на самом деле маркетинговый термин, объединяющий несколько стандартов беспроводной мобильной передачи данных. Но я не буду тут приводить спецификации и технические характеристики этих стандартов - любители могут это узнать из Википедии. А для неискушённого пользователя важно знать, что 3G - это переходный стандарт к мобильным сетям следующего поколения 4G. Стандарт, например, в случае использования технологии HSPA+ может обеспечивать передачу данных со скоростью до 70 Мбит/с. Для сравнения: более старые стандарты второго поколения 2G обеспечивают скорость до 474 кбит/с.

Естественно, высокая скорость передачи данных в 3G обеспечивается более высокой мощностью сигнала радиосвязи, что, само собой, значительно быстрее разряжает батарею мобильного устройства. Кстати, эффективное использование электроэнергии при использовании 3G зависит и от настроек передающих станций мобильного оператора. Поэтому, вспоминая слова Фернандо

Алонсо, можно сказать, что для эффективного энергосбережения смартфонов гении нужны не только среди производителей мобильной техники, но и желательно среди ОПСОСов.

Эффективное использование электроэнергии при использовании 3G зависит и от настроек передающих станций мобильного оператора.

### [Алгоритмы](#)

Кроме изменения частот, сборщики зачастую добавляют в ядро новые алгоритмы управления энергосбережением (автоматическим управлением частотой процессора), которые, по их мнению, могут показать лучшие результаты в сравнении со стандартными. Почти все из них базируются на используемом по умолчанию в новых версиях Android алгоритме Interactive, суть которого заключается в том, чтобы резко поднять частоту процессора до максимальной в случае повышения нагрузки, а затем постепенно снижать до минимальной. Он пришел на смену используемому раньше алгоритму OnDemand, который плавно регулировал частоту в обе стороны соразмерно нагрузке, и позволяет сделать систему более отзывчивой. Сборщики альтернативных ядер предлагают на замену Interactive следующие алгоритмы:

- SmartAssV2 — переосмысление алгоритма Interactive с фокусом на сохранение батареи. Основное отличие в том, чтобы не дергать процессор на высокие частоты в случае кратковременных всплесков нагрузки, для которых хватит и низкой производительности процессора. По умолчанию используется в ядре Matrix.

Interactive/ —тюнингованный алгоритм Interactive, главная особенность которого в залочке процессора на минимальной указанной пользователем частоте и обесточивании второго ядра процессора во время отключения экрана. По умолчанию используется в Leankernel. LulzactiveV2 — по сути, изобретенный заново OnDemand. Когда нагрузка на процессор превышает указанную (по умолчанию 60%), алгоритм поднимает частоту на определенное число делений (по умолчанию 1), при понижении нагрузки —опускает. Особый интерес представляет тем, что позволяет самостоятельно задавать параметры работы, поэтому подходит для прожженных гиков.

Вообще, сборщики ядер очень любят придумывать новые алгоритмы энергосбережения по причине простоты их реализации, поэтому можно найти еще с десяток других. Большинство из них полный шлак, и при выборе планировщика следует руководствоваться правилом: либо один из трех описанных выше, либо стандартный Interactive, который, кстати, очень неплох. Сделать выбор можно с помощью все той же Trickster MOD.

### [Андервольтинг](#)

Теперь поговорим о тяжелой артиллерией. Ни для кого не секрет, что один из самых прожорливых компонентов смартфона — это процессор. Его энергопотребление может быть даже больше потребления экрана (а точнее, его подсветки), и все потому, что он работает на очень высоких частотах, которые требуют подачи высоких напряжений. Поначалу может показаться, что сохранить жизнь от батареи в этом случае можно, просто понизив максимальную частоту работы процессора и отключив «лишние» ядра. Однако, скорее всего, это ни к чему не приведет: несмотря на пониженное потребление энергии, процессор будет исполнять код дольше, и в конечном счете энергопотребление может даже возрасти.

Вместо этого следует провести операцию андервольтинга, то есть просто понизить максимальное подаваемое напряжение для всех возможных частот. Для этого необходимо установить кастомное ядро с поддержкой данной функции. О том, как это сделать и какое ядро выбрать, я во всех

подробностях рассказывал в одном из предыдущих номеров журнала, поэтому не буду повторяться, а просто скажу, что если у тебя один из нексусов, то достаточно установить franco. Kernel updater и с его помощью скачать и установить ядро. Все происходит в автоматическом режиме.

Далее устанавливаем платную версию Trickster MOD (бесплатная не сохраняет настройки напряжений) или CPU Adjuster; для ядер franco также подойдет платный franco. Kernel updater. Переходим на страницу регулировки вольтажа (в Trickster MOD нужные настройки находятся внизу четвертой страницы) и начинаем аккуратно убавлять по 25 мВ для каждой из возможных частот процессора. После убавления сворачиваем приложение и некоторое время тестируем смартфон, запуская тяжелые приложения, затем снова убавляем и снова тестируем.

В 90% случаев процессор без всяких последствий выдержит понижение на 100 мВ, а это даст нам дополнительный час-два в режиме активного использования. Если тебе повезет, то процессор сможет выдержать и -150, а в особо счастливых случаях даже -200, все зависит от партии процессора и конкретного экземпляра. Слишком сильное занижение напряжения приведет к перезагрузке, после которой достаточно будет поднять напряжение на 25 мВ и сохранить значение в дефолтовом профиле (в Trickster MOD это кнопка «Профиль» сразу над значениями).

Смартфон с AMOLED-экраном будет работать дольше, если использовать приложения с черным фоном. Чтобы сделать системные приложения темными, можно использовать прошивку AOKP или один из модулей Xposed.

Зачастую механизм автоматической регулировки яркости экрана выставляет слишком высокие значения. Если управлять яркостью вручную, можно продлить жизнь смартфона еще на пару часов.

Продвинутые функции фирменных прошивок некоторых производителей смартфонов, такие как управление жестами, голосовое управление или автоматическое включение экрана, приводят к чрезмерному расходу заряда аккумулятора. По возможности их следует отключить.

С пособы продлить жизнь заряда аккумулятора от одного заряда:

1. Первое что я посоветую, это отключать Wi-Fi, когда вы им не пользуетесь. Так как даже в момент его неиспользования постоянная поддержка подключения потребляет не малую долю заряда аккумулятора до 330 мА/ч
2. Также по возможности отключать сети 3G. Почему? Всё просто, сети третьего поколения обладают много канальностью, то есть если в сетях 2G во время разговора интернет сервисы отключаются (что позволяет экономить заряд батареи), то в сетях 3G в то время, когда вы разговариваете по телефону, активность интернет не останавливается и исходя из этого идёт гораздо большее потребление энергии. А также, когда к примеру, вы находитесь вне покрытия сетей 3G, а режим только 2G не включён телефон будет постоянно искать сети 3G сканируя сети (что тоже потребляет не мало питания)
3. Продолжу по поводу мобильных сетей. Я думаю у всех бывает такое, когда вы попадаете в зону плохо покрытия мобильной связи и ваш телефон начинает лихорадочно искать доступные сети либо пытаться поддержать связь с уже подключённой БС. Так вот в этом "Режиме поиска" ваш телефон потребляет много энергии по сравнению с обычной работой в сетях. Что делать в данной ситуации решать вам, но как вариант (если дорог заряд батареи) включить автономный режим (режим в самолёте)

4. По поводу GPS много писать не буду, просто напишу, что в режиме активной работы, полного заряда батареи хватает на очень малое время от 3-х до 5-ти часов максимум. Исходя из этого в то время, когда вы им не пользуетесь советую его отключать.
5. Акселерометр. Как известно датчик движения в нашем телефоне постоянно бодрствует (если он есть конечно), ну а постоянно работающий модуль - постоянно потребляет питание, питание не маленькое порядка 4 мА. Как вариант для продления жизни заряда аккумулятора отключить автоповорот дисплея.
6. Процессор. Спорный вопрос. По заявлению производителей все процессоры в новых моделях телефонов динамически изменяют свою частоту в зависимости от требований системы, чем обеспечивают экономию энергопотребления. Но многие с этим не согласны и я в том числе. Далеко не всегда процессор снижает частоту своевременно, чем повышает расход заряда аккумулятора. Я вам порекомендую вручную регулировать использование процессора! Для этого есть замечательная программа SetCPU, с помощью которой даже можно создавать различные режимы работы процессора, чем снизить потребление питания.
7. Последнее что хочу посоветовать, отключайте режим передачи данных, когда он вам не нужен, я к примеру, выключаю на ночь все варианты передачи данных в сети (Wi-Fi, мобильные сети), чем добиваюсь минимального разряда аппарата во время моего сна. И ещё один нюанс, для владельцев аппаратов AMOLED дисплеями советую ставить в качестве фона тёмные обои, они позволяют экономить энергию вашего аккумулятора.

## Технологии

Пройдя по списку самых энергозатратных приложений с помощью Wakelock Detector, легко понять, что основные причины пробуждения устройства — это разные виды синхронизации и регулярное обновление информации о местоположении. Это значит, что, отключив эти функции полностью, можно избавиться от большинства случаев пробуждения и серьезно сэкономить батарею.

Я бы рекомендовал сперва зайти в настройки Google-аккаунта («Настройки -> Аккаунты -> Google -> user@gmail.com») и аккаунтов других приложений и отключить все ненужные виды синхронизации. Мне, например, не нужны синхронизация календаря, стандартного браузера, контактов Google+ и «данных приложений», так что я могу спокойно избавиться от них. Так же следует поступить и со всеми остальными зарегистрированными на смартфоне аккаунтами, а в настройках сторонних приложений отключить автоматическую синхронизацию (тебе действительно нужна автосинхронизация Twitter и RSS?). Редко используемые приложения лучше удалить вовсе.

Последние версии Android не позволяют отключить определение местоположения полностью, но зато могут использовать очень консервативный и почти не влияющий на жизнь смартфона режим под названием (сюрприз!) «Экономия заряда батареи», который обновляет информацию только тогда, когда происходит подключение к Wi-Fi-сети или переход на другую сотовую вышку.

Если приложение садит аккумулятор, а удалять его нельзя и в настройках нет опций синхронизации или автообновления, то его можно просто заморозить. Делается это с помощью великолепного приложения под названием Greenify. Оно подавляет возможность приложения просыпаться самостоятельно и заставляет его работать только тогда, когда ты сам этого захочешь. Пользоваться очень просто. Запускаем Greenify, нажимаем на кнопку + в левом нижнем углу и видим, какие

приложения дольше всего работают в фоне. На скриншоте видно, что наиболее прожорливые — это OTransfer Target, используемый для удаленного включения переадресации (оно вообще постоянно бодрствует), а также Beautiful Widgets и Carbon, которые периодически просыпаются для разного рода синхронизаций. OTransfer Target я ставил для теста, так что могу спокойно его удалить (оно, кстати, также есть в числе «лидеров» в Wakelock Detector). Beautiful Widgets просыпается для обновления виджета на рабочем столе, поэтому его я оставлю в покое. А вот Carbon, занявший пятое место по версии Wakelock Detector, можно заморозить. Для этого достаточно просто тапнуть по имени и нажать галочку в правом верхнем углу.

#### *Пять вредных советов по энергосбережению*

1. Убийство фоновых процессов с помощью таск-киллера. Одна из самых глупых идей из всех, что только могут прийти в голову. Следует просто запомнить: фоновые процессы не потребляют энергию, обычно ее потребляют запущенные ими сервисные службы, которые либо вообще не убиваются таск-киллерами, либо имеют способность к самовоскрешению. А вот убийство самих фоновых приложений приводит к необходимости их повторного запуска, на что энергия таки тратится.
2. Отключение Wi-Fi дома. В энергосберегающем режиме (когда смартфон спит) модуль Wi-Fi потребляет очень мало энергии, настолько мало, что на включение и выключение модуля зачастую расходуется гораздо больше. Имеет смысл разве что на планшете, который берешь в руки два-три раза в день, чтобы почитать новости или книгу.
3. Автоматическое переключение между 2G и 3G. Аналогичная история. При скачках между типами сетей происходит повторный поиск вышек и повторное же соединение, а в это время радиомодуль работает на полную мощность. Приложения, автоматически включающие 2G во время сна, почти всегда приводят к еще большему расходу энергии.
4. Приложения с названиями вроде Ultimate Battery Saver. В 99% (если не в ста) случаев это либо плацебо, либо все тот же таск-киллер, снабженный механизмом, который отключает разные компоненты смартфона при достижении определенного уровня заряда. Сначала происходит перевод на 2G и отключение GPS, затем отключается интернет, а под самый конец телефон переводится в режим полета. Проблема здесь в том, что описанный механизм работы скорее мешает и все это удобнее сделать самому в нужное время.
5. Калибровка батареи с помощью рекавери. С давних пор существует миф о том, что удаление файла /data/system/batterystats.bin с помощью CWM приводит кбросу настроек батареи, так что она начинает показывать «более правильный» уровень заряда. Миф настолько въелся в умы, что некоторые индивидуумы начали делать «калибровку» ежедневно, заявляя, что так можно продлить жизнь батареи и даже повысить ее емкость. На самом деле файл нужен для сохранения статистики использования энергии (той самой инфы из «Настройки -> Батарея») между перезагрузками и ни на что не влияет.

#### *Производительность в режиме энергосбережения*

Чтобы вникнуть в тонкости реализации технологии, нужно почитать руководства, заглянуть в BIOS, прогуляться в Панель управления («Электропитание»). Последний пункт выполнить проще всего, установив галочки во всех рекомендуемых окошках. И тогда... Собственно, а что будет тогда? ПК — не ноутбук, которому нужно беречь аккумулятор ради многочасовой работы. Дело в том, что именно для ПК энергосбережение практически не несет никаких положительных факторов. Не

берем в расчет режим гибернации – это несколько другой процесс, более важный. Скорее, при некотором размышлении, можно сообразить, что «зеленые функции» ПК несут больше вреда.

Не пытайтесь возмущаться, это так. Представьте себе лампочку в ванной или коридоре, которую включают и выключают по сто раз за день. Именно при очередном включении она и перегорает, отработав свой ресурс и не выдержав множество циклов максимальной нагрузки – спираль-то светится далеко не от радости, а от мощного тока, который запускает кучу всяких процессов – физических (износ, старение), химических (взаимодействие с инертными газами, но далеко не чистыми) и т.д. На долговечность работы лампы влияют влажность, температура, пыль и страшно подумать, что еще. И после этого нам говорят, что периодическое выключение подсистем ПК оказывается на нем положительным образом. Возможно, если программы будут выключать винчестер каждые 20 минут, то при этом можно будет сэкономить пару десятков ватт, но первое же включение обнулит экономию – запуск систем всегда происходит на всплеске используемой мощности. Но страшно даже не это: каждое включение, как и в лампе накаливания, разогревает микросхемы, проводники, двигатели и т.д. По закону физики нагретое тело расширяется, а потом сужается, и получается волна микродвижения в платах, причем неоднородная. Со временем это приведет к обязательному разрыву связей в схеме – если долго расшатывать здоровый зуб, то даже он выпадет. Итак, эфемерная мода на «зеленое» чуть-чуть экономит деньги, которые придется потратить на новый компьютер гораздо раньше планируемого срока. Причем – более мощный. Такое вот оно – энергосбережение.

#### *Беспроводной зарядные устройства*

Несмотря на то что первые устройства с беспроводной зарядкой появились около пяти лет назад, ввиду своей дороговизны и специфики их рынок пока растет крайне медленно

Беспроводные зарядные устройства набирают популярность: по словам экспертов, этот рынок «взорвется» уже в этом году и с выпуском Apple и Samsung соответствующих устройств достигнет объема \$2 млрд. Решения по бесконтактной зарядке также активно развиваются и в автомобильной отрасли.

Когда Nokia выпустила свой первый смартфон с беспроводной зарядкой, к этому оказались не очень готовы покупатели — им было привычнее заряжать смартфон, используя стандартное устройство. В свою очередь, производители аксессуаров также не могли себе позволить обслуживать единственное устройство, пусть даже и инновационное.

#### *Нестандартные способы зарядки*

Заводная рукоятка. Возможно, Вам понравится чувство, что Вы действительно заработали заряд «яблочного» гаджета, работая мускулами? Устройство Eton's Boost Turbine 2000 создано как раз для таких пользователей. Для тех, кто не в курсе, компания Eton – поставщик ручных вспомогательных устройств (фонариков, радиоприемников и прочих товаров, ориентированных на любителей активного отдыха и выживания в экстремальных условиях).

Отчаянные потенциальные пользователи смартфонов получат около 30 секунд в режиме разговора, проворачивая рукоятку Boost Turbine 2000 в течение одной минуты, с возможностью достичь полной зарядки. При емкости встроенного аккумулятора в 2000 мА\*ч вы не сможете эффективно зарядить ноутбук, но айпод, айпад или телефон будут работать просто отлично.

Стоимость покупки – 94 доллара. Дороговато, но разве это так важно, когда оказываешься в трудном положении?

Ветровая турбина. Думаете, статус «iFan» носят только те пользователи, которые перед выпуском новой модели готовы целую ночь дежурить возле магазинов Apple? Вовсе нет!

Спроектированная Тьеердом Веенховеном, дизайнером 3D интерьеров и экстерьеров, и введенная в действие с помощью модифицированного компьютера, ветровая турбина iFan позволяет использовать для зарядки айпода, айпада и других гаджетов «зеленую» энергию ветра.

Чтобы зарядить девайсы в общей сложности нужно шесть часов. Но энтузиасты (с хорошей физической подготовкой) могут ускорить процесс прикрепив устройство к велосипеду. Крутя педали, можно быстрее «заполнить» аккумулятор.

Зарядка через огневую поверхность. Угадайте, что заряжает Ваш айпод, айпад, когда Вы подогреваете суп? BioLite's CampStove – портативная плита для эксплуатации на свежем воздухе. Используя тепло от костра, она производит электричество посредством термоэлектрического генератора, который затем запускает вентилятор для создания потока воздуха, улучшая сгорание.

Избыток электроэнергии направляется в USB-порт для зарядки электронных гаджетов. Даже если это звучит немного сложно, это совсем не так. Просто, подключите Apple девайс и наслаждайтесь зефиром на огне, а всего через два часа ваш разряженный «яблочный» любимец вернется к нормальной деятельности.

Возможно, это не самая полезная вещь из представленных на рынке, но стоит отдать должное ее создателям – они вышли за привычные рамки мышления.

Солнце. Оказывается, что солнце ежесуточно вырабатывает целых 400 триллионов триллионов ватт – примерно такое количество энергии требуется, чтобы обеспечить 500 000 лет существования нашей текущей цивилизации. Да, это включает и подзарядку гаджетов.

Конечно, освоение солнечных лучей – сложная задача. Именно поэтому, нужно быть благодарными за устройство Solio BOLT Solar Charger + Battery Pack от компании Better Energy Systems («Лучшие энергетические системы»). Солнечные зарядки помогут зарядить айфон, айпод или айпад. Хоть максимальное напряжение всего 5 вольт, существует несколько способов зарядить девайсы. В среднем для полной зарядки батареи телефона потребуется до трех часов времени.

Кусочек фрукта. Что может быть гармоничней, чем подзарядка iPhone настоящим яблоком? При условии наличия необходимых инструментов, некоторого количества времени и фрукта, эту идею легко можно реализовать.

По сути, нужно создать схему (вроде тех, которые мастерились в старших классах на уроках физики), используя растительный аккумулятор, созданный путем наклеивания полосок цинка и меди внутрь выбранного фрукта или корнеплода.

Электроэнергия поступает от окисления цинка, органическое вещество служит в качестве проводящего барьера, а медь замыкает цепь. Один фрукт/овощ будет генерировать около половины вольта электроэнергии, а стеки чередующихся слоев овощей, цинка и меди, создающих ряд батарей в виде лазаньи, увеличат общее напряжение.

Автомобиль. Отличный друг и помощник при отсутствии электричества в доме! Сейчас практически каждый автовладелец уже обзавёлся автомобильным зарядником, ну а в качестве более продуманной альтернативы можно приобрести вот такую заряжалку сразу на 2 порта.

Портативная USB-батарея. При выборе подобного аксессуара обращайте внимание на его ёмкость. К примеру, вот такая батарея на 6000 mAh полностью зарядит ваш айфон 4 раза, прежде чем сама исчерпает собственный запас

### [Троттлинг](#)

Троттлинг (от английского throttling) представляет собой механизм защиты процессора от термических повреждений при перегреве системы. Чем выше температура действует на микропроцессор, тем больше машинных тактов он пропускает. Такты пропускаются, соответственно снижается эффективность и производительность – это и есть троттлинг процессора.

Дросселирование тиков (проф. жарг. троттлинг, троттлинг от англ. throttling) — механизм пропуска части машинных тактов (циклов) в цифровой электронике с целью синхронизации работы различных компонентов (например, в интерфейсе SCSI) или их защиты, в том числе процессора, от термического повреждения при перегреве.

Интерфейс автоматического управления конфигурацией и питанием (ACPI) - это промышленный стандарт, который определяет функции управления питанием и другие сведения о конфигурации компьютера. Некоторые предыдущие версии BIOS не поддерживают интерфейс ACPI, поэтому компьютеры не могут успешно переходить в дополнительные режимы питания, например в ждущий или спящий режим.

Интерфейс автоматического управления конфигурацией и питанием (ACPI) пришел на смену уже устаревшего APM (Advanced Power Management).

### [Программы-уборщики](#)

Стоит доверять различным программам-уборщикам. Ведь захламленный жесткий диск может не только усложнить рабочий процесс из-за большого количества времени, которое уйдет на поиск необходимой информации среди множества бесполезных файлов, но и скажется на производительности ОС. Поэтому от ненужных временных файлов, которые создаются браузерами, самой операционной системой и приложениями, во время их функционирования, необходимо периодически избавляться.

Для этого лучше всего использовать специальные программы-уборщики, которые справляются с поставленной задачей в кратчайшие сроки. Они с легкостью избавят носители информации от лишних данных.

### [Замыкание](#)

Замыкание почти всегда является экстремальным режимом его работы и поэтому крайне нежелательным, и опасным. Последствия замыкания носят, как правило, катастрофический характер прежде всего для самого источника электропитания. Короткое замыкание возникает при случайном соединении положительного и отрицательного полюсов источника напряжения любым проводником с малым сопротивлением.

При коротком замыкании аккумулятора происходит предельно быстрый его разряд. Этот процесс сопровождается выделением большого количества тепловой энергии, происходит стремительный перегрев источника питания с образованием в результате интенсивных химических реакций газов. Температура и давление внутри батареи быстро повышаются, что может привести к нарушению её герметичности и даже взрыву.

Каждый ИБП оснащается функциями тестирования – проверки исправности своих внутренних узлов. При этом осуществляется контроль внештатных ситуаций (возникновение перегрузки или короткого замыкания), анализируется состояние батарей, степень их разряда, а также правильность подключения ИБП. При подаче питания на ИБП автоматически запускается процедура тестирования, которая затем повторяется через определенные промежутки времени. Этот процесс можно запустить и вручную, нажав соответствующую кнопку (если таковая у ИБП имеется).

Компания Открытые Технологии предлагает услуги по созданию суперкомпьютеров гибридной архитектуры на базе графических процессоров, отличающихся от традиционного более низкого энергопотребления, существенно более низкой стоимостью и компактностью.

Для примера, если ранее для решения задачи сейсмоанализа требовался суперкомпьютер, состоящий из 2000 вычислительных узлов, потребляющий 1 200 кВт электроэнергии и стоящий около \$8 млн., то современный гибридный суперкомпьютер, способный решить ту же задачу за то же время, занимает более чем в 30 раз меньше места, почти в 30 раз экономичнее по энергопотреблению и в 20 раз дешевле.

#### [Приложения](#)

Приложения с названиями вроде Ultimate Battery Saver. В 99% (если не в ста) случаев это либо плацебо, либо все тот же таск-киллер, снабженный механизмом, который отключает разные компоненты смартфона при достижении определенного уровня заряда. Сначала происходит перевод на 2G и отключение GPS, затем отключается интернет, а под самый конец телефон переводится в режим полета. Проблема здесь в том, что описанный механизм работы скорее мешает и все это удобнее сделать самому в нужное время. Установил и забыл. Как говорится, жить стало лучше, жить стало веселей с осознанием того факта, что Battery Doctor заботится о моей батарее.

Почитав описание пары десятков аналогов "Доктора" можно условно разделить их на две категории:

- информаторы - программы, которые сообщают о расходе батареи
- оптимизаторы - это подобные "Доктору" нещадно убивающие ненужные приложения для экономии энергии

На открытие и закрытие программы тратится больше энергии, чем на её поддержание в фоновом режиме, таким образом вместо оптимизации получался усиленный расход.

Одна из главных проблем современных смартфонов – недолгий срок работы от батареи. Дать объективную оценку, какая из трех систем имеет лучшее энергосбережение сложно – нельзя использовать одинаковые методы сравнения. Традиционно считается, что iOS менее энергоемкая, однако смартфоны на Android и Windows комплектуются мощными аккумуляторами. Еще одно

преимущество Android – возможность самостоятельно установить энергосберегающий режим и произвести тонкую настройку системы, отключив ненужные программы, дающие повышенный расход аккумулятора.

Windows Phone также позволяет экономить электроэнергию, а вот в iOS серьезно не хватает энергосберегающих опций. Особенно это характерно для последней, 7-й версии ОС.

### Symbian Blackberry OS

Известно, что при включенном Bluetooth устройство разряжается в разы быстрее. Существуют ли алгоритмы, чтобы это максимально обходить? Особенno для умных часов и прочих устройств, которые постоянно работают на Bluetooth.

В частности, появились поддержка Wi-Fi, что кардинально изменило реализацию работы с гаджетами, и активация режима энергосбережения при включенном экране.

Благодаря поддержке Wi-Fi пользователи теперь не ограничены радиусом работы Bluetooth (плюс более высокая стабильность сигнала). Часы могут работать с данными телефона, в какой бы точке мира они не находились (к примеру, телефон в Москве, а часы – в Хабаровске). «Общение» девайсов осуществляется через облачные сервисы Google (важно только, чтобы оба изделия были активны и имели подключение к интернету).

Есть режим экстремального энергосбережения, в котором Apple Watch работает как часы. Не в смысле точности (хотя и это тоже), — просто единственной функцией становится демонстрация текущего времени.

### *Режим сна*

Некоторые пользователи (которые используют свой ноутбук каждый день) предпочитают и вовсе не выключать его, скажем, на ночь, а вместо выключения использовать ждущий режим. (sleep). В таком случае не нужно ждать загрузки операционной системы и фоновых программ (этот процесс, как правило, не быстрый — особенно на маломощных ноутбуках или нетбуках).

Не стоит забывать о том, что ждущий режим предназначен для временного прекращения работы (скажем, на пару часов не более), но никак не отменяет выключение.

Во время ждущего режима, компьютер отключает дисплей, жесткий диск, адAPTERы... Работают только оперативная память и процессор в самом экономичном режиме, а также клавиатура и тачпад, чтобы обеспечить возможность выхода из этого режима.

Т.е. компьютер все-таки работает в этом режиме, хоть и не в полном смысле этого слова.

Во время ждущего режима на все устройства подается питание, в т.ч. батарея находится во включенном режиме; система так же выделяет тепло и т.п.

Постоянная работа без выключения — так же вредна для батареи и ее ресурс исчерпывается еще быстрее.

Как вариант – используйте спящий режим (hibernate, гибернация) вместо ждущего (если все же не хотите отказываться от возможности быстрого старта ноутбука).

## *Жесткий диск*

Жесткий диск(и) на сетевом накопителе может перейти в спящий режим позже, чем указанно в настройках. Фактически это связано с особенностями встроенной системы Linux. При сохранении данных на сетевой накопитель данные сначала записываются в системный кэш, а затем на жесткий диск(и) сетевого накопителя. Время записи зависит от размера данных. После записи данных, если нет доступа к жесткому диску, жесткий диск перейдет в спящий режим в соответствии с настройками в меню.

## *Типы аккумуляторов*

Свинцово-кислотные аккумуляторы. Одна из старейших аккумуляторных систем. Эта недорогая, надежная и переносящая перегрузки батарея; но она имеет низкую удельную энергию и ограниченный срок службы. Свинцовый кислотный аккумулятор используется в автомобильном транспорте, в инвалидных колясках, в системах аварийного освещения и в источниках бесперебойного питания (ИБП).

Никель-кадмиеевые (NiCd) аккумуляторы. Также является одной из старейших и хорошо изученных аккумуляторных систем. Эти источники питания используется там, где необходим длительный срок службы, высокий ток разрядки, экстремальные температуры и низкая стоимость. Из-за того, что NiCd аккумуляторы наносят значительный вред окружающей среде, их заменяют другими типами систем. Основные области применения: электроинструмент, рации, авиационный транспорт, ИБП. В Европе запретили продавать потребительские товары с такими типами аккумуляторов, но в России их можно приобрести.

Никель-металлгидридные (NiMH) аккумуляторы. Фактически являются заменой никель-кадмиеевых; имеет более высокую удельную энергию и меньшее количество токсичных металлов. NiMH аккумуляторы используется в медицинском оборудовании, в гибридных автомобилях, в ракетно-космической технике, в промышленности.

Литий-ионные (Li ion) аккумуляторы. Самый перспективный тип аккумуляторных систем; используется в портативных потребительских товарах, также, как и в электромобилях. Li ion аккумуляторы чувствительны к превышению напряжения при заряде и, для обеспечения безопасности, в них добавляется защитный контур, но не всегда. Эти типы аккумуляторов дороже, чем описанные выше.

Семейство литий-ионных систем можно разделить на три основных типа батарей в зависимости от материала катода – это кобальт лития, литий-марганцевая шпинель и литий-феррофосфат. Характеристики этих литий-ионных систем приведены ниже.

Кобальт лития или литий оксид кобальта (LiCoO<sub>2</sub>). Обладает высокой удельной энергией, переносит умеренные нагрузки и обладает небольшим сроком службы. Применяется в сотовых телефонах, ноутбуках, цифровых фотоаппаратах и других гаджетах.

Литий-марганцевая шпинель или литий-марганцевый (LiMn<sub>2</sub>O<sub>4</sub>). Переносит высокий ток заряда и разряда, но имеет низкую удельную энергию и небольшой срок службы; используется в электроинструментах, медицинском оборудовании и в электрических силовых агрегатах.

Литий-феррофосфатный (LiFePO<sub>4</sub>). Схож с литий-марганцевым; номинальное напряжение 3,3 В/элемент; более долговечный, но обладает более высокой скоростью саморазряда, чем другие литий-ионные системы.

Существует и множество других типов литий-ионных аккумуляторов, некоторые из которых будут описаны позднее на этом сайте. Здесь отсутствует популярный литий-полимерный тип аккумуляторов. В то время как литий-ионные системы получили свое название благодаря материалу катодов, литий-полимерные системы заслужили свое название благодаря архитектуре. Также здесь не упоминаются литий-металлические (Li-metal) аккумуляторы. Этот тип источника тока еще требует доработки, но, скорее всего, в скором времени они будут обладать необыкновенно высокой удельной энергией и хорошей удельной мощностью.

#### *Алгоритмы управления энергосбережением*

Кроме изменения частот, сборщики зачастую добавляют в ядро новые алгоритмы управления энергосбережением (автоматическим управлением частотой процессора), которые, по их мнению, могут показать лучшие результаты в сравнении со стандартными. Почти все из них базируются на используемом по умолчанию в новых версиях Android алгоритме Interactive, суть которого заключается в том, чтобы резко поднять частоту процессора до максимальной в случае повышения нагрузки, а затем постепенно снижать до минимальной. Он пришел на смену используемому раньше алгоритму OnDemand, который плавно регулировал частоту в обе стороны соразмерно нагрузке, и позволяет сделать систему более отзывчивой. Сборщики альтернативных ядер предлагают на замену Interactive следующие алгоритмы:

- SmartAssV2 — переосмысление алгоритма Interactive с фокусом на сохранение батареи. Основное отличие в том, чтобы не дергать процессор на высокие частоты в случае кратковременных всплесков нагрузки, для которых хватит и низкой производительности процессора. По умолчанию используется в ядре Matr1x.
- InteractiveX — тюнингованный алгоритм Interactive, главная особенность которого в залочке процессора на минимальной указанной пользователем частоте и обесточивании второго ядра процессора во время отключения экрана. По умолчанию используется в Leankernel.
- LulzactiveV2 — по сути, изобретенный заново OnDemand. Когда нагрузка на процессор превышает указанную (по умолчанию 60%), алгоритм поднимает частоту на определенное число делений (по умолчанию 1), при понижении нагрузки — опускает. Особый интерес представляет тем, что позволяет самостоятельно задавать параметры работы, поэтому подходит для прожженных гиков.

Вообще, сборщики ядер очень любят придумывать новые алгоритмы энергосбережения по причине простоты их реализации, поэтому можно найти еще с десяток других. Большинство из них полный шлак, и при выборе планировщика следует руководствоваться правилом: либо один из трех описанных выше, либо стандартный Interactive, который, кстати, очень неплох. Сделать выбор можно с помощью все той же Trickster MOD.

#### *Калибровка батареи*

В процессе активной эксплуатации существенно сокращается срок службы аккумулятора. Как свидетельствует статистика, корректно проработать батарея смартфона может примерно около

года. Очень часто бывает так, что проблемы с аккумулятором видны невооруженным глазом. Неисправная батарея начинает попросту выделять токсичный газ, ввиду этого она раздувается и теряет первоначальную форму. Также на внешней оболочке аккумулятора при этом могут появиться следы коррозии – бело-зеленые пятна.

Достаточно сложно обнаружить симптомы выхода из строя аккумулятора, если он является несъемным. Но когда батарея начнет раздуваться, это непременно отразится на корпусе, он будет попросту деформироваться – растрескиваться и менять форму. Достаточно часто деформация корпуса приводит к растрескиванию дорогостоящего экрана гаджета.

Обратите внимание: отнюдь не всегда аккумулятор раздувается быстро. Следовательно, на ранних стадиях разрушительного процесса вы можете просто не заметить, что батарея уже начала постепенно раздуваться.

Если ваш гаджет укомплектован съемным аккумулятором, определить неполадки с ним вы сможете при помощи простейшего теста. Для проведения тестирования нужно просто вынуть аккумулятор из устройства и попробовать покрутить его на ровной горизонтальной поверхности. В случае, если по инерции аккумулятор продолжает вращаться, вы имеете дело с батареей, которую уже тронули первые симптомы коррозии. Аккумулятор нужно срочно менять!

Решение о замене аккумулятора должно приниматься не от того, что в отчете о состоянии аккумулятора вашего устройства, а от вашего желания. Например, если отчет показал, что работоспособность аккумулятора на уровне 40%, но его работа и срок автономной работы вас устраивает — то зачем его менять? Но нужно понимать, что срок автономной работы будет со временем уменьшаться.

В то же время, если срок автономной работы резко снизился, а вам нужно, чтобы аккумулятор работал дольше — то, наверное, пришло время заменить его. Но, обязательно перед заменой аккумулятора, проверьте устройство на наличие приложений снижающих время автономной работы.

#### *Отличия режимов сна и гибернации*

Режим сна в компьютере предназначен для своеобразной постановки "на паузу" всего процесса. Комп остается работающим и потребляет энергию (правда значительно меньше чем в обычном режиме), но при этом выключается его экран, система охлаждения и жесткий диск, а на процессор, оперативную память, беспроводные модули и прочие компоненты в таком режиме подается минимальное напряжение. Для возобновления работы и выхода, из режима сна достаточно двинуть мышку или нажать любую кнопку на клавиатуре. В итоге Вы получите систему, какой она была в тот момент, когда ушла в сон - со всеми открытыми приложениями.

Режим гибернации - это с виду то же самое, но внутри всё гораздо сложнее. Комп выключается и обесточивается полностью (остается только питание на кварцевой генератор для поддержания даты/времени в BIOS). При этом на жесткий диск записывается точная копия оперативной памяти ПК в момент выключения и при включении ПК этот образ восстанавливается с жесткого диска обратно в оперативную память. Иными словами - комп запоминает состояние системы перед уходом в режим гибернации, а затем восстанавливает все вкладки и настройки какие были. Похоже на сон? Да. Только тут задействован жесткий диск.

Этот процесс занимает больше времени, чем включение из режима сна (около 10 секунд против 1-2 сек.), но в любом случае намного быстрее обычной загрузки Windows (1 минута и более).

Кроме того, после загрузки из режима гибернации можно продолжить работу ровно с того места, на котором она была завершена (как и в случае с режимом сна), но при этом можно не опасаться за разряд батареи.

Поэтому режим гибернации больше подходит для ноутбуков.

Если не хотите снова открывать все вкладки и программы при следующем включении компа - выбирайте режим сна. Только его лучше использовать до 8ми часов. Для более длительного "хранения" данных используйте режим Гибернации.

#### *Режимы Stamina и Ultra Stamina*

В смартфонах Sony имеется ряд активных функций, позволяющих сэкономить заряд аккумулятора. В большинстве моделей имеется режим STAMINA, принцип работы которого весьма прост – когда он активен, включается расширенный режим ожидания, который экономит энергию батареи за счет отключения передачи мобильных данных и Wi-Fi в тот момент, когда экран смартфона отключен, когда же экран работает, то и устройство функционирует в нормальном режиме. Стоит помнить, что в таком случае передача данных приложений отключена, но в настройках можно указать приложения, которые необходимы для работы даже в данном энергосберегающем режиме и тогда они будут нормально функционировать. Также в настройках "Стамина" можно активировать "Продленное использование" – ограничение производительности устройства для экономии заряда.

Ultra Stamina. Простыми словами данный режим можно описать как резервное отключение всего ненужного. При его активации будут отключены все ненужные приложения, функции и процессы, которые расходуют заряд аккумулятора, но при этом остаются все самые необходимые возможности – в общем, смартфон превратится в телефон. При этом обои приобретают черно-белый оттенок. Так, при 40% оставшегося заряда и активации Ultra STAMINA устройство может прожить более 4 дней. Приложение, доступные в режиме Ultra STAMINA: Телефон; Сообщения; Контакты; Камера; Альбом; Календарь; Будильники и часы; Калькулятор; FMрадио; Настройки. Стоит отметить, что его активация перезагружает смартфон.

## Глава 4. Языки программирования

### Языки программирования и их краткий обзор

Ну вот мы и подобрались к рассмотрению одной из самых важных и трепетных тем. Исторически сложилось, что языки программирования стали самым главным инструментом программиста. Это первое, но далеко не самое важное, знание, которым должен обладать любой разработчик. Программа - это своего рода речь программиста, обращение к машине. Чем правильнее и понятней сформулирована эта речь для машины, тем лучше результат на выходе и квалификация автора этой речи.

Язык программирования определяют, как формальную знаковую систему, предназначенную для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — ЭВМ) под её управлением.

### Начало развития

Можно сказать, что первые языки программирования возникали ещё до появления современных электронных вычислительных машин: уже в XIX веке были изобретены устройства, которые можно с долей условности назвать программируемыми — к примеру, механические пианино и ткацкие станки. Значимым можно считать «язык», на котором леди Ада Августа графиня Лавлейс написала программу для вычисления чисел Бернулли для Аналитической машины Чарльза Бэббиджа, ставшей бы, в случае реализации, первым компьютером — хотя и механическим, с паровым двигателем — в мире. Одну из первых попыток создать полноценный язык программирования предпринял немецкий учёный Конрад Цузе, который в период с 1943 по 1945 год разработал язык Plankalkül. Это был очень перспективный язык, фактически являвшийся языком высокого уровня, однако из-за военных действий он не получил практической реализации, а его описание было опубликовано только в 1972 году.

### Машинный язык

Неизвестно, насколько бы ускорилось развитие программирования, если бы наработки Цузе стали доступны другим учёным в конце 40-х годов, но на практике с развитием компьютерной техники сначала получил распространение машинный язык. С его помощью программист мог задавать команды, оперируя с ячейками памяти, полностью используя возможности машины. Суть этого языка — набор кодов, обязательно понятных процессору, к кому обращаются. Части («слова») этого языка называются инструкциями, каждая из которых представляет собой одно элементарное действие для центрального процессора, как, например, считывание информации из ячейки памяти. Тогда ещё компьютеры были простыми вычислительными машинами, применяемыми для различных математических расчётов. Но они развивались, а использование большинства компьютеров на уровне машинного языка затруднительно, особенно сложным было чтение и модификация подобных программ, что усугублялось использованием абсолютной адресации памяти. Поэтому со временем от использования машинных кодов пришлось отказаться.

Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т.п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающие наличие и типы ошибок, которые необходимо

анализировать. Уже одно обращение к процессору громоздко, а анализ ошибок и вовсе представляется невообразимым, особенно, если не именно с этим процессором приходиться работать. Таким образом, набор команд машинного языка сильно зависит от типа процессора.

Чтобы решить эту проблему, стали пользоваться специальными программами-сборщиками программ из маленьких кусочков кодов — ассемблерами. Начался новый этап развития.

Но даже работа с ассемблером достаточно сложна и требует специальной подготовки. Например, для процессора Zilog Z80 машинная команда 00000101 предписывает процессору уменьшить на единицу свой регистр В. На языке ассемблера это же будет записано как DEC B.

### Языки высокого уровня

В 1954 году был разработан Фортран (англ. FORTRAN — FORmula TRANslator), компилятор для которого впервые появился в апреле 1957 года. К разработке такого языка подтолкнули новые возможности внедрённого в 1954 году компьютера IBM 704, в котором на аппаратном уровне были реализованы индексная адресация и операции с плавающей точкой. Вслед за ним появились и некоторые другие языки, например: LISP, ALGOL 58, FACT (ещё один предшественник языка COBOL). Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека, с помощью них можно писать программы до нескольких тысяч строк длиной. Условными словами можно было, как привычно человеку, гораздо более просто выразить сложную программную операцию из битов.

### Появление структурного программирования

К концу 1960-х годов в связи с ростом сложности программ и дальнейшим развитием программных средств возникла необходимость увеличить производительность труда программистов, что привело к разработке структурного программирования. Основоположником данной методологии считается Эдсгер Дейкстра, который в 1968 году опубликовал своё знаменитое письмо «Оператор Goto считается вредным», а также описал основные принципы структурного программирования. Основным аргументом было то, что код с goto трудно форматировать, так как он может нарушать иерархичность выполнения (парадигму структурного программирования) и потому отступы, призванные отображать структуру программы, не всегда могут быть выставлены правильно. Goto также мешает оптимизации компиляторами управляющих структур. Впервые подобные сомнения высказал Хайнц Земанек (Heinz Zemanek) на совещании по языку Алгол в начале 1959 года в Копенгагене. С развитием структурного программирования следующим достижением были процедуры и функции. Общий код программы в данном случае становится меньше. Это способствовало созданию модульных программ.

Структуры — это составные типы данных, построенные с использованием других типов данных. Структурное программирование предполагает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций безусловного перехода (GOTO), автономные подпрограммы, поддержка рекурсии и локальных переменных. Суть такого подхода заключается в возможности разбиения программы на составляющие элементы с увеличением читабельности программного кода.

Примерно в то же время были созданы функциональные (аппликативные) языки (Пример: Lisp — англ. LISt Processing, 1958) и логические языки (пример: Prolog — англ. PROGramming in LOGic, 1972).

Хотя внедрение структурного программирование дало положительный результат, даже оно оказывалось несостоятельным тогда, когда программа достигала определённой длины. Для того чтобы написать более сложную и длинную программу, нужен был новый подход к программированию.

## ООП

При использовании структур данных в программе вырабатываются и соответствующие им функции для работы с ними. это привело к мысли их объединить и использовать совместно, так появились классы.

Класс — это структура данных, содержащая в себе не только переменные, но и функции, которые работают с этими переменными.

Коротко, это достижение в области программирования было очень велико. Теперь программирование можно было разбить на классы и тестировать не всю программу, состоящую из 10 000 строк кода, а разбить программу на 100 классов, и тестировать каждый класс. Это существенно облегчило написание программного продукта.

В итоге в конце 1970-х и начале 1980-х были разработаны принципы объектно-ориентированного программирования. ООП сочетает лучшие принципы структурного программирования с новыми концепциями инкапсуляции, полиморфизма подтипов и наследования.

Первым объектно-ориентированным языком программирования является Симула-67, в котором впервые появились классы. Концепции ООП получили дальнейшее развитие в языке Smalltalk, в котором также были заложены основы систем с оконным управлением. Более поздними примерами объектно-ориентированных языков являются Object Pascal, C++, Java, C# и др.

ООП позволяет оптимально организовывать программы, разбивая проблему на составные части, и работая с каждой по отдельности. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче.

## Стандартизация языков программирования

Для многих широко распространённых языков программирования созданы международные стандарты. Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка. В рамках таких комитетов продолжается разработка и модернизация языков программирования и решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

## Типы данных

Современные цифровые компьютеры являются двоичными и данные хранят в двоичном (бинарном) коде (хотя возможны реализации и в других системах счисления). Эти данные как правило отражают информацию из реального мира (имена, банковские счета, измерения и др.), представляющую высокоДУРНевые концепции.

Особая система, по которой данные организуются в программе, — это система типов языка программирования; разработка и изучение систем типов известна под названием теория типов. Языки можно поделить на имеющие *статическую типизацию* и *динамическую типизацию*, а также *бестиповые языки* (например, Forth).

Статически типизированные языки могут быть в дальнейшем подразделены на языки с *обязательной декларацией*, где каждая переменная и объявление функции имеет обязательное объявление типа, и языки с *выводимыми типами*.

### Структуры данных

Системы типов в языках высокого уровня позволяют определять сложные, составные типы, так называемые структуры данных. Как правило, структурные типы данных образуются как декартово произведение базовых (атомарных) типов и ранее определённых составных типов.

Основные структуры данных (списки, очереди, хеш-таблицы, двоичные деревья и пары) часто представлены особыми синтаксическими конструкциями в языках высокого уровня. Такие данные структурируются автоматически.

### Семантика языков программирования

Существует несколько подходов к определению семантики языков программирования.

Наиболее широко распространены разновидности следующих трёх: операционного, деривационного (аксиоматического) и денотационного (математического).

При описании семантики в рамках операционного подхода обычно исполнение конструкций языка программирования интерпретируется с помощью некоторой воображаемой (абстрактной) ЭВМ.

Аксиоматическая (Деривационная) семантика описывает последствия выполнения конструкций языка с помощью языка логики и задания пред- и постусловий.

Денотационная семантика оперирует понятиями, типичными для математики — множества, соответствия, а также суждения, утверждения и др.

### Парадигма программирования

Язык программирования строится в соответствии с той или иной *базовой моделью вычислений и парадигмой программирования*.

Несмотря на то, что большинство языков ориентировано на *императивную модель вычислений*, задаваемую фон-неймановской архитектурой ЭВМ, существуют и другие подходы. Можно упомянуть языки со *стековой вычислительной моделью* (Форт, Factor, PostScript и др.), а также функциональное (Лисп, Haskell, ML, F#, РЕФАЛ, основанный на модели вычислений, введённой советским математиком А. А. Марковым-младшим и др.) и логическое программирование (Пролог).

Далее, мы рассмотрим эту тему чуть подробней.

### Способы реализации языков

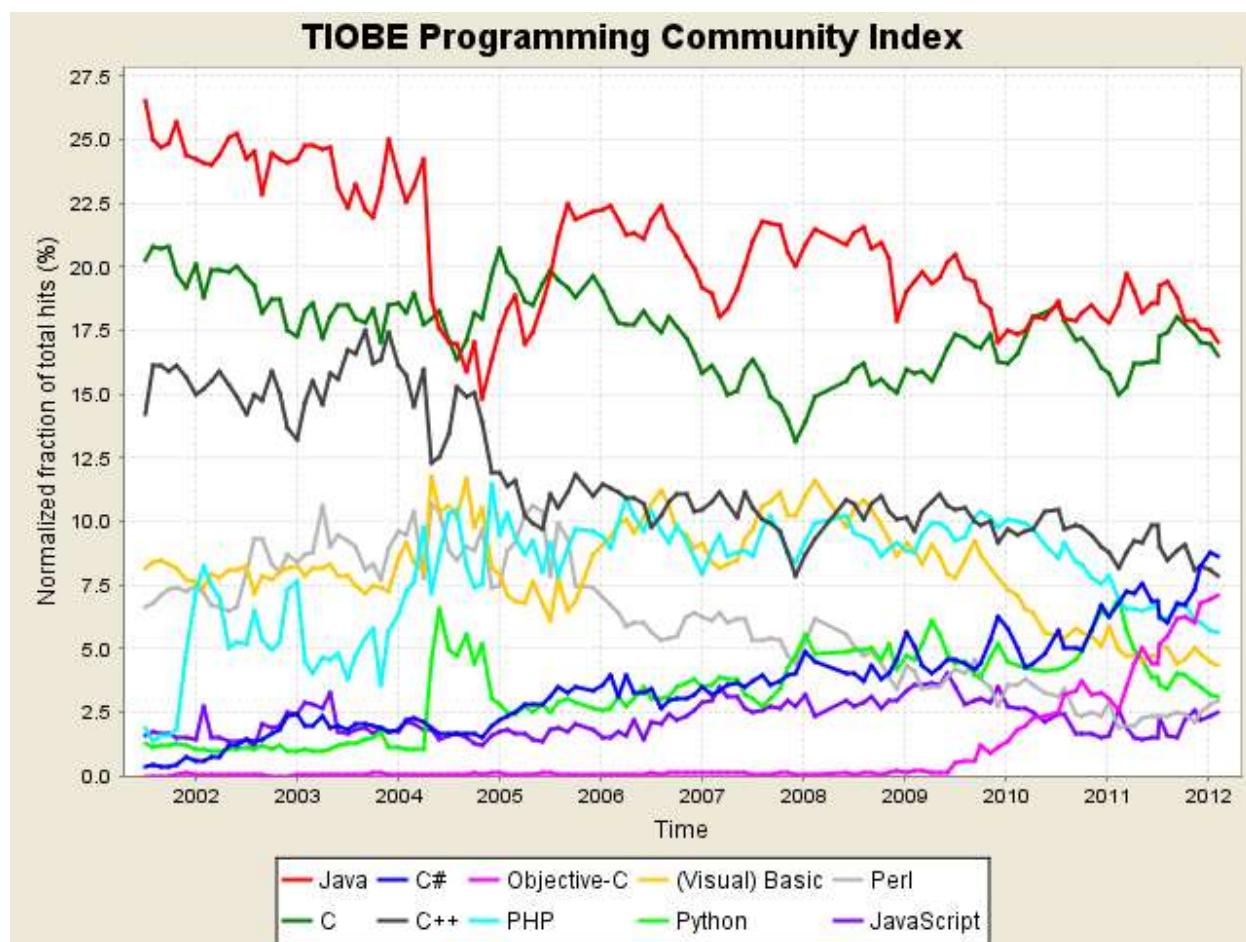
Языки программирования могут быть реализованы как *компилируемые, интерпретируемые и встраиваемые*. Намного чаще можно встретить комбинацию этих реализаций. Подробней об этом поговорим позже.

Как уже было сказано, языки программирования также разделяются на языки **высокого** и **низкого** уровней.

Недостатком некоторых языков высокого уровня является большой размер программ в сравнении с программами на языках низкого уровня. С другой стороны, для алгоритмически и структурно сложных программ при использовании суперкомпиляции преимущество может быть на стороне языков высокого уровня. Сам текст программ на языке высокого уровня меньше, однако, если взять в байтах, то код, изначально написанный на ассемблере, будет более компактным. Поэтому в основном языки высокого уровня используются для разработки программного обеспечения компьютеров и устройств, которые имеют большой объём памяти. А разные подвиды ассемблера применяются для программирования других устройств, где критичным является размер программы.

### Каким будет наше будущее?

Спрогнозировать наше будущее – задача очень сложная, поэтому будем смотреть на факты и делать выводы. Ребята из компании TIOBE иногда проводят анализ и вычисляют индекс популярности.



#### 1. Java

Java является одним из самых популярных языков для бэкэнд-разработки современных корпоративных веб-приложений. С Java и основанными на нём фреймворками разработчики могут создавать масштабируемые веб-приложения для широкого круга пользователей. Java — также

основной язык, используемый для разработки родных Android-приложений для смартфонов и планшетов.

## 2. JavaScript

Каждый современный сайт использует JavaScript. Это ключевой язык для создания интерактивности сайта или построения пользовательских интерфейсов с одним из десятка популярных JavaScript-фреймворков.

## 3. C#

C # является основным языком для разработки на платформах и сервисах Microsoft. Будь то разработка современных веб-приложений с использованием Azure и .NET, приложений для «девайсов» Windows или мощных «настольных» приложений для бизнеса, C# — самый быстрый способ использовать всё, что может предложить Microsoft. Кроме того, это и один из основных языков движка для разработки игр Unity.

## 4. PHP

Пишите веб-приложение для работы с данными? Язык PHP наряду с базами данных (например, MySQL) является важным инструментом для создания современных веб-приложений. На PHP разработано большинство сайтов, ориентированных на большой объём данных. Это также основополагающая технология мощных систем управления контентом, как WordPress.

## 5. C++

Если для максимальной отдачи мощности процессора вам необходимо подключиться непосредственно к железу, поможет язык C++. Это идеальный выбор для разработки мощного «настольного» программного обеспечения, игр с функцией аппаратного ускорения, а также приложений для ПК, консолей и мобильных устройств, требующих большого объёма памяти для работы.

## 6. Python

Python может сделать почти всё вышеперечисленное. Веб-приложения, пользовательские интерфейсы, анализ данных, статистика — для какой бы задачи вам не предстояло найти решение, в Python, скорее всего, найдётся подходящий фреймфорк. Совсем недавно учёные пришли к выводу, что Python можно использовать в качестве основного инструмента для обработки гигантских объёмов данных практически в любой отрасли.

## 7. C

Почему язык C по-прежнему популярен? Из-за размера: маленький, быстрый и мощный. Если вы разрабатываете программное обеспечение для встраиваемых систем, работаете с системными ядрами или просто хотите выжать из имеющихся под рукой ресурсов всё до последней капли, C — то, что нужно.

## 8. SQL

Данные — всеобъемлющие и всепроникающие. SQL даёт возможность найти необходимую информацию быстрым и надёжным способом. Используя SQL, вы можете легко запрашивать и извлекать значительные объёмы данных из больших и сложных баз данных.

### 9. Ruby

Хотите запустить проект в рекордно короткие сроки или создать прототип новой идеи для крутого веб-приложения? С помощью Ruby (и Ruby on Rails) это возможно довольно быстро. Обладая невероятной мощностью, язык прост в освоении. Плюс на нём написаны тонны популярных веб-приложений по всему миру.

### 10. Objective-C

Собираетесь написать приложение для iOS? Тогда вы просто обязаны знать Objective-C. Несмотря на прошлогоднюю шумиху вокруг нового языка Apple Swift, Objective-C по-прежнему остаётся основополагающим языком приложений для экосистемы Apple. С Objective-C и официальным инструментом разработки ПО от Apple XCode до App Store — рукой подать.

### 11. Perl

Можно ли назвать Perl эзотерическим языком? Да. Сбивает ли он с толку? Да. Является ли он супермощным языком и ключевым компонентом в арсенале кибербезопасности? Снова да. Разработчики используют Perl с самых истоков интернета, и он до сих пор считается ключевым инструментом для любого ИТ-специалиста.

### 12. .NET

Хотя и не язык сам по себе, .NET является ключевой платформой Microsoft для разработки облачных и не очень сервисов и приложений. Становится более продвинутым и ценным с каждым новым релизом. Благодаря последним усилиям Microsoft в области разработки с открытым исходным кодом, .NET теперь приходит на платформы Google и Apple. Как результат, вы можете использовать .NET с различными языками программирования для создания мультиплатформенных приложений.

### 13. Swift

За менее чем год существования язык программирования Swift привлёк внимание разработчиков во всём мире как новый, простой и быстрый способ разработки для операционных систем OS X и iOS. Широкие полномочия и дружественный синтаксис Swift позволяют написать очередное убойное приложение для пользователей Apple.

## Выводы

Опираясь на статистику и индекс популярности ЯП, можно однозначно говорить о том, что будущее за динамическими языками программирования (с динамической типизацией). Динамический язык — это такой язык программирования и такой транслятор, которые позволяют определять типы данных и осуществлять синтаксический анализ и трансляцию "на лету", непосредственно на этапе выполнения (JavaScript, Python, Ruby, Basic, Tcl, Smalltalk, PHP). Однако вместе с языками активно набирающими всеобщую популярность, существуют и вымирающие или мертвые языки. К таким можно отнести все семейство COBOL, ada, Haskell, assembler, Basic.

На сегодняшний день языков программирования бесчисленное множество, но лишь немногие из них являются популярными. Все дело в сложившейся концепции разработки, основанной на конкретных методах и парадигмах. Однако никто не гарантирует, что эти стандарты сохранятся. Скорее всего так и будет происходить в дальнейшем. Поэтому развитие языков, на мой взгляд, очень важная тема, и за этим развитием нужно следить.

## JavaScript

### Введение в JavaScript

**JavaScript** – это *интерпретируемый язык* программирования с объектно-ориентированными возможностями.

**Интерпретируемый язык программирования** — язык программирования, в котором исходный код программы не преобразовывается в машинный код для непосредственного выполнения центральным процессором (как в компилируемых языках), а исполняется с помощью специальной программы-интерпретатора.

С точки зрения синтаксиса базовый язык JavaScript напоминает C, C++ и Java такими программными конструкциями, как инструкция if, цикл while и оператор &&. Однако это подобие ограничивается синтаксической схожестью.

JavaScript – это язык с *динамической типизацией*, т. е. в нем не требуется определять типы переменных. Объекты в JavaScript отображают имена свойств на произвольные значения.

Ядро языка JavaScript поддерживает работу с такими простыми типами данных, как числа, строки и булевые значения. Помимо этого, он обладает встроенной поддержкой массивов, дат и объектов регулярных выражений.

Обычно JavaScript применяется в веб-браузерах, а расширение его возможностей за счет введения объектов позволяет организовать взаимодействие с пользователем, управлять веб-браузером и изменять содержимое документа, отображаемое в пределах окна веб-браузера. Эта встроенная версия JavaScript запускает сценарии, внедренные в HTML-код веб-страниц. Как правило, эта версия называется *клиентским языком* JavaScript, чтобы подчеркнуть, что сценарий исполняется на клиентском компьютере, а не на веб-сервере.

В основе языка JavaScript и поддерживаемых им типов данных лежат международные стандарты, благодаря чему обеспечивается прекрасная совместимость между реализациями. Некоторые части клиентского JavaScript формально стандартизированы, другие части стали стандартом де-факто, но есть части, которые являются специфическими расширениями конкретной версии браузера.

Совместимость реализаций JavaScript в разных браузерах зачастую приносит немало беспокойств программистам, использующим клиентский язык JavaScript.

### JavaScript – это не Java

Одно из наиболее распространенных заблуждений о JavaScript состоит в том, что этот язык представляет собой упрощенную версию Java, языка программирования, разработанного в компании Sun Microsystems. Кроме некоторой синтаксической схожести и способности предоставлять исполняемое содержимое для веб-браузеров, эти два языка между собой ничто не связывает.

Схожесть имен – не более чем уловка маркетологов (первоначальное название языка – LiveScript – было изменено на JavaScript в последнюю минуту). Однако JavaScript и Java могут взаимодействовать друг с другом.

#### *JavaScript не простой язык*

Поскольку JavaScript является интерпретируемым языком, очень часто он позиционируется как язык сценариев, а не как язык программирования, при этом подразумевается, что языки сценариев проще и в большей степени ориентированы не на программистов, а на обычных пользователей. В самом деле, при отсутствии контроля типов JavaScript прощает многие ошибки, которые допускают неопытные программисты. Благодаря этому многие веб-дизайнеры могут использовать JavaScript для решения ограниченного круга задач, выполняемых по точным рецептам.

Однако за внешней простотой JavaScript скрывается полноценный язык программирования, столь же сложный, как любой другой, и даже более сложный, чем многие. Программисты, пытающиеся решать с помощью JavaScript нетривиальные задачи, часто разочаровываются в процессе разработки из-за того, что недостаточно понимают возможности этого языка.

### [История JavaScript](#)

#### *Предпосылки*

В 1992 году компания Nombas начала разработку встраиваемого скриптового языка Стм (Си-минус-минус), который, по замыслу разработчиков, должен был стать достаточно мощным, чтобы заменить макросы, сохраняя при этом схожесть с Си, чтобы разработчикам не составляло труда изучить его. Главным отличием от Си была работа с памятью. В новом языке всё управление памятью осуществлялось автоматически: не было необходимости создавать буферы, объявлять переменные, осуществлять преобразование типов. В остальном языки сильно походили друг на друга: в частности, Стм поддерживал стандартные функции и операторы Си.

Стм был переименован в ScriptEase, поскольку исходное название звучало слишком негативно, а упоминание в нём Си «отпугивало» людей. На основе этого языка был создан проприетарный продукт CEnvi. В конце ноября 1995 года Nombas разработала версию CEnvi, внедряемую в веб-страницы. Страницы, которые можно было изменять с помощью скриптового языка, получили название Espresso Pages — они демонстрировали использование скриптового языка для создания игры, проверки пользовательского ввода в формы и создания анимации. Espresso Pages позиционировались как демоверсия, призванная помочь представить, что случится, если в браузер будет внедрён язык Стм. Работали они только в 16-битовом Netscape Navigator под управлением Windows.

#### *JavaScript*

Перед Брэнданом Эйхом, нанятым в компанию Netscape 4 апреля 1995 года, была поставлена задача внедрить язык программирования Scheme или что-то похожее в браузер Netscape. Поскольку требования были размыты, Эйха перевели в группу, ответственную за серверные продукты, где он проработал месяц, занимаясь улучшением протокола HTTP. В мае разработчик был переброшен обратно, в команду, занимающуюся клиентской частью (браузером), где он немедленно начал разрабатывать концепцию нового языка программирования.

Менеджмент разработки браузера, включая Тома Пакина, Михаэля Тоя, Рика Шелла, был убеждён, что Netscape должен поддерживать язык программирования, встраиваемый в HTML-код страницы.

Помимо Брэндана Эйха в разработке участвовали сооснователь Netscape Communications Марк Андрессен и сооснователь Sun Microsystems Билл Джой: чтобы успеть закончить работы над языком к релизу браузера, компании заключили соглашение о сотрудничестве в разработке. Они ставили перед собой цель обеспечить «язык для склеивания» составляющих частей веб-ресурса: изображений, плагинов, Java-апплетов, который был бы удобен для веб-дизайнеров и программистов, не обладающих высокой квалификацией.

Первоначально язык назывался Mocha, затем он был переименован в LiveScript и предназначался как для программирования на стороне клиента, так и для программирования на стороне сервера (там он должен был называться LiveWire). На синтаксис оказали влияние языки Си и Java, и, поскольку Java в то время было модным словом, 4 декабря 1995 года LiveScript переименовали в JavaScript, получив соответствующую лицензию у Sun. Анонс JavaScript со стороны представителей Netscape и Sun состоялся накануне выпуска второй бета-версии Netscape Navigator. В нём декларируется, что 28 лидирующих ИТ-компаний выразили намерение использовать в своих будущих продуктах JavaScript как объектный скриптовый язык с открытым стандартом.

В 1996 году компания Microsoft выпустила аналог языка JavaScript, названный JScript. Анонсирован этот язык был 18 июля 1996 года. Первым браузером, поддерживающим эту реализацию, был Internet Explorer 3.0.

По инициативе компании Netscape была проведена стандартизация языка ассоциацией ECMA. Стандартизированная версия имеет название ECMAScript, описывается стандартом ECMA-262. Первой версии спецификации соответствовал JavaScript версии 1.1, а также языки JScript и ScriptEasy.

#### *Популярность*

В статье «Самый неправильно понятый язык программирования в мире стал самым популярным в мире языком программирования» Дуглас Крокфорд утверждает, что лидирующую позицию JavaScript занял в связи с развитием AJAX, поскольку браузер стал превалирующей системой доставки приложений.

Он также констатирует растущую популярность JavaScript, то, что этот язык встраивается в приложения, отмечает значимость языка.

Согласно TIOBE Index, базирующемуся на данных поисковых систем Google, MSN, Yahoo!, Википедия и YouTube, в апреле 2015 года JavaScript находился на 6 месте (год назад на 9).

(Индекс TIOBE (TIOBE programming community index) — индекс, оценивающий популярность языков программирования, на основе подсчета результатов поисковых запросов, содержащих название языка).

JavaScript является самым популярным языком программирования, используемым для разработки веб-приложений на стороне клиента (англ.)

## Дополнительные возможности JavaScript

### Библиотеки

Для обеспечения высокого уровня абстракции и достижения приемлемой степени кросс-браузерности при разработке веб-приложений используются библиотеки JavaScript. Они представляют собой набор многократно используемых объектов и функций.

Среди известных JavaScript библиотек можно отметить Adobe life, Dojo Toolkit, Extjs, jQuery, Mootools, Prototype, Qooxdoo, Underscore.

### Фреймворки

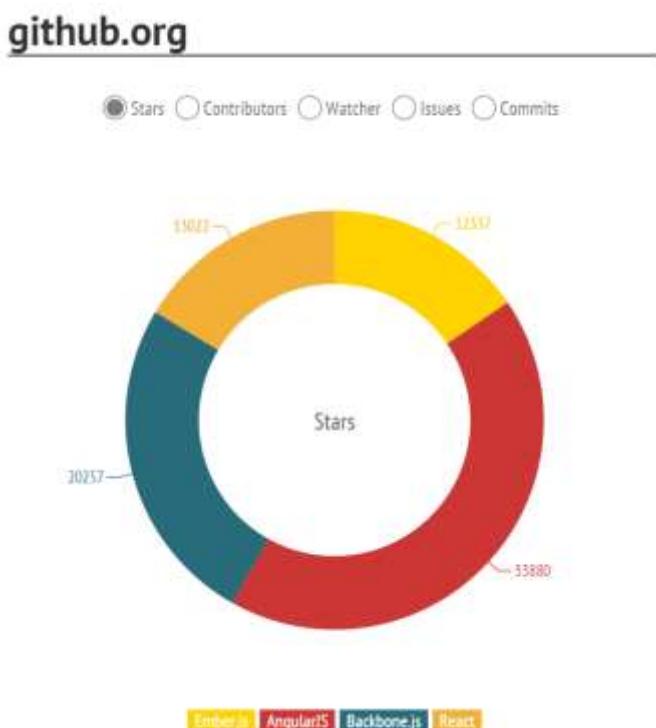
Интересными возможностями JavaScript также являются различные фреймворки, библиотеки и утилиты, построенные на его базе. Например, MV\* фреймворки Angular.js, Backbone.js, Ember.js, React.js, Ext.js. Фреймворки – это такие библиотеки, которые позволяют уйти от спагетти-кода, они помогают поддерживать структуру и организованность в проекте, облегчают ремонтопригодность в будущем. Есть даже фреймворк Matreshka.js. Все эти библиотеки очень помогают при работе с веб-приложениями и прочими.

### Популярность фреймворков

Интерес к Javascript MV\* фреймворкам вызвал их подъем.

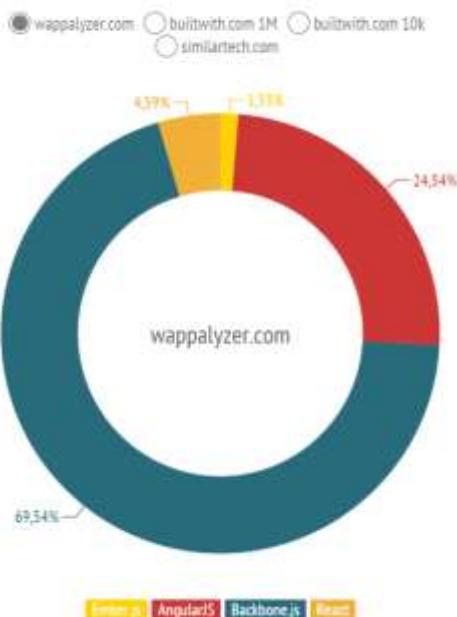
Meteor, Ember, Angular, Backbone, все они действительно популярны на Github. Измерить популярность довольно сложно, но хорошим показателем может быть количество Github-звездочек.

Звездочки на Github-е:



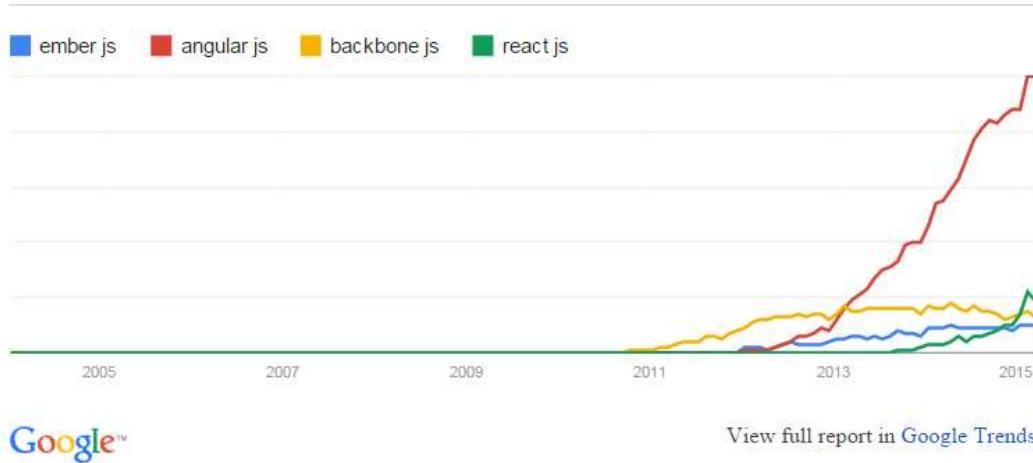
Зрелость проектов:

## Market Reach



## Google Trends

Interest over time. Web Search. Worldwide, 2004 - present.



[View full report in Google Trends](#)

## Jswiki

Jswiki — это проект на github, который постарался собрать все достойные JavaScript библиотеки и ресурсы. На страницах описания библиотек, так же собраны ссылки на статьи для начинающих, чтобы читатель мог как можно быстрее начать использовать ту или иную библиотеку.

## Область применения

### Веб-приложения

JavaScript используется в клиентской части веб-приложений: клиент-серверных программ, в котором клиентом является браузер, а сервером — веб-сервер, имеющих распределённую между

сервером и клиентом логику. Обмен информацией в веб-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами.

#### *AJAX*

JavaScript используется в AJAX, популярном подходе к построению интерактивных пользовательских интерфейсов веб-приложений, заключающемся в «фоновом» асинхронном обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе (без применения AJAX).

#### *Comet*

Comet — широкое понятие, описывающее механизм работы веб-приложений, использующих постоянные HTTP-соединения, что позволяет веб-серверу отправлять данные браузеру без дополнительного запроса со стороны браузера. Для таких приложений используются технологии, непосредственно поддерживаемые браузерами. В частности, в них широко используется JavaScript.

#### *Браузерные операционные системы*

JavaScript широко используется в браузерных операционных системах. Так, например, исходный код IndraDesktop WebOS на 75 % состоит из JavaScript, код браузерной операционной системы IntOS — на 70 %. Доля JavaScript в исходном коде eyeOS — 5 %, однако и в рамках этой операционной системы JavaScript играет важную роль, участвуя в визуализации на клиенте и являясь необходимым механизмом для коммуникации клиента и сервера.

#### *Букмарклеты*

JavaScript используется для создания небольших программ, размещаемых в закладки браузера. При этом используются URL-адреса со спецификатором javascript::.

#### *Пользовательские скрипты в браузере*

Пользовательские скрипты в браузере — это программы, написанные на JavaScript, выполняемые в браузере пользователя при загрузке страницы. Они позволяют автоматически заполнять формы, переформатировать страницы, скрывать нежелательное содержимое и встраивать желательное для отображения содержимое, изменять поведение клиентской части веб-приложений, добавлять элементы управления на страницу и т. д.

Для управления пользовательскими скриптами в Mozilla Firefox используется расширение Greasemonkey; Opera и Google Chrome предоставляют средства поддержки пользовательских скриптов и возможности для выполнения ряда скриптов Greasemonkey.

#### *Серверные приложения*

Приложения, написанные на JavaScript, могут исполняться на серверах, использующих Java 6 и более поздних версий. Это обстоятельство используется для построения серверных приложений, позволяющих обрабатывать JavaScript на стороне сервера.

Помимо Java 6, существует ряд платформ, использующих существующие движки (интерпретаторы) JavaScript для исполнения серверных приложений. (Как правило, речь идёт о повторном использовании движков, ранее созданных для исполнения кода JavaScript в браузерах WWW.)

## Платформы исполнения серверных приложений на JavaScript

Название	Используемый движок JavaScript	Языки, на которых написан движок и платформа	Лицензия
Jaxer	SpiderMonkey	C++, C	GPL 3
persevere-framework	Rhino	Java	Модифицированная лицензия BSD
Helma	Rhino	Java, JavaScript	BSD-подобная Helma License 2.0
v8cgi	V8	C++, JavaScript	Лицензия BSD
node.js	V8	C++	Лицензия MIT

JavaScript на стороне сервера используется в проектах Google. Так например, Google Sites допускает подстройку с помощью JavaScript-сценариев, исполняемых движком Rhino.

Можно привести пример, когда JavaScript на стороне сервера удобнее использовать, чем тот же язык Java. Возьмем для рассмотрения Node.js.

Node.js обладает полезными свойствами, такими как поддержка большого числа соединений и быстрая их обработка, в отличие от Java. Также Node.js работает в разы быстрее.

Интеграция с клиентом осуществляется гораздо проще: один и тот же язык на обеих сторонах, передача JSON не составляет никакого труда, дополнительные утилиты не требуются.

Запросы для некоторых относительно новых баз данных, например, MongoDB, CouchDB пишутся на языке JavaScript.

## Мобильные приложения

Перевод мобильных устройств Palm на использование Palm webOS в качестве операционной системы с Mojo SDK в качестве комплекта средств разработки позволяет использовать JavaScript в качестве языка разработки мобильных приложений.

Native script (NS) – это библиотека, позволяющая делать кроссплатформенные приложения, используя XML, CSS, JavaScript. Native script решает ту же задачу, что и уже всем известный phonegap (создание кроссплатформенных приложений), но подходы у них разные. Phonegap использует движок браузера, чтобы отобразить ваш UI (фактически вы получаете веб-страницу), Native script использует нативный рендеринг, использует элементы нативного UI. Следующее важное отличие: чтобы получить доступ к камере, gps и так далее в phonegap необходимо устанавливать плагины, в то время как NS дает доступ из коробки.

Стоит подчеркнуть, что приложения можно писать для Android 4.2 и выше, и для iOS 7.1 и выше. Для этих целей также можно использовать библиотеку React Native.

### *Виджеты*

Виджет — вспомогательная мини-программа, графический модуль которой размещается в рабочем пространстве соответствующей родительской программы, служащая для украшения рабочего пространства, развлечения, решения отдельных рабочих задач или быстрого получения информации из интернета без помощи веб-браузера.

JavaScript используется как для реализации виджетов, так и для реализации движков виджетов. В частности, при помощи JavaScript реализованы Apple Dashboard, Microsoft Gadgets, Yahoo! Widgets, Google Gadgets, Klipfolio Dashboard.

### *Прикладное программное обеспечение*

JavaScript используется для написания прикладного ПО. Например, 16,4 % исходного кода Mozilla Firefox написано на JavaScript.

Google Chrome OS в качестве прикладного ПО использует веб-приложения.

### *Манипуляция объектами приложений*

JavaScript также находит применение в качестве скриптового языка доступа к объектам приложений. Платформа Mozilla (XUL/Gecko) использует JavaScript. Среди сторонних продуктов, например, Java, начиная с версии 6, содержит встроенный интерпретатор JavaScript на базе Rhino. Сценарии JavaScript поддерживаются в таких приложениях Adobe, как Adobe Photoshop, Adobe Dreamweaver, Adobe Illustrator и Adobe InDesign.

### *Офисные приложения*

JavaScript используется в офисных приложениях для автоматизации рутинных действий, написания макросов, организации доступа со стороны веб-служб.

### *Microsoft Office*

В Excel Services 2010 добавились два новых интерфейса программирования приложений: REST API и JavaScript Object Model (JSOM).

Excel Services 2010 REST API позволяет осуществлять доступ к объектам рабочих книг, таким как таблицы, диаграммы и именованные серии данных; получать изображения, HTML, Atom, рабочие книги; устанавливать значения и обновлять вычисления перед запрашиванием элементов.

JSOM даёт возможность реагировать на действия пользователя в отношении Excel Web Access (EWA), программно взаимодействовать с составляющими EWA. Использование JSOM осуществляется при помощи помещения кода JavaScript на страницу, содержащую компоненты EWA.

### *OpenOffice.org*

JavaScript — один из языков программирования, используемых для написания макросов в приложениях, входящих в состав OpenOffice.org. В OpenOffice.org интегрирован интерпретатор JavaScript Rhino. По состоянию на декабрь 2009 года поддержка JavaScript носила ограниченный характер. Ограничения, присущие разработке макросов OpenOffice.org на JavaScript:

- Среда выполнения JavaScript поддерживает загрузку лишь тех классов Java, которые развёрнуты сценарием JavaScript;

- Среда выполнения JavaScript не предоставляет сообщения об ошибках, произошедших во время выполнения скрипта;
- Ещё не реализована поддержка интерактивной разработки JavaScript-сценариев.

В OpenOffice.org имеется редактор и отладчик JavaScript-сценариев.

#### *Обучение информатике*

JavaScript обладает пропедевтической ценностью, позволяя сочетать при обучении информатике интенсивную практику программирования и широту используемых технологий. Преподавание данного языка в школе позволяет создать базу для изучения веб-программирования, использовать на уроках творческие проекты. Соответствующий курс позволяет обеспечить углубленный уровень изучения информатики и его имеет смысл включать в элективные курсы углубленного уровня подготовки.

JavaScript — подходящий язык для обучения программированию игр. По сравнению с альтернативами, он функционально достаточен, прост в изучении и в применении, снижает сложность для обучения, мотивирует обучаемых делиться своими играми с другими.

Не включённые в книгу Николаса Закаса «Professional JavaScript for Web Developers» части о реализации на JavaScript классических алгоритмов, техник, структур данных, послужили началу проекта Computer science in JavaScript.

#### *Версии*

JavaScript	Соответствующая версия JScript	Существенные изменения
1.0 (Netscape 2.0, март 1996)	1.0 (ранние версии IE 3.0, август 1996)	Оригинальная версия языка JavaScript.
1.1 (Netscape 3.0, август 1996)	2.0 (поздние версии IE 3.0, январь 1997)	В данной версии был реализован объект Array и устраниены наиболее серьёзные ошибки.
1.2 (Netscape 4.0, июнь 1997)		Реализован переключатель switch, регулярные выражения. Практически приведён в соответствии с первой редакцией спецификации ECMA-262.
1.3 (Netscape 4.5, октябрь 1998)	3.0 (IE 4.0, октябрь 1997)	Совместим с первой редакцией ECMA-262.
1.4 (только Netscape Server)	4.0 (Visual Studio 6, нет версии IE)	Применяется только в серверных продуктах Netscape.

	5.0 (IE 5.0, март 1999)	
	5.1 (IE 5.01)	
1.5 (Netscape 6.0, ноябрь 2000; также поздние версии Netscape и Mozilla)	5.5 (IE 5.5, июль 2000)	Редакция 3 (декабрь 1999). Совместим с третьей редакцией спецификации ECMA-262.
	5.6 (IE 6.0, октябрь 2001)	
1.6 (Gecko 1.8, Firefox 1.5, ноябрь 2005)		Редакция 3 с некоторыми совместимыми улучшениями: E4X, дополнения к Array (например, Array.prototype.forEach), упрощения для Array и String
1.7 (Gecko 1.8.1, Firefox 2.0, осень 2006), расширение JavaScript 1.6		Редакция 3, с добавлением всех улучшений из JavaScript 1.6, генераторов и списочных выражений (list comprehensions, [a*a for (a in iter)]) из Python, блоковых областей с использованием let и деструктурирующего присваивания (var [a, b] = [1, 2]).
	JScript .NET(ASP.NET; нет версии IE)	(Считается, что JScript .NET разработан при участии других членов ECMA)
1.8 (Gecko 1.9, Firefox 3.0, осень 2008), расширение JavaScript 1.7		Новая форма записи для функций, сходная с типичными лямбда-выражениями, генераторы (англ.), новые методы итеративной обработки массивов reduce() и reduceRight().
1.8.1 (Gecko 1.9.1, Firefox 3.5)		Встроенная поддержка JSON, метод getPrototypeOf() у Object, методы trim(), trimLeft(), trimRight() у String
2.0		Редакция 4 (разработка не закончена, название зарезервировано ECMA, но не было использовано для публикации)

Редакция 5 (ранее известная под названием ECMAScript 3.1. Финальная версия принята 3 декабря 2009 года)

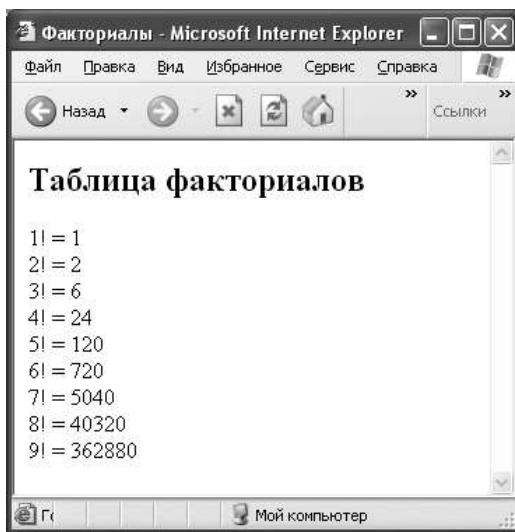
## Примеры использования клиентского JavaScript

Веб-браузер, оснащенный интерпретатором JavaScript, позволяет распространять через Интернет исполняемое содержимое в виде JavaScript-сценариев. В примере 1.1 показана простая программа на языке JavaScript, которая и представляет собой сценарий, встроенный в веб-страницу.

### Пример 1.1. Простая программа на языке JavaScript

```
<html><head><title>Факториалы</title></head>
<body>
    <h2>Таблица факториалов</h2>
    <script>
        var fact = 1;
        for (i = 1; i < 10; i++) {
            fact = fact*i;
            document.write(i + " ! = " + fact + "<br>");
        }
    </script>
</body>
</html>
```

После загрузки в браузер, поддерживающий JavaScript, этот сценарий выдаст результат, показанный на рисунке.

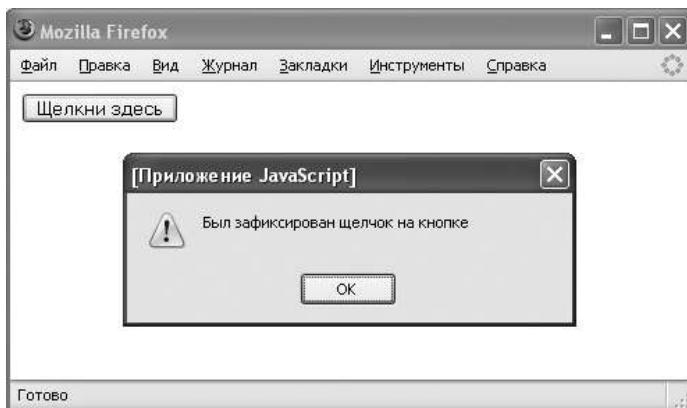


Как видно из этого примера, для встраивания JavaScript-кода в HTML-файл были использованы теги `<script>` и `</script>`. Главное, что демонстрируется в данном примере, – это использование метода `document.write()`. Этот метод позволяет динамически выводить HTML-текст внутри HTML-документа по мере его загрузки веб-браузером. JavaScript обеспечивает возможность управления не только содержимым HTML-документов, но и их поведением. Другими словами, JavaScript-программа может реагировать на действия пользователя: ввод значения в текстовое поле или щелчок мышью в области изображения в документе. Это достигается путем определения *обработчиков событий* для документа – фрагментов JavaScript\_кода, исполняемых при возникновении определенного события, например, щелчка на кнопке. В примере 1.2 показан простой фрагмент HTML\_кода, который включает в себя обработчик события, вызываемый в ответ на такой щелчок.

Пример 1.2. HTML кнопка с обработчиком события на языке JavaScript

```
<button onclick="alert('Был зафиксирован щелчок на кнопке');">  
Щелкни здесь  
</button>
```

На рисунке показан результат щелчка на кнопке.



Атрибут onclick из примера 1.2 – это строка JavaScript\_кода, исполняемого, когда пользователь щелкает на кнопке. В данном случае обработчик события onclick вызывает функцию alert(). Как видно из рисунка, функция alert() выводит диалоговое окно с указанным сообщением.

Примеры 1.1 и 1.2 демонстрируют лишь простейшие возможности клиентского JavaScript. Реальная его мощь состоит в том, что сценарии имеют доступ к содержимому HTML-документов.

### Утечки памяти

Иногда могут возникать утечки памяти, они в реальной жизни проявляют себя в двух ситуациях:

- Приложение, в котором посетитель все время на одной странице и работает со сложным JavaScript-интерфейсом. В этом случае утечки могут постепенно съедать доступную память.
- Страница регулярно делает что-то, вызывающее утечку памяти. Посетитель (например, менеджер) оставляет компьютер на ночь включенным, чтобы не закрывать браузер с кучей вкладок. Приходит утром — а браузер съел всю память и сильно тормозит.

Утечки бывают из-за ошибок браузера, ошибок в расширениях браузера и, гораздо реже, по причине ошибок в архитектуре JavaScript-кода.

### Изучение JavaScript

Наиболее очевидный подход к изучению JavaScript – это написание простых сценариев. Одно из достоинств клиентского JavaScript состоит в том, что любой, кто имеет веб-браузер и простейший текстовый редактор, имеет и полноценную среду разработки. Для того чтобы начать писать программы на JavaScript, нет необходимости в покупке или загрузке специального ПО.

Например, чтобы вместо факториалов вывести последовательность чисел Фибоначчи, пример 1.1 можно переписать следующим образом:

```
<script>

    document.write("<h2>Числа Фибоначчи </h2>");
    for (i=0, j=1, k=0, fib =0; i<50; i++, fib=j+k, j=k, k=fib) {
        document.write("Fibonacci (" + i + ") = " + fib);
        document.write("<br>");
    }
</script>
```

Этот отрывок может показаться запутанным, но для того чтобы поэкспериментировать с подобными короткими программами, достаточно набрать код и запустить его в веб-браузере в качестве файла с локальным URL-адресом. Обратите внимание, что для вывода результатов вычислений используется метод document.write().

Это полезный прием при экспериментах с JavaScript. В качестве альтернативы для отображения текстового результата в диалоговом окне можно применять метод alert():

```
alert("Fibonacci (" + i + ") = " + fib);
```

Отметим, что в подобных простых экспериментах с JavaScript можно опускать теги <html>, <head> и <body> в HTML\_файле.

Для еще большего упрощения экспериментов с JavaScript можно использовать URL-адрес со спецификатором псевдопротокола javascript: для вычисления значения JavaScript-выражения и получения результата. Такой URL-адрес состоит из спецификатора псевдопротокола (javascript:), за которым указывается произвольный JavaScript-код (инструкции отделяются одна от другой точками с запятой). Загружая URL-адрес с псевдопротоколом, браузер просто исполняет JavaScript-код. Значение последнего выражения в таком URL-адресе преобразуется в строку, и эта строка выводится веб-браузером в качестве нового документа. Например, для того чтобы проверить свое понимание некоторых операторов и инструкций языка JavaScript, можно набрать следующие URL-адреса в адресном поле веб-браузера:

```
javascript:5%2  
javascript:x = 3; (x < 5)? "значение x меньше": "значение x больше"  
javascript:d = new Date(); typeof d;  
javascript:for(i=0,j=1,k=0,fib=1; i<5; i++,fib=j+k,k=j,j=fib) alert(fib);  
javascript:s=""; for(i in navigator) s+=i+":"+navigator[i]+\n; alert(s);
```

В веб-браузере Firefox однострочные сценарии вводятся в JavaScript-консоли, доступ к которой можно получить из меню Инструменты. Просто введите выражение или инструкцию, которую требуется проверить. При использовании JavaScript-консоли спецификатор псевдопротокола (javascript:) можно опустить.

### Тестирование и дебаг

Не любой код, написанный вами при изучении JavaScript, будет работать так, как ожидается, и вам захочется его отладить. Базовая методика отладки JavaScript-кода совпадает с методикой для многих других языков: вставка в код инструкций, которые будут выводить значения нужных переменных так, чтобы можно было понять, что же на самом деле происходит. Как мы уже видели, иногда для этих целей можно использовать метод document.write() или alert().

Также существуют методы тестирования кода. После завершения разработки новой функции (возможен вариант написания тестов и до начала разработки) девелопер пишет специальный код для тестирования своего кода. В коде для тестирования нужно сымитировать различные ситуации и возвращаемые значения.

QUnit пользуется особой популярностью среди JavaScript разработчиков. Во-первых, она отлично документирована и проста в использовании, а во-вторых она создана авторами jQuery. Библиотека подходит как для тестирования кода, созданного на базе jQuery, так и нативного JavaScript.

Альтернативы QUnit.js: Buster.JS, Capybara, Mocha, FuncUnit, Hiro, Jasmine, Laika, RobotFramework, TapeDeck, Casper.js

Если же речь идет о работе с Node.js, то с помощью команды node-debug filename Вы сможете дебажить программу с помощью консольной утилиты Node Debugger.

Также можно установить отдельно Node Inspector для более удобного пользования.

### Конкуренты JavaScript

- **VBScript**

Visual Basic Scripting Edition (или просто VBScript) — это язык программирования от компании Microsoft, предназначенный для создания сценариев (скриптов). Он является подмножеством языка Visual Basic и широко используется при создании административных сценариев в системе Windows.

- **JScript**

JScript — это язык программирования от компании Microsoft. Он предназначен для создания сценариев и является реализацией стандарта ECMAScript.

- **Python**

Python (питон) — интерпретируемый, объектно-ориентированный язык программирования высокого уровня. Он поддерживает классы, модули (которые могут быть объединены в пакеты), обработку исключений, а также многопоточную обработку.

- **Tcl**

Tcl (Tool Command Language) — интерпретируемый язык программирования высокого уровня.

- **Ruby**

Ruby — интерпретируемый скриптовый язык высокого уровня для быстрого и удобного объектно-ориентированного программирования.

- **PHP**

PHP (пи-эйч-пи) — интерпретируемый скриптовый язык программирования, созданный для генерации HTML-страниц на веб-сервере и работы с базами данных

- **Perl**

Perl — интерпретируемый скриптовый язык программирования, один из самых распространённых в области веб-программирования.

### Языки на базе JavaScript

- **CoffeeScript**

CoffeeScript это небольшой язык, который транслируется в Javascript. Рубистам он кажется похожим на руби, питонистам он похож на питон, и конечно же, он похож на яваскрипт. CoffeeScript старается упростить использование яваскрипта, сохранив все его сильные стороны.

- **Dart**

Dart это объектно-ориентированный язык с полноценной системой классов, лексическими скоупами, замыканиями, и опционально — статической типизацией. Dart помогает создавать структурированные веб приложения и легок в изучении для широкого круга разработчиков.

- **TypeScript**

TypeScript отличается от JavaScript возможностью явного определения типов (статическая типизация), поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей.

- ClojureScript  
ClojureScript — это расширение языка Clojure, с возможностью компиляции в Javascript.  
Напоминает Lisp.
  - Opal  
Компилятор из Ruby в Javascript.
  - IcedCoffeeScript  
IcedCoffeeScript это надстройка над CoffeeScript, упрощающая контроль за асинхронными операциями. Вместо колбеков вводятся два новых оператора: await и defer.
  - LiveScript  
Ещё один форк от CoffeeScript. Добавляет поддержку функционального стиля программирования, а также вводит небольшие улучшения в текущую ООП-модель
  - Kaffeine  
Расширяет синтаксис яваскрипта, не изобретая ещё один язык программирования. Код на Kaffeine строка к строке соответствует скомпилированому яваскрипту коду. Данная фича должна существенно упростить отладку приложения.
  - Roy  
Экспериментальный язык программирования, преобразующий код в Javascript. Включает в себя возможности статичных функциональных языков.

Необычное поведение языка JavaScript лишь таковым кажется

Некоторые примеры необычного поведения языка:

```
> '5' - 3
2           // weak typing + implicit conversions * headaches
> '5' + 3
'53'        // Because we all love consistency
> '5' - '4'
1           // string - string * integer.
> '5' + + '5'
'55'
> 'foo' + + 'foo'
'fooNaN' // Marvelous.
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '-2'
'52'        // Apparently it's ok

> var x * 3;
> '5' + x - x
50
> '5' - x + x
5
```

```

NaN           ===      NaN; //          false
typeof NaN; // type Number

0.1+          0.2        !==      0.3          (0.3000000000000004)
'3'-+--+'1'+1'/'3'*'6+'2'

```

Операции выполняются в порядке приоритета:

- $'1' / '3' = 0.3333333333333333$
- $0.3333333333333333 * '6' = 2$
- $'3'-+--+'1' = 2$
- $2 + 2 = 4$
- $4 + '2' = '42'$

Также для проверки является ли переменная NaN существует функция isNaN, но она работает не до конца корректно. С самых ранних версий функции isNaN её поведение для не числовых переменных или литералов было довольно-таки запутанным. Когда аргументом функции isNaN является переменная, тип которой не Number, она преобразуется к типу Number. К примеру, если мы возьмем строку var hello = «Hello»; и проверим isNaN(hello), то получим true. Но на самом деле, hello имеет тип String, но из-за неудавшегося преобразования к числу, получаем соответствующий результат. Но если мы возьмем пустую строку: isNaN(""), то получим false, так как пустая строка может быть преобразована к числу 0. Аналогично функция возвращает false, если строка состоит из числа. Даже isNaN('NaN') возвращает true. Эта функция является одной из ошибок логики языка JavaScript. От этого программисты ее просто не используют, а значение NaN проверяется из того, что NaN != NaN.

Больше странностей JS можно обнаружить на ресурсе:

<http://wtfjs.com/>

## Перспективы на 2015-2016 годы

- Новый стандарт ECMAScript 6. Утверждение, реализация в браузерах, адаптация в сообществе и фреймворках.

Новый стандарт несет давно ожидаемые возможности, которые существенно облегчат создание сложных решений: классы, модули, коллекции, итераторы, генераторы, прокси, типизированные массивы, обещания, новые методы и свойства для стандартных объектов и новые синтаксические возможности и еще много чего.

- Рост использования TypeScript в реальных проектах, развитие альтернативных проектов и их взаимное обогащение.

TypeScript – разработка компании Microsoft.

TypeScript является надмножеством JavaScript. То есть любой корректный код на JavaScript также является корректным кодом на TypeScript.

TypeScript использует статическую типизацию, то есть все типы проверяются при компиляции. Кстати сам компилятор TypeScript написан на TypeScript и является open source.

TypeScript добавляет возможность объявлять модули, классы и интерфейсы. Это позволяет масштабировать разработку сложных JavaScript приложений.

На выходе получается обычный JavaScript, который не требует дополнительных библиотек или специальных компонентов.

С TypeScript можно работать на Visual Studio, при несовпадении типов компилятор будет «ругаться».

Код TypeScript является открытым.

```
// TypeScript
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

Прелест TypeScript в том, что, пока вы пишите код, вы получаете возможность удобно описывать сложные структуры данных, а компилятор при этом помогает вам отслеживать, что вы нигде ничего не напутали и правильно работаете с типами. Еще одно замечательное свойство TS, точнее его компилятора (который, кстати, открыт также, как и сам язык!), состоит в том, что в результате компиляции получается чистый код на JavaScript, причем, примерно такой, какой вы бы и сами написали, следуя современным практикам:

```
// TypeScript to JavaScript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
```

Таким образом, на выходе получается код, работающий в современных браузерах на любых операционных системах. К слову, под Node.js тоже можно писать на TypeScript.

<http://www.typescriptlang.org/>

- Развитие инструментов для кроссплатформенной разработки на JS, продолжение стирания границ между сайтами и приложениями.

В первой задаче критичным является стремление научить веб-сайты делать то, что умеют делать приложения: например, интегрироваться в операционную систему – от иконок и живых плиток до пуш-уведомлений и поддержки локальных контрактов. При этом важным

остается система обновления содержимого таких приложений через веб-сайт, что позволяет сохранять преимущества веб-подхода.

- Рост умных телевизоров и консолей с разработкой на JavaScript, нативная разработка на JS на многих современных платформах (но не всех).
- Развитие API доступа к нативным возможностям устройства из JavaScript, адаптация NUI в JS. (Затягивается на несколько лет.)

С самыми базовыми вещами вроде геолокации или ориентации устройства мы уже научились работать, но впереди большая работа по стандартизации и реализации в движках браузеров большого блока возможностей, доступных в случае нативной разработки, но, как правило, неподвластных в случае разработки для браузера:

- Вибрация
- Статус батареи
- Сенсоры (например, света)
- Камера и микрофон
- и др.
- Новые переработанные версии популярных библиотек, повышение входного порога для создания комплексных фреймворков, нишевые решения на базе ES6.
- Адаптацию веб-компонент браузерами, принятие новых технологий разработчиками элементов управления и различных фреймворков.
- Принятие менеджеров пакетов и систем сборки для JavaScript в корпоративной и учебной среде, интеграция в популярные инструменты веб-разработки.
- Развитие графических библиотек на JS, показательная адаптация новых технологий крупными или заметными игроками рынка (игры и интерактивный контент — основные драйверы).
- Unity 5 с рендерингом в WebGL, развитие 3d и игровых библиотек, потенциальный прорыв через социальные сети.
- Адаптация Node.js в корпоративной среде, адаптация нового ES6 в самом Node.js. Запасаемся попкорном и смотрим историю с ответвлением io.js.

Кстати, в конце 2015 нас ждет большой праздник — 20-летие JavaScript ☺

## Трансляторы

### Общие понятия

**Транслятор** — программа или техническое средство, выполняющее *трансляцию программы*.

**Трансляция программы** — преобразование программы, представленной на одном из языков программирования, в программу на другом языке и, в определённом смысле, равносильную первой.

Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Язык, на котором представлена входная программа, называется *исходным языком*, а сама программа — *исходным кодом*. Выходной язык называется *целевым языком*, а выходная (*результатирующая*) программа — *объектным кодом*.

В общем случае, понятие трансляции относится не только к языкам программирования, но и к другим языкам — как формальным компьютерным (вроде языков разметки типа HTML), так и естественным (русскому, английскому и т. п.)

### Виды трансляторов

Существует несколько видов трансляторов:

- *Диалоговый* транслятор — транслятор, обеспечивающий использование языка программирования в режиме разделения времени.
- *Синтаксически-ориентированный* (*синтаксически-управляемый*) транслятор — транслятор, получающий на вход описание синтаксиса и семантики языка, текст на описанном языке и выполняющий трансляцию в соответствии с заданным описанием.
- *Однопроходной* транслятор — транслятор, создающий объектный модуль при однократном последовательном чтении исходного кода (за один проход).
- *Многопроходной* транслятор — транслятор, создающий объектный модуль после нескольких чтений исходного кода (за несколько проходов).
- *Оптимизирующий* транслятор — транслятор, выполняющий оптимизацию создаваемого кода перед записью в объектный файл.
- *Тестовый* транслятор — транслятор, получающий на вход исходный код и выдающий на выходе изменённый исходный код. Запускается перед основным транслятором для добавления в исходный код отладочных процедур. Например, транслятор с языка ассемблера может выполнять замену макрокоманд на код.
- *Обратный* транслятор — транслятор, выполняющий преобразование машинного кода в текст на каком-либо языке программирования.

### Реализации

Цель трансляции — преобразование текста с одного языка на язык, понятный адресату. При трансляции компьютерной программы адресатом может быть:

- устройство — процессор (трансляция называется *компиляцией*);
- программа — интерпретатор (трансляция называется *интерпретацией*).

### Виды трансляции

- компиляция;
- интерпретация;
- динамическая компиляция.

### Компилятор

**Компиляция** — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера). Входной информацией для компилятора (исходный код) является описание алгоритма или программа на объектно-ориентированном языке, а на выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

### *Виды компиляторов*

- *Векторизующий*. Транслирует исходный код в машинный код компьютеров, оснащённых векторным процессором.
- *Гибкий*. Сконструирован по модульному принципу, управляет табличами и запрограммирован на языке высокого уровня или реализован с помощью компилятора компиляторов.
- *Диалоговый*.
- *Инкрементальный*. Повторно транслирует фрагменты программы и дополнения к ней без перекомпиляции всей программы.
- *Интерпретирующий (пошаговый)*. Последовательно выполняет независимую компиляцию каждого отдельного оператора (команды) исходной программы.
- *Компилятор компиляторов*. Транслятор, воспринимающий формальное описание языка программирования и генерирующий компилятор для этого языка.

Синтаксис выражается в виде Форма Бэкуса — Наура (формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. БНФ используется для описания контекстно-свободных формальных грамматик, Пример: от пример БНФ-конструкции, описывающей правильные скобочные последовательности: `<правпосл> ::= <пусто> | (<правпосл>) | <правпосл><правпосл>` или её производной и должен удовлетворять правилам того метода синтаксического анализа, который будет использоваться в генерируемом компиляторе.

Семантика языка обычно описывается путём ассоциирования процедуры генерации кода с каждой синтаксической конструкцией, причём необходимая процедура вызывается всякий раз, когда соответствующая конструкция распознаётся программой синтаксического анализа. Таким образом, пользователю компилятора компиляторов в любом случае нужно разработать исполняющие структуры и выбрать способ преобразования каждой входной синтаксической конструкции в операции выходного языка или в машинные операции, после чего нужно написать собственно процедуры генерации кода. Следовательно, компилятор компиляторов — это полезное средство, помогающее писать компиляторы, но не более того.

### *Компилятор компиляторов Bison*

Bison – это GNU-выпуск известной программы YACC, предназначеннной для порождения компиляторов по описанной пользователем КС-грамматике.

- *Отладочный*. Устраняет отдельные виды синтаксических ошибок.
- *Резидентный*. Постоянно находится в оперативной памяти и доступен для повторного использования многими задачами.
- *Самокомпилируемый*. Написан на том же языке, с которого осуществляется трансляция. Метод создания транслятора для некоторого языка программирования, при котором транслятор пишется на том же языке программирования; создание транслятором исполняемых файлов из исходного кода самого транслятора. Используется для переноса трансляторов на новые платформы. Появился в середине 1950-х годов. Позволяет создать

транслятор, который генерирует сам себя. Применялся для создания трансляторов многих языков программирования, включая языки BASIC, Алгол, Си, Паскаль, ПЛ/1, Factor, Haskell, Modula-2, Oberon, OCaml, Common Lisp, Scheme, Java, Python, Scala, Nemerle и другие.

- **Универсальный.** Основан на формальном описании синтаксиса и семантики входного языка. Составными частями такого компилятора являются: ядро, синтаксический и семантический загрузчики

### *Виды компиляции*

- **Пакетная.** Компиляция нескольких исходных модулей в одном пункте задания.
- **Построчная.** То же, что и интерпретация.
- **Условная.** Компиляция, при которой транслируемый текст зависит от условий, заданных в исходной программе директивами компилятора. Так, в зависимости от значения некоторой константы, можно включать или выключать трансляцию части текста программы.

### *Процесс компиляции*

Процесс компиляции состоит из следующих этапов:

- **Лексический анализ.** На этом этапе последовательность символов исходного файла преобразуется в последовательность лексем.
- **Синтаксический (грамматический) анализ.** Последовательность лексем преобразуется в дерево разбора.
- **Семантический анализ.** Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением/кодом», и может быть дополненным деревом разбора, новым деревом, абстрактным набором команд или чем-то ещё, удобным для дальнейшей обработки.
- **Оптимизация.** Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах — например, над промежуточным кодом или над конечным машинным кодом.
- **Генерация кода.** Из промежуточного представления порождается код на целевом языке.

В конкретных реализациях компиляторов эти этапы могут быть разделены или, наоборот, совмещены в том или ином виде.

### *Генерация кода*

#### *Генерация машинного кода*

Большинство компиляторов переводит программу с некоторого высокоуровневого языка программирования в машинный код, который может быть непосредственно выполнен процессором. Как правило, этот код также ориентирован на исполнение в среде конкретной операционной системы, поскольку использует предоставляемые ею возможности (системные вызовы, библиотеки функций). Архитектура (набор программно-аппаратных средств), для которой производится компиляция, называется *целевой машиной*.

Результат компиляции — исполимый модуль — обладает максимальной возможной производительностью, однако привязан к определённой операционной системе и процессору (и не будет работать на других).

Для каждой целевой машины (IBM, Apple, Sun и т. д.) и каждой операционной системы или семейства операционных систем, работающих на целевой машине, требуется написание своего компилятора. Существуют также так называемые *кросс-компиляторы*, позволяющие на одной машине и в среде одной ОС генерировать код, предназначенный для выполнения на другой целевой машине и/или в среде другой ОС. Кроме того, компиляторы могут оптимизировать код под разные модели из одного семейства процессоров (путём поддержки специфичных для этих моделей особенностей или расширений наборов инструкций). Например, код, скомпилированный под процессоры семейства Pentium, может учитывать особенности распараллеливания инструкций и использовать их специфичные расширения — MMX, SSE и т. п.

Некоторые компиляторы переводят программу с языка высокого уровня не прямо в машинный код, а на язык ассемблера (примером может служить PureBasic, транслирующий бейсик-код в ассемблер FASM). Это делается для упрощения части компилятора, отвечающей за кодогенерацию, и повышения его переносимости (задача окончательной генерации кода и привязки его к требуемой целевой платформе перекладывается на ассемблер), либо для возможности контроля и исправления результата компиляции программистом.

#### [Генерация байт-кода](#)

Результатом работы компилятора может быть программа на специально созданном низкоуровневом языке, подлежащем интерпретации *виртуальной машиной*. Такой язык называется псевдокодом или байт-кодом. Как правило, он не является машинным кодом какого-либо компьютера и программы на нём могут исполняться на различных архитектурах, где имеется соответствующая виртуальная машина, но в некоторых случаях создаются аппаратные платформы, напрямую поддерживающие псевдокод какого-либо языка. Например, псевдокод языка Java называется байт-кодом Java и выполняется в Java Virtual Machine, для его прямого исполнения была создана спецификация процессора picoJava. Для платформы .NET Framework псевдокод называется Common Intermediate Language (CIL), а среда исполнения — Common Language Runtime (CLR).

Некоторые реализации интерпретируемых языков высокого уровня (например, Perl) используют байт-код для оптимизации исполнения: затратные этапы синтаксического анализа и преобразование текста программы в байт-код выполняются один раз при загрузке, затем соответствующий код может многократно использоваться без промежуточных этапов.

#### [Динамическая компиляция](#)

Из-за необходимости интерпретации байт-код выполняется значительно медленнее машинного кода сравнимой функциональности, однако он более переносим (не зависит от операционной системы и модели процессора). Чтобы ускорить выполнение байт-кода, используется *динамическая компиляция*, когда виртуальная машина транслирует псевдокод в машинный код непосредственно перед его первым исполнением (и при повторных обращениях к коду исполняется уже скомпилированный вариант).

CIL-код также компилируется в код целевой машины JIT-компилятором, а библиотеки .NET Framework компилируются заранее.

### *Декомпиляция*

Существуют программы, которые решают обратную задачу — перевод программы с низкоуровневого языка на высокоуровневый. Этот процесс называют декомпиляцией, а такие программы — декомпиляторами. Но поскольку компиляция — это процесс с потерями, точно восстановить исходный код, скажем, на C++, в общем случае невозможно. Более эффективно декомпилируются программы в байт-кодах — например, существует довольно надёжный декомпилятор для Flash. Разновидностью декомпилирования является дизассемблирование машинного кода в код на языке ассемблера, который почти всегда выполняется успешно (при этом сложность может представлять самомодифицирующийся код или код, в котором собственно код и данные не разделены). Связано это с тем, что между кодами машинных команд и командами ассемблера имеется практически взаимно-однозначное соответствие.

### *Раздельная компиляция*

Трансляция частей программы по отдельности с последующим объединением их компоновщиком в единый загрузочный модуль.

Исторически особенностью компилятора, отражённой в его названии (англ. *compile* — собирать вместе, составлять), являлось то, что он производил как трансляцию, так и компоновку, при этом компилятор мог порождать сразу абсолютный код. Однако позже, с ростом сложности и размера программ (и увеличением времени, затрачиваемого на перекомпиляцию), возникла необходимость разделять программы на части и выделять библиотеки, которые можно компилировать независимо друг от друга. При трансляции каждой части программы компилятор порождает объектный модуль, содержащий дополнительную информацию, которая потом, при компоновке частей в исполняемый модуль, используется для связывания и разрешения ссылок между частями.

Появление раздельной компиляции и выделение компоновки как отдельной стадии произошло значительно позже создания компиляторов. В связи с этим вместо термина «компилятор» иногда используют термин «транслятор» как его синоним: либо в старой литературе, либо когда хотят подчеркнуть его способность переводить программу в машинный код (и наоборот, используют термин «компилятор» для подчёркивания способности собирать из многих файлов один).

### *Интерпретатор*

**Интерпретатор** — программа (разновидность транслятора), выполняющая *интерпретацию*.

**Интерпретация** — пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса (в отличие от компиляции, при которой программа транслируется без её выполнения).

### *Типы интерпретаторов*

- **Простой интерпретатор** анализирует и тут же выполняет (собственно интерпретация) программу покомандно (или построчно), по мере поступления её исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды (или строки) с ошибкой.
- **Интерпретатор компилирующего типа** — это система из компилятора, переводящего исходный код программы в промежуточное представление, например, в байт-код, и

собственно интерпретатора, который выполняет полученный промежуточный код (так называемая виртуальная машина). Достоинством таких систем является большее быстродействие выполнения программ (за счёт выноса анализа исходного кода в отдельный, разовый проход, и минимизации этого анализа в интерпретаторе). Недостатки — большее требование к ресурсам и требование на корректность исходного кода. Применяется в таких языках, как Java, PHP, Tcl, Perl, REXX (сохраняется результат парсинга исходного кода<sup>1</sup>), а также в различных СУБД.

В случае разделения интерпретатора компилирующего типа на компоненты получаются компилятор языка и простой интерпретатор с минимизированным анализом исходного кода. Причём исходный код для такого интерпретатора не обязательно должен иметь текстовый формат или быть байт-кодом, который понимает только данный интерпретатор, это может быть машинный код какой-то существующей аппаратной платформы. К примеру, виртуальные машины вроде QEMU, Bochs, VMware включают в себя интерпретаторы машинного кода процессоров семейства x86.

Некоторые интерпретаторы (например, для языков Лисп, Scheme, Python, Бейсик и других) могут работать в режиме диалога или так называемого цикла чтения-вычисления-печати (англ. *read-eval-print loop, REPL*). В таком режиме интерпретатор считывает законченную конструкцию языка (например, s-expression в языке Лисп), выполняет её, печатает результаты, после чего переходит к ожиданию ввода пользователем следующей конструкции.

Уникальным является язык Forth, который способен работать как в режиме интерпретации, так и компиляции входных данных, позволяя переключаться между этими режимами в произвольный момент, как во время трансляции исходного кода, так и во время работы программ.

Следует также отметить, что режимы интерпретации можно найти не только в программном, но и аппаратном обеспечении. Так, многие микропроцессоры интерпретируют машинный код с помощью встроенных микропрограмм, а процессоры семейства x86, начиная с Pentium (например, на архитектуре Intel P6), во время исполнения машинного кода предварительно транслируют его во внутренний формат (в последовательность микроопераций).

#### *Алгоритм работы простого интерпретатора*

1. прочитать инструкцию;
2. проанализировать инструкцию и определить соответствующие действия;
3. выполнить соответствующие действия;
4. если не достигнуто условие завершения программы, прочитать следующую инструкцию и вернуться к пункту 2.

#### *Достиныства и недостатки*

##### *Достиныства*

- Большая переносимость интерпретируемых программ — программа будет работать на любой платформе, на которой есть соответствующий интерпретатор.
- Как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах.
- Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.

## Недостатки

- Интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. Сам интерпретатор при этом может быть очень компактным.
- Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени в сравнении с непосредственным исполнением машинного кода, в который мог бы быть скомпилирован исходный код.
- Практически отсутствует оптимизация кода, что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

## Динамический компилятор

**JIT-компиляция** (англ. *Just-in-time compilation*, компиляция «на лету»), **динамическая компиляция** (англ. *dynamic translation*) — технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы. Таким образом достигается высокая скорость выполнения по сравнению с интерпретируемым байт-кодом (сравнимая с компилируемыми языками) за счёт увеличения потребления памяти (для хранения результатов компиляции) и затрат времени на компиляцию. JIT базируется на двух более ранних идеях, касающихся среды исполнения: *компиляции байт-кода* и *динамической компиляции*.

Так как JIT-компиляция является, по сути, одной из форм динамической компиляции, она позволяет применять такие технологии, как адаптивная оптимизация и динамическая рекомпиляция. Из-за этого JIT-компиляция может показывать лучшие результаты в плане производительности, чем статическая компиляция. Интерпретация и JIT-компиляция особенно хорошо подходят для динамических языков программирования, при этом среда исполнения справляется с поздним связыванием типов и гарантирует безопасность исполнения.

Проекты LLVM, GNU Lightning, libJIT (часть проекта DotGNU) и RPython (часть проекта PyPy) могут быть использованы для создания JIT интерпретаторов любого скриптового языка.

## Особенности реализации

JIT-компиляция может быть применена как ко всей программе, так и к её отдельным частям. Например, текстовый редактор может на лету компилировать регулярные выражения для более быстрого поиска по тексту. С AOT-компиляции (компиляция перед исполнением) такое сделать не представляется возможным, так как данные предоставляются во время исполнения программы, а не во время компиляции. JIT используется в реализациях Java, JavaScript, .NET Framework, в одной из реализаций Python — PyPy. Существующие наиболее распространённые интерпретаторы языков PHP, Ruby, Perl, Python и им подобных, которые имеют ограниченные или неполные JIT.

Большинство реализаций JIT имеют последовательную структуру: сначала (AOT) приложение компилируется в байт-код виртуальной машины среды исполнения, а потом JIT компилирует байт-код непосредственно в машинный код. В итоге приложение подтормаживает при запуске, что, впоследствии, компенсируется более быстрой его работой.

## Описание

В языках, таких как Java, PHP, C#, Lua, Perl, GNU CLISP, исходный код транслируется в одно из промежуточных представлений, называемое байт-кодом. Байт-код не является машинным кодом

какого-либо конкретного компьютера и может переноситься на различные компьютерные архитектуры и исполняться точно так же. JIT читает байт-код из некоторых секторов (редко сразу из всех) и компилирует их в машинный код. Этим сектором может быть файл, функция или любой фрагмент кода. Однажды скомпилированный код может кэшироваться и в дальнейшем повторно использоваться без перекомпиляции.

Динамически компилируемая среда — это среда, в которой компилятор может вызываться приложением во время выполнения. Например, большинство реализаций Common Lisp содержат функцию `compile`, которая может создать функцию во время выполнения; в Python это функция `eval`. Это удобно для программиста, так как он может контролировать, какие части кода действительно подлежат компиляции. Также с помощью этого приёма можно компилировать динамически генерированный код, что в некоторых случаях приводит даже к лучшей производительности, чем реализация в статически скомпилированном коде. Однако стоит помнить, что подобные функции могут быть опасны, особенно когда данные передаются из недоверенных источников.

Основная цель использования JIT — достичь и превзойти производительность статической компиляции, сохранив при этом преимущества динамической компиляции:

- Большинство тяжеловесных операций, таких как парсинг исходного кода и выполнение базовых оптимизаций происходит во время компиляции (до развёртывания), в то время как компиляция в машинный код из байт-кода происходит быстрее, чем из исходного кода
- Байт-код более переносим (в отличие от машинного кода)
- Среда может контролировать выполнение байт-кода после компиляции, поэтому приложение может быть запущено в песочнице (стоит отметить, что для нативных программ такая возможность тоже существует, но реализация данной технологии сложнее)
- Компиляторы из байт-кода в машинный код легче в реализации, так как большинство работы по оптимизации уже было проделано компилятором

JIT, как правило, эффективней, чем интерпретация кода. К тому же в некоторых случаях JIT может показывать большую производительность по сравнению со статической компиляцией за счёт оптимизаций, возможных только во время исполнения:

1. Компиляция может осуществляться непосредственно для целевого CPU и операционной системы, на которой запущено приложение. Например, JIT может использовать векторные SSE2 расширения процессора, если он обнаружит их поддержку. Однако, до сих пор нет основных реализаций JIT, где этот подход бы использовался, ведь чтобы обеспечить подобный уровень оптимизации, сравнимый со статическими компиляторами, потребовалось бы либо поддерживать бинарный файл под каждую платформу, либо включать в одну библиотеку оптимизаторы под каждую платформу.
2. Среда может собирать статистику о работающей программе и производить оптимизации с учётом этой информации. Некоторые статические компиляторы также могут принимать на вход информацию о предыдущих запусках приложения.
3. Среда может делать глобальные оптимизации кода (например, встраивание библиотечных функций в код) без потери преимуществ динамической компиляции и без накладных расходов, присущим статическим компиляторам и линкерам.
4. Более простое пристраивание кода для лучшего использования кэша

### *Задержка при запуске, средства борьбы с ней*

Типичная причина задержки при запуске JIT-компилятора — расходы на загрузку среды и компиляцию приложения в байт-код. В общем случае, чем лучше и чем больше оптимизаций выполняет JIT, тем дольше получается задержка. Поэтому разработчикам JIT приходится искать компромисс между качеством генерируемого кода и временем запуска. Однако, часто оказывается так, что узким местом в процессе компиляции оказывается не сам процесс компиляции, а задержки системы ввода вывода (так, например, *rt.jar* в Java Virtual Machine (JVM) имеет размер 40МВ, и поиск метаданных в нём занимает достаточно большое количество времени).

Ещё одно средство оптимизации — компилировать только те участки приложения, которые используются чаще всего. Этот подход реализован в PyPy и Sun's HotSpot Java Virtual Machine.

В качестве эвристики может использоваться счётчик запусков участков приложения, размер байт-кода или детектор циклов.

Порой достаточно сложно найти правильный компромисс. Так, например, Sun's Java Virtual Machine имеет два режима работы — клиент и сервер. В режиме клиента количество компиляций и оптимизаций минимально для более быстрого запуска, в то время как в режиме сервера достигается максимальная производительность, но из-за этого увеличивается время запуска.

Ещё одна техника, называемая pre-JIT, компилирует код до запуска. Преимуществом данной техники является ускоренное время запуска, в то время недостатком является плохое качество скомпилированного кода по сравнению с runtime JIT.

### *История*

Самую первую реализацию JIT можно отнести к LISP, написанную McCarthy in 1960<sup>[5]</sup>. В его книге *Recursive functions of symbolic expressions and their computation by machine, Part I*, он упоминает функции, компилируемые во время выполнения, тем самым избавив от необходимости вывода работы компилятора на перфокарты.

Другой ранний пример упоминания JIT можно отнести к Кену Томпсону, который в 1968 году впервые применил регулярные выражения для поиска подстрок в текстовом редакторе QED. Для ускорения алгоритма Томпсон реализовал компиляцию регулярных выражений в машинный код IBM 7094.

Важный метод получения скомпилированного кода был предложен Митчелом в 1970 году, когда он реализовал экспериментальный язык *LC<sup>2</sup>*.

Smalltalk (1983) был пионером в области JIT-технологий. Трансляция в машинный код выполнялась по требованию и кэшировалась для дальнейшего использования. Когда память кончалась, система могла удалить некоторую часть закэшированного кода из оперативной памяти и восстановить его, когда он снова потребуется. Язык программирования Self некоторое время был самой быстрой реализацией Smalltalk-а и работал всего-лишь в два раза медленней C, будучи полностью объектно-ориентированным.

Self был заброшен Sun, но исследования продолжились в рамках языка Java. Термин «Just-in-time компиляция» был заимствован из производственного термина «Точно в срок» и популяризован

Джеймсом Гослингом, использовавшим этот термин в 1993. В данный момент JIT используется почти во всех реализациях Java Virtual Machine.

Также большой интерес представляет диссертация, защищённая в 1994 году в Университете ETH (Швейцария, Цюрих) Михаэлем Францем «Динамическая кодогенерация — ключ к переносимому программному обеспечению» и реализованная им система Juice динамической кодогенерации из переносимого семантического дерева для языка Оберон. Система Juice предлагалась как плагин для интернет-браузеров.

#### *Безопасность*

Так как JIT составляет исполняемый код из данных, возникает вопрос безопасности и возможных уязвимостей.

JIT компиляция включает в себя компиляцию исходного кода или байт-кода в машинный код и его выполнение. Как правило, результат записывается в память и исполняется сразу же, не используя диск и не вызывая код как отдельную программу. В современных архитектурах для повышения безопасности произвольные участки памяти не могут быть исполнены. Для корректной работы память должна быть помечена, как исполняемая (NX bit); для большей безопасности флаг должен быть поставлен *после* загрузки кода в память, а эта память должна быть помечена как доступная только для чтения, так как перезаписываемая и исполняемая память есть ничто иное, как дыра в безопасности.

## Глава 5. Базы данных

### Реляционные базы данных. SQL

**Реляционная база данных** — база данных, основанная на реляционной модели данных. Слово «реляционный» происходит от англ. *relation* («отношение», «зависимость», «связь»). Для работы с реляционными БД применяют реляционные СУБД.

Использование реляционных баз данных было предложено доктором Коддом из компании IBM в 1970 году.

#### Нормализация

Целью нормализации реляционной базы данных является устранение недостатков структуры базы данных, приводящих к избыточности, которая, в свою очередь, потенциально приводит к различным аномалиям и нарушениям целостности данных.

Теоретики реляционных баз данных в процессе развития теории выявили и описали типичные примеры избыточности и способы их устранения.

#### 1ая нормальная форма

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия *отношение*. Что же касается различных *таблиц*, то они могут не быть правильными *представлениями отношений* и, соответственно, могут не находиться в 1НФ.

Сотрудник	Номер телефона
Иванов И. И.	283-56-82 390-57-34
Петров П. П.	708-62-34

Сотрудник	Номер телефона
Иванов И. И.	283-56-82
Иванов И. И.	390-57-34
Петров П. П.	708-62-34

#### 2ая нормальная форма

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут неприводимо (функционально полно) зависит от ее потенциального ключа.

Сотрудник	Должность	Зарплата	Наличие компьютера
Гришин	Кладовщик	20000	Нет
Васильев	Программист	40000	Есть
Иванов	Кладовщик	25000	Нет

<b>Сотрудник</b>	<b>Должность</b>	<b>Зарплата</b>
Гришин	Кладовщик	20000
Васильев	Программист	40000
Иванов	Кладовщик	25000

<b>Должность</b>	<b>Наличие компьютера</b>
Кладовщик	Нет
Программист	Есть

(Наличие компьютера полностью зависит от должности)

### *Зая нормальная форма*

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых. (По сути повторяет вторую нормальную форму).

### Триггер

**Триггер** (англ. *trigger*) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением `INSERT`, удалением `DELETE` строки в заданной таблице, или изменением `UPDATE` данных в определенном столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

### SQL

**SQL** (*structured query language* — «язык структурированных запросов») — формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД).

SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL можно назвать языком программирования, при этом он не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

- создание в базе данных новой таблицы;
- добавление в таблицу новых записей;
- изменение записей;

- удаление записей;
- выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
- изменение структур таблиц.

Со временем SQL усложнился — обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) — и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся единственным механизмом связи между прикладным программным обеспечением и базой данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

Каждое предложение SQL — это либо **запрос** данных из базы, либо обращение к базе данных, которое приводит к изменению данных в базе. В соответствии с тем, какие изменения происходят в базе данных, различают следующие типы запросов:

- запросы на создание или изменение в базе данных новых или существующих объектов (при этом в запросе описывается тип и структура создаваемого или изменяемого объекта);
- запросы на получение данных;
- запросы на добавление новых данных (записей);
- запросы на удаление данных;
- обращения к СУБД.

Основным объектом хранения реляционной базы данных является таблица, поэтому все SQL-запросы — это операции над таблицами. В соответствии с этим, запросы делятся на:

- запросы, оперирующие самими таблицами (создание и изменение таблиц);
- запросы, оперирующие с отдельными записями (или строками таблиц) или наборами записей.

Каждая таблица описывается в виде перечисления своих полей (столбцов таблицы) с указанием

- типа хранимых в каждом поле значений;
- связей между таблицами (задание первичных и внешних ключей);
- информации, необходимой для построения индексов.

Запросы первого типа в свою очередь делятся на запросы, предназначенные для создания в базе данных новых таблиц, и на запросы, предназначенные для изменения уже существующих таблиц. Запросы второго типа оперируют со строками, и их можно разделить на запросы следующего вида:

- вставка новой строки;
- изменение значений полей строки или набора строк;
- удаление строки или набора строк.

Самый главный вид запроса — это запрос, возвращающий (пользователю) некоторый набор строк, с которым можно осуществить одну из трёх операций:

- просмотреть полученный набор;
- изменить все записи набора;
- удалить все записи набора.

Таким образом использование SQL сводится, по сути, к формированию всевозможных выборок строк и совершению операций над всеми записями, входящими в набор.

### [Преимущества](#)

#### **Наличие стандартов**

Наличие стандартов и набора тестов для выявления совместимости и соответствия конкретной реализации SQL общепринятым стандарту только способствует «стабилизации» языка. Правда, стоит обратить внимание, что сам по себе стандарт местами чересчур формализован и раздут в размерах (например, базовая часть стандарта SQL:2003 состоит из более чем 1300 страниц текста).

#### **Декларативность**

С помощью SQL программист описывает только то, какие данные нужно извлечь или модифицировать. То, каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса. Однако не стоит думать, что это полностью универсальный принцип — программист описывает набор данных для выборки или модификации, однако ему при этом полезно представлять, как СУБД будет разбирать текст его запроса. Чем сложнее сконструирован запрос, тем больше он допускает вариантов написания, различных по скорости выполнения, но одинаковых по итоговому набору данных.

#### **Интересный факт**

Хотя SQL и задумывался как средство работы конечного пользователя, в конце концов он стал настолько сложным, что превратился в инструмент программиста.

### [ORM](#)

**ORM** (англ. *object-relational mapping*, рус. *объектно-реляционное отображение*) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

В объектно-ориентированном программировании объекты в программе представляют объекты из реального мира. В качестве примера можно рассмотреть адресную книгу, которая содержит список людей с нулём или более телефонов и нулём или более адресов. В терминах объектно-ориентированного программирования они будут представляться объектами класса «Человек», которые будут содержать следующий список полей: имя, список (или массив) телефонов и список адресов.

Суть задачи состоит в преобразовании таких объектов в форму, в которой они могут быть сохранены в файлах или базах данных, и которые легко могут быть извлечены в последующем, с сохранением свойств объектов и отношений между ними. Эти объекты называют «хранимыми» (англ. *persistent*). Исторически существует несколько подходов к решению этой задачи.

Разработано множество пакетов, устраняющих необходимость в преобразовании объектов для хранения в реляционных базах данных.

Некоторые пакеты решают эту проблему, предоставляя библиотеки классов, способных выполнять такие преобразования автоматически. Имея список таблиц в базе данных и объектов в программе, они автоматически преобразуют запросы из одного вида в другой. В результате запроса объекта «человек» (из примера с адресной книгой) необходимый SQL-запрос будет сформирован и выполнен, а результаты «волшебным» образом преобразованы в объекты «номер телефона» внутри программы.

С точки зрения программиста система должна выглядеть как постоянное хранилище объектов. Он может просто создавать объекты и работать с ними как обычно, а они автоматически будут сохраняться в реляционной базе данных.

На практике всё не так просто и очевидно. Все системы ORM обычно проявляют себя в том или ином виде, уменьшая в некотором роде возможность игнорирования базы данных. Более того, слой транзакций может быть медленным и неэффективным (особенно в терминах генерированного SQL). Все это может привести к тому, что программы будут работать медленнее и использовать больше памяти, чем программы, написанные «вручную».

Но ORM избавляет программиста от написания большого количества кода, часто однообразного и подверженного ошибкам, тем самым значительно повышая скорость разработки. Кроме того, большинство современных реализаций ORM позволяют программисту при необходимости самому жестко задать код SQL-запросов, который будет использоваться при тех или иных действиях (сохранение в базу данных, загрузка, поиск и т. д.) с постоянным объектом.

## Нереляционные базы данных

NoSQL (not only SQL, не только SQL), в информатике — термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL. Применяется к базам данных, в которых делается попытка решить проблемы масштабируемости (англ. scalability) и доступности (англ. availability) за счёт атомарности (англ. atomicity) и согласованности данных (англ. Consistency).

## История названия

Изначально слово NoSQL являлось акронимом из двух слов английского языка: No («Не») и SQL (сокращение от англ. Structured Query Language — «структурированный язык запросов»), что даёт термину смысл «отрицающий SQL». Возможно, что первые, кто стал употреблять этот термин, хотели сказать «No RDBMS» («не реляционная СУБД») или «no relational» («не реляционный»), но NoSQL звучало лучше и в итоге прижилось (в качестве альтернативы предлагалось также NonRel). Позднее для NoSQL было придумано объяснение «Not Only SQL» («не только SQL»). NoSQL стал общим термином для различных баз данных и хранилищ, но он не обозначает какую-либо одну конкретную технологию или продукт.

## Развитие идеи

Сама по себе идея нереляционных баз данных не нова, а использование нереляционных хранилищ началось ещё во времена первых компьютеров. Нереляционные базы данных процветали во

времена мэйнфреймов, а позднее, во времена доминирования реляционных СУБД, нашли применение в специализированных хранилищах, например, иерархических службах каталогов. Появление же нереляционных СУБД нового поколения произошло из-за необходимости создания параллельных распределённых систем для высокомасштабируемых интернет-приложений, таких как поисковые системы.

В начале 2000-х годов Google построил свою высокомасштабируемую поисковую систему и приложения: Gmail, Google Maps, Google Earth и т. п., решая проблемы масштабируемости и параллельной обработки больших объёмов данных. В результате была создана распределённая файловая система и распределённая система координации, хранилище семейств колонок (англ. column family store), среда выполнения, основанная на алгоритме MapReduce. Публикация компанией Google описаний этих технологий привела к всплеску интереса среди разработчиков открытого программного обеспечения, в результате чего был создан Hadoop и запущены связанные с ним проекты, призванные создать подобные Google технологии. Через год, в 2007 году, примеру Google последовал Amazon.com, опубликовав статьи о высокодоступной базе данных Amazon DynamoDB.

Поддержка гигантов индустрии менее чем за пять лет привела к широкому распространению технологий NoSQL (и подобных) для управления «большими данными», а к делу присоединились другие большие и маленькие компании, такие как: IBM, Facebook, Netflix, Ebay, Hulu, Yahoo!, со своими проприетарными и открытыми решениями.

### Основные черты

Традиционные СУБД ориентируются на требования ACID к транзакционной системе: атомарность (англ. atomicity), согласованность (англ. consistency), изолированность (англ. isolation), надёжность (англ. Durability).

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние.

Сильная согласованность (Strong consistency) - после завершения обновления, любой последующий доступ к данным (Процессом A, B или C) вернет обновленное значение.

Изолированность - во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

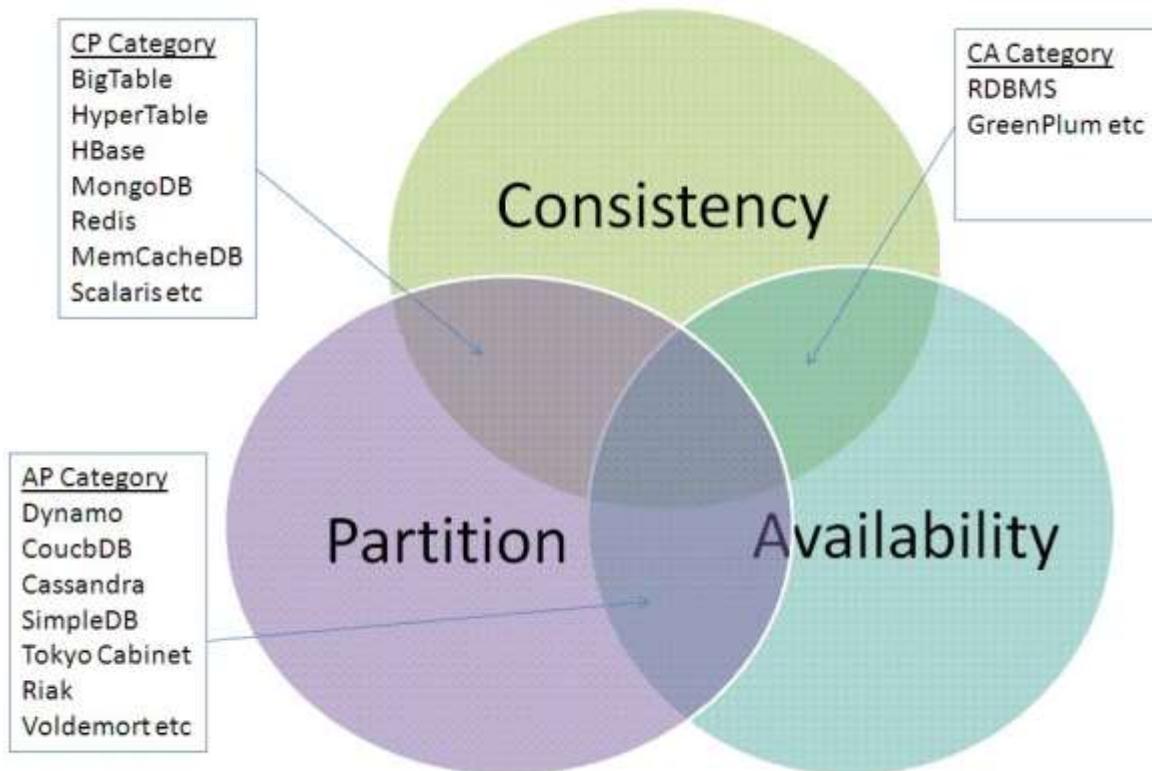
Надежность - независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

В NoSQL вместо ACID может рассматриваться набор свойств BASE:

- базовая доступность (англ. basic availability) — каждый запрос гарантированно завершается (успешно или безуспешно).
- гибкое состояние (англ. soft state) — состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных.
- согласованность в конечном счёте (англ. eventual consistency) — данные могут быть некоторое время рассогласованы, но приходят к согласованию через некоторое время.

При сбое в некоторых узлах системы отказ получает только часть приложений, взаимодействующих с вышедшими из строя узлами. В ходе взаимодействия используются протоколы без состояния, что снижает нагрузку на отдельные узлы и позволяет ее перераспределять. Наконец, допустима временная несогласованность данных в разных узлах системы при условии, что информация будет синхронизирована через некоторый обозримый промежуток времени. BASE используется для наиболее общего описания требований к распределенным NoSQL-системам, подпадающих под утверждение теоремы CAP и не удовлетворяющих требованиям ACID.

Термин «BASE» был предложен Эриком Брюером, автором теоремы CAP, согласно которой в распределённых вычислениях можно обеспечить только два из трёх свойств: согласованность данных, доступность или устойчивость к разделению (англ. partition tolerance — расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций).



Разумеется, системы на основе BASE не могут использоваться в любых приложениях: для функционирования биржевых и банковских систем использование транзакций является необходимостью. В то же время, свойства ACID, какими бы желанными они ни были, практически невозможно обеспечить в системах с многомиллионной веб-аудиторией, вроде [amazon.com](#). Таким образом, проектировщики NoSQL-систем жертвуют согласованностью данных ради достижения двух других свойств из теоремы CAP. Некоторые СУБД, например, [Riak](#), позволяют настраивать требуемые характеристики доступности-согласованности даже для отдельных запросов путём задания количества узлов, необходимых для подтверждения успеха транзакции.

### Преимущества NoSQL:

- **Масштабируемость.** Горизонтальное масштабирование существующих традиционных СУБД обычно является трудоемкой, дорогостоящей и эффективной только до определенного уровня задачей. В то же время многие NoSQL-решения проектировались исходя из необходимости масштабироваться горизонтально и делать это «на лету». Поэтому эта процедура обычно проще и прозрачнее в NoSQL, чем в РСУБД.
- **Производительность** БД на одном узле, а не в кластере также является немаловажным параметром. Для многих задач такие свойства традиционных СУБД, как транзакционность, изолированность изменений, надежность в пределах одного узла или даже сама реляционная модель, не всегда нужны в полном объеме. Поэтому отказ от этих свойств (всех или некоторых) позволяет NoSQL иногда добиваться большей производительности на одном узле, чем традиционным решениям.
- **Надежная работа в условиях**, когда отказ железа или сетевая недоступность – обычное дело, является одним из свойств многих решений NoSQL. Основной способ ее обеспечения – это репликация. Сама по себе репликация отнюдь не является уникальной особенностью NoSQL, но здесь, как и при масштабировании, важную роль играют эффективность и легкость внесения изменений в существующую инсталляцию. Переход БД к работе в режиме репликации – это простая задача для большинства NoSQL-решений.
- **Простота разработки и администрирования** – также важный аргумент в пользу NoSQL-технологий. Целый ряд задач, связанных с масштабированием и репликацией, представляющих значительную сложность и требующих обширной специальной экспертизы на традиционных СУБД, у NoSQL занимает считанные минуты. Задачи установки и настройки, само использование NoSQL-решений обычно существенно проще и менее трудоемки, чем в случае с РСУБД. Поэтому NoSQL-системы стали очевидным выбором для многих стартапов, где скорость разработки и внедрения является ключевым фактором. NoSQL-решения не обязательно означают замену и полный отказ от РСУБД. Как обычно, инструмент должен выбираться под задачу, а не наоборот.
- **Специализированная модель данных.** Реляционная модель не обязательно является самым подходящим способом представления данных для всех задач. При разработке приложений уже давно стало нормой использование специальных «прослоек», отображающих реляционную модель на модель данных приложения, и наоборот. Это увеличивает накладные расходы и усложняет систему в целом. NoSQL предлагает широкий спектр моделей данных и их реализаций, остается лишь выбрать оптимальную для конкретной задачи модель: данные в виде «документов» из наборов полей, записей «ключ–значение», графов и т. д.

С другой стороны, согласованность базы данных явно принесена в жертву эффективности. Модель данных приложения не накладывает никаких ограничений; можно создать неограниченное число экземпляров одного и того же объекта. Например, благодаря автогенерации ключей в Google App Engine все экземпляры будут иметь уникальные ключи, но все остальное будет идентично. Кроме того, не поддерживается возможность каскадного удаления, поэтому если применить тот же подход к хранению отношений типа "один-ко-многим", то возможна ситуация, при которой родительский объект будет удален, а дочерние останутся в базе данных. Разумеется, ничто не мешает вам реализовать собственную схему обеспечения согласованности, но в этом-то и кроется проблема: вам придется делать это самостоятельно (примерно так же, как мы реализовывали остальную функциональность).

Таким образом, работа с нереляционными базами данных требует определенной дисциплины. Если, например, начать создавать различные типы соревнований, некоторые с названиями, некоторые без них, одни - со свойством `date`, другие – с `race_date`, то это приведет к головной боли и для вас самих, и для других разработчиков, которые будут использовать ваш код.

Другими характерными чертами NoSQL-решений являются:

- Возможность разработки базы данных без задания схемы.
- Скорость: даже при небольшом количестве данных конечные пользователи могут оценить снижение времени отклика системы с сотен миллисекунд до миллисекунд.

#### Типы хранилищ данных

Описание схемы данных в случае использования NoSQL-решений может осуществляться через использование различных структур данных: хеш-таблиц, деревьев и других.

В зависимости от модели данных и подходов к распределённости и репликации можно выделить четыре типа хранилищ: «ключ-значение» (key-value store), документно-ориентированные (document store), хранилища семейств колонок (column database), графовые базы данных (graph database).

#### *Хранилище «ключ-значение»*

Хранилища «ключ-значение» является простейшим хранилищем данных, использующим ключ для доступа к значению. Такие хранилища используются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в системах, спроектированных с прицелом на масштабируемость. Примеры таких хранилищ — Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB.

Такие БД очень производительны, просты в обращении и легко масштабируются.

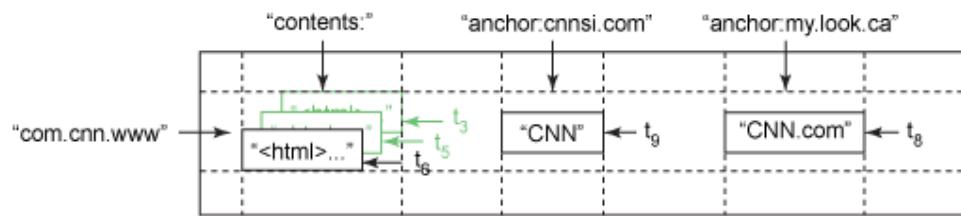
Часто встречающиеся случаи применения:

- Кеширование - быстрое и частое сохранение данных для будущего использования.
- Очередь - некоторые БД типа ключ-значение поддерживают списки, наборы и очереди.
- Распределение информации/задач - используется для реализации паттерна Pub/Sub.
- Живое обновление информации - приложения использующие состояния.

### *Хранилище семейств колонок (или Bigtable-подобные базы данных)*

В этом хранилище данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование, а также задачи, связанные с большими данными, с пониженными требованиями к согласованности данных. Примерами СУБД данного типа являются: Apache Hbase, Apache Cassandra, Apache Accumulo , Hypertable, SimpleDB (Amazon.com).

Пример таблицы, в которой URL-адреса используются в качестве ключей строк, а различные аспекты веб-страниц используются в качестве имен столбцов. Содержимое веб-страниц хранится в единственном столбце, который сохраняет несколько версий страницы с метками момента времени, в который они были выбраны.



Хранилища семейств колонок и документно-ориентированные хранилища имеют близкие сценарии использования: системы управления содержимым, блоги, регистрация событий. Использование отметок времени (timestamp) позволяет использовать этот вид хранилища для организации счётчиков, а также регистрации и обработки различных данных, связанных со временем.

Хранилища семейств колонок (англ. column family stores) не следует путать с колоночными хранилищами (англ. column stores). Последние являются реляционными СУБД с раздельным хранением колонок (в отличие от более традиционного построчного хранения данных).

Такие системы баз данных очень эффективны и могут быть использованы для хранения важной информации больших объемов. Может они где-то не очень гибки в плане данных, зато они функциональны и производительны.

Основные области применения:

- Хранение неструктурированных, не разрушаемых данных - если вам необходимо хранить большие объемы данных в течение долгого времени, то такие БД очень хорошо справляются с задачей.
- Масштабирование - по задумке такие базы данных легко масштабируются. Они легко справляются с любым объемом данных.

### *Документо-ориентированная СУБД*

Документо-ориентированные СУБД служат для хранения иерархических структур данных. Находят своё применение в системах управления содержимым, издательском деле, документальном поиске и т. п. СУБД данного типа — CouchDB, Couchbase, MarkLogic, MongoDB, eXist, Berkeley DB XML и т.д.

Немного подробней на примере MongoDB:

Основой MongoDB является концепция документа (document), который представляется в виде упорядоченного набора ключей с ассоциированными значениями; коллекция (collection) — это группа таких документов. Если документ является в MongoDB аналогом строки в реляционной базе данных, то коллекция может считаться аналогом таблицы.

Коллекции не имеют схем. Это означает, что документы в рамках одной коллекции могут иметь любое количество различных форм. Например, оба следующих документа могли бы храниться в одной коллекции.

```
{"empID" : "E12345", "fname" : "John", "lname" : "Smit", "city" : "Sydney",  
"age": 32}
```

```
{"postID" : "P1", "postText" : "This is my blog post"}
```

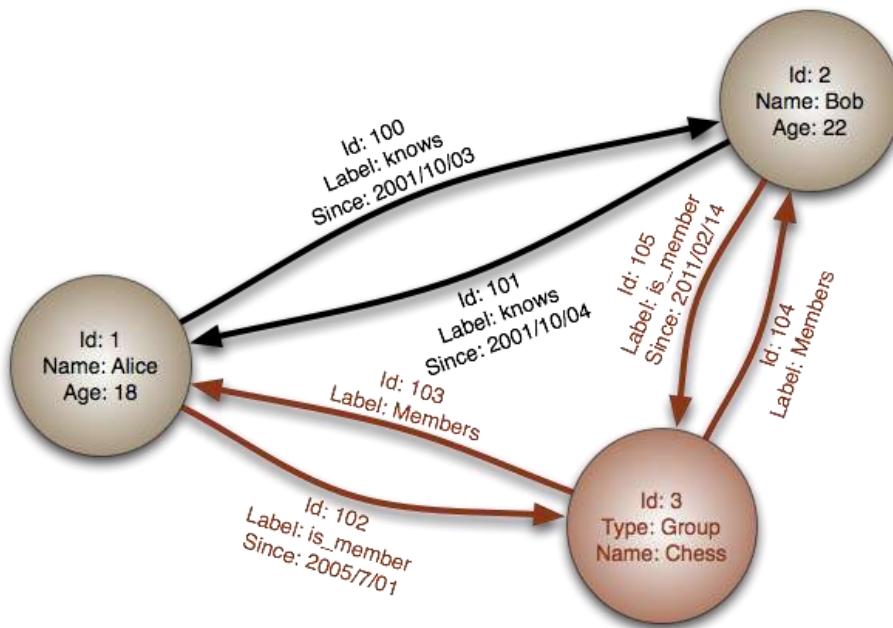
Документ-ориентированные хранилища отлично хранят несвязанную информацию больших объемов, даже если она очень разнится от сущности к сущности.

#### Преимущества:

- Вложенная информация - документо-ориентированные хранилища отличаются тем, что работают с глубоко вложенной, сложной информацией.
- Поддержка JavaScript - одна из отличительных особенностей документо-ориентированных хранилищ это то, как они работают с другими приложениями: поддержка JSON.

#### *Базы данных на основе графов*

Графовые базы данных применяются для задач, в которых данные имеют большое количество связей, например, социальные сети, выявление мошенничества. Примеры: Neo4j, OrientDB, AllegroGraph, Blazegraph (RDF-хранилище, ранее называлось Bigdata), InfiniteGraph , FlockDB, Titan.



Так как ребра графа материализованы, то есть, являются хранимыми, обход графа не требует дополнительных вычислений (как JOIN в SQL), но для нахождения начальной вершины обхода требуется наличие индексов. Графовые базы данных как правило поддерживают ACID, а также имеют различные языки запросов, вроде Gremlin и Cypher (Neo4j).

Такие типы БД хранят информацию совершенно особенно, совсем не как все остальные СУБД.

Часто встречающиеся примеры использования:

- Работа со сложно связанный информацией - как было сказано во вступлении, хранилища типа графа отличноправляются со сложно связанный информацией. Например, хранения связей между двумя сущностями и целого ряда разноуровневых связей между сущностями не связанных с первыми напрямую.
- Моделирование и поддержка классификаций - такие БД преуспели везде где есть связи. Моделирование данных и классификация различной информации по связям можно с легкостью представить используя эти БД.

### UnQL

В июле 2011 компания Couchbase, разработчик CouchDB, Memcached и Membase, анонсировала создание нового SQL-подобного языка запросов — UnQL (Unstructured Data Query Language). Работы по созданию нового языка выполнили создатель SQLite Ричард Гипп и основатель проекта CouchDB Дэмиен Кац. Разработка передана сообществу на правах общественного достояния.

Этот шаг сделан в сторону стандартизации языка запросов к NoSQL базам данных, на данный момент каждая база данных обладает собственным синтаксисом запросов, например:

CQL (Cassandra):

```
SELECT * FROM ad_click WHERE reseller_id = 'supabooobs' AND day = '2013-11-29';
```

MongoDB:

```
db.users.find( { username: "joe", age: 27 } );
```

DQL (DynamoDB):

```
SELECT * FROM foobars WHERE foo = 'bar' SAVE out.json;
```

UnQL имеет SQL-подобный синтаксис и поддерживает такие команды, как SELECT, DELETE, INSERT и UPDATE, что делает новый язык запросов привычным для большинства разработчиков. Тем не менее, в отличие от SQL, UnQL обладает рядом расширенных возможностей, позволяющих манипулировать и выбирать информацию в хранилищах документов со сложной и неоднородной структурой. Для определения представления документов используется формат JSON (JavaScript Object Notation).

Вместо таблиц UnQL манипулирует коллекциями разнородных документов, структура которых жестко не определена и может варьироваться в разных документах (структура документа задается в самом документе, общая схема данных отсутствует). Для создания коллекций по аналогии с SQL-выражением "CREATE/DROP TABLE" используется "CREATE/DROP COLLECTION", при этом коллекция служит для логического разделения различных наборов документов.

Присутствующие в каждом документе поля определяются через JSON, в простейшем случае JSON-представление может состоять из одной строковой или числовой переменной. Каждое из таких полей может фигурировать в блоке "WHERE" запроса SELECT, при этом запрос коснется только документов, в которых определены данные поля. В запросе также могут быть заданы критерии сортировки (ORDER BY), группировки (GROUP BY, HAVING), ограничения размера выборки (LIMIT...OFFSET) и объединения (UNION, INTERSECT, EXCEPT). Поддерживается создание индексов (CREATE INDEX) и использование атомарных транзакций (BEGIN, ROLLBACK, COMMIT). Операции добавления, удаления и изменения данных (INSERT, DELETE, UPDATE) могут выполняться как в синхронном, так и в асинхронном (возвращение управления, не дожидаясь фактического выполнения) режимах.

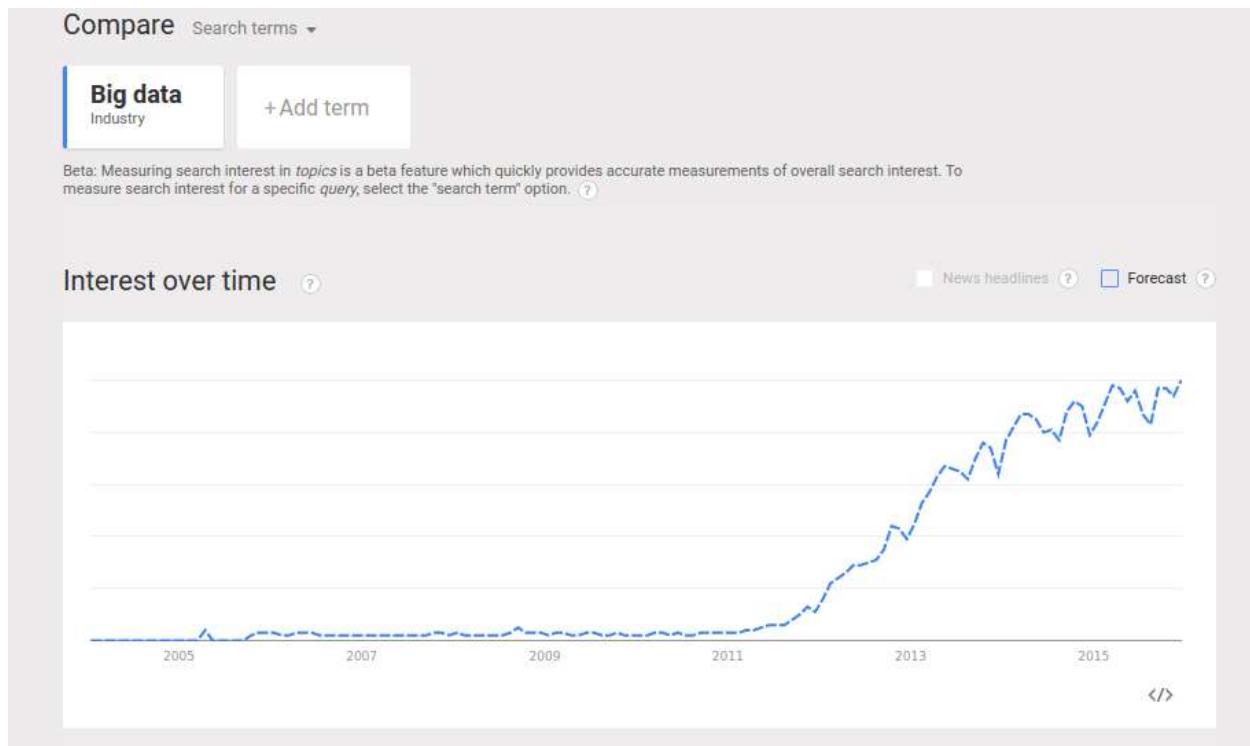
### [Немного о будущем](#)

Согласованность данных - самое большое препятствие на пути NoSQL. Пока что нереляционные БД применяются там, где масштабируемость и производительность важнее строгой согласованности. Реляционные СУБД никуда не денутся, NoSQL не заменяет, а дополняет их. А, может, в будущем мы сможем наблюдать постепенное срастание обеих технологий в гибридную экосистему и поглощения молодых лидеров рынка NoSQL производителями традиционных РСУБД.

## [Обработка больших данных. Подходы к формированию больших данных](#)

### [Определение термина](#)

Термин Big Data появился сравнительно недавно. Google Trends показывает начало активного роста употребления словосочетания начиная с 2011 года.



Встречаются разные определения:

- Big Data – это когда данных больше, чем 100ГБ (500ГБ, 1ТБ, кому что нравится)
- Big Data – это такие данные, которые невозможно обрабатывать в Excel
- Big Data – это такие данные, которые невозможно обработать на одном компьютере

И даже такие:

- Big Data – это вообще любые данные.
- Big Data не существует, ее придумали маркетологи.

Однако будем придерживаться формального определения:

Большие данные (англ. *big data*) — серия подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объёмов и значительного многообразия для получения воспринимаемых человеком результатов, эффективных в условиях непрерывного прироста, распределения по многочисленным узлам вычислительной сети, сформировавшихся в конце 2000-х годов, альтернативных традиционным системам управления базами данных и решениям класса Business Intelligence. В данную серию включают средства массово-параллельной обработки неопределённо структурированных данных, прежде всего, решениями категории NoSQL, алгоритмами MapReduce, программными каркасами и библиотеками проекта Hadoop.

В качестве определяющих характеристик для больших данных отмечают «три V»: объём (англ. *volume*, в смысле величины физического объема), скорость (англ. *velocity* в смысле как скорости прироста, так и необходимости высокоскоростной обработки и получения результатов),

многообразие (англ. variety, в смысле возможности одновременной обработки различных типов структурированных и полуструктурных данных).

Несколько примеров того, что может быть источником данных, для которых необходимы методы работы с большими данными:

- Логи поведения пользователей в интернете
- GPS-сигналы от автомобилей для транспортной компании
- Данные, снимаемые с датчиков в большом адронном коллайдере
- Оцифрованные книги в Российской Государственной Библиотеке
- Информация о транзакциях всех клиентов банка
- Информация о всех покупках в крупной ритейл сети и т.д.

Количество источников данных стремительно растёт, а значит технологии их обработки становятся всё более востребованными.

### История

Введение термина «большие данные» относят к Клиффорду Линчу, редактору журнала *Nature*, подготовившему к 3 сентября 2008 года специальный номер журнала с темой «Как могут повлиять на будущее науки технологии, открывающие возможности работы с большими объёмами данных?», в котором были собраны материалы о феномене взрывного роста объёмов и многообразия обрабатываемых данных и технологических перспективах в парадигме вероятного скачка «от количества к качеству»; термин был предложен по аналогии с расхожими в деловой англоязычной среде метафорами «большая нефть», «большая руда».

Несмотря на то, что термин вводился в академической среде, и прежде всего, разбиралась проблема роста и многообразия научных данных, начиная с 2009 года термин широко распространился в деловой прессе, а к 2010 году относят появление первых продуктов и решений, относящихся исключительно и непосредственно к проблеме обработки больших данных. К 2011 году большинство крупнейших поставщиков информационных технологий для организаций в своих деловых стратегиях используют понятие о больших данных, в том числе IBM, Oracle, Microsoft, Hewlett-Packard, EMC, а основные аналитики рынка информационных технологий посвящают концепции выделенные исследования.

В 2011 году Gartner отмечает большие данные как тренд номер два в информационно-технологической инфраструктуре (после виртуализации и как более существенный, чем энергосбережение и мониторинг). Прогнозируется, что внедрение технологий больших данных наибольшее влияние окажет на информационные технологии в производстве, здравоохранении, торговле, государственном управлении, а также в сферах и отраслях, где регистрируются индивидуальные перемещения ресурсов.

С 2013 года большие данные как академический предмет изучаются в появившихся вузовских программах по науке о данных и вычислительным наукам, и инженерии.

### Принципы работы с большими данными

Исходя из определения Big Data, можно сформулировать основные принципы работы с такими данными:

- Горизонтальная масштабируемость. Поскольку данных может быть сколь угодно много – любая система, которая подразумевает обработку больших данных, должна быть расширяемой. В 2 раза вырос объём данных – в 2 раза увеличили количество железа в кластере и всё продолжило работать.
- Отказоустойчивость. Принцип горизонтальной масштабируемости подразумевает, что машин в кластере может быть много. Например, Hadoop-кластер Yahoo имеет более 42000 машин (Facebook — 1400, Ebay - 532, Spotify — 120, Adobe - 30). Это означает, что часть этих машин будет гарантированно выходить из строя. Методы работы с большими данными должны учитывать возможность таких сбоев и переживать их без каких-либо значимых последствий.
- Локальность данных. В больших распределённых системах данные распределены по большому количеству машин. Если данные физически находятся на одном сервере, а обрабатываются на другом – расходы на передачу данных могут превысить расходы на саму обработку. Поэтому одним из важнейших принципов проектирования BigData-решений является принцип локальности данных – по возможности обрабатываем данные на той же машине, на которой их храним.

Все современные средства работы с большими данными так или иначе следуют этим трём принципам. Для того, чтобы им следовать – необходимо придумывать какие-то методы, способы и парадигмы разработки средств разработки данных.

### Методы анализа

Методы и техники анализа, применимые к большим данным:

- методы класса Data Mining: обучение ассоциативным правилам (англ. association rule learning), классификация (методы категоризации новых данных на основе принципов, ранее применённых к уже наличествующим данным), кластерный анализ, регрессионный анализ;
- краудсорсинг — категоризация и обогащение данных силами широкого, неопределённого круга лиц, привлечённых на основании публичной оферты, без вступления в трудовые отношения;
- смешение и интеграция данных (англ. data fusion and integration) — набор техник, позволяющих интегрировать разнородные данные из разнообразных источников для возможности глубинного анализа, в качестве примеров таких техник, составляющих этот класс методов приводятся цифровая обработка сигналов и обработка естественного языка (включая тональный анализ);
- машинное обучение, включая обучение с учителем и без учителя, а также Ensemble learning — использование моделей, построенных на базе статистического анализа или машинного обучения для получения комплексных прогнозов на основе базовых моделей (англ. constituent models,ср. со статистическим ансамблем в статистической механике);
- искусственные нейронные сети, сетевой анализ, оптимизация, в том числе генетические алгоритмы;
- распознавание образов;
- прогнозная аналитика;
- имитационное моделирование;

- пространственный анализ (англ. Spatial analysis) — класс методов, использующих топологическую, геометрическую и географическую информацию в данных;
- статистический анализ, в качестве примеров методов приводятся A/B-тестирование и анализ временных рядов;
- визуализация аналитических данных — представление информации в виде рисунков, диаграмм, с использованием интерактивных возможностей и анимации как для получения результатов, так и для использования в качестве исходных данных для дальнейшего анализа.

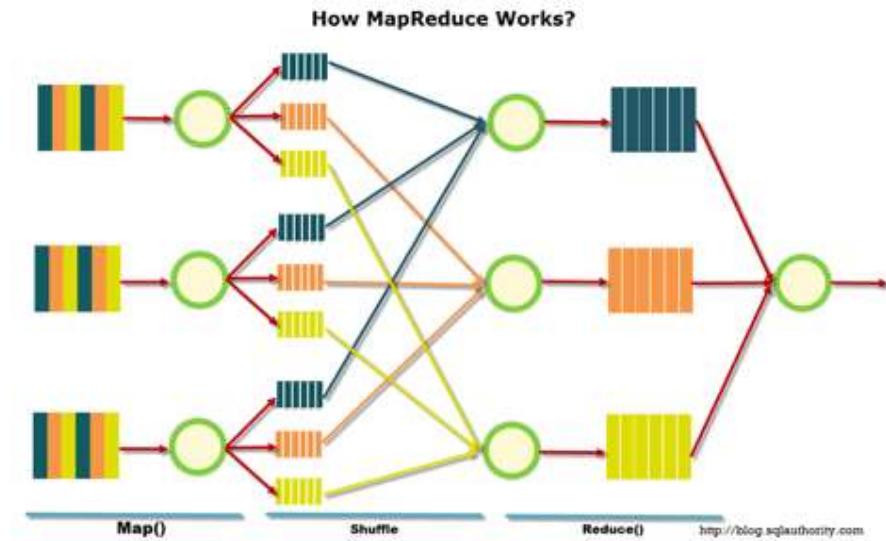
## Технологии

Наиболее часто указывают в качестве базового принципа обработки больших данных в SN-архитектуру (англ. Shared Nothing Architecture), обеспечивающую массивно-параллельную обработку, масштабируемую без деградации на сотни и тысячи узлов обработки. При этом, кроме рассматриваемых большинством аналитиков технологий NoSQL, MapReduce, Hadoop, R, иногда включается в контекст применимости для обработки больших данных также технологии Business Intelligence и реляционные системы управления базами данных с поддержкой языка SQL.

Рассмотрим наиболее популярные технологии подробней.

### *MapReduce*

MapReduce – это модель распределенной обработки данных, предложенная компанией Google для обработки больших объемов данных на компьютерных кластерах.



MapReduce предполагает, что данные организованы в виде некоторых записей. Обработка данных происходит в 3 стадии:

- Стадия Map. На этой стадии данные предобращаются при помощи функции map(), которую определяет пользователь. Работа этой стадии заключается в предобратке и фильтрации данных. Работа очень похожа на операцию map в функциональных языках программирования – пользовательская функция применяется к каждой входной записи. Функция map() примененная к одной входной записи и выдаёт множество пар ключ-

значение. Множество – т.е. может выдать только одну запись, может не выдать ничего, а может выдать несколько пар ключ-значение. Что будет находиться в ключе и в значении – решать пользователю, но ключ – очень важная вещь, так как данные с одним ключом в будущем попадут в один экземпляр функции reduce.

- Стадия Shuffle. Проходит незаметно для пользователя. В этой стадии вывод функции map «разбирается по корзинам» – каждая корзина соответствует одному ключу вывода стадии map. В дальнейшем эти корзины послужат входом для reduce.
- Стадия Reduce. Каждая «корзина» со значениями, сформированная на стадии shuffle, попадает на вход функции reduce(). Функция reduce задаётся пользователем и вычисляет финальный результат для отдельной «корзины». Множество всех значений, возвращённых функцией reduce(), является финальным результатом MapReduce-задачи.

Несколько дополнительных фактов про MapReduce:

- Все запуски функции map работают независимо и могут работать параллельно, в том числе на разных машинах кластера.
- Все запуски функции reduce работают независимо и могут работать параллельно, в том числе на разных машинах кластера.
- Shuffle внутри себя представляет параллельную сортировку, поэтому также может работать на разных машинах кластера. Пункты 1-3 позволяют выполнить принцип горизонтальной масштабируемости.
- Функция map, как правило, применяется на той же машине, на которой хранятся данные – это позволяет снизить передачу данных по сети (принцип локальности данных).
- MapReduce – это всегда полное сканирование данных, никаких индексов нет. Это означает, что MapReduce плохо применим, когда ответ требуется очень быстро.

### *Hadoop*

Парадигму MapReduce предложила компания Google в 2004 году в своей статье MapReduce: Simplified Data Processing on Large Clusters. Поскольку предложенная статья содержала описание парадигмы, но реализация отсутствовала – несколько программистов из Yahoo предложили свою реализацию в рамках работ над web-краулером nutch. С этого и началась история Hadoop. Изначально Hadoop был, в первую очередь, инструментом для хранения данных и запуска MapReduce-задач, сейчас же Hadoop представляет собой большой стек технологий, так или иначе связанных с обработкой больших данных (не только при помощи MapReduce). Основными (core) компонентами Hadoop являются:

- Hadoop Distributed File System (HDFS) – распределённая файловая система, позволяющая хранить информацию практически неограниченного объёма.
- Hadoop YARN – фреймворк для управления ресурсами кластера и менеджмента задач, в том числе включает фреймворк MapReduce.
- Hadoop common.

Также существует большое количество проектов, непосредственно связанных с Hadoop, но не входящих в Hadoop core:

- Hive – инструмент для SQL-like запросов над большими данными (превращает SQL-запросы в серию MapReduce-задач);

- Pig – язык программирования для анализа данных на высоком уровне. Одна строчка кода на этом языке может превратиться в последовательность MapReduce-задач;
- Hbase – колоночная база данных, реализующая парадигму BigTable;
- Cassandra – высокопроизводительная распределенная key-value база данных;
- ZooKeeper – сервис для распределённого хранения конфигурации и синхронизации изменений этой конфигурации;
- Mahout – библиотека и движок машинного обучения на больших данных.

Отдельно хотелось бы отметить проект Apache Spark, который представляет собой движок для распределённой обработки данных. Apache Spark обычно использует компоненты Hadoop, такие как HDFS и YARN для своей работы, при этом сам в последнее время стал популярнее, чем Hadoop.

### [Big Data в действии](#)

Зрелость технологий Больших Данных в организации определяется составом задач, для которых они используются, что, в свою очередь, сильно зависит от внутренней среды организации. Выделяются пять уровней зрелости для бизнес-модели организации с точки зрения интегрированных групп задач, которые решаются на основе технологий Больших Данных: мониторинг бизнеса, анализ бизнеса, оптимизация бизнеса, монетизация данных, трансформация бизнеса. Интерес представляют высокие уровни зрелости. Так, уровень монетизации данных предполагает, что организация не только обладает высоким цифровым потенциалом для того, чтобы собирать и обрабатывать данные, но и способна создавать на этой основе новый цифровой продукт или услугу, опираясь на аналитику Больших Данных. Развитая аналитика Больших Данных создает базис для разработки стратегий развития предприятия, направленных на существенную трансформацию бизнес-моделей с точки зрения новых продуктов, услуг и рынков.

Наиболее важные перемены связаны с изменением управления данными в организации.

Учитывая разнообразие структуры Больших Данных, их содержания, характеристик источников, предполагаемого использования, необходимо опираться на применение существенно большего, по сравнению с традиционным, спектра инструментов и методов, а также на многоплатформность программных и аппаратных решений и тщательный подход к формированию персонала, ответственного за управление.

Особую сложность представляет управление потоковыми данными, поступающими в реальном времени. Задачи обработки в этом случае направлены на выявление в потоке значимой для решения задач информации и событий, сопоставление с существующей информацией и ее сохранение. Инструменты для обработки сложных событий (Complex Event Processing, CEP) позволяют настраиваться на особенности определенных потоковых данных, проводить корреляцию данных из многих источников, обеспечивая более высокую отдачу от их использования.

Требования к перечню необходимых компетенций аналитика в сфере Больших Данных еще только формируются. В общем виде компетенции специалиста по исследованию данных можно представить в виде диаграммы Венна.



Предполагается, что аналитические навыки включают знание как традиционной аналитики, которая объясняет, что происходит и почему, так и аналитики предсказательной и предписывающей. Последняя направлена не только на поддержку решений, но и на автоматизацию принятия решений.

Бизнес-навыки предполагают умение правильно формулировать вопросы к аналитике для решения бизнес-задач и производить отбор критериев для оценки результатов, а также знание существующих ограничений, методов принятия решений, способов обеспечения прозрачности применяемых механизмов и методов. ИТ-навыки включают знание технологий сбора, хранения и обработки данных. Важными являются компетенции в области высокопроизводительных вычислений и доставки данных.

Компетенции и навыки в области исследования данных связаны с пониманием свойств данных, их моделей, соответствующих аналитических методов, способов представления и интерпретации данных. В области пересечения находятся интегрированные компетенции, позволяющие решать задачи бизнеса на основе аналитики Больших Данных. Это компетенции, обеспечивающие способность находить нужные данные, понимать их, осуществлять управление ими в организации, находить на основе их исследования новые возможности и решения для организации.

## Глава 6. Разработка программного обеспечения

### Разница между разработкой и производством программного обеспечения

**Разработка программного обеспечения** (*software development*) — это род деятельности и процесс, направленный на создание и поддержание работоспособности, качества и надежности программного обеспечения, используя технологии, методологию и практики из информатики, управления проектами, математики, инженерии и других областей знания.

**Производство**, в экономическом смысле — процесс создания какого-либо продукта. Понятие производства характеризует специфически человеческий тип обмена веществами с природой, или, более точно, — процесс активного преобразования людьми природных ресурсов в какой-либо продукт.

Как отмечает Философская энциклопедия, процесс производства всегда носит общественный характер: производство обособленного одиночки вне общества представляет собой, по выражению Маркса, такую же бессмыслицу, как развитие языка без совместно живущих индивидов.

Важные замечания:

1. Производство программного обеспечения (ПО) предполагает активное преобразование энергии, знаний и опыта людских ресурсов в конченый продукт, то есть, само ПО.
2. Производство — одна из стадий (завершающая) разработки ПО.
3. Следует понимать, что разработка отличается от производства своей глобальностью и общностью. Разработка предоставляет каскад для команды / компании / корпорации, на который они могут навешивать различные парадигмы, по-разному распределять ресурсы и просчитывать риски в зависимости от желаемого продукта. Производство, как было сказано, это процесс, то есть, движение от точки А к точке Б. Можно провести такое сравнение: разработка — это механизм в выключенном состоянии, который можно дополнять или урезать, а производство — это тот же механизм (или некоторые его частные аспекты) во включённом состоянии.

Разработка программного обеспечения имеет дело с проблемами качества, стоимости и надёжности, об этом уже неоднократно говорилось. Эти аспекты характеризуют вроде бы исключительно техническую задачу с экономической точки зрения. Аналогичным образом строится и производство ПО. В данном случае, проблемы затратности и оптимизации рисков целиком попадают под производственный вопрос.

Разработка программного обеспечения может быть разделена на несколько разделов.

- **Требования к программному обеспечению:** извлечение, анализ, спецификация и ратификация требований для программного обеспечения.
- **Проектирование программного обеспечения:** проектирование программного обеспечения средствами Автоматизированной Разработки Программного Обеспечения (CASE) и стандарты формата описаний, такие как Унифицированный Язык Моделирования (UML), используя различные подходы: проблемно-ориентированное проектирование и т. д.
- **Инженерия программного обеспечения:** создание программного обеспечения с помощью языков программирования.

- **Тестирование программного обеспечения:** поиск и исправление ошибок в программе.
- **Сопровождение программного обеспечения:** программные системы часто имеют проблемы совместимости и переносимости, а также нуждаются в последующих модификациях в течение долгого времени после того, как закончена их первая версия. Подобласть имеет дело с этими проблемами.
- **Управление конфигурацией программного обеспечения:** так как системы программного обеспечения очень сложны и модифицируются в процессе эксплуатации, их конфигурации должны управляться стандартизованным и структурированным методом.
- **Управление разработкой программного обеспечения:** управление системами программного обеспечения имеет заимствования из управления проектами, но есть нюансы, не встречающиеся в других дисциплинах управления из-за специфики ресурсов и конечного продукта. Во многом, конечный продукт неосозаем. Об этом очень хорошо было сказано у Тарасова (тема SaaS).
- **Процесс разработки программного обеспечения:** процесс построения программного обеспечения горячо обсуждается среди практиков, основными парадигмами считаются agile или waterfall.
- **Инструменты разработки программного обеспечения**, см. CASE: методика оценки сложности системы, выбора средств разработки и применения программной системы.
- **Качество программного обеспечения:** методика оценки критериев качества программного продукта и требований к надёжности.
- **Локализация программного обеспечения**, ветвь языковой промышленности.

На протяжении нескольких десятилетий стоит задача поиска повторяемого, предсказуемого процесса или методологии, которая бы улучшила продуктивность, качество и надёжность разработки. Одни пытались систематизировать и формализовать этот, по-видимому, малопредсказуемый процесс. Другие применяли к нему методы управления проектами и методы программной инженерии. Третьи считали, что без постоянного контроля со стороны заказчика разработка ПО выходит из-под контроля, съедая лишнее время и средства.

Опыт управления разработкой программ отражается в соответствующих руководствах, обычаях и стандартах.

Производство ПО не начинается, пока не будут определены все стандарты, на которые должно опираться производство. Т.е., механизм без всех гвоздиков не заработает.

Однако, это в теории всё так хорошо. На практике же всегда процесс производства, а именно цикл жизни проекта, начинается тогда, когда ещё не все стандарты могут быть определены.

К тому же, стоит заметить, что производство ПО не такой уж и закрытый объект, каким его хотелось бы видеть. Почему? Да потому что на проекты берут новых людей, которые не имеют понятия о том, с каких нулей стартовало производство. Но зато им понятно, какой кусок работы необходимо сделать. Однако, на мой взгляд, нельзя данную «слепоту нового разработчика» считать ужасной и неприемлемой: напротив, в некоторых ситуациях подробное знание всего, что происходило на проекте, может отвлечь разработчика от сути проекта и выполнения своих должностных обязанностей. А это стоит времени и денег, а это затягивает производственный процесс.

## Участники процесса разработки ПО

- Пользователь
- Заказчик
- Разработчик
- Руководитель проекта
- Аналитик
- Тестировщик
- Поставщик

Меняется ли как-то этот список на этапе производства ПО? Незначительно. Уходят из списка тестировщики. Во многом, аналитика во время производства уходит куда-то на задний план, тем не менее, оставаясь одним из самых важных столпов, обеспечивающих положительную обратную связь с пользователем.

В самом определении производства ПО смыслы рамки настолько, что трудно понять: входит ли сам цикл работы над проектом внутрь производства или нет? В каких-то методиках он входит, а в каких-то производство ограничивается лишь настройкой конвейера, который поставляет ПО пользователю и следит за тем, чтобы можно было добавить такого, чтобы и пользователь был рад, и прибыли было побольше.

Если следовать строгой логике данного ранее определения термина «производство», то цикл разработки проекта входит в него практически весь, лишь с оговорками, которые носят весьма субъективный характер в силу размытого определения производства ПО и его проекции на реалии разработки ПО.

Можно даже такую параллель провести: разработка – набор спецификационных документов и тестов, которым должна следовать и которые должна пройти определённая программа. А эта программа и есть производство ПО.

## Проблемы разработки ПО

Наиболее распространёнными проблемами, возникающими в процессе разработки ПО, считают:

- **Недостаток прозрачности.** В любой момент времени сложно сказать, в каком состоянии находится проект и каков процент его завершения. **В грамотно наложенном производственном механизме этой проблемы нет: механизм работает как часы / разработка подходит к завершению.**
- **Недостаток контроля.** Без точной оценки процесса разработки срываются графики выполнения работ и превышаются установленные бюджеты. Сложно оценить объём выполненной и оставшейся работы. **Встречается и в производстве (это уже перерастает в проблему халатности).**
- **Недостаток мониторинга.** Невозможность наблюдать ход развития проекта не позволяет контролировать ход разработки в реальном времени. С помощью инструментальных средств менеджеры проектов принимают решения на основе данных, поступающих в реальном времени. **(Аналогично прозрачности процессов в ПО).**
- **Неконтролируемые изменения.** У потребителей постоянно возникают новые идеи относительно разрабатываемого программного обеспечения. Влияние изменений может

быть существенным для успеха проекта, поэтому важно оценивать предлагаемые изменения и реализовывать только одобренные, контролируя этот процесс с помощью программных средств. (**Встречается в производстве, из-за чего процесс = механизм приходится перенастраивать, а, следовательно, возвращаться к расчётом и т.д. и т.п.**)

- **Недостаточная надёжность.** Самый сложный процесс — поиск и исправление ошибок в программах на ЭВМ. Поскольку число ошибок в программах заранее неизвестно, то заранее неизвестна и продолжительность отладки программ, и отсутствие гарантий отсутствия ошибок в программах. Следует отметить, что привлечение доказательного подхода к проектированию ПО позволяет обнаружить ошибки в программе до её выполнения. В этом направлении много работали Кнут, Дейкстра и Вирт. Профессор Вирт при разработке Паскаля и Оберона за счет строгости их синтаксиса добился математической доказуемости завершаемости и правильности программ, написанной на этих языках. (**В производстве вряд ли бывает, это больше проектирование ПО**)
- **Неправильный выбор методологии** разработки программного обеспечения. (Тоже больше проектирование, однако, способ налаживания конвейера в первую очередь как средства общения между разработчиком и заказчиком напрямую зависит от методологии).
- **Отсутствие гарантий качества и надежности программ** из-за отсутствия гарантий отсутствия ошибок в программах вплоть до формальной сдачи программ заказчикам. Данная проблема не является проблемой, относящейся исключительно к разработке ПО. Гарантия качества — это проблема выбора поставщика товара (не продукта). (**Да, это проблема производства, над которой разработчикам надо беспокоиться настолько, насколько их код должен проходить все необходимые тесты!**)

Основные виды простых производств можно описать как:

- линейное производство
- расходящееся производство
- сходящееся производство
- смешанное (из простых) производство

К сложным видам производствам можно отнести:

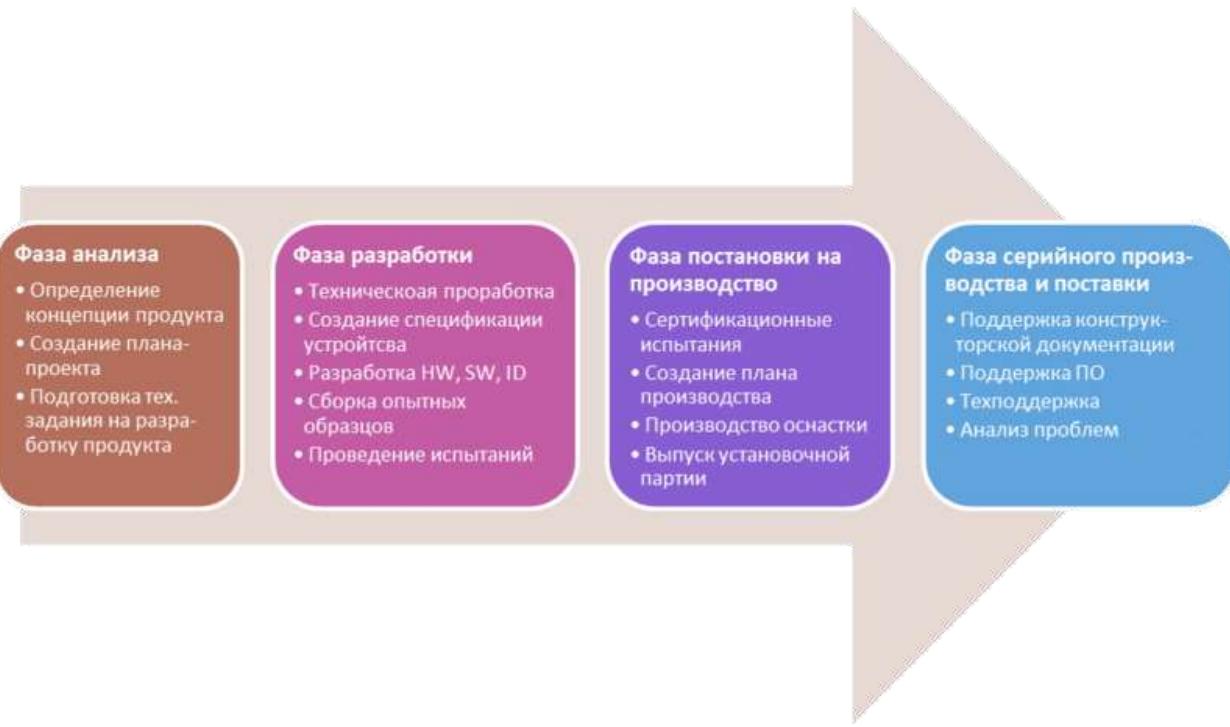
- циклическое производство
- смешанное (из простых и сложных) производство

Реальное производство является зачастую смешанным производством, но для оптимизации производства или для расчёта расчётных цен необходимо понимание видов производства (организации структуры производственных факторов).

#### Частные примеры (разработка продуктов в сфере электроники)

Презентации новых гаджетов Apple, Samsung и других брендов — это только видимая часть айсберга, под которой скрывается человеко-десятилетия труда людей самых разных специализаций: инженеры, программисты, дизайнеры, логисты, руководители различных уровней, продавцы и так далее. Пока ты не погружаешься в эту внутреннюю кухню, может показаться, что процесс довольно простой и понятный: была бы идея, хорошая команда и достаточное финансирование. Однако не все так просто.

Упрощенно этапы создания нового устройства выглядят так:



#### *Фаза №1. Анализ*

У всех эта фаза проходит и называется по-разному: «Расчеты», «Проектирование», «Сбор требований». Одни компании сразу же стартуют с тех. задания, написанного инженером «в теме», другие уделяют чертовски много времени планированию и проработке. Почему же это может быть важно? Причины просты:

- **Планирование бюджета.** Как правило, процесс разработки продукта довольно длинный и будет неприятно, если после 8—9 месяцев инвестирования у вас внезапно закончатся деньги. Матерые компании и продукт-менеджеры имеют собственные технологии планирования для новых продуктов, которые банки даже принимают в качестве гарантий по кредитам.
- **Оптимизация себестоимости.** Если вы производите 50 тыс. устройств в год, будет полезно экономить каждый доллар в себестоимости устройства. – вот и производственная аналитика.
- **Управление.** В разработку любого продукта, даже довольно простого, завязаны тысячи людей (прямо или косвенно: поставщики компонентов, инженеры, маркетологи, рабочие завода-изготовителя и т.д.). Задачи нужно будет делегировать правильным специалистам в правильное время.
- **Цена ошибки,** она очень высока, и чем раньше мы ее выявляем, тем лучше.

Главная задача фазы анализа — определить четкую концепцию продукта, над которым предстоит работа на последующих этапах, и синхронизировать ее с остальными участниками по мере необходимости. Следует учитывать вопросы конечных клиентов, себестоимость, требования производственных площадок, планы проекта, контрактные обязательства, бюджеты,

финансирование, исполнителей, разрешающие и сертифицирующие документы, выбор материалов и компонентов.

Если говорить о планировании нового продукта, то это своего рода искусство, которым занимаются продукт-менеджеры.

Финальный этап фазы анализа — подготовка технического задания, в нем мы определяем функциональные и нефункциональные требования к продукту, прорабатываем технический дизайн и создаем скетчи устройства:

В электронике мы всегда говорим об ограниченности ресурса. Разработчики ПО вне темы embedded уже давно забыли про дефицит, например, оперативной памяти. А тем временем подобные ограничения делают работу инженера крайне увлекательной. Для инженеров-электронщиков технические параметры устройства — это рамки творчества.

Я уже говорил про цену ошибки, внесение изменений в электронике — это долго и дорого. Для ПО можно выпустить обновления, а что можно выпустить для реального предмета? Только другой реальный предмет, в котором недостатки устранены.

Некоторые продукты в сфере электроники «не взлетали» именно по причине нерабочего софта.

Даже самое тщательное планирование не защитит от ошибок на 100%. Вы наверняка читали подобные заголовки новостей:

- Apple отзывает iPod nano: в плеерах найден дефект аккумулятора
- Nokia отзывает 14 миллионов потенциально бракованных зарядных устройств
- Intel отзывает партию чипсетов для процессоров из-за обнаруженного дефекта
- Garmin отзывает более миллиона GPS-навигаторов из-за перегрева аккумуляторов
- Lenovo отзывает свои компьютеры из-за риска внезапного возгорания

Итак, что мы получаем в итоге фазы проектирования? Набор документации, контрактных обязательств и команду, которая понимает продукт и заряжена на успех.

### *Фаза №2. Разработка*

Как правило, электронное изделие состоит из следующих компонентов, которыми занимается инженерный отдел:

1. Железо
2. Корпус
3. Софт
4. Технологии производства

А вот эти компоненты остаются за границами инженерного отдела (мы к ним вернемся позже):

- Упаковка
- Брэнд
- Логистика
- Производство
- Каналы продаж

- Сервисное обслуживание
- Инфраструктура
- Интеллектуальная собственность (патенты и т.д.)

**Железо.** Благодаря современным САПРам разработка аппаратного обеспечения — это достаточно стандартизованный процесс, который при хорошо сформированных вводных не длится долго (от 2 недель до пары месяцев максимум). Самая сложная штука — это конструкция. Дело в том, что физическое расположение одних элементов относительно других играет большую роль и оказывает непосредственное влияние на работоспособность современного устройства.

**Промышленный дизайн.** Внешний вид корпуса плотно привязан к технологии производства. Технология — это материалы, а значит — стоимость. Дизайнер может нарисовать абсолютно волшебную штуку, но для ее воплощения в жизнь понадобится труд конструктора и технолога. Технологические лидеры ставят уникальный дизайн во главу угла, и потому непрерывно создают новые технологии производства.

**Программное обеспечение** в электронике работает в условиях ограниченных ресурсов. Подход к его разработке кардинально отличается от подходов, принятых в индустрии разработки ПО для веба, десктопов и т.п. Используются другие инструменты и другая среда разработки: версии ОС для встраиваемых систем Windows CE, Linux Embedded и т.п.

Программисты-электронщики сильно завязаны на железе. Если разработчик десктопных приложений не задумывается о корректности работы своей аппаратной платформы, то инженер-программист сталкивается с этим постоянно, особенно при работе с новыми компонентами. У меня были проекты, в которых баги в программе возникали из-за ошибок в работе процессора либо от плохого монтажа. К слову последнее — это действительно серьезная проблема для дизайн-центров.

Еще пример: программист и дизайнер могут создать интерфейс пользователя, который будет отлично работать на тестовой платформе, но вызовет перегрузку оперативной памяти на реальном устройстве. Причем такие ошибки не всегда можно заметить сразу. Сутки устройство работает корректно, а потом перезагружается.

Все эти особенности существенно усложняют жизнь программистов, которые работают со встраиваемым ПО, хотя многие специалисты выбирают электронику как раз ради таких увлекательных задач.

Как правило, промежуточным результатом фазы разработки являются **опытные образцы устройства**. Процесс их сборки — это вообще отдельная история. Со всего мира в одну точку на карте съезжается сотня компонентов, каждый упакован в отдельную посылку. На специализированном предприятии по специальным технологиям делаются рабочие образцы. Ваше положение на карте мира практически не имеет значения при разработке софта. А вот при разработке железа сроки производства образца могут различаться в десятки раз в зависимости от положения разработчика и изготовителя. Компании в Китае собирают образец за 2 недели, в России этот срок может доходить до пары месяцев.

После сборки образцы уходят на **тестирование**. По его результатам мы можем повторить весь процесс с самого начала, чтобы исправить найденные косяки либо понять, что в таком исполнении продукт никому не нужен. Такой вот суровый scrum с циклом в год.

#### *Опытные образцы и испытания*

Очевидно, что цель проведения всех видов тестирования — это снижение риска неприятных неожиданностей на следующих этапах.

Итак, у нас завершился этап разработки, и мы имеем готовые опытные образцы. Что же это такое и для чего нужно? Опытный образец (или прототип) — это устройство, которое было собрано с применением технологий прототипирования согласно нашей конструкторской документации. Чтобы убедиться в том, что опытные образцы соответствуют всем необходимым требованиям, мы проводим самые разные испытания, их никогда не бывает мало, они зависят от компании и продукта. Основные тесты включают в себя предсертификационные испытания, функциональное тестирование, ЭМС, электробезопасность, климатику и др. Если продукт работает в какой-то системе, проводятся испытания «в поле», обычно они дают намного больше ценных данных, чем синтетические тесты устройства. Можно до бесконечности тестировать роутер на синтетических тестах, но пока не включишь его в реальную сеть и не начать решать реальные задачи, нельзя понять, работает он нормально или нет.

В рамках одного проекта может быть сделано огромное количество опытных образцов — для выбора материалов корпуса и комплектующих. Есть вещи, которые нельзя смоделировать даже в самых современных САПРах: тактильные ощущения, насколько устройство приятно держать в руках, маркость пластика, его блеск, ощущение веса устройства. Только живое сравнение разных вариантов позволяет сделать оптимальный выбор.

В наших реалиях опытные образцы делаются на оборудовании для серийного производства, но по другим технологиям. Например, пластик не отливается, а фрезеруется или выращивается, поскольку создание литьевой пресс-формы — это длительный и затратный этап.

Главная цель всех испытаний — убедиться в том, что продукт готов и что именно его хотят покупатели. На этом завершается фаза разработки. Следующий шаг — **постановка изделий на серийное производство**.

#### *Фаза постановки на производство*

Теперь наша цель — создать технологию производства устройства в определенных тиражах с заданными сроками, себестоимостью и показателями качества.

Для начала нужно определиться с производственной площадкой. Как правило, крупные компании выбирают себе производственных партнеров заранее. Для стартапов выбор производства — отдельная история, зачастую именно на этой фазе они задумываются о процессе производства как таковом, не раньше. Поэтому можно увидеть множество прототипов, разработанных в различных небольших компаниях и при этом очень мало изделий, которые продаются сериями.

В электронике, как правило, устройства разрабатываются под конкретное производство, т.к. на нем завязаны технологические процессы, поставка компонентов, логистика, контракты и т.д. В противном случае на этапе постановки на производство нам предстоит определить техпроцессы, цепочку поставок, согласовать логистику, связать все это в ясный план.

Выбор производственной площадки — это как выбор невесты. Несколько раз проводилась эта процедура в Юго-Восточной Азии и отсеивались неподходящие производители в поисках «того самого», который и качество обеспечит, и приемлемые коммерческие условия. Когда производство выбрано и все нюансы согласованы, начинается реальная работа. Одна из задач — создание оснастки.

#### *Производство оснастки*

Важная веха в постановке на производство — разработка и производство оснастки. Самая распространенная — пресс-форма для литья пластиковых деталей. Может производиться и любая другая оснастка, которая позволяет реализовать различные технологические операции: гибочная, штамповочная и т.д.

В общем, оснастка — это важно, качество литьевой формы напрямую скажется на качестве отлитого пластика.

Еще один интересный момент — **проверка качества оснастки**. Один корпус можно рассмотреть, изнюхать, измерить и убедиться: это то, что нужно. А как же быть, если их производятся тысячи? Как правило, делается *golden sample* — тот самый Золотой Образец, с которым необходимо сравнивать все остальные образцы в случае сомнений. Этакий эталон килограмма в палате мер и весов.

#### *Контроль качества на производстве*

В процессе производства тестируется всё и на всех этапах, т.к. любой косяк может стоить много денег и лучше основательно подготовиться с проведением испытаний. Во всяком случае, так должно быть.

Что тестируется:

- Каждый компонент у производителя компонентов. Когда эти компоненты прибывают на фабрику, они тестируются повторно непосредственно перед монтажом (выборочно или все подряд). Большинство производств тестирует всё подряд, чтобы обеспечить максимальное качество.
- Печатная плата (и у производителя, и перед запайкой).
- Оптический контроль качества монтажа.
- Внутрисхемное тестирование уже собранной платы.
- Функциональное тестирование. Для сложных устройств создаются стенды, способные быстро и качественно полностью протестировать функционал устройства и убедиться в том, что все работает.

И все равно после всех этих манипуляций средний показатель качества по индустрии — 0,5% процента брака. Для Китая приемлем 1%, т.е. каждое сотое устройство будет с косяком. Это довольно много для серии, поэтому хорошие компании устанавливают планку в 0,1% или еще ниже.

Как-то мы общались с представителем китайской торговой братии по поводу приобретения одного электронного компонента. На открытом рынке его цена была приблизительно 15 долларов за штуку, а этот парень предлагал за 2\$. На вопрос «а почему собственно так дешево» он отвечал: «Потому что там ровно 30% брака». «Так уж ровно 30%? А если больше?» — спросили мы и в ответ получили заманчивое предложение: «Тогда я вам еще насыплю».

К слову контрафактные компоненты — это серьезная опасность для любого производства. Количество поддельных, не рабочих и бракованных компонентов на рынке Юго-Восточной Азии поражает воображение.

#### *Выпуск установочной партии*

Итак, у нас готова оснастка, мы привезли нужные компоненты, понимаем свой план тестирования и готовы стартовать производство.

Все начинается с маленькой серии, например, с 10 устройств. На них мы отлавливаем ошибки в технологических цепочках. Могут быть проблемы с компонентами, с тестами, с рабочими, с фазой луны. Все эти вещи необходимо отследить и сделать так, чтобы это не влияло на качество и работоспособность продукта. Часто вносятся необходимые поправки, чтобы все работало как часы.

После того как все проблемы устранены, выпускается чуть большая партия. Скажем, 100 штук. Разные рынки накладывают разные требования к сертификации и проверке. Сертификацию часто делают на устройствах из первой партии, после чего появляется возможность легально ставить заветные CE, FCA, и так далее.

#### *Фаза серийного производства и поставки*

Серийное производство в управлеченческой теории — это хороший и понятный процесс: где-то есть «черная коробочка», в которую загружаешь кэш, а она выдает готовые изделия, которые можно грузить в корабли/самолеты и отправлять покупателям.

**На практике производство — это каждодневное решение проблем, не только технологических.** Например, компонент подорожал. Или приходит заявка на замену компонента без подвижек в сроках выпуска. Или, что хуже, в результате «оптимизации себестоимости» устройство перестало работать.

В итоге на фазу постановки и самого производства необходимо держать инженерный штат, который будет разбираться в проблемах, допиливать документацию и, конечно, допиливать ПО.

Но все проблемы решаемы, рано или поздно все начинает работать и после этого фокус перемещается на измерение качества готовой продукции.

#### *Поддержка и обслуживание*

Про это следует задуматься уже на этапе разработки продукта. Нужно ответить на вопросы: будет ли устройство обслуживаться вообще или после поломки оно должно отправиться в мусорное ведро? Кто его будет обслуживать? На территории каких стран/регионов будет обеспечиваться поддержка?

Крупные компании зачастую формируют собственные центры техподдержки близко к центрам продаж. Компании поменьше передают это на аутсорсинг.

Стоимость поддержки, как правило, изначально включена в стоимость товара. У компаний есть статистика по своему продукту, средние показатели по отрасли, а также собственные целевые показатели. Например, задана планка по затратам в 2\$ на каждое устройство. И потом менеджеры ломают голову, как это организовать. Таким образом, поддержка — это чисто затратная часть, а, значит, ее качество мало у кого является хорошим.

## *Что осталось вне инженерного отдела?*

Часть компонентов процесса разработки продукта остается за границами инженерного отдела:

- Упаковка
- Брэнд
- Логистика
- Производство
- Каналы продаж
- Сервисное обслуживание
- Инфраструктура
- Интеллектуальная собственность (патенты и т.д.)

Большинство этих вопросов находятся в компетенции **продукт-менеджера**, который, по сути, отвечает за то, чтобы продукт был успешен. Каким должен быть процесс планирования, как считать продукт «на коленке», как составлять требования и взаимодействовать с подрядчиками, чтобы получилось хорошо? Ответам на эти вопросы хотелось бы посвятить отдельный пост про продукт-менеджмент в электронике. Традиционно, при наличии интереса со стороны читателей.

## [Обзор процесса разработки программного обеспечения](#)

### *Введение*

Прежде, чем предложить обзор процесса разработки, сложившегося в результате накопления опыта за последние годы, я хотел бы сделать несколько общих пояснений, которые мне кажутся существенными.

Я работаю в ИТ последние 15 лет, хотя программированием начал заниматься значительно раньше. Основное направление моей деятельности как системного архитектора была организация разработки программ, разработка концепций и верхнеуровневой архитектуры и контроль выполнения концепции на протяжении проекта. Кроме управления разработкой ПО и создания архитектуры, я время от времени занимаюсь решением сложных технических проблем и написанием некоторых критически важных участков кода, где необходимо не только знание самого языка и среды разработки, но и их внутренней организации, иногда преподносящей неприятные сюрпризы.

Проекты, над которыми я работаю, чаще всего связаны с разработкой заказного или инвестиционного программного обеспечения. Также мне приходилось работать с встроенным ПО и программами, ориентированными на выпуск «хитов» (что, с лёгкой руки Джоэля Спольски, я называю далее игровым ПО, хотя на самом деле некоторые игровые проекты ближе к инвестиционным).

Заказное программное обеспечение может быть предназначено для внутреннего или внешнего заказчика. Эксклюзивные права на разработанную систему получает заказчик, и работа над развитием системы в дальнейшем может быть передана другому исполнителю.

В отличие от заказного ПО, работа над инвестиционным программным обеспечением ведётся самим исполнителем на деньги внутреннего или внешнего инвестора. Как правило, права на код системы остаются у исполнителя, что стимулирует непрерывную работу по улучшению своего продукта и последовательный выпуск версий с более развитой функциональностью.

Встроенное программное обеспечение поставляется вместе с аппаратной частью и, грубо говоря, не подлежит сопровождению, поскольку отзыв партии устройств производителем – дело очень затратное и потому исключительное.

Разработка игровых хитов также практически не содержит фазы сопровождения. Кроме того, пользователи игровых программ, даже столкнувшись с ошибкой в игре, очень редко загружают обновлённую версию. Поэтому разработка игр, как правило, имеет свою экономику и свой процесс разработки.

Нашиими заказчиками являются органы власти, крупные государственные и коммерческие организации и, конечно, мы сами. Поэтому в смысле заказного ПО в нашем процессе часто присутствует некоторая разница между процессами разработки продуктов для внутреннего и для внешнего заказчиков. Некоторые нюансы я укажу в этой статье. Уровень формализации отношений с заказчиком у нас варьируется от проекта к проекту очень широко. В целом, чем больше бюджет проекта, тем выше формальность. Государственный заказчик или крупные коммерческие предприятия (особенно с государственным участием) обычно имеют законодательные ограничения на формирование, размещение заказа и приёмку результатов работ. Ещё одним ограничением крупных организаций является тот факт, что их персонал, являющийся источником требований и основным пользователем наших систем, имеет очень ограниченную доступность для исполнителей, хотя бы вследствие своей занятости. Однако для небольших организаций уровень формализации падает и иногда уходит в противоположную крайность, где возникает недостаточный уровень ответственности заказчика в рамках проекта.

Другая сторона наших заказных проектов – высокие требования к функциональности. Это и высокая нагрузка на все системы, и большая географическая распределённость, и высокие требования к точности вычислений при очень ограниченных временных рамках. Часто в наших проектах появляются элементы исследовательской работы и творческого поиска, направленного на решение нетривиальных проектных задач. Иногда нам приходится комбинировать в рамках одного процесса разработки разные методологии, например, вставляя в общий процесс, близкий к RUP, один или несколько этапов почти чистого scrum, порождая что-то вроде проекта в проекте. Это позволяет нам сохранять невысокий уровень вовлеченности пользователей, связанный с природой проекта, с гибкостью разработки в условиях высокой неопределенности требований. В этом плане для меня важен именно подготовительный этап, во время которого можно выбрать необходимую методологию и выстроить оптимальный процесс разработки. Один из примеров применения гибкой методологии описан в статье «Применение agile при разработке проекта для государственного заказчика».

В качестве примера работы над инвестиционным проектом я могу привести разработку комплексной системы безопасности, которую мы создавали как «коробочный» продукт. Под моим руководством было выпущено последовательно четыре версии этой системы, пользователями которой стали самые разные коммерческие и государственные организации, включая мэрию Москвы, АФК «Система», банки, бизнес-центры и, конечно, наш собственный офис. Первая версия была не очень успешной, но у нас была стратегия развития, которая позволила нам успешно захватить рынок и пережить сложные времена кризиса. Опыт работы над этим и ещё несколькими инвестиционными проектами тоже был учтён при формировании используемого мной процесса разработки.

Наш процесс представляет собой последовательность определённых этапов. Приведённая мной классификация ПО сделана только, чтобы показать возможную разницу в организации разработки различных программных средств. Делая обзор процесса разработки, я остановлюсь только на различиях именно самого процесса касаюо разных видов ПО. Однако надо помнить, что различия между процессами разработки разных видов ПО гораздо глубже, поэтому при планировании каждого этапа необходимо учитывать эти нюансы.

Важно понимать, что переход процесса от одного этапа к другому не имеет чёткой границы. Как правило, работы следующего этапа начинаются по мере выполнения 80-90% работ по предыдущему этапу. Особенно это касается разработки требований, когда в ряде случаев снятие неопределённости происходит лишь к концу проекта. Безусловно, наличие такой неопределённости в проекте является существенным риском и должно находиться под постоянным контролем.

#### *Процесс разработки заказного ПО*

Обзор процесса разработки начнём с наиболее общего случая – разработки заказного программного обеспечения. Схема процесса приведена на рисунке 1.

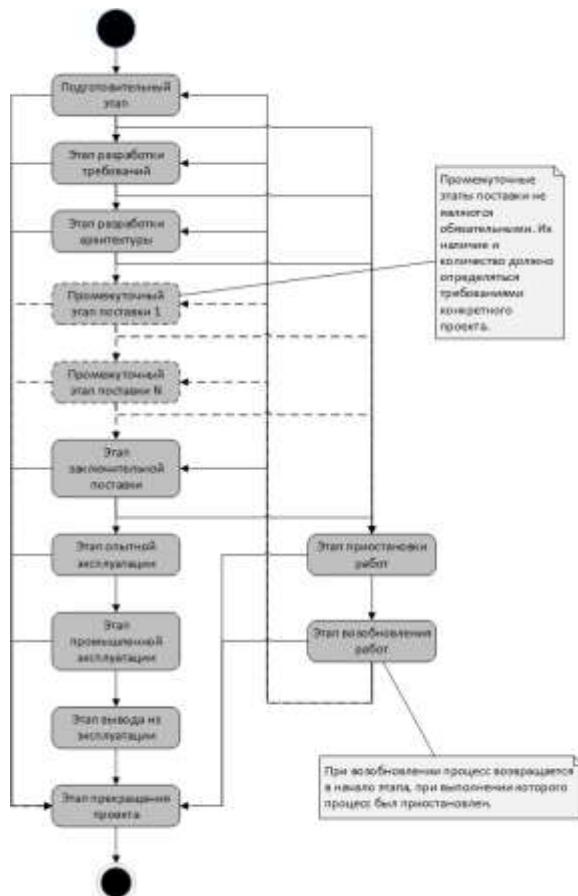


Рисунок 1. Процесс разработки заказного программного обеспечения.

Работа над проектом начинается с подготовительного этапа. Цель этапа состоит в том, чтобы на основе предложений заказчика создать некоторую концепцию будущей системы и, отталкиваясь от этой концепции, провести оценку востребованности и реализуемости проекта. Если решение о

привлечении исполнителя принимается заказчиком на конкурсной основе, то предварительный этап фактически является стадией подготовки потенциального исполнителя к конкурсу, включая формирование необходимой документации.

Не нужно тратить время и ресурсы на проект, чья концепция признаётся невостребованной или нереализуемой. Такой проект должен быть завершён. В ряде случаев требуется некоторая итеративная работа с заказчиком по коррекции концепции проекта, пока либо не будет достигнут приемлемый баланс требований заказчика и затрат исполнителя, либо не будет принято решение о сворачивании работ.

Проект, концепция которого выглядит приемлемой для реализации, выходит на этап разработки требований. На этом этапе исполнитель должен сформировать перечень всех явных и скрытых потребностей заказчика. Часто оказывается, что заказчик либо не определился со своими потребностями, либо его потребности вступают в противоречие между собой, с возможностями заказчика или с возможностями исполнителя. Целями этапа являются выявление всех скрытых потребностей, решение конфликтов требований, формирование целостного технического решения и анализ реализуемости подготовленного решения.

Иногда уточнение требований приводит к пересмотру концепции проекта. Если после уточнения всех требований не удается найти приемлемого технического решения, проект приходится сворачивать либо откладывать на некоторое время в ожидании более приемлемых обстоятельств.

Если техническое решение найдено, исполнитель приступает к разработке архитектуры будущей системы. Цель этапа – определение верхнеуровневой логической и физической архитектуры, полностью покрывающей все требования заказчика. При разработке архитектуры проводится рецензирование и уточнение концепции, требований и предварительного технического решения, что даёт возможность предупредить наиболее опасные риски.

После завершения проектирования архитектуры необходимо снова провести ревизию основных параметров проекта и решить, в состоянии ли исполнитель завершить проект. Полезно на стадии разработки архитектуры отказаться от излишних и слишком громоздких функций. Оптимизация архитектурного решения часто помогает вписаться в приемлемые параметры проекта. В иных случаях требуется более радикальное сокращение функционала разрабатываемой системы. Однако даже остановка проекта на этой стадии, если она происходит по веским причинам, должна восприниматься как победа: продолжение работ в таком случае может привести только к ещё большим потерям.

Если баланс был найден, и удалось создать приемлемую архитектуру системы, исполнитель может переходить к реализации и поставке системы. Реализация может проходить в один или несколько этапов. Для небольших проектов одноэтапная поставка всего функционала системы может быть вполне приемлемой. Однако, чем больше проект, тем выше зависимости подсистем внутри создаваемой системы. В этих условиях следует делить реализацию на несколько этапов так, чтобы в конце каждого этапа команда разработчиков имела готовый к поставке продукт. При этом самый важный, фундаментальный функционал должен разрабатываться на ранних этапах, а надстройки, работающие поверх этих основных компонентов, следует реализовывать позднее. В таком случае наиболее опасные для системы ошибки будут исправлены на первых этапах, и риск того, что

прикладная функциональность системы будет основана на нестабильной основе, будет значительно снижен.

После поставки полностью завершённой системы проект заказного ПО обычно переходит к этапу опытной эксплуатации. Цель этого этапа заключается в проверке качества работы разработанной системы в реальных условиях эксплуатации. Как правило, на этом этапе исполнитель совместно с заказчиком проводит измерение количественных метрик, позволяющих определить качество созданной системы. В первую очередь проверяются функциональные характеристики качества, затем – нефункциональные. При наличии несоответствий исполнитель корректирует код системы.

Полностью отлаженная и настроенная система вводится в промышленную эксплуатацию. Как правило, исполнитель должен сопровождать систему, по крайней мере, в течение срока гарантии. Выявляемые несоответствия должны исправляться. Пользователи и обслуживающий персонал заказчика должны получат оперативную консультативную поддержку.

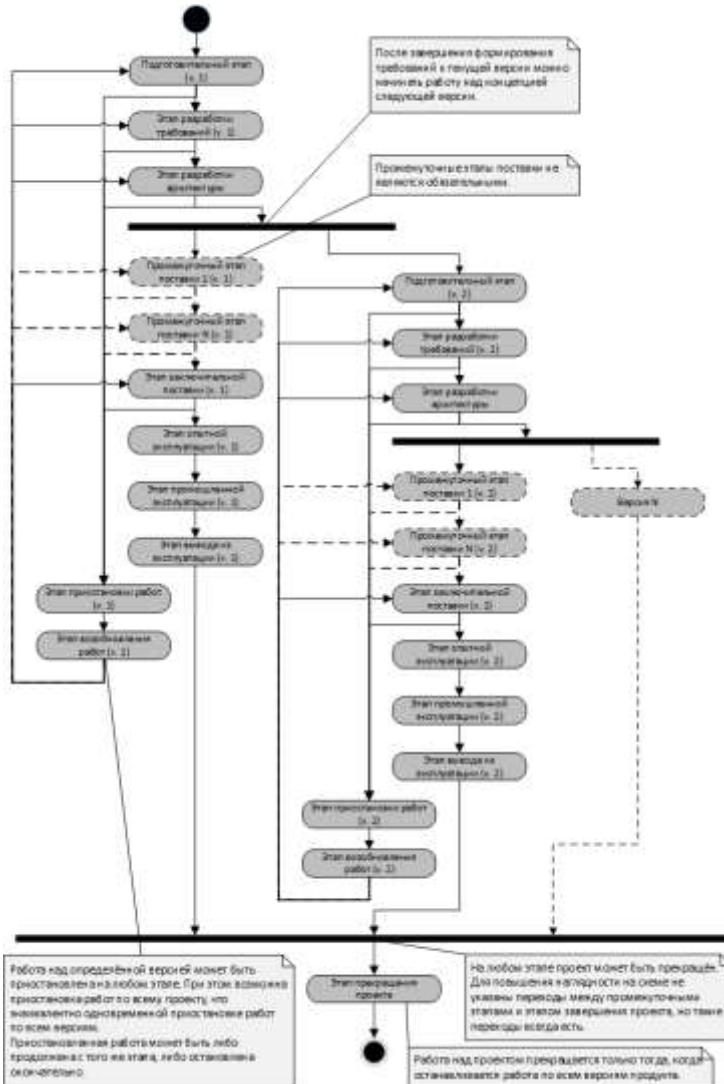
Наконец, приходит момент, когда система перестаёт устраивать заказчика по какой-либо причине. Наступает этап вывода системы из эксплуатации. Впрочем, для заказного ПО этот этап не всегда актуален, поскольку заказчик может воспользоваться своими эксклюзивными правами на систему и отстранить исполнителя от дальнейших работ по сопровождению и развитию системы ещё до того, как она потеряет актуальность.

Любой проект в конечном счёте приходит к своему завершению. Этап прекращения проекта имеет целью анализ результатов, внесение изменений в процесс разработки на основе полученного опыта и пополнение базы знаний разработчиков новыми эффективными решениями и предостережениями, а также новыми готовыми компонентами, которые можно будет использовать в следующих проектах.

Осталось отметить ещё два этапа процесса разработки. Бывает, что обстоятельства не позволяют продолжать реализацию проекта, но результаты проделанной работы показывают, что у проекта может быть будущее. Закрывать такой проект преждевременно. Поэтому вместо полной остановки работ исполнитель может временно приостановить деятельность по проекту, зафиксировав достигнутые результаты. Как только обстоятельства позволят, проект можно будет возобновить, расконсервировав инфраструктуру, вернув в проект разработчиков и восстановив состояние проекта. Важно, однако, возобновлять работу с того этапа, на котором проект был прерван, повторно проведя ревизию достигнутых результатов.

### *Процесс разработки инвестиционного ПО*

Процесс разработки инвестиционного ПО отличается тем, что параллельно может идти работа сразу над несколькими версиями продукта: пока первая версия сопровождается, вторая уже реализуется, а для третьей формулируются требования. Процесс показан на рисунке 2.



**Рисунок 2. Процесс разработки инвестиционного программного обеспечения.**  
Как нетрудно заметить, при разработке инвестиционного ПО имеют место те же этапы, которые были рассмотрены выше для процесса разработки заказного программного обеспечения. Но отличие состоит в том, что этапы относятся не ко всему продукту, а к отдельной версии продукта. Исключение составляет этап прекращения проекта: проект не может завершиться, пока идёт работа хотя бы над одной версией продукта.

Обратите внимание на момент начала работ над следующей версией продукта. Этот момент настает, как только пройден этап создания архитектуры текущей разрабатываемой версии. До этого на этапах формирования требований и создания архитектуры, как правило, идёт обсуждение, какие функции следует реализовать в текущей версии, а какие перенести на будущее. И только тогда, когда требования к текущей версии сформулированы, рецензированы и подтверждены архитектурой системы, имеет смысл думать о следующей версии.

Кроме того, после разработки архитектуры, как правило, у аналитиков и архитекторов проекта появляется некоторая свобода действий, поскольку на этапах поставки основная нагрузка ложится

на программистов. Эту свободу можно использовать для проработки концепции и требований для следующей версии.

В принципе, можно перенести начало работ над следующей версией на более поздний срок. Например, вполне допустимо сначала ввести текущую версию в опытную или даже промышленную эксплуатацию, и только после этого начать работу над следующей версией. Но нужно помнить, что такое решение неприменимо в случае высокой конкуренции: вас просто опередят и выдавят с рынка. Решение нужно принимать, исходя из всего комплекса обстоятельств, влияющих на ваш бизнес.

Говоря о процессе разработки инвестиционного ПО, нужно понимать, что работа над несколькими версиями имеет ряд явных и скрытых взаимозависимостей между параллельными ветками процесса.

Во-первых, исправления несоответствий, выявленных в ранней версии, должны вноситься и в версию, где они были обнаружены, и во все более поздние версии, включая разрабатываемые. Это касается не только кода программы, но и всех остальных артефактов проекта: технической и пользовательской документации, справочной системы, оценок и планов работ и т.п. Причём исправления должны вноситься немедленно, поскольку уменьшить стоимость исправлений вам не удастся, но, если не внести исправления сразу, их стоимость на более поздних стадиях может увеличиться в десятки и даже сотни раз.

Во-вторых, для параллельной работы над несколькими версиями нужна особая инфраструктура проекта, включая организацию контроля версий кода и документации, контроля заданий и несоответствий, утилит автоматической сборки и тестирования и т.п. Нельзя допустить, чтобы работа над одной версией продукта блокировала выполнение задач по другим версиям только из-за того, что инфраструктура проекта не позволяет запустить два процесса сборки одновременно для разных версий продукта.

Особое внимание нужно уделить стендам, на которых проводится тестирование: на них должны быть развернуты все версии продукта, которые были выпущены ранее (по меньшей мере, те версии, которые сопровождаются), и все версии, разработка которых ведётся в настоящий момент.

В-третьих, в работе над несколькими версиями могут быть одновременно задействованы одни и те же участники. Имеется большой риск, что ключевой сотрудник может погрязнуть в работе над одной версией программы и допустить существенное превышение сроков по задачам, связанным с другой версией.

В-четвёртых, имеет место обратная ситуация, когда персонал, работающий над одной версией, ничего не знает о том, какие решения принимаются в рамках работ над другой версией. Частично проблема снимается, если исправления всей документации и кода будут немедленно распространяться на все более поздние версии, о чём я говорил выше. Но одними исправлениями дело не должно ограничиваться. Нужно, чтобы команда, работающая над одной версией, понимала, почему были приняты те или иные решения при работе над другой версией. Для этого нужна база знаний для разработчиков – специальная информационная система, в которой должны описываться все проблемы, с которыми столкнулись разработчики при работе над той или иной версией продукта, и способы решения этих проблем. База знаний должна рассыпать всем

участникам проекта уведомления о поступлении новых записей. Нельзя пускать на самотёк взаимодействие двух команд, работающих над разными версиями одного продукта.

### **Процесс разработки встроенного ПО**

Как уже отмечалось выше, встроенное ПО отличается от заказного тем, что его крайне сложно сопровождать.

Допустим, вы выпускаете программы для холодильников. После того, как ПО поставлено производителю, десятки тысяч устройств начинают расходиться по всему миру, и вы понятия не имеете, где они окажутся. И если один из холодильников выйдет из строя по вине вашего софта, то проще заплатить неустойку, чем возвращать холодильник на завод и проводить диагностику. Конечно, можно подготовить инженеров для дилерских центров, которые смогут провести диагностику на месте и заменить прошивку вашей системы, но это всё равно очень дорого.

Таким образом, при разработке встроенного ПО возникает сразу несколько важных ограничений.

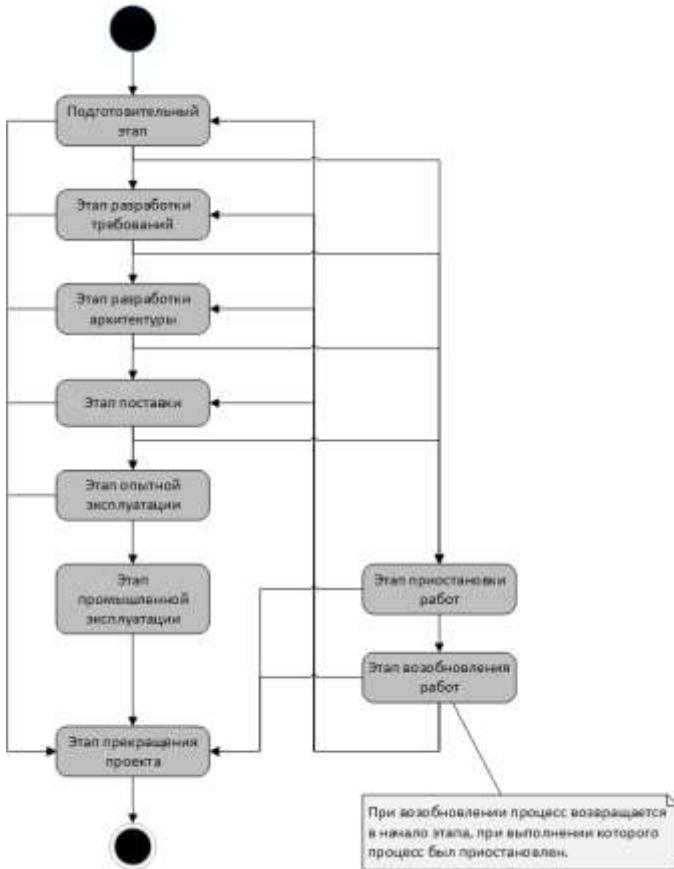
Во-первых, поставка выполняется в рамках только одного этапа: никто не будет встраивать в устройства наполовину работающую программу.

Во-вторых, при поставке вы должны уделить особое внимание качеству программы, поскольку с момента внедрения её внутрь железного ящика менять её будет очень сложно. Особое внимание нужно уделить этапу опытной эксплуатации, когда программа внедряется в ограниченную партию устройств, и эти устройства проходят комплексные испытания в различных режимах эксплуатации. Вы должны собрать максимум информации о динамике поведения вашей системы, проанализировать эту информацию и доработать ПО.

В-третьих, когда устройство с вашим ПО ушло в серию, вы имеете очень мало возможностей для исправления ошибок. По факту, такие исправления возможны только в случае брака ПО, приводящего к неработоспособности всей партии устройств, из-за чего производитель будет вынужден отзывать эту партию, а вы получите большое чёрное пятно на свою репутацию.

Наконец, в-четвёртых, этапа вывода из эксплуатации у встроенного ПО нет. Программу просто выбрасывают вместе с устройством. Поэтому, как только для партии устройств, в которых работает ваше ПО, истекает гарантийный срок, можно переходить к закрытию проекта.

Процесс разработки встроенного ПО показан на рисунке 3.



*Рисунок 3. Процесс разработки встроенного программного обеспечения.*

#### *Процесс разработки игр*

Игровое программное обеспечение было выделено мной по причине специфики их производства и эксплуатации. Бизнес игрового ПО основан на выпуске хитов. Один успешный хит оплачивает расходы на создание нескольких игр, которые остаются незамеченными пользователями. Поэтому процесс разработки одной игры взаимосвязан с процессами разработки других игр.

Ещё одним фактором, выделяющим производство игр, является тот факт, что игра интересна пользователю либо пока он не прошёл последний уровень, либо пока у него не произошла фатальная ошибка. Это значит, что вторую версию игры он не будет покупать или даже бесплатно загружать только ради исправлений нескольких ошибок.

Указанные факторы сказываются на процессе разработки игрового ПО. Процесс представлен на рисунке 4.

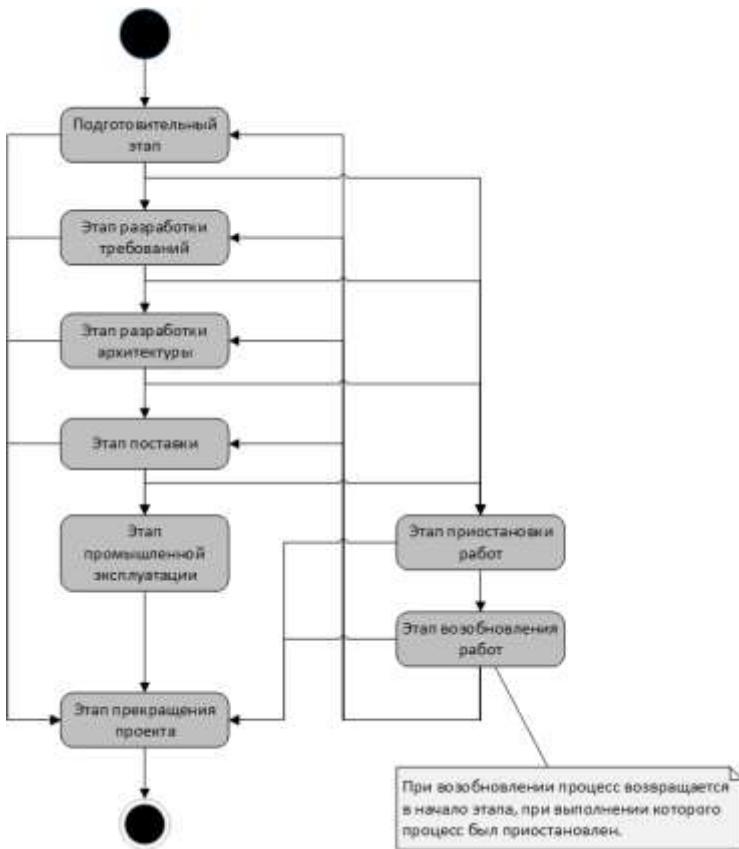


Рисунок 4. Процесс разработки игрового программного обеспечения.

Нужно отметить следующие особенности процесса разработки игрового ПО.

Прежде всего, при производстве игр крайне важно качество концепции. Если концепция игры не позволяет создать хит, то дальнейшая работа бессмысленна. Ситуация, когда большинство проектов заканчиваются на подготовительном этапе, для разработки игрового ПО типична.

При разработке требований и архитектуры для игрового ПО часто повторно используются наработки, полученные при работе над предыдущими проектами. В этом плане также дополнительный вес получает этап прекращения проекта, когда все полезные наработки должны быть зафиксированы в базе знаний разработчиков.

Поставка игрового программного обеспечения происходит в рамках одного единственного этапа. Даже если сначала создаётся некое ядро, «движок» игровой системы, его работу невозможно проверить без реализации всего функционала системы.

Для игрового ПО нет этапов опытной эксплуатации и вывода из эксплуатации. Игры сразу поступают в продажу, а после использования просто удаляются пользователем по мере утраты интереса к ним.

### Заключение

Каждый этап процесса, безусловно, нуждается в отдельном обсуждении с обязательным учётом особенностей разрабатываемых программных средств.

Отмечу, что рассматриваемая здесь схема процесса является результатом обобщения моего личного опыта разработки различных программных средств. Как любое обобщение, моя схема является абстракцией. И, как любая абстракция, у неё есть свои границы применимости. Нельзя бездумно применять эту схему к конкретному проекту. Важно понимать, что каждый проект имеет свои нюансы, влияющие на организацию процесса разработки. И поэтому для каждого проекта приведённую здесь схему нужно адаптировать, а в ряде случаев потребуется разработать принципиально другой подход.

## Парадигмы программирования

### Введение

**Парадигма программирования** — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Своим современным значением в научно-технической области термин «парадигма» обязан, по-видимому, Томасу Куна и его книге «Структура научных революций». Кун называл парадигмами устоявшиеся системы научных взглядов, в рамках которых ведутся исследования. Согласно Куну, в процессе развития научной дисциплины может произойти замена одной парадигмы на другую (как, например, геоцентрическая небесная механика Птолемея сменилась гелиоцентрической системой Коперника), при этом старая парадигма ещё продолжает некоторое время существовать и даже развиваться благодаря тому, что многие её сторонники оказываются по тем или иным причинам неспособны перестроиться для работы в другой парадигме.

Термин «парадигма программирования» впервые применил в 1978 году Роберт Флойд - лауреат премии Тьюринга.

Флойд отмечает, что в программировании можно наблюдать явление, подобное парадигмам Куна, но, в отличие от них, парадигмы программирования не являются взаимоисключающими:

Если прогресс искусства программирования в целом требует постоянного изобретения и усовершенствования парадигм, то совершенствование искусства отдельного программиста требует, чтобы он расширял свой репертуар парадигм.

Таким образом, по мнению Роберта Флойда, в отличие от парадигм в научном мире, описанных Куном, парадигмы программирования могут сочетаться, обогащая инструментарий программиста.

Важно отметить, что парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм (мультипарадигмальное программирование). Так, на языке Си, который не является объектно-ориентированным, можно работать в соответствии с принципами объектно-ориентированного программирования, хотя это и сопряжено с определёнными сложностями; функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции.

Первой парадигмой, с которой знакомят современные ВУЗы в первую очередь - ООП. Она легка в освоении, за счет проецирования разработки на повседневную жизнь. Но ведь *список парадигм* не

ограничивается объектно-ориентированным. Он намного больше: автоматное программирование, аспектно-ориентированное, визуальное, вычисления с откатами, декларативное, императивное, конкатенативное, логическое, матричное, метапрограммирование, мультипарадигма, на уровне значений, на уровне функций, строгое, нестрогое, обмен сообщениями, обобщённое, объектно-ориентированное, параллельное, потоковое, правила переписывания, предметно-ориентированное, прототипное, процедурное, рефлексивное, скалярное, стек-ориентированное, структурное, табличное, функциональное, эзотерическое (более 30 штук).

## Объектно-ориентированное программирование

Представляет программу как набор объектов и их взаимодействий.

### Основные понятия

- *объект* — элементарная сущность, описываемая определенными свойствами (хранищимися в виде атрибутов объекта) и поведением (реализованным в виде методов);
- *класс* описывает структуру свойств и поведения одного типа объектов. Каждый объект программы является экземпляром некоторого класса;
- классы могут *наследовать атрибуты и методы* их родительских классов, в то же время добавляя свои собственные. Иерархия классов позволяет моделировать сущности решаемой задачи на нескольких уровнях детализации и в дальнейшем использовать класс, отвечающий уровню детализации, необходимому для решения конкретной подзадачи;
- *инкапсуляция* подразумевает, что некоторые детали реализации класса скрыты от взаимодействующих с ним объектов. У каждого класса есть интерфейс, описывающий взаимодействие объектов этого класса с прочими объектами, и реализация, описывающая то, как это взаимодействие отражается на объекте этого класса.

## Императивное программирование

Исторически сложилось так, что подавляющее большинство вычислительной техники, которую мы программируем имеет состояние и программируется инструкциями, поэтому первые языки программирования в основном были чисто императивными, т.е. не поддерживали никаких парадигм кроме императивной. Это были машинные коды, языки ассемблера и ранние высокоуровневые языки, вроде Fortran.

### Ключевые моменты

В этой парадигме вычисления описываются в виде инструкций, шаг за шагом изменяющих состояние программы.

В низкоуровневых языках (таких как язык ассемблера) состоянием могут быть память, регистры и флаги, а инструкциями — те команды, что поддерживает целевой процессор.

В более высокоуровневых (таких как Си) состояние — это только память, инструкции могут быть сложнее и вызывать выделение и освобождение памяти в процессе своей работы.

В совсем высокоуровневых (таких как Python, если на нем программировать императивно) состояние ограничивается лишь переменными, а команды могут представлять собой комплексные операции, которые на ассемблере занимали бы сотни строк.

### Языки поддерживающие данную парадигму

Как основную:

- Языки ассемблера
- Fortran
- Algol
- Cobol
- Pascal
- C
- C++
- Ada

Как вспомогательную:

- Python
- Ruby
- Java
- C#
- PHP
- Haskell (через монады)

Стоит заметить, что большая часть современных языков в той или иной степени поддерживает императивное программирование. Даже на чистом функциональном языке Haskell можно писать императивно.

### Процедурное программирование

Опять-же возрастающая сложность программного обеспечения заставила программистов искать другие способы описывать вычисления.

Собственно, еще раз были введены дополнительные понятия, которые позволили по-новому взглянуть на программирование.

Этим понятием на этот раз была процедура.

В результате возникла новая методология написания программ, которая приветствуется и по сей день — исходная задача разбивается на меньшие (с помощью процедур) и это происходит до тех пор, пока решение всех конкретных процедур не окажется тривиальным.

### Ключевые моменты

**Процедура** — самостоятельный участок кода, который можно выполнить как одну инструкцию.

В современном программировании процедура может иметь несколько точек выхода (return в C-подобных языках), несколько точек входа (с помощью yield в Python или статических локальных переменных в C++), иметь аргументы, возвращать значение как результат своего выполнения, быть перегруженной по количеству или типу параметров и многое еще.

### Функциональное программирование

Парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Противопоставляется парадигме императивного программирования, которая описывает процесс вычислений как последовательное изменение состояний (в значении, подобном таковому в теории автоматов). При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например, как список.

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь побочные эффекты и менять состояние внешних переменных. Таким образом, в императивном программировании при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, можно получить разные данные на выходе из-за влияния на функцию состояния переменных. А в функциональном языке при вызове функции с одними и теми же аргументами мы всегда получим одинаковый результат: выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках кешировать результаты функций и вызывать их в порядке, не определяемом алгоритмом и распараллеливать их без каких-либо дополнительных действий со стороны программиста.

### Аспектно-ориентированное программирование

Существующие парадигмы программирования — процедурное, модульное, объектно-ориентированное программирование (ООП) и предметно-ориентированное проектирование — предоставляют определённые способы для разделения и выделения функциональности: функции, модули, классы, но некоторую функциональность с помощью предложенных методов невозможно выделить в отдельные сущности. Такую функциональность называют сквозной (от англ. *scattered* — разбросанный или англ. *tangled* — переплетённый), так как её реализация распределена по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному коду, сложному для понимания и сопровождения.

Ведение лога и обработка исключений — типичные примеры сквозной функциональности. Другие примеры: трассировка; аутентификация и проверка прав доступа; контрактное программирование (в частности, проверка пред- и постусловий). Для программы, написанной в парадигме ООП, любая функциональность, по которой не была проведена декомпозиция, является сквозной.

Все языки АОП предоставляют средства для выделения сквозной функциональности в отдельную сущность. Так как *AspectJ* является родоначальником этого направления, используемые в этом расширении концепции распространились на большинство языков АОП.

### Основные понятия

- **Аспект** (англ. aspect) — модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.

- **Совет** (англ. advice) — средство оформления кода, которое должно быть вызвано из точки соединения. Совет может быть выполнен до, после или вместо точки соединения.
- **Точка соединения** (англ. join point) — точка в выполняемой программе, где следует применить совет. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.
- **Срез** (англ. pointcut) — набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в AspectJ применяются Java-сигнатуры) и позволяют их повторное использование с помощью переименования и комбинирования.
- **Внедрение** (англ. introduction, введение) — изменение структуры класса и/или изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (англ. metaobject protocol, МОР).

### Метапрограммирование

Предусматривает написание программ, которые работают с другими программами в качестве данных. Язык обрабатывающей программы называется *метаязыком*, язык обрабатываемой — *объектным языком*.

Простейшим примером метапрограммирования является любой компилятор, преобразующий код, написанный на языке высокого уровня, в низкоуровневый машинный язык или ассемблер. Очевидно, что большинство языков, поддерживающих работу со строками, могут использоваться для непосредственной генерации кода для других языков. Тем не менее, термин “метапрограммирование” обычно подразумевает, что в качестве метаязыка и объектного языка выступает один и тот же язык, и более того, такое его использование предусмотрено дизайном языка.

Примеры языков: Rust, Perl, диалекты Lisp: Clojure, Common Lisp, Scheme.

### Программирование на уровне значений

Программирование на уровне значений (также известное как модель фон Неймана) представляет программу в виде последовательности значений, преобразующихся друг в друга. Выполнение программы начинается с исходных данных, которые комбинируются и преобразуются в другие значения до тех пор, пока не получаются нужные результаты. Новые значения конструируются из имеющихся при помощи предопределенного набора операций.

Эта парадигма сосредотачивается на изучении типов данных, т.е. значений и элементарных операций над ними, их структуры и свойств. Обычно элементарные операции образуют алгебру над пространством значений.

Большинство современных языков используют понятия типов данных, переменных и операторов присваивания, и, следовательно, реализуют эту парадигму.

Программирование на уровне значений является противоположностью программирования на уровне функций.

Примеры языков: ECMAScript, C++, C#, Java, Brainfuck, Scala.

## Программирование на уровне функций

Программирование на уровне функций (другое русское название — комбинаторное программирование) предполагает, что программа строится из элементарных функций, комбинируемых при помощи функционалов (функциональных форм).

Эта парадигма не использует понятия переменной или операции присваивания, а вместо этого сосредотачивается на изучении элементарных функций и функциональных форм.

### Основные понятия

- **Атомы** - единицы данных, с которыми оперируют функции. Данные появляются только на входе и выходе программы, и нигде внутри. Атомы могут быть скалярами или множествами других атомов.
- **Функции** - инструменты, преобразующие атомы в другие атомы. Язык задает начальный набор функций, а программист может определять новые, используя функциональные формы. Сама программа тоже является функцией.
- **Функциональные формы** - инструменты, преобразующие функции в другие функции. Язык задает начальный набор функциональных форм, и либо позволяет создание новых форм (FFP), либо нет (FP). Таким образом, язык задает алгебру функциональных форм над пространством функций.

Программирование на уровне функций является противоположностью программирования на уровне значений и разновидностью функционального программирования (с ограничением на то, как создаются новые функции).

Примеры языков: APL, FP, J.

### Строгое программирование:

В строгом языке программирования могут быть определены только строгие функции, то есть функции, которые придерживаются строго модели вычислений.

Строгая модель вычислений означает, что аргументы всегда вычисляются полностью до применения функции к ним. Нестрогая модель вычислений означает, что аргументы не вычисляются до тех пор, пока их значение не используется в теле функции. В ряде языков булевые выражения имеют нестрогий порядок вычисления, называемый в русской литературе «вычислениями по короткой схеме», где вычисления прекращаются, как только результат становится однозначно предсказуем — например, значение «истина» в операции дизъюнкции, «ложь» в операции конъюнкции, и так далее. Операторы ветвления зачастую также имеют ленивую семантику вычислений, то есть возвращают результат всего оператора, как только однозначная ветвь его породит.

### Нестрогое программирование:

Нестрогий язык программирования позволяет пользователю определять нестрогие функции и, как следствие, использовать ленивые вычисления.

В большинстве нестрогих языков «нестрогость» также применяется к конструкторам данных. Это позволяет управлять бесконечными структурами данных (к примеру, списком всех простых чисел)

точно так же, как и обычными конечным. Это облегчает использование очень больших, но конечных структур, к примеру, таких, как полное дерево игры в шахматы.

Нестрогость имеет некоторые недостатки, которые помешали её повсеместному использованию: \* из-за неопределённости касательно того, будут ли и когда будут вычислены выражения, нестрогие языки должны быть чисто функциональными для удобного использования; *большинство распространённых архитектур оптимизированы для строгих языков, т. о. лучшие компиляторы для нестрогих языков обычно производят код хуже, чем лучше компиляторы строгих;* пространственную сложность (space complexity) нестрогих программ сложно понять или предсказать.

Термины «энергичные языки программирования» (eager) и «ленивые языки программирования» (lazy) часто используются как синонимы «строгие языки программирования» и «нестрогие языки программирования» соответственно.

Во многих строгих языках некоторые положительные стороны нестрогих функций могут быть достигнуты с помощью макросов.

Примеры языков: Аналитик, Miranda.

### Обобщенное программирование

Состоит в написании алгоритмов в терминах абстрактных типов данных; когда алгоритм используется для конкретных типов данных, создается экземпляр этого алгоритма с типами данных, переданными в качестве параметров. Такой стиль программирования позволяет использовать универсальный код для похожих задач, имеющих дело с разными типами данных, и таким образом уменьшить дублирование кода.

Обобщённое программирование широко используется для реализации универсальных контейнеров и алгоритмов. Так, стандартная библиотека шаблонов STL в C++ предоставляет набор контейнеров (динамический массив, связный список, очередь, множество и т.д.) и алгоритмов, применимых к этим или пользовательским контейнерам.

### Стек-ориентированное программирование

Стек-ориентированная парадигма программирования использует для передачи параметров модель стека.

Стек-ориентированный язык программирования оперирует одним или несколькими стеками и обычно использует префиксную или постфиксную нотацию вместо инфиксной, обычной для других языков. Две основные операции, которые выполняются над данными в стеке — *pop* (удалить верхний элемент и вернуть его) и *push* (добавить элемент в верх стека). Иногда стек-ориентированные языки предоставляют и более сложные операции, например, *dup* (скопировать верхний элемент стека и добавить его в верх стека), *swap* (поменять местами два верхних элемента стека), *roll* (циклически переставить элементы в заданной части стека) и *drop* (удалить верхний элемент стека, не возвращая его).

### Скалярное программирование

Низкоуровневая парадигма, диктующая отсутствие в языке матричных операций. Каждая операция применяется к отдельным скалярным величинам, но не ко всему массиву. Таким образом,

программист должен организовать обработку массива как последовательность скалярных операций.

### Эзотерическое программирование

Языки, поддерживающие данную парадигму создаются не с серьезными намерениями, а в качестве шутки или вызову самому себе.

Примеры языков: Brainfuck, Cat.

### Заключение

Важно заметить, что универсальной парадигмы, или просто самой хорошей парадигмы не существует. Также крайне сложно сказать возможно ли это вообще. Все парадигмы имеют свои области применения, свои плюсы и минусы. Даже то же ООП, которое в современном программировании занимает лидирующие позиции, не обделено своими минусами. (Статья приложения "почему ООП провалилось"). Но вместе с тем, оно заняло свою нишу. Причин тому много, но одна из них кроется в то, что программирование — это искусство оперирования абстрактной информацией. Это очень сложное искусство — поскольку человеческий мозг вообще не приспособлен для оперирования абстракциями. А ООП перекладывает абстракции на вполне осозаемые сущности (в общем случае). Сейчас программист, непонимающий основ ООП, - либо плохой специалист, либо специалист, занявший свою нишу.

### Приложение

Почему ООП провалилось ([http://blogerator.ru/page/oop\\_why-objects-have-failed](http://blogerator.ru/page/oop_why-objects-have-failed))

Абсурдопедия (<http://absurdopedia.wikia.com/wiki/ООП>)

## Стили программирования

### Введение

Стили программирования можно сопоставить со стилями одежды. Классический стиль приемлем в любой ординарной обстановке, хотя иногда может быть и функционально неудобным, и не очень подходящим в других отношениях.

Поскольку стиль программирования — неформализуемое понятие очень высокого уровня, строгого определения дать невозможно. Поэтому охарактеризуем его и другие, тесно взаимосвязанные с ним, понятия следующим образом.

Под стилем программирования понимается внутренне согласованная совокупность программ, обладающих общими фундаментальными особенностями, как логическими, так и алгоритмическими, и базовых концепций, связанных с этими программами.

Стиль программирования реализуется через методологии программирования, заключающиеся в совокупности соглашений о том, какие базовые концепции языков программирования и какие их сочетания считаются приемлемыми или неприемлемыми для данного стиля. Методология включает в себя модель вычислителя для данного стиля.

Перейдем теперь к конкретным рассмотрениям. Поскольку теории стилей в настоящий момент просто нет, рассмотрим практически сложившиеся основные стили построения программ, несколько упорядочив и отцензуривав их список в соответствии с концепциями:

- Программирование от состояний;
- Структурное программирование;
- Сентенциальное программирование;
- Программирование от событий;
- Программирование от процессов и приоритетов;
- Функциональное программирование;
- Объектно-ориентированное программирование;
- Программирование от переиспользования;
- Программирование от шаблонов.

Это перечисление не претендует на теоретическую полноту и даже на полную обоснованность. Здесь мы стремимся сосредоточиться на классификации методов и отвлечься от конкретных частностей выражения.

*Парадигма программирования* — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). В статье Дэниела Боброва парадигма определяется как «стиль программирования как описания намерений программиста».

Часто в статьях, когда пишут про парадигмы программирования, в скобочках пишут словосочетание «стиль программирования».

Так что многие не выделяют отличий между этими понятиями. Однако источников, описывающих парадигмы или стили достаточно много. Так что иногда их классификация может быть различна в разных источниках.

Почти всегда для языка есть вариант писать под разные стили. Тот же LISP является функциональным ЯП, но также обладает чертами императивности, также позволяет реализовать ООП.

Однако некоторые языки, можно сказать, относят к некоторым стилям, полагаясь на то, что в основном на этих языках работают именно на этих конкретных стилях.

К примеру, к структурному стилю относят Fortran, Pascal, C. К функциональному Lisp и Haskell, к ООП Java, C++, Python.

### [Программирование от состояний](#)

Это — пожалуй, самый старый стиль программирования. Он соответствует теоретическому понятию конечного автомата. На этот стиль программирования наталкивает само устройство существующих вычислительных машин, которое представляют собой гигантские конечные автоматы.

Суть программирования от состояний можно охарактеризовать следующим образом.

Определяются:

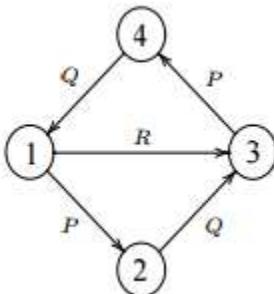
- Множество так называемых состояний, которые может принимать конечный автомат;
- Переходы между состояниями, которые осуществляются под внешним воздействием (например, под воздействием перерабатываемых данных).

Программа, написанная в таком стиле, является *перечнем команд*, фиксирующих переходы между состояниями. Если же говорить в более ‘теоретических’ терминах, то для каждой возможной пары (состояние, внешнее воздействие) указывается очередное состояние.

Описание такой программы может быть произведено разными способами. Многие из них хорошо изучены теоретически и поддерживаются развитыми методиками программирования. Современные методики программирования от состояний базируются на таблицах состояний, подобных таблице состояний конечного автомата. Эти таблицы часто также представляются в виде графов, что особенно удобно, когда не все возможные переходы между состояниями реализуемы.

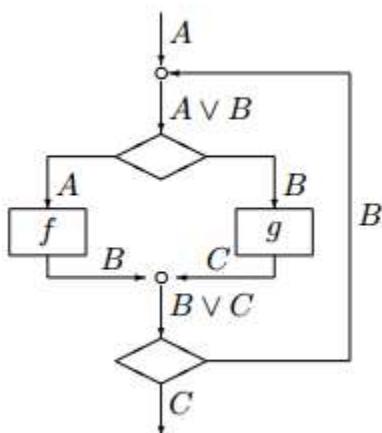
	a	b	c	d	
1	2	3	4	1	
2	3	4	1	2	
3	4	1	2	3	
4	1	2	3	4	

Здесь состояния идентифицируются порядковыми номерами, а воздействия — буквами.



Заметим, что естественно рассматривать таблицы состояний и переходов как недетерминированные, поскольку после выполнения действия вполне может оказаться истинно сразу несколько условий, соответствующих выходящим ребрам.

Исторически первой моделью программирования от состояний, использованной и на практике, и для теоретических исследований, явилось представление программы в виде *блок-схемы*, узлы которой представляют собой состояния.



Заметим, что, операции в программировании от состояний глобальны, а условия локальны. Проверка условия не изменяет состояния всей системы (ни одного из ее параметров или характеристик), она лишь ведет нас в то или иное состояние самой программы.

Важно подчеркнуть, что для стиля программирования от состояний *характеристическим качеством* является не конкретное представление автомата, а то, что программист заставляет себя думать о

разрабатываемой или исследуемой программе, как о таком автомате. Во многих случаях эта позиция оправдана и приносит существенные плоды (например, когда требуется перерабатывать потоки данных). Но далеко не всякая обработка соответствует автоматному мышлению. В частности, когда сложность алгоритма превышает определенные пределы и обозримость автомата не может быть достигнута, данный стиль мышления и, как следствие, стиль программирования становится неприемлемым.

### Структурное программирование

В теории схем программ было замечено, что некоторые случаи блок-схем легче поддаются анализу. Поэтому естественно было выделить такой класс блок-схем, что и сделали итальянские ученые С. Бем и К. Джакопини в 1966 г. Они доказали, что любую блок-схему можно преобразовать к структурированному виду, использовав несколько дополнительных булевых переменных. Э. Дейкстра подчеркнул, что программы в таком виде, как правило, являются легче понимаемыми и модифицируемыми, так как каждый блок имеет один вход и один выход. Эти наблюдения послужили основой стиля программирования, который, начиная с 70-х гг. XX века, занимает практически монопольное положение в преподавании и исключительно широко используется в практике программирования. Этот стиль называется **структурным программированием**.

Структурное программирование основано главным образом на теоретическом аппарате теории рекурсивных функций. Программа рассматривается как частично-рекурсивный оператор над библиотечными подпрограммами и исходными операциями. Структурное программирование базируется также на теории доказательств, прежде всего на естественном выводе. Структура программы соответствует структуре простейшего математического рассуждения, не использующего сложных лемм и абстрактных понятий.

Структурное программирование естественно возникает во многих классах задач, прежде всего в таких, где задача естественно расщепляется на подзадачи, а информация — на достаточно независимые структуры данных. Для подобных задач — это, безусловно, лучший стиль программирования.

### Сентенциальное программирование

Сентенциальное программирование отличается следующими особенностями:

- Память представляется как единая большая структура данных. Состояние программы (поле зрения) задается как выражение либо совокупность выражений. В некоторых из вариантов сентенциального программирования эти выражения могут содержать переменные, а в других — нет.
- В поле зрения по некоторым правилам может быть выделена активная часть, т. е. рассматриваемая в данный момент.
- На каждом шаге выполнения программы глобальные свойства активной части памяти подвергаются анализу. Выполняется не тот оператор, который стоит следующим в программе либо на который явно передано управление, а тот, который соответствует условию, выполненному для данного состояния внутреннего мира программы.
- Каждое действие программы состоит в замене одного выражения на другое: в традиционной для таких языков записи, левой его части на правую.

- Выполняется тот оператор, для которого активная часть выражения получается из левой части оператора некоторой подстановкой значений переменных, их конкретизацией либо унификацией. Полученные при конкретизации (унификации) значения переменных используются для конкретизации правой части (а при унификации — и всего поля памяти). Активная часть выражения заменяется на правую часть правила.

При сентенциальном стиле образ мышления программиста подобен выводу предложения по грамматике (отсюда и наименование стиля). Этот стиль удобен, если перерабатываемые данные естественно представлять в виде сложной структуры единиц, которые достаточно глобально распознаются и достаточно регулярно, но глобально, меняются. В отличие от структурного программирования, когда структурируются и локализуются прежде всего действия, здесь делается упор на локальности активной (преобразуемой) части данных, а действия глобальны.

### Программирование от событий

Есть довольно обширный круг задач, которые естественно описывать как совокупность реакций на события, возникающие в среде выполнения программы. Вообще говоря, так можно трактовать любую программу, обрабатывающую данные: поступление очередного данного — это внешнее событие, требующее реакции, которая, как минимум, должна быть связана с вводом этого данного. Понятно, что такая трактовка далеко не всегда продуктивна. Но она оправдана, например, когда есть много событий, порядок которых не определяет логику обработки, когда реакция на каждое событие автономна, т. е. не зависит от реакции на другие события.

Общая характеристика подобных ситуаций сводится к трем условиям, которые можно считать определяющими для целесообразного применения стиля программирования от событий, или *событийно-ориентированного стиля программирования*:

- Процессы генерации событий отделены от процессов их обработки;
- Процессы отработки разных реакций не зависят друг от друга;
- Можно определить единый механизм установления контакта между событием и реакцией на него, никак не связанный с обработкой.

Стиль событийного программирования — это создание для каждого события собственной процедуры-обработчика. Порядок, в котором обработчики описываются в программе, не имеет никакого значения. Более того, они могут, а во многих случаях и должны быть приписаны к разным структурным единицам программы, в рамках которых только и осмыслена реакция на событие. Как следствие, продуктивно разбивать реакцию на событие на части, за которые отвечают такие структурные единицы, и иметь несколько реакций (разных) структурных единиц на одно событие.

До сих пор мы ничего не говорили о том, какие события возможны при программировании в событийно-ориентированном стиле. Исторически этот стиль как определенный “художественный прием” сформировался в области разработки операционных систем, где естественно связывать понятие события с прерываниями. *Прерывание* — это сигнал от одного из устройств (может быть, и от самого процессора), который говорит о том, что произошло нечто, на что нужно обратить внимание. Когда происходит прерывание, операционная система распознает, какая причина его вызвала, и далее формирует событие как информационный объект, вызывающий реакцию программной системы. Возможны разные способы реагирования на события, в том числе и

передача его для обработки той программе, при выполнении которой возникло прерывание, породившее это событие. Из потребности такой обработки, собственно говоря, и сформировался событийно-ориентированный стиль программирования.

### Программирование от приоритетов

При программировании в стиле событий может возникнуть потребность упорядочивания выполнения нескольких конкурирующих между собой реакций на события. Достаточно универсальным средством удовлетворения этой потребности является приписывание реакциям приоритетов: сначала выполняются те реакции, которые имеют больший приоритет.

Этот прием, как выяснилось на практике, пригоден для решения достаточно широкого круга задач. Он стал основой для формирования самостоятельного стиля **программирования от приоритетов**.

В программировании от приоритетов, как и в сценарийном программировании, порядок расположения операторов в программе имеет малое значение, зато имеет значение его приоритет, т. е. некоторое значение, принадлежащее в самом общем случае частично-упорядоченному множеству и почти всегда рассматриваемое как элементарное. После завершения очередного оператора среди оставшихся выбирается оператор с максимальным приоритетом. Если таких операторов с равными или несравнимыми приоритетами несколько, то, вообще говоря, в идеале надо было бы выбирать один из них недетерминированно.

В целом программирование от приоритетов является мощным, но специфическим орудием для описания глобальных совместных, параллельных или распределенных процессов. Его элементы проникают и в традиционные системы программирования в виде обработки исключительных ситуаций в программах.

### Функциональное программирование

При функциональном программировании основным понятием является *функция*. Мы интересуемся определением и переопределением функции как целого. Соответственно, тогда сами функции становятся значениями.

Функциональное программирование базируется на исчислении, на теории рекурсивных схем и на денотационной семантике, дающей модели этих понятий.

Надо сказать, что при работе с функционалами высших типов возникают большие сложности, и для понимания их теории нужно хорошо знать логику, современную алгебру и топологию. Специалистов, владеющих этой теорией, и практикой программирования, практически нет.

Наиболее богатый опыт и развитые традиции функционального программирования имеет язык LISP. Методы составления LISP-программ вполне сложились, и можно подвести некоторые итоги. Стиль программирования на LISP'е не удалось погубить даже внедрением совершенно чуждого функциональности присваивания. В то же время, этот стиль оказался гибок по отношению к освоению новых концепций, когда они не противоречат базовым средствам языка: и абстрактные типы данных, и объектная ориентированность вполне успешно прививаются на стройное дерево списочной структуры LISP'a.

Как показывает опыт LISP'a, взаимопроникновение стилей возможно, и особенно успешно в случае концептуально хорошо проработанной базы. Но далеко не всегда оно приводит к ожидаемому

росту выразительности. Многочисленные примеры демонстрируют обратное. И всегда неудачи заимствований обусловлены нечеткостью проработки базовых концепций, отходом от того, что логически и теоретически предопределено для данного стиля. Вот почему крайне важно проанализировать существующие стили и выявить их сущность, а не поверхностные сходства и различия. Опыт показал, что владение функциональным стилем программирования является элементом фундаментального образования программиста и мощным средством проектирования программ на концептуальном уровне. Тот, кто осмыслил концепции функционального программирования, гораздо глубже овладевает современными высокоуровневыми методами объектно-ориентированного проектирования и дизайна и гораздо эффективнее их применяет.

Выбирая функциональное программирование, мы почти полностью отказываемся от UML ввиду его объектной природы. Так, диаграммы классов, объектов и последовательности практически теряют смысл. В Haskell нет классов или объектов, нет инкапсулированной мутабельности, — а есть только процесс преобразования данных. Диаграмму последовательности, необходимую для описания взаимодействий этих самых объектов, можно было бы как-то использовать для цепочек функций, но штука в том, что сами цепочки будут более читабельны и понятны. Все прочие диаграммы, тем не менее, вполне применимы. В большой программе на ФП также имеются подсистемы или компоненты, а значит, в строю остаются диаграммы компонентов, пакетов и коммуникации. Диаграмма состояний — универсальна: процессы и конечные автоматы встречаются очень часто. Ее можно было бы применять даже в иных областях, не только в разработке ПО. Наконец, диаграмма вариантов использования вообще имеет мало отношения к дизайну ПО; она связывает бизнес-требования с системными требованиями на этапе анализа.

Но если внимательно присмотреться к «применимым» диаграммам, можно прийти к выводу, что они слабо помогут в высокоуровневом дизайне кода (который располагается ниже дизайна архитектуры), а то и вовсе могут навредить, подталкивая к императивному мышлению.

Во многих источниках делается вывод, что функциональные языки очень декларативные и можно сказать, что программы, на них написанные, документируют сами себя.

*Главным недостатком функциональных языков программирования считается плохая совместимость с самыми популярными из императивных языков программирования, на которых сейчас написано подавляющее большинство широко используемого программного обеспечения. Также не стоит забывать про плохую переносимость программ на функциональных языках на различные платформы и низкую популярность этих языков (то есть, получается рекурсия - языки не популярны из-за того, что все опасаются серьёзно их использовать из-за их низкой популярности). Есть ещё ряд причин, но все они также, в большинстве своём, упираются в инерционность индустрии программного обеспечения, в том числе и в инерционность людей, которые работают в ней.*

"Функциональный программист смотрится как средневековый монах, отвергающий удовольствия жизни в надежде, что это сделает его добродетельным. Для тех, кто заинтересован в материальных выгодах, эти "преимущества" не очень убедительны. (...) Ясно, что такая характеристика функционального программирования неадекватна". Джон Хьюз

## Объектно-ориентированный подход

В современном программировании *объектно-ориентированный подход (ООП)* является одним из популярнейших. Он превратился в стандарт во многих фирмах и во многих сообществах программистов. Этот стиль вместе с соответствующими организационными нововведениями позволил резко повысить эффективность коллективной работы программистов.

Сейчас можно сказать, что есть мода на ООП, так как по статистике это самый популярный стиль программирования. Но также как и моде на одежду стоит руководствоваться не только какими-то тенденциями, но и задумываться о том, какой стиль в конкретной ситуации больше подходит. К примеру, если в моде открытые платья, то вряд ли лучшей идеей будет одеться по этой моде в холод -20 градусов. Также и ООП подходит отнюдь не всегда.

ООП базируется на интенсивном использовании сложных структур данных: объектов. Объекты соединяют внутри себя данные и методы. Чаще всего объекты организуются таким образом, что к данным прямого доступа извне нет, мы можем обращаться к ним лишь через методы. Это позволяет быстро заменять внутреннее представление данных, что исключительно важно для обеспечения перестраиваемости сложных программных систем.

Более того, для замены внутреннего представления данных в объектно-ориентированном программировании имеется механизм наследования. Можно описать новый тип объектов, базирующийся на уже имеющемся, и переопределяющий его методы. Как следствие, при присваивании переменной-объекту могут замениться не только значения данных, но и обрабатывающие их функции.

Объектный подход является прежде всего структурой достаточно высокоуровневых понятий, которые надстраиваются над базисом программных конструкций какого-то стиля первого уровня. В частности, объектная программа может с равным успехом базироваться и на структурном программировании, и на программировании от состояний, и на программировании от приоритетов.

Очень важной стадией развития ООП является *объектно-ориентированное проектирование (дизайн) (ООД)*. Здесь система описывается с нескольких сторон, сначала со стороны пользователя (диаграммы использования), потом со стороны данных (диаграммы классов) и с других сторон реализации (диаграммы состояний и т. п.) Поддерживать целостность системы описаний и переходить от диаграмм к прототипам программ позволяет система *UML* (Unified Modelling Language). Это уже не язык программирования, а язык формализованных спецификаций программ.

## Три технологических стиля программирования

Настоящий параграф посвящен несколько иным стилям, чем представленные выше, зависящим не столько от языков и от задач, сколько от программного и технологического окружения программиста. Эти стили рассчитаны не на составление единственной программы, а на серийное их производство (в том или ином смысле).

В одном случае в качестве серии рассматривается набор программ, в которых повторяется применение одних и тех же фрагментов. Это стиль программирования от переиспользования.

В другом случае серия — это набор различных программ, которые в принципе могут быть построены автоматически (что в реальности означает либо полуавтоматически, либо вручную, как

правило, по жестко фиксированному методу) из одной общей программы с учетом особенностей применения. Это стиль специализирующего программирования.

Наконец, третий случай, для которого серийность вырождается. Она подменяется парой, один компонент которой — макет, образец, демонстрационная разработка или же что-то отличное от программы, выполняющей решение задачи, но способное показать некоторые ее особенности либо в другом смысле приблизить разработку к цели. Это нечто называется образцом, макетом, шаблоном либо прототипом, по которому может строиться программа. Второй компонент пары — производственный вариант программы, изготавливаемый на основе информации о макетном образце, о его разработке или путем технологической процедуры работы с первым компонентом пары (например, это могут быть правила заполнения шаблона). Это стиль программирования от образца.

## Clean code

### Введение

Робин Мартин (автор книги «Clean code» и не только) утверждает, что всех программистов, которые добиваются успеха в мире разработки ПО, отличает один общий признак: они больше всего заботятся о качестве создаваемого программного обеспечения. Это – основа для них. Потому что они являются профессионалами своего дела. То есть написание чистого кода является одним из важных аспектов программирования.

Некоторые принципы написания чистого кода, безусловно, есть у всех языков программирования, однако часто эти принципы выделяют для каждого языка в отдельности. Такой свод правил обычно называют “Best practices”, в интернете с легкостью можно найти “Best practices” для любого конкретного языка.

Однако для языков низкого уровня Clean Code не предусмотрен.

### Best practices JavaScript

- Избегайте глобальных переменных. Вместо этого используйте локальные переменные либо замыкания.
- Всегда объявляйте переменные, используйте ключевое слово “var”.
- Стараться делать все объявления как можно выше. Это сделает код чище, все переменные будут объявлены в одном месте, что поможет избежать возможности переопределения переменных.

Это правило распространяется также и на циклы:

```
var i;  
for (i = 0; i < 10; i++) {...}
```

- Инициализировать переменные при объявлении
- Не объявлять переменные как объектные типы, т.е. не использовать ключевые слова “String”, “Boolean”, “Number”.

```
var x = "John";  
var y = new String("John");
```

```
(x === y) // is false because x is a string and y is an object.
```

---

```
var x = new String("John");
var y = new String("John");
(x == y) // is false because you cannot compare objects.
```

- Не использовать “new Object()”  
Use {} instead of new Object()  
Use "" instead of new String()  
Use 0 instead of new Number()  
Use false instead of new Boolean()  
Use [] instead of new Array()  
Use /()/ instead of new RegExp()  
Use function (){} instead of new function()
- Использовать типы для того, для чего они предназначены. В числовых выражениях не следует использовать строки-числа и т.д.
- Использовать сравнение “==”
- Проверять аргументы функции на undefined. Это на случай, если при вызове функции аргумент не был передан.
- Заканчивать конструкцию “switch” с “default”.
- Не использовать функцию eval() – функция для запуска текста в качестве кода.
- Использовать “use strict”.

#### Best Practices C++

- Названия своих типов с большой буквы: MyClass
- Названия функций с маленькой буквы: myMethod
- Константы большими буквами: const double PI = 3.14;
- Названия переменных писать либо camelCase, либо snake\_case
- Private переменные называть, начиная с “m\_” (member)
- Параметры функции называть, начиная с “t\_”
- Не начинать названия с символа “\_”
- Для выходных и входных файлов создавать отдельные папки
- Использовать лишь односторонние комментарии “//”. Это поможет впоследствии избежать конфликтов с многострочными комментариями, добавленными для дебага.
- Не использовать “using namespace” в Header-файле
- Всегда использовать "{}", даже если выполняется лишь одна команда в блоке
- Не писать код в ширину. Если, к примеру, имеет место длинное условие, следует раскидать его на несколько строк.
- Инициализировать переменные класса
- Всегда использовать namespace
- Не мешать табы с пробелами

## Знание Clean Code при устройстве на работу

Вакансии с такими требованиями имеют место. И можно предположить, раз Clean Code – отражение профессионализма программиста, то возможно на серьезные должности его знание является обязательным.

В Беларуси к фирмам с такими требованиями к разработчикам можно отнести:

- Cedon BLR
- Ciklum
- Itransition
- Specific-Group
- DAROO
- ИООО СИБ софтвэр
- Intetics
- Прогрессив Софт
- Иностр. п. ПИКСОЛИО
- Zagreby Ltd.
- ООО МКСмедиа

## Рефакторинг

### Введение

*Рефакторинг* (англ. refactoring) или реорганизация кода — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований. Поскольку каждое преобразование маленькое, программисту легче проследить за его правильностью, и в то же время вся последовательность может привести к существенной перестройке программы и улучшению её согласованности и четкости.

### Цели рефакторинга

Цель рефакторинга — сделать код программы легче для понимания; без этого рефакторинг нельзя считать успешным.

Рефакторинг следует отличать от оптимизации производительности. Как и рефакторинг, оптимизация обычно не изменяет поведение программы, а только ускоряет её работу. Но оптимизация часто затрудняет понимание кода, что противоположно рефакторингу.

С другой стороны, нужно отличать рефакторинг от реинжиниринга, который осуществляется для расширения функциональности программного обеспечения. Как правило, крупные рефакторинги предваряют реинжиниринг.

### Трудности при отсутствии рефакторинга

Первая сложность состоит в том, что новички на проекте не смогут ничего понять в коде, а постоянный коллектив на одном проекте – это слишком идеальная ситуация, в любом случае имеет место текучка кадров.

Также трудности могут возникнуть на этапе введения нового функционала или расширения старого. Все равно придется провести рефакторинг хотя бы тех кусков, которые затронули новые изменения.

Из-за политики «сделать хоть как, не рефакторить впринципе» очень сильно может пострадать производительность программы.

### Причины применения рефакторинга

Рефакторинг нужно применять постоянно при разработке кода. Основными стимулами его проведения являются следующие задачи:

1. Необходимо добавить новую функцию, которая недостаточно укладывается в принятное архитектурное решение;
2. Необходимо исправить ошибку, причины возникновения которой сразу не ясны;
3. Преодоление трудностей в командной разработке, которые обусловлены сложной логикой программы.

### Примеры плохого кода

Во многом при рефакторинге лучше полагаться на интуицию, основанную на опыте. Тем не менее имеются некоторые видимые проблемы в коде (англ. code smells), требующие рефакторинга:

1. Дублирование кода;
2. Длинный метод;
3. Большой класс;
4. Длинный список параметров;
5. «Жадные» функции — это метод, который чрезмерно обращается к данным другого объекта;
6. Избыточные временные переменные;
7. Классы данных;
8. Несгруппированные данные.

### Методы рефакторинга

Наиболее употребимые методы рефакторинга:

- Изменение сигнатуры метода (Change Method Signature)
- Инкапсуляция поля (Encapsulate Field)
- Выделение класса (Extract Class)
- Выделение интерфейса (Extract Interface)
- Выделение локальной переменной (Extract Local Variable)
- Выделение метода (Extract Method)

«Выделение метода» - один из наиболее часто проводимых типов рефакторинга. Находим метод, кажущийся слишком длинным, или код, требующий комментариев, объясняющих его назначение. Тогда преобразуем этот фрагмент кода в отдельный метод.

По ряду причин предпочтительны короткие методы с осмысленными именами. Во-первых, если выделен мелкий метод, повышается вероятность его использования другими методами. Во-

вторых, методы более высокого уровня начинают выглядеть как ряд комментариев. Замена методов тоже упрощается, когда они мелко структурированы.

- Генерализация типа (Generalize Type)
- Встраивание (Inline)
- Введение фабрики (Introduce Factory)
- Введение параметра (Introduce Parameter)
- Подъём метода (Pull Up Method)
- Спуск метода (Push Down Method)
- Переименование метода (Rename Method)
- Перемещение метода (Move Method)
- Замена условного оператора полиморфизмом (Replace Conditional with Polymorphism)

## Пример рефакторинга

### Проблема

У вас есть условный оператор, который, в зависимости от типа или свойств объекта, выполняет различные действия.

```
class Bird {  
    //...  
    double getSpeed() {  
        switch (type) {  
            case EUROPEAN:  
                return getBaseSpeed();  
            case AFRICAN:  
                return getBaseSpeed() - getLoadFactor() * number_of_coconuts;  
            case NORWEGIAN_BLUE:  
                return (isNailed) ? 0 : getBaseSpeed(voltage);  
        }  
        throw new RuntimeException("Should be unreachable");  
    }  
}
```

### Решение

Создайте подклассы, которым соответствуют ветки условного оператора. В них создайте общий метод и переместите в него код из соответствующей ветки условного оператора. Впоследствии

замените условный оператор на вызов этого метода. Таким образом, нужная реализация будет выбираться через полиморфизм в зависимости от класса объекта.

```
abstract class Bird {  
    //...  
    abstract double getSpeed();  
}  
  
class European extends Bird {  
    double getSpeed() {  
        return getBaseSpeed();  
    }  
}  
  
class African extends Bird {  
    double getSpeed() {  
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;  
    }  
}  
  
class NorwegianBlue extends Bird {  
    double getSpeed() {  
        return (isNailed) ? 0 : getBaseSpeed(voltage);  
    }  
}  
  
// Somewhere in client code  
speed = bird.getSpeed();
```

#### Причины рефакторинга

Этот рефакторинг может помочь, если у вас в коде есть условные операторы, которые выполняют различную работу, в зависимости от:

- Класса объекта или интерфейса, который он реализует;
- Значения какого-то из полей объекта;
- Результата вызова одного из методов объекта.

При этом если у вас появится новый тип или свойство объекта, нужно будет искать и добавлять код во все схожие условные операторы. Таким образом, польза от данного рефакторинга

увеличивается, если условных операторов больше одного, и они разбросаны по всем методам объекта.

### Достоинства

- Этот рефакторинг реализует принцип *говори, а не спрашивай*: вместо того, чтобы спрашивать объект о его состоянии, а потом выполнять на основании этого какие-то действия, гораздо проще просто сказать ему, что нужно делать, а как это делать он решит сам.
- Убивает дублирование кода. Вы избавляетеесь от множества почти одинаковых условных операторов.

Если вам потребуется добавить новый вариант выполнения, все, что придётся сделать, это добавить новый подкласс, не трогая существующий код (*принцип открытости/закрытости*).

- Замена наследования делегированием (Replace Inheritance with Delegation)
- Замена кода типа подклассами (Replace Type Code with Subclasses)

### Авторефакторинг в различных IDE

Существует множество различных IDE, и не хватит целой книги, чтобы описать возможности авторефакторинга во всех, так что приведем пример *IntelliJ IDEA* и некоторых ее интересных возможностей.

- Ctrl + Alt + L - Сделать форматирование кода
- Ctrl + Alt + O - Удалить неиспользуемые импорты
- Alt + F7 - Найти использования кода
- Shift + F6 - Переименовать
- Ctrl + F6 - Изменить сигнатуру
- Ctrl + Alt + N - Встроить
- Ctrl + Alt + M - Поместить в метод
- Ctrl + Alt + V - Поместить в переменную
- Ctrl + Alt + F - Поместить в поле
- Ctrl + Alt + C - Поместить в константу
- Ctrl + Alt + P - Поместить в параметр
- И проч.

### Литература по рефакторингу

- Refactoring: Improving the Design of Existing Code («Рефакторинг: улучшение существующего кода»). Martin Fowler, Kent Beck, John Brant и William Opdyke (Мартин Фаулер, Кент Бек, Джон Брант и Вильям Опдейк)
- Чистый код. Создание, анализ и рефакторинг. Робин Мартин.
- Эффективная работа с унаследованным кодом. Майкл К. Физерс.

## Паттерны проектирования

### Что это такое?

**Шаблон проектирования** или **паттерн** (англ. design pattern) — архитектурная конструкция, представляющая собой решение конкретной проблемы в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть преобразован прямо в код. Это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты будут использоваться.

### Классификация

Классифицируют шаблоны всегда по-разному. В зависимости от применения могут разбиваться на группы. Чаще всего разбиение производится по уровням. «Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются **идиомами**. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные. На наивысшем уровне существуют **архитектурные шаблоны**, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

### История

- В 1970-е годы архитектор Кристофер Александр составил набор шаблонов проектирования. В области архитектуры эта идея не получила такого развития, как позже в области программной разработки.
- В 1987 году Кент Бэк (Kent Beck) и Вард Каннингем (Ward Cunningham) взяли идеи Александра и разработали шаблоны применительно к разработке программного обеспечения для разработки графических оболочек на языке Smalltalk.
- В 1988 году Эрих Гамма (Erich Gamma) начал писать докторскую диссертацию при цюрихском университете об общей переносимости этой методики на разработку программ.
- В 1989—1991 годах Джеймс Коплин (James Coplien) трудился над разработкой идиом для программирования на C++ и опубликовал в 1991 году книгу Advanced C++ Idioms.

В этом же году Эрих Гамма заканчивает свою докторскую диссертацию и переезжает в США, где в сотрудничестве с Ричардом Хеллом (Richard Helm), Ральфом Джонсоном (Ralph Johnson) и Джоном Влиссидесом (John Vlissides) публикует книгу **Design Patterns** — Elements of Reusable Object-Oriented Software. В этой книге описаны 23 шаблона проектирования. Также команда авторов этой книги известна общественности под названием «Банда четырёх» (англ. *Gang of Four*, часто сокращается до *GoF*). Именно эта книга стала причиной роста популярности шаблонов проектирования.

### Паттерны проектирования классов/объектов

Описание системы в терминах классов/объектов следует считать низшим уровнем ее представления. В свою очередь, при моделировании системы на уровне классов/объектов обычно проводят дополнительную типологизацию в двух аспектах, а именно, описывают структуру системы в терминах микроскопических элементов и то, каким образом такая система обеспечивает

требуемый функционал. Соответственно, среди паттернов проектирования выделены **структурные паттерны** и **паттерны распределения обязанностей между классами/объектами**. Поскольку отдельные объекты создаются и уничтожаются в процессе работы системы, выделена еще одна большая группа **паттернов проектирования**, которые служат для создания объектов.

**Примеры:** Адаптер(Adapter), Декоратор (Decorator), Мост (Bridge), Стратегия (Strategy), Прототип (Prototype), Абстрактная фабрика (Abstract Factory), Одиночка (Singleton), Строитель (Builder).

#### Архитектурные системные паттерны

Архитектурные системные паттерны объединены в группы в соответствии с теми задачами, для решения которых они разработаны. Для организации классов или объектов системы в базовые подструктуры используются **структурные архитектурные паттерны**. С другой стороны, для обеспечения требуемого системного функционала первостепенное значение имеет адекватная организация взаимодействия отдельных архитектурных элементов системы - этой цели служат **паттерны управления**.

**Примеры:** MVC (modal-view-controller), MVVM (modal-view-viewmodal), Активная запись (Active Record).

#### Паттерны интеграции корпоративных информационных систем

Паттерны интеграции информационных систем представляют собой, как это описано в разделе 2, верхний уровень классификации паттернов проектирования. Аналогично паттернам более низких уровней классификации, среди паттернов интеграции выделена группа **структурных паттернов**. Структурные паттерны описывают основные компоненты единой интегрированной метасистемы. В свою очередь, для описания взаимодействия отдельных корпоративных систем, включенными в интегрированную метасистему, организована группа паттернов, объединенных в соответствии с тем или иным **методом интеграции**. Далее, интеграция корпоративных информационных систем подразумевает тем или иным способом организованный обмен данными между системами. Для организации обмена информацией между отдельными системами, включенными в интегрированную метасистему, служат **паттерны интеграции по типу обмена данными**. Следует отметить, что в отличие от паттернов проектирования классов/объектов и архитектурных системных паттернов, отнесение отдельного паттерна интеграции к тому или иному виду является менее условным.

**Примеры:** Взаимодействие «точка - точка», Файловый обмен, Удаленный вызов процедур.

#### Паттерны параллельного программирования (Concurrency)

Представляют собой примитивы синхронизации потоков, обеспечивают асинхронный и безопасный доступ к ресурсу, наблюдают за состояниями объектов.

**Примеры:** Замок (Lock), активный объект (Active Object), Планировщик (Scheduler).

#### Популярные паттерны

##### *Мост*

<b>Проблема</b>	Требуется отделить абстракцию от реализации так, чтобы и то, и другое можно было изменять независимо. При использовании наследования
-----------------	--

	реализация жестко привязывается к абстракции, что затрудняет независимую модификацию.
<b>Решение</b>	Поместить абстракцию и реализацию в отдельные иерархии классов.
<b>Рекомендации</b>	Можно использовать если, например, реализацию необходимо выполнять во время реализации программы.
<b>Пример</b>	"Абстракция" определяет интерфейс "Абстракции" и хранит ссылку на объект "Реализация", "УточненнаяАбстракция" расширяет интерфейс, определенный "Абстракцией". "Реализация" определяет интерфейс для классов реализаций, он не обязан точно соответствовать интерфейсу класса "Абстракция" - оба интерфейса могут быть совершенно различны. Обычно интерфейс класса "Реализация" предоставляет только примитивные операции, а класс "Абстракция" определяет операции более высокого уровня, базирующиеся на этих примитивных. "КонкретнаяРеализация" содержит конкретную реализацию класса "Реализация". Объект "Абстракция" перенаправляет своему объекту "Реализация" запросы "Клиента".
<b>Преимущества</b>	<p>Отделение реализации от интерфейса, то есть, "Реализацию" "Абстракции" можно конфигурировать во время выполнения. Кроме того, следует упомянуть, что разделение классов "Абстракция" и "Реализация" устраняет зависимости от реализации, устанавливаемые на этапе компиляции: чтобы изменить класс "Реализация" вовсе не обязательно перекомпилировать класс "Абстракция".</p>

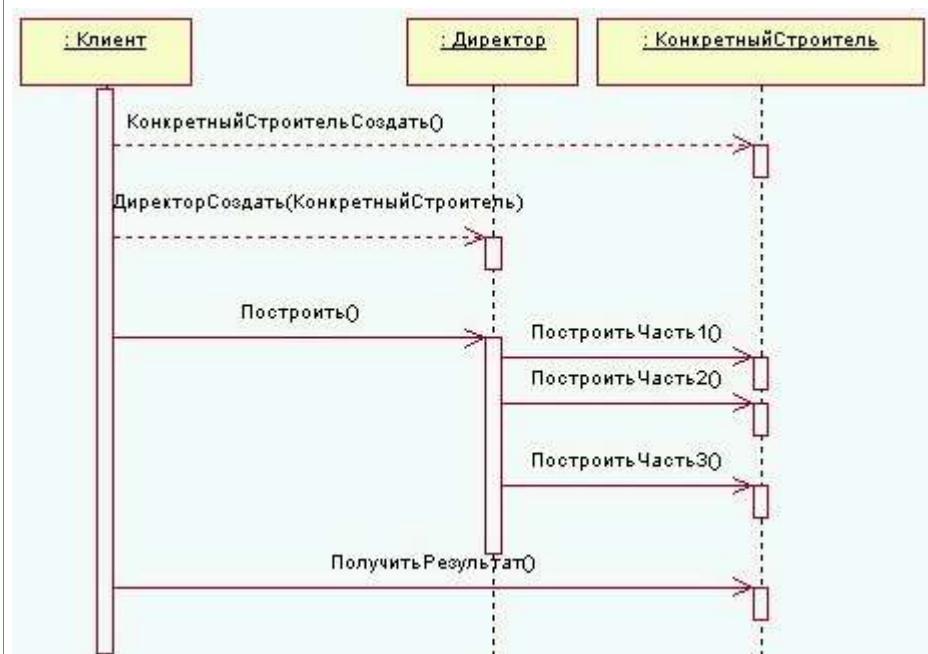
### Строитель

<b>Проблема</b>	Отделить конструирование сложного объекта от его представления, так чтобы в результате одного и того же конструирования могли получаться различные представления. Алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой.
-----------------	--

"Клиент" создает объект - распорядитель "Директор" и конфигурирует его объектом - "Строителем". "Директор" уведомляет "Строителя" о том, что нужно построить очередную часть "Продукта". "Строитель" обрабатывает запросы "Директора" и добавляет новые части к "Продукту", затем "Клиент" забирает "Продукт" у "Строителя".



#### Решение



#### Преимущества

Объект "Строитель" предоставляет объекту "Директор" абстрактный интерфейс для конструирования "Продукта", за которым может скрыть представление и внутреннюю структуру продукта, и, кроме того, процесс сборки "продукта". Для изменения внутреннего представления "Продукта" достаточно определить новый вид "Строителя". Данный паттерн изолирует код, реализующий создание объекта и его представление.

#### [Абстрактная фабрика](#)

#### Проблема

Создать семейство взаимосвязанных или взаимозависимых объектов (не специфицируя их конкретных классов).

<b>Решение</b>	Создать абстрактный класс, в котором объявлен интерфейс для создания конкретных классов.
<b>Пример</b>	Какой класс должен отвечать за создание объектов - адаптеров при использовании паттерна "Адаптер". Если подобные объекты создаются неким объектом уровня предметной области, то будет нарушен принцип разделения обязанностей.
<b>Преимущества</b>	Изолирует конкретные классы. Поскольку "Абстрактная фабрика" инкапсулирует ответственность за создание классов и сам процесс их создания, то она изолирует клиента от деталей реализации классов. Упрощена замена "Абстрактной фабрики", поскольку она используется в приложении только один раз при инстанцировании.
<b>Недостатки</b>	Интерфейс "Абстрактной фабрики" фиксирует набор объектов, которые можно создать. Расширение "Абстрактной фабрики" для изготовления новых объектов часто затруднительно.

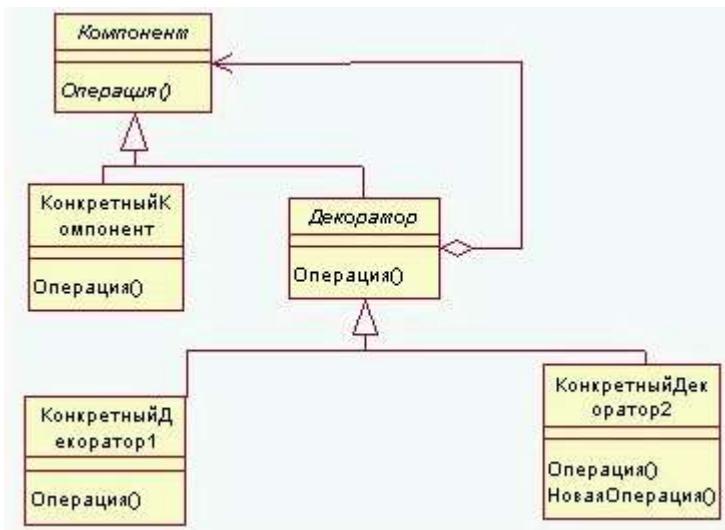
### *Адаптер*

<b>Проблема</b>	Необходимо обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких компонентов с разными интерфейсами.
<b>Решение</b>	Конвертировать исходный интерфейс компонента к другому виду с помощью промежуточного объекта - адаптера, то есть, добавить специальный объект с общим интерфейсом в рамках данного приложения и перенаправить связи от внешних объектов к этому объекту - адаптеру.

### *Декоратор(Обертка)*

<b>Проблема</b>	Возложить дополнительные обязанности (прозрачные для клиентов) на отдельный объект, а не на класс в целом.
<b>Рекомендации</b>	Применение нескольких "Декораторов" к одному "Компоненту" позволяет произвольным образом сочетать обязанности, например, одно свойство можно добавить дважды.
<b>Решение</b>	Динамически добавить объекту новые обязанности, не прибегая при этом к порождению подклассов (наследованию). "Компонент" определяет интерфейс для объектов, на которые могут быть динамически возложены дополнительные обязанности, "Конкретный Компонент" определяет объект, на который возлагаются дополнительные обязанности, "Декоратор" - хранит ссылку на объект "Компонент" и определяет интерфейс, соответствующий интерфейсу "Компонента".

"КонкретныйДекоратор" возлагает дополнительные обязанности на компонент. "Декоратор" переадресует запросы объекту "Компонент".



#### Преимущества

Большая гибкость, чем у статического наследования: можно добавлять и удалять обязанности во время выполнения программы в то время как при использовании наследования надо было бы создавать новый класс для каждой дополнительной обязанности. Данный паттерн позволяет избежать перегруженных методами классов на верхних уровнях иерархии - новые обязанности можно добавлять по мере необходимости.

#### Недостатки

"Декоратор" и его "Компонент" не идентичны, и, кроме того, получается, что система состоит из большого числа мелких объектов, которые похожи друг на друга и различаются только способом взаимосвязи, а не классом и не значениями своих внутренних переменных - такая система сложна в изучении и отладке.

### *Синглтон*

<b>Проблема</b>	Какой специальный класс должен создавать "Абстрактную фабрику", и как получить к ней доступ? Необходим лишь один экземпляр специального класса, различные объекты должны обращаться к этому экземпляру через единственную точку доступа.
<b>Решение</b>	Создать класс и определить статический метод класса, возвращающий этот единственный объект.
<b>Рекомендации</b>	Разумнее создавать именно статический экземпляр специального класса, а не объявить требуемые методы статическими, поскольку при использовании методов экземпляра можно применить механизм наследования и создавать подклассы. Статические методы в языках

программирования не полиморфны и не допускают перекрытия в производных классах.

Решение на основе создания экземпляра является более гибким, поскольку впоследствии может потребоваться уже не единственный экземпляр объекта, а несколько.

## Достоинства

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению свое имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

## Недостатки

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла (Стивен Макконнелл — американский программист, автор книг по разработке программного обеспечения. Журнал «Software Development» дважды удостоил его книги премии Jolt Excellence как лучшие книги года о разработке программного обеспечения), с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Многие шаблоны в ООП можно рассматривать как идиоматическое воспроизведение элементов функциональных языков. Пол Грэхэм (Пол Грэм — американский предприниматель, программист, известный сторонник и пропагандист использования языка программирования Lisp. В числе прочего, создал диалект Lisp, названный им Arc. Автор ряда книг по программированию) считает саму идею шаблонов проектирования — анти-паттерном, сигналом о том, что система не обладает достаточным уровнем абстракции, и необходима её тщательная переработка. Нетрудно видеть, что само определение шаблона как «готового решения, но не прямого обращения к библиотеке» по сути означает отказ от повторного использования в пользу дублирования. Это, очевидно, может быть неизбежным для сложных систем при использовании языков, не поддерживающих комбинаторы и полиморфизм типов, и это в принципе может быть исключено в языках, обладающих этими свойствами (хотя и не обязательно эффективно), так как любой шаблон может быть реализован в виде исполнимого кода.

## Анти-паттерны

Это классы наиболее часто внедряемых плохих решений проблем. Они изучаются, как категория, в случае, когда их хотят избежать в будущем, и некоторые отдельные случаи их могут быть распознаны при изучении неработающих систем.

Термин происходит от паттернов.

- Дым и зеркала (Smoke and mirrors): Демонстрация того, как будут выглядеть ненаписанные функции (название происходит от двух излюбленных способов, которыми мифокусники скрывают свои секреты)
- Раздувание ПО (Software bloat): Разрешение последующим версиям системы требовать всё больше и больше ресурсов
- Функции для галочки: Превращение программы в конгломерат плохо реализованных и не связанных между собой функций (как правило, для того, чтобы заявить в рекламе, что функция есть)
- Полтергейст (Poltergeist): Объекты, чьё единственное предназначение — передавать информацию другим объектам
- Слепая вера (Blind faith): Недостаточная проверка корректности исправления ошибки или результата работы подпрограммы
- Воняющий подгузник (The Diaper Pattern Stinks): Сброс флага ошибки без её обработки или передачи вышестоящему обработчику
- Мыльный пузырь (Soap bubble): Объект, инициализированный мусором, максимально долго притворяется, что содержит какие-то данные
- Бензиновая фабрика (Gas factory): Необязательная сложность дизайна
- Золушкина туфелька: Попытка "натянуть" на объект уже имеющийся малоподходящий по смыслу интерфейс, вместо создания нового.
- Коммит-убийца (Commit assassin): Внесение отдельных изменений в систему контроля версий без проверки влияния их на другие части системы. Как правило, после подобных коммитов работа коллектива парализуется на время исправления проблем в местах, которые ранее работали безошибочно.

## Разработка мобильных приложений

### Типы мобильных приложений

Познакомимся с видами приложений, которые подразделяются по определенным критериям. Очень часто при разработке приложения, программисты и заказчики формируют целый перечень пунктов в Договоре и техническом задании. Вид устройства и виды ОС, а также девайсов, с которыми будет взаимодействовать мобильное приложение является не менее важным условием, нежели прототипирование или разработка функционала.

На практике можно разделить приложения для мобильных устройств на три типа.

#### *Мобильные сайты, веб-приложения*

Это самый распространенный тип приложений для мобильных устройств. Современные смартфоны в состоянии отобразить обычный сайт. Им доступно все то, что мы привыкли видеть в десктопных приложениях — поддержка HTML5 делает свое дело. Веб-приложения отлично подходят для стартапа: именно они позволяют получить большой результат за маленькие деньги и за небольшой срок. Еще один плюс мобильного сайта по сравнению с другими мобильными приложениями — это кроссплатформенность. Однако есть и минус, притом весомый: с ними достаточно сложно заработать.

### *Гибридные приложения*

При таком подходе вы получаете доступ ко всем плюсам API операционной системы: приложение обращает push-уведомлениями и другими приятными бонусами, кроме того, теперь ваш продукт можно размещать в магазинах. При этом основной контент все еще представляет собой платформонезависимую страничку с версткой, размещенную на сервере. Это позволяет вносить косметические изменения в продукт без выпуска новой версии: достаточно добавить изменения на сервер. Гибридные приложения – отличное решение для тех, кто начинает бизнес или хочет проверить свою идею, показать ее инвестору, друзьям.

### *Нативные приложения*

Этот вид приложений самый ресурсоемкий, но вместе с этим он позволяет по максимуму использовать возможности, предлагаемые каждой конкретной операционной системой. Как следствие, нативные приложения выигрывают как по функционалу, так и по скорости работы у других типов мобильных приложений. Именно к такому подходу сейчас приходят те компании, которые делали комбинированные приложения. Например, Facebook начинала с комбинированного приложения: нативные контролы (переключатели, вкладки и так далее) и веб-страница в качестве контента. Несмотря на то, что это неплохое решение, проблемы с производительностью приводят к тому, что разработчики отходят от комбинации с вебом. Именно такие мобильные приложения используются банками для входа в личный кабинет пользователя и проведения операций. Здесь работает множество структур, и система безопасности по праву занимает первое место.

### Процесс создания мобильного приложения

#### *Анализ бизнес-модели и изучение конкурентов*

От успешности и целесообразности идеи зависит его дальнейший рост и перспектива. Если Вы решили разрабатывать мобильную игрушку, это не лучший выбор, так как конкуренция в игровой категории баснословная. Что касается анализа конкурентов, существуют отличные сервисы, позволяющие определить наиболее активных игроков рынка по соответствующим критериям. Есть, как платные зарубежные сервисы, так и бесплатные российские. Например, Appintop, модуль CPIera Spy, позволяющий при регистрации и предоставления аккаунта получить весьма интересную статистику.

Необходимо исследовать наиболее конкурентную нишу, в которой ежедневно идет борьба за ТОП в маркете, целевую аудиторию и ее платежеспособность,

понять, какими особенностями обладает Ваш продукт и что нового Вы можете дать пользователям.

#### *User Story*

На этом этапе важно проработать все возможные сценарии, чтобы не было неприятных сюрпризов на более поздних этапах разработки.

Важно понимать, что за каждым пунктом в вашем to-do листе скрывается огромный айсберг функционала. Страйтесь фрагментировать и конкретизировать задачи.

### *Проектирование и дизайн*

После составления User Story начинается проектирование и разработка дизайна. На этом этапе используются прототипы, которые можно вешать на доску и стрелочками показывать, как будет происходить навигация.

При разработке дизайна обязательно используются гайдлайны.

Гайдлайн в общем понимании – это документ, который выпускает компания, и по которому дизайнеры и разработчики понимают принцип построения взаимодействия приложения с пользователем. Условно говоря, для iOS кнопки надо делать круглыми, а для Windows Phone – квадратными. Однако используются и внутренние гайдлайны для разработчиков. Таким образом результат работы дизайнера чаще всего состоит из макетов, гайдлайнов и нарезки графики. Макеты лучше всего подавать «перелинкованными», например с помощью ProtoType, чтобы была понятна логика переходов. Гайдлайны содержат в себе информацию об отступах, размерах, визуальных эффектах, механике анимации и пр. Этот этап можно пропустить, если в вашем проекте один дизайнер и один разработчик, сидящие рядом друг с другом. Третья часть результата — нарезка графики — должна содержать минимум необходимых графических ресурсов (заботимся о весе приложения), иметь версии для разных разрешений экранов.

### *Передача в разработку. Обсуждение и необходимые правки описания*

После получения макетов, гайдлайна и нарезки, начинается работа разработчика. Передается в разработку все то, что придумали, и ожидается ранний результат. Это не значит, что работа над архитектурой и пользовательским интерфейсом закончена. Иногда у разработчиков появляются интересные идеи, которые вносят корректиды в изначальный план. Когда разработка завершена, наступает стадия тестирования.

### *Тестирование*

Существует немалое количество способов протестировать приложение. В мобильной разработке тестировщик – это человек, вокруг которого одни телефоны. Почему не эмуляторы? Зачастую эмуляторы очень требовательны к ресурсам, так как наиболее качественные из них эмулируют работу приложения с самых низких уровней. То, что приложение работает на эмуляторе, не значит практически ничего, ведь пользователи будут запускать приложения на реальных мобильных телефонах, которые всегда отличаются даже от самых лучших эмуляторов. Тестирование на целевом мобильном телефоне – это самый верный способ убедиться в правильном функционировании приложения, поскольку вы выполняете приложение на том же аппаратном обеспечении, которое будет у ваших пользователей.

Чаще всего тестирование производится по тест-кейсам - если внедряется новая функция, по ее описанию составляется тест-план.

Существуют сервисы, помогающие в тестировании. Например, HockeyApp – приложение, позволяющее раздавать продукт бета-тестерам. Можно написать в социальных сетях: «Ребята, у нас новое крутое приложение. Кто хочет попробовать?» Желающие получают билд, пользуются приложением, а сервис собирает статистику, составляет креш-репорт и отправляет все это Вам. Также есть сервисы, позволяющие протестировать приложение на разных операционных системах – например, все Android-прошивки версии 2.1 или 2.3. Вы отдаете приложение, сервис скриншотит весь путь, который вы задали, присыпает картинки вам на почту, и вы проверяете, все ли в порядке.

## *Мониторинг*

Итак, вы разработали, протестирували приложение, добавили его в магазин. Для отслеживания статистики скачиваний можно использовать сервис Distimo. Он показывает статистику по пользователям, которые приходят в магазин, чтобы скачать приложения, и агрегирует комментарии.

Важно понимать, что люди более склонны оставлять негативные комментарии. Если у человека все хорошо, он чаще всего просто пользуется приложением, не комментируя. Поэтому имейте в виду, что комментарии — это не полная оценка вашей работы, скорее еще один баг-трекер. Изменить ситуацию может довольно распространенных «хак» — окно Rate Us. С предложением оставить положительный комментарий в store, а в случае проблем написать разработчику. Эффект достаточно сильный, главное — правильно продумать алгоритм показывания диалога пользователю.

Помимо комментариев Distimo показывает количество скачиваний, заработанные деньги, а также откуда скачивают ваши приложения.

Еще один интересный мониторинговый сервис — Flurry. Он помогает собирать клиентскую статистику. Flurry предоставляет отчет о том, что делает пользователь в вашем приложении: сколько раз он нажал на кнопку, сколько раз возвращался в приложение и более общие параметры — аудитория, география, пол, возраст и пр.

Несмотря на большое количество сторонних сервисов, полезно иметь собственную статистику. Какими бы хорошими не были внешние источники, их нужно проверять.

## *Что еще нужно помнить?*

- На каждой новой платформе пользователь ожидает увидеть богатое приложение. Он рассуждает следующим образом: «Я сидел на плохом Java-телефоне, при этом мог пользоваться аськой. Я купил новый телефон, захожу в Marketplace, а там нет ICQ? Вы чем там занимаетесь?» Пользователь не принимает в расчет того, что вам приходится делать приложение с нуля. Новая платформа — это новые девайсы, новая документация, новые ресурсы.
- Чем популярнее платформа, тем больше у вас конкурентов. На данный момент существует два store, на примере которых это отлично видно: AppStore и Google Play. Если у вас есть идея приложения, которое легко монетизируется или просто получит много скачиваний, вбейте ключевые слова в поиске и скорее всего вы обнаружите, что такое приложение существует. Чем популярнее платформа, тем больше конкурентов. В таких случаях надо тщательно изучать аналоги, смотреть статистику, пытаться понять, по каким параметрам существующие решения можно превзойти.
- Важно понимать, как пользователи выбирают приложения. Изначально человек не собирается покупать конкретное приложение, он просто смотрит список. Например, по запросу «бесплатная музыка». Иконка и первые две строчки описания — это то, что человек видит и оценивает в первую очередь. Если иконка приличная, можно покупать; соответственно, плохая иконка уменьшает количество скачиваний.
- Очень важно попасть в топ магазина приложений. Попасть в топ store — очень хорошо, закрепиться там — залог успеха. Зачем это нужно? Когда пользователь хочет что-нибудь скачать, он заходит в чарт и видит, какое приложение сейчас на первом месте по числу скачиваний. Очень важно туда попасть, потому что это своего рода замкнутый круг.

Приложение попадает в чарт, его видят пользователи, они его скачивают, оно снова попадает на первое место, и дальше итерации продолжаются. Поэтому всеми силами добывайте скачки и рейтинг: просите мам, бабушек, соседей ставить приложению пятерки. Стоит сказать, что на рынке полно решений для гарантированного вывода программы в топ. Однако органических пользователей это приносит мало, что не мешает продолжать эксперименты.

- Помните, что время публикации может доходить до нескольких недель. Допустим, вы разработали и протестирували приложение, обзвонили всех блоггеров или СМИ и сказали: «У меня новое приложение, приходите на пресс-конференцию». Вы его отправляете в AppStore и вынуждены ждать семь рабочих дней в России. В этом случае Google Play – рай для оперативных обновлений, где публикация занимает несколько часов.
- Фрагментация операционной системы. Если вы разрабатываете приложения под Android, учитывайте фрагментацию и существование целого зоопарка устройств. Это сказывается на времени разработки на всех этапах: проектирование, дизайн, разработка и особенно тестирование.
- В store невозможно общаться с пользователями. Например, пользователь пишет, что у него в ICQ не ходят сообщения. Он недоволен и выражается очень красочно, но не очень информативно. Нет возможности с ним связаться и узнать о проблеме подробнее. Все, что можно сделать – залезть в мониторинг.
- Сейчас некоторые store обзаводятся админкой, в которой можно задать вопросы пользователю, выяснить его контактные данные, посмотреть, в каких странах скачивают ваше приложение, но пока ситуация не слишком улучшилась. Стоит отметить, что Google Play продолжает добавлять эту возможность некоторым разработчикам.

### Платформы разработки

Каждая из платформ для мобильных приложений имеет интегрированную среду разработки, предоставляющую инструменты, позволяющие разработчику программировать, тестировать и внедрять приложения на целевую платформу. Их достаточно много, языки программирования используются тоже разные. То есть если вы очень хотите написать свое приложение под iOS, необязательно в срочном порядке учить Objective C. Пример — webMethods Mobile Designer подходит для разработки под Java ME, Android, BREW, BlackBerry, Nintendo DS, iOS (iPhone/iPad), Palm/webOS, Sony PSP, Samsung bada, Symbian, Windows Mobile, Windows Phone 7, Windows Desktop, OSX. Язык программирования — Java.

Основные платформы — Android ([Java](#), частично [C](#), [C++](#), [Delphi](#)), IOS SDK (Objective-C, Object Pascal, C++), Windows Phone (C#).

Что касается популярных движков для разработки мобильных приложений, согласно исследованию, проведенному журналом Game Developer, Unity является самым популярным движком среди мобильных разработчиков. Более половины опрошенных журналом инсайдеров заявили, что используют для разработки именно этот пакет. Если быть точнее, то на Unity «сидят» 53,1% опрошенных британским журналом разработчиков. На втором месте по популярности идет с 17,7% Cocos2D, что касается таких движков как Marmalade и Corona, то они вдвоем делят третье место: с каждым из них работают порядка 5,3% опрошенных разработчиков.

Так же существуют платформы для самостоятельного конструирования мобильных приложений, не обращаясь к услугам IT-специалистов.

Например, BuildAnApp - конструктор для самостоятельной генерации приложений BlackBerry, Windows, iOS и Android. Сборка осуществляется в шесть шагов, на выходе получаем нативное приложение или веб-приложение в зависимости от пожеланий клиента.

### [И что дальше?](#)

Давайте немного поговорим о деньгах. Стоимость регистрации в Google Play — 25\$, в App Store — 99\$, в MarcketPlace если Вы регистрируетесь как простой разработчик (Individual), то будете платить абонентскую плату 99\$ в год, если же Вы являетесь студентом, то выбирайте Student, и Вам не надо будет платить этот взнос, однако нужно будет еще подтвердить то, что Вы являетесь студентом очного дневного отделения.

Комиссия Apple, Google play — 30%. До тех пор, пока выручка с приложения не достигнет \$25000, Microsoft будет брать с нее 30 процентов. По достижению \$25000 отметки, доля Microsoft будет снижена до 20 процентов.

Как видите, с такими тратами стратегия зарабатывания на мобильных приложениях просто необходима. Вот самые распространенные из них.

### [Платное скачивание](#)

Эта стратегия предполагает единовременную оплату пользователем всей стоимости приложения. Такой подход отлично работает с играми, развлекательными, навигационными и новостными приложениями. Однако, чем выше будет стоимость приложения, тем меньше пользователей будет устраивать реклама или необходимость дополнительной покупки определенных функций в приложении. Вы можете также предложить как бесплатную, так и платную версии приложения с предоставлением дополнительного контента и сведением к минимуму или полным исключением какой-либо рекламы.

### [Реклама внутри приложения](#)

Это один из наиболее популярных методов монетизации приложений, функционирующий за счет отображения рекламы на предварительно выделенном пространстве в интерфейсе и генерирующий прибыль в расчете за определенное количество просмотров или/и переходов. Этот метод по-настоящему эффективно работает в играх, новостных и развлекательных приложениях, а также в мессенджерах. Однако, основная трудность заключается в том, чтобы создать приложение, которое будут использовать постоянно.

### [Встроенные покупки или подписки](#)

Покупки внутри приложений позволяют пользователю приобретать дополнительные функции или бонусы. Эта стратегия также отлично работает в играх и новостных приложениях. Однако, если вы используете такой метод монетизации, вам необходимо обзавестись действительно преданными пользователями вашего продукта, которые будут не против платы за дополнительные функции.

### [Спонсорство](#)

Суть спонсорства заключается в запуске приложения от имени другого издателя в обмен на популярность и признание. Кроме того, в приложении может быть размещен логотип известной компании. Этот вариант подходит приложениям, которые привязаны к определенному местоположению или событию. Однако, главный недостаток этого метода заключается в том, что это одноразовый источник дохода.

### *Push-уведомления*

Рекламные объявления отображаются в виде оповещений на устройстве пользователя. CTR и показатели конверсии в данном случае очень высоки, так как этот метод не предполагает случайных нажатий, которые часто встречаются при использовании внутренней рекламы. Рекламодатели платят за клики. В отличие от традиционных мобильных объявлений, эти уведомления размещаются не внутри приложения. Каждое объявление обозначается тегом и ссылкой для отклонения предложения.

### *Иконки*

Иконки представляют собой ярлыки (Иконки), которые помещаются на экран «Домой» вместе с устанавливаемым приложением, подобно тому, как приложения предварительно устанавливаются на новые смартфоны. Рекламодатели платят за каждую установленную иконку.

### *Реклама во всплывающих окнах*

Подобные рекламные объявления отображаются в виде диалогового окна внутри приложения. Они обычно содержат некий призыв к действию и кнопки OK и Отмена. К примеру, такое объявление может предложить пользователю «Скачать новую бесплатную 3D-игру» с возможностью выбора соответствующей кнопки. Здесь рекламодатели платят на основе CPC-основе.

### *Офферы*

Это рекламные объявления для отдельных рекламных предложений, которые размещаются в промежуточной вставке внутри приложения, когда пользователь может решить, будет ли он участвовать в программе или нет. В этом случае рекламодатели платят на основе CPC и CPA.

### *Видео-заставки*

Это видеоролики, которые можно просмотреть внутри приложения. Чаще всего такие объявления демонстрируются только тем пользователям, которые использует wi-fi или 4g-подключение к интернету, так как это повышает вероятность того, что пользователь сможет просмотреть ролик без остановки от начала и до конца. Рекламодатели платят на базе CRM.

### *Rich Media*

Это полноценная целевая страница, встроенная в приложение. Объявление отображается во внутренней рекламной вставке, с фиксированной кнопкой закрытия в самом верху. Рекламодатели оплачивают такую услугу на базе CPC.

### *Без нарушений!*

Будьте аккуратны при выкладывании своих приложений в store-ы, обязательно прочитайте условия на их официальных сайтах. Помните, ваше приложение могут не просто удалить, могут заблокировать весь аккаунт. Вот некоторые правила:

- Если у вас есть похожие приложения (похожие на то, которое заблокировали) — немедленно их удалите, даже если они стоили вам большого труда или денег. Вы спасёте ваш бесценный аккаунт, потеря которого неисправима.
- Используйте только тот контент, авторским правом на который вы обладаете. Не используйте чужие картинки, видео или звуки, даже если ваше приложение будет способствовать развитию автора этого контента.
- Будьте особенно внимательны к заголовку вашего приложения. От греха подальше — не добавляйте туда названия компаний или каких-либо популярных продуктов.

- Аккуратнее с тегами.
- Для своих личных проектов — найдите альтернативу Google play. Их предостаточно. Там вас вряд ли заблокируют. И на худой конец — никто не мешает вам разместить APK на вашем личном сайте или блоге.
- Будьте краткими! Если у вас в описании приложения (или в описании обновления) будет слишком много слов в принципе (хоть и не ключевых) — вас могут заблокировать.
- 6% нарушений в App Store касаются п 10.6 правил — «Apple и наши клиенты высоко ценят простой, изысканный, творческий, хорошо продуманный интерфейс. Такой проект требует больше усилий, но он этого стоит. Apple устанавливает высокую планку. Если пользовательский интерфейс окажется сложным и недостаточно хорошим, это может привести к отклонению заявки». Думаю, объяснять не надо.

## Тестирование

**Тестирование программного обеспечения** — процесс исследования, испытания программного продукта, имеющий две различные цели:

- продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
- выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации

## Виды тестирования

Все **виды тестирования программного обеспечения**, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

- Функциональные
- Нефункциональные
- Связанные с изменениями

### *Функциональные виды тестирования*

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены одни из самых распространенных видов функциональных тестов:

- Функциональное тестирование (Functional testing)
- Тестирование безопасности (Security and Access Control Testing)
- Тестирование взаимодействия (Interoperability Testing)

### *Нефункциональные виды тестирования*

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как" система работает. Далее перечислены основные виды нефункциональных тестов:

- Все виды тестирования производительности:

- нагрузочное тестирование (Performance and Load Testing)
- стрессовое тестирование (Stress Testing)
- тестирование стабильности или надежности (Stability / Reliability Testing)
- объемное тестирование (Volume Testing)
- Тестирование установки (Installation testing)
- Тестирование удобства пользования (Usability Testing)
- Тестирование на отказ и восстановление (Failover and Recovery Testing)
- Конфигурационное тестирование (Configuration Testing)

#### *Связанные с изменениями виды тестирования*

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть пере тестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- Дымовое тестирование (Smoke Testing)
- Регрессионное тестирование (Regression Testing)
- Тестирование сборки (Build Verification Test)
- Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

**Понятие дымовое тестирование** пошло из инженерной среды:

"При вводе в эксплуатацию нового оборудования ("железа") считалось, что тестирование прошло удачно, если из установки не пошел дым."

В области же программного обеспечения, *дымовое тестирование рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции.*

#### **Тестирование сборки:**

Тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом Дымового Тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

Тестирование на разных уровнях производится на протяжении всего жизненного цикла разработки и сопровождения программного обеспечения. Уровень тестирования определяет то, **над чем** производятся тесты: над отдельным модулем, группой модулей или системой, в целом. Проведение тестирования на всех уровнях системы - это залог успешной реализации и сдачи проекта.

#### *Уровни Тестирования*

1. Компонентное или Модульное тестирование (Component Testing or Unit Testing)
2. Интеграционное тестирование (Integration Testing)
3. Системное тестирование (System Testing)
4. Приемочное тестирование (Acceptance Testing)

*Компонентное (модульное) тестирование* проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится, вызывая код, который необходимо проверить и при поддержке сред разработки, таких как фреймворки (frameworks - каркасы) для модульного тестирования или инструменты для отладки. Все найденные дефекты, как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это *подготовка автоматизированных тестов* до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор, пока все тесты не будут успешно пройдены.

#### [Разница между компонентным и модульным тестированием](#)

По-существу эти уровни тестирования представляют одно и тоже, разница лишь в том, что в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании - конкретные значения.

**Интеграционное тестирование** предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения в системе в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;
- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

#### [Тест дизайн](#)

**Тест дизайн** – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

### *План работы над тест дизайном*

- анализ имеющихся проектных артефактов: документация (спецификации, требования, планы), модели, исполняемый код и т.д.
- написание спецификации по тест дизайну (Test Design Specification)
- проектирование и создание тестовых случаев (Test Cases)

### *Роли, ответственные за тест дизайн*

- Тест аналитик - определяет "**ЧТО** тестировать?"
- Тест дизайнер - определяет "**КАК** тестировать?"

Попросту говоря, задача тест аналитиков и дизайнеров сводится к тому, чтобы, используя различные стратегии и техники тест дизайна, создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого приложения. Однако, на большинстве проектов эти роли не выделяются, а доверяется обычным тестирующим, что не всегда положительно сказывается на качестве тестов, тестировании и, как из этого следует, на качестве программного обеспечения (конечного продукта).

## **Стандартизация**

### *Введение*

Концепция языка программирования неотрывно связана с его реализацией. Для того, чтобы компиляция одной и той же программы различными компиляторами всегда давала одинаковый результат, разрабатываются стандарты языков программирования. Существует ряд организаций, которые занимаются вопросами стандартизации. Вот некоторые из них: ANSI (American National Standards Institute), IEEE (Institute of Electrical and Electronic Engineers), ISO (International Organization for Standardization), ECMA (European Computer Manufacturers Association), The Open Group.

Как правило, при создании языка выпускается частный стандарт, определяемый разработчиками языка. Если язык получает широкое распространение, то со временем появляются различные версии компиляторов, которые не точно следуют частному стандарту. В большинстве случаев идет расширение зафиксированных первоначально возможностей языка. Для приведения наиболее популярных реализаций языка в соответствие друг с другом разрабатывается согласительный стандарт. Очень важным фактором стандартизации языка программирования является своевременность появления стандарта – до широкого распространения языка и создания множества несовместимых реализаций. В процессе развития языка могут появляться новые стандарты, отражающие современные нововведения. Так, язык FORTRAN первоначально был стандартизирован в 1966 году. В результате был издан стандарт FORTRAN 66. Далее этот стандарт несколько раз пересматривался (в 1977 году был выпущен FORTRAN 77, затем появился и FORTRAN 90).

В процессе развития языка некоторые его конструкции и функции устаревают. Однако с целью обратной совместимости новые версии должны поддерживать и все устаревающие возможности. Это ведет к "разбуханию" компиляторов. В последнее время в реализациях введено понятие не рекомендуемой и устаревшей возможности (*deprecated*). В первом случае следующий стандарт еще будет поддерживать не рекомендуемую возможность, но может перевести ее в категорию устаревшей. Во втором случае стандарт может исключить поддержку возможности, объявленной ранее как устаревшая. Введение не рекомендуемых и устаревших возможностей предоставляет

разработчикам временной интервал, в течение которого они могут модифицировать код в соответствии с новыми требованиями стандарта.

Внедрение новых стандартов языка происходит крайне медленно. Причем чем более используема технология, тем более медленно внедряются новые стандарты. Так, например, готовится стандарт Java 9, а огромные игроки глобального рынка только-только, с треском, переехали на Java 7.

Посмотрим на несколько языков программирования с точки зрения стандартов.

### Стандарты C++

C++, как и другие языки, стандартизован. На конец 2015 года, современным стандартом является C++14. Наиболее используемым является C++11. Ведется работа над C++17. Официальное издание стандарта публикуется ISO, и стоит денег. Его можно купить на организации ANSI: [ansi.org](http://ansi.org). Также можно увидеть черновики стандарта: на сайте [open-std.org](http://open-std.org) публикуются рабочие документы комитета по стандартизации, в том числе и черновики стандарта.

Когда очередная версия стандарта готова, публикуется "финальный черновик" (Final Draft), который затем отправляется в ISO. Он практически ничем не отличается от официального издания стандарта. Однако после публикации официального издания доступ к финальному черновику закрывается. Также существует возможность следить за репозиторием на GitHub. Некоторые исходные документы размещены на [github.com/cplusplus/draft](https://github.com/cplusplus/draft). Их можно скомпилировать в .pdf и получить самый свежий черновик.

### Стандарты JavaScript

JavaScript является реализацией языка ECMAScript, спецификации ECMA-262 (несколько интересных стандартов ECMA будут рассмотрены далее). А значит все спецификации JavaScript - редакции спецификации ECMA-262. На данный момент актуальная спецификация языка - ECMAScript 6 (ECMAScript 2015). Любую информацию по последней спецификации можно найти в сети Интернет.

### Стандарты Java

Первая спецификация была выпущена в 26 августа 1996 года. Последняя на конец 2015 года спецификация - Java 8, выпущенная 19 марта 2014 года. Будущая спецификация, Java 9, планируется к выпуску в сентябре 2016 года. Всю информацию о стандартах языка можно найти на сайте [docs.oracle.com/javase/specs/](http://docs.oracle.com/javase/specs/). Спецификация общедоступна и совершенно бесплатна.

### Интересные стандарты Ecma

**Ecma International** — основанная в 1961 году ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий. Изначально ассоциация называлась **ECMA** — European Computer Manufacturers Association, однако она сменила название в 1994 году в связи с глобализацией её деятельности. Вследствие этого название Ecma перестало быть аббревиатурой и больше не пишется заглавными буквами. Ассоциация создана для создания (в сотрудничестве с организациями аналогичной направленности, но локального масштаба) стандартов и технических отчётов в порядке поддержки и стандартизации использования информационных и сетевых систем, а также для публикации их в электронном и бумажном виде. Распространение такого рода документов, по мнению организации, должно быть бесплатно и неограниченно.

Примеры стандартов:

- ECMA-1 — Стандарт на 6-битный код символа ввода-вывода (Standard for a 6-bit Input/Output character code)
- ECMA-13 — Файловая структура и размечивание магнитных лент (File Structure and Labelling of Magnetic Tapes) (позднее был принят ISO 1001)
- ECMA-58 — 8-дюймовая дискета
- ECMA-107 — Файловая система FAT (аналогичен ISO/IEC 9293)
- ECMA-119 — Файловая система CD-ROM (впоследствии утвержден как ISO 9660:1988)
- ECMA-262 — ECMAScript (стандартизированный JavaScript)
- ECMA-334 — Язык программирования C#
- ECMA-372 — C++/CLI
- ECMA-404 — JSON
- ECMA-408 — Dart Programming Language Specification

## Глава 7. Инновационные концепции и технологии

### Искусственный интеллект

#### Направления искусственного интеллекта

В настоящее время различают два основных подхода к моделированию искусственного интеллекта: машинный интеллект, заключающийся в строгом задании результата функционирования, и искусственный разум, направленный на моделирование внутренней структуры системы. Разделение работ по искусственному интеллекту на два направления связано с существованием двух точек зрения на вопрос, каким образом строить системы искусственного интеллекта. Сторонники одной точки зрения убеждены, что важнее всего результат, т.е. хорошее совпадение поведения искусственно созданных и естественных интеллектуальных систем, а что касается внутренних механизмов формирования поведения, то разработчик искусственного интеллекта вовсе не должен копировать или даже учитывать особенности естественных, живых аналогов. Другая точка зрения состоит в том, что именно изучение механизмов естественного мышления и анализ данных о способах формирования разумного поведения человека могут создать основу для построения систем искусственного интеллекта,

Первое направление, таким образом, рассматривает продукт интеллектуальной деятельности человека, изучает его структуру, и стремится воспроизвести этот продукт средствами современной техники. Моделирование систем машинного интеллекта достигается за счет использования законов формальной логики, теории множеств, графов, семантических сетей и других достижений науки в области дискретных вычислений.

Второе направление искусственного интеллекта рассматривает данные о нейрофизиологических и психологических механизмах интеллектуальной деятельности и, в более широком плане, разумного поведения человека. Оно стремиться воспроизвести эти механизмы с помощью тех или иных технических устройств, с тем чтобы поведение таких устройств хорошо совпадало с поведением человека в определенных, заранее задаваемых пределах. Развитие этого направления тесно связано с успехами наук о человеке. Для него характерно стремление к воспроизведению более широкого, чем в машинном интеллекте, спектра проявлений разумной деятельности человека. Системы искусственного разума базируются на математической интерпретации деятельности нервной системы во главе с мозгом человека.

Первым шагом в первом направлении можно считать разработку GPS-универсального решателя задач. В его основу было положено представление об эвристическом поиске, в процессе которого обеспечивалось разбиение задачи на подзадачи до тех пор, пока не будет получена легко решаемая подзадача.

Попытки уйти от не оправдавших себя универсальных эвристик при решении интеллектуальных задач привели к появлению систем, базирующихся на знаниях. Знаниями можно назвать сложную совокупность сведений о некоторой предметной области, совокупности объектов или просто объекте. Такие системы стали называть экспертными. Первой экспертной системой была Dendral, которая умела определять структуру неизвестного органического соединения.

Базовая структура системы, базирующейся на знаниях состоит из следующих блоков: базы знаний, содержащей знания о некоторой ограниченной предметной области; решателя, или блока логического вывода, осуществляющего активизацию знаний, соответствующих текущей ситуации;

блока верификации БЗ, обеспечивающего добавление новых знаний и корректировку уже существующих; блока объяснения, позволяющего пользователю прослеживать всю цепочку рассуждений системы, приводящих к конечному результату, и, наконец, интерфейса, обеспечивающего удобную связь между пользователем и системой.

Так как нейроподобные сети в последнее время являются одним из самых перспективных направлений в области искусственного интеллекта и постепенно входят в бытность людей в широком спектре деятельности, то будем рассматривать их в качестве примера второго направления.

Нейроподобные сети прошли длинный путь становления и развития, от полного отрицания возможности их применения до воплощения во многие сферы деятельности человека, в том числе и измерения.

Современные цифровые вычислительные машины способны с высоким быстродействием и точностью решать формализованные задачи с вполне определенными данными по заранее известным алгоритмам. Однако в тех случаях, когда задача не поддается формализации, а входные данные неполны, зашумлены или противоречивы, применение традиционных компьютеров становится неэффективным. Альтернативой им становятся специализированные компьютеры, реализующие нетрадиционные нейросетевые технологии. Сильной стороной этих комплексов является нестандартный характер обработки информации. Она кодируется и запоминается не в отдельных ячейках памяти, а в распределении связей между нейронами и в их силе, поэтому состояние каждого отдельного нейрона определяется состоянием многих других нейронов, связанных с ним. Следовательно, потеря одной или нескольких связей не оказывает существенного влияния на результат работы системы в целом, что обеспечивает ее высокую надежность.

Нейроподобные сети могут решать следующие задачи:

- обработка высокоскоростных цифровых потоков;
- быстрый поиск и классификация информации в реальном масштабе времени;
- решение трудоемких задач оптимизации;
- адаптивное управление и предсказание.
- обработка и анализ изображений;
- распознавание речи и перевод;

Одно из важнейших свойств нейроподобной сети - способность к самоорганизации, самоадаптации с целью улучшения качества функционирования. Это достигается обучением сети, алгоритм которого задается набором обучающих правил. Обучающие правила определяют, каким образом изменяются связи в ответ на входное воздействие. Многие из них являются развитием высказанной Д. О. Хеббом идеи о том, что обучение основано на увеличении силы связи между одновременно активными нейронами. Таким образом, часто используемые в сети связи усиливаются, что объясняет феномен обучения путем повторения и привыкания.

### Представление знаний в СИИ

Важное место в теории искусственного интеллекта занимает проблема представления знаний, являющаяся, по мнению многих исследователей, ключевой.

Первоначально вычислительная техника была ориентирована на обработку данных. Это было связано как с уровнем развития техники и программного обеспечения, так и со спецификой решаемых задач. Дальнейшее усложнение решаемых задач и их интеллектуализация ставят задачу создания машин обработки знаний. Знания характеризуются наличием ситуативных связей, определяющих ситуативную совместимость отдельных событий и фактов, позволяющих устанавливать причинно-следственные связи.

Некоторые исследователи предпринимали попытки определить типы знаний, которые должны быть представлены в системах ИИ. Так, например, этот перечень может охватывать: структуру, форму, свойства, функции и возможные состояния объекта; возможные отношения между объектами, возможные события, в которых эти объекты могут участвовать; физические законы; возможные намерения, цели, планы, соглашения. Но никакого универсального способа не придумали, поэтому существуют разные способы представления знаний (СПЗ). Существует ряд общих для всех СПЗ проблем. К ним можно отнести проблемы: приобретения новых знаний и их взаимодействие с уже существующими, организации ассоциативных связей.

Модели представления знаний можно условно разделить на декларативные и процедуральные.

В процедуральном представлении знания содержатся в процедурах - небольших программах, которые определяют, как выполнять специфичные действия (как поступать в специфичных ситуациях). При этом можно не описывать все возможные состояния среды или объекта для реализации вывода.

В декларативных моделях не содержатся в явном виде описания выполняемых процедур. Эти модели обычно представляют собой множество простых утверждений и сложного механизма их вывода, оперирующего этими этими утверждениями.

В реальных системах представления знаний используются в равной мере элементы и сочетания указанных выше моделей.

#### *Модели представления знаний*

Продукционные модели представляют собой набор правил в виде "условие - действие", где условия являются утверждениями о содержимом БД (фактов), а действия есть некоторые процедуры, которые могут модифицировать содержимое БД. Продукционные модели из-за модульного представления знаний, легкого расширения и модификации нашли широкое применение в экспертных системах.

Другая важная схема представления знаний - семантические сети, представляющие собой направленный граф, в котором вершинам ставятся в соответствие конкретные объекты, а дугам, их связывающим, - семантические отношения между этими объектами. Семантические сети могут использоваться как для декларативных, так и для процедуральных знаний.

Перспективной формой представления знаний являются фреймы, которые быстро завоевали популярность у разработчиков систем ИИ благодаря своей универсальности и гибкости.

Принципиальным методом для логического представления знаний является использование логики предикатов первого порядка (исчисление предикатов). При таком подходе знания о некоторой предметной области могут рассматриваться как совокупность логических формул. Изменения в

модели представления знаний происходят в результате добавления или удаления логических формул.

В редукционных моделях осуществляется декомпозиция исходной задачи на ряд подзадач, решая которые последовательно определяют решение поставленной задачи.

### *Игровой искусственный интеллект*

Игровой ИИ не должен быть наделен чувствами и самосознанием, ему нет необходимости обучаться чему-либо за пределами рамок игрового процесса. Подлинная цель ИИ в играх состоит в имитации разумного поведения и в предоставлении игроку убедительной, правдоподобной задачи.

### *Принятие решений*

Основным принципом, лежащим в основе работы ИИ, является принятие решений. Для выбора при принятии решений система должна влиять на объекты с помощью ИИ. При этом такое воздействие может быть организовано в виде «вещания ИИ» или «обращений объектов».

В системах с «вещанием ИИ» система ИИ обычно изолирована в виде отдельного элемента игровой архитектуры. Такая стратегия зачастую принимает форму отдельного потока или нескольких потоков, в которых ИИ вычисляет наилучшее решение для заданных параметров игры. Когда ИИ принимает решение, оно передается всем участвующим объектам. Такой подход лучше всего работает в стратегиях реального времени, где ИИ анализирует общий ход событий во всей игре.

Системы с «обращениями объектов» лучше подходят для игр с простыми объектами. В таких играх объекты обращаются к системе ИИ каждый раз, когда объект «думает» или обновляет себя. Такой подход отлично подходит для систем с большим количеством объектов, которым не нужно «думать» слишком часто, например, в шутерах. Такая система также может воспользоваться преимуществами многопоточной архитектуры, но для нее требуется более сложное планирование.

### *Базовое восприятие*

Чтобы искусственный интеллект мог принимать осмысленные решения, ему необходимо каким-либо образом воспринимать среду, в которой он находится. В простых системах такое восприятие может ограничиваться простой проверкой положения объекта игрока. В более сложных системах требуется определять основные характеристики и свойства игрового мира, например, возможные маршруты для передвижения, наличие естественных укрытий на местности, области конфликтов. При этом разработчикам необходимо придумывать способ выявления и определения основных свойств игрового мира, важных для системы ИИ. Например, укрытия на местности могут быть заранее определены дизайнерами уровней или заранее вычислены при загрузке или компиляции карты уровня. Некоторые элементы необходимо вычислять на лету, например, карты конфликтов и ближайшие угрозы.

### *Системы на основе правил*

Простейшей формой искусственного интеллекта является система на основе правил. Такая система дальше всего стоит от настоящего искусственного интеллекта. Набор заранее заданных алгоритмов определяет поведение игровых объектов. С учетом разнообразия действий конечный результат может быть неявной поведенческой системой, хотя такая система на самом деле вовсе не будет «интеллектуальной». Классическим игровым приложением, где используется такая система,

является Pac-Man. Игрока преследуют четыре привидения. Каждое привидение действует, подчиняясь простому набору правил. Одно привидение всегда поворачивает влево, другое всегда поворачивает вправо, третье поворачивает в произвольном направлении, а четвертое всегда поворачивает в сторону игрока. Если бы на экране привидения появлялись по одному, их поведение было бы очень легко определить, и игрок смог бы без труда от них спастись. Но поскольку появляется сразу группа из четырех привидений, их движения кажутся сложным и скоординированным выслеживанием игрока. На самом же деле только последнее из четырех привидений учитывает расположение игрока.

#### *Адаптивный ИИ*

Если же в игре требуется большее разнообразие, если у игрока должен быть более сильный и динамичный противник, то ИИ должен обладать способностью развиваться, приспосабливаться и адаптироваться. Адаптивный ИИ часто используется в боевых и стратегических играх со сложной механикой и огромным количеством разнообразных возможностей в игровом процессе.

#### *Тактический ИИ*

Роль тактического ИИ состоит в координации усилий групп объектов в игре. Группы более эффективны, поскольку члены группы могут поддерживать друг друга, могут действовать как единое подразделение, обмениваться информацией и распределять действия по получению информации.

Принцип тактического ИИ построен вокруг динамики группы. Игра должна отслеживать различные группы объектов. Каждую группу необходимо обновлять отдельно от индивидуальных объектов. Для этого можно использовать выделенный модуль обновления, отслеживающий различные группы, их цели и их состав. Недостаток этого метода состоит в том, что требуется разработка отдельной системы для игрового движка. Поэтому будет проще использовать метод командира группы.

Одному юниту в составе группы можно назначить роль командира группы. Все остальные члены группы связаны со своим командиром, их поведение определяется информацией, полученной из приказов командира. Командир группы обрабатывает все вычисления тактического ИИ для всей группы.

#### *Стратегический ИИ*

Стратегический ИИ — это ИИ более высокого порядка, он управляет целой армией и вырабатывает оптимальные стратегии.

Стратегический ИИ обычно применяется в стратегиях в реальном времени, но в последнее время его все чаще реализуют и в тактических шутерах от первого лица. Управляемый игроком командир может представлять собой отдельную систему или может быть настроен как пустой объект, не имеющий места и графического изображения, но обновляемый и размышляющий.

Командиры подчиняются иерархическим системам правил и конечным автоматам, которые управляют такой деятельностью, как сбор ресурсов, изучение дерева технологий, создание армии и т. д. Как правило, для базового поддержания игровых элементов не требуется особо сложных размышлений. Интеллект нужен главным образом при взаимодействии с другими игроками.

Управление этим взаимодействием (или сражением) — вот где прежде всего работает ИИ. Командир должен изучить карту игры, чтобы обнаружить игрока, выявить основные области интереса, например, узкие проходы, выстроить оборону и проанализировать оборону другого игрока. Как именно это сделать? Очевидного ответа на этот вопрос нет, но для упрощения можно использовать карты решений.

Карты решений представляют собой двухмерные массивы, приближенно соответствующие игровой карте. Каждая ячейка массива соответствует определенной области в игре и содержит важную информацию об этой области. Эти карты помогают стратегическому ИИ принимать грамотные решения относительно игры в целом.

## Виртуальная реальность

### Общие понятия

**Виртуальная реальность**, VR, искусственная реальность, электронная реальность, компьютерная модель реальности (*virtual reality, VR*) — созданный техническими средствами мир, передаваемый человеку через его ощущения: зрение, слух, обоняние, осязание и другие. Виртуальная реальность имитирует как воздействие, так и реакции на воздействие. Для создания убедительного комплекса ощущений реальности компьютерный синтез свойств и реакций виртуальной реальности производится в реальном времени.

Не следует путать **виртуальную** реальность с **дополненной**. Их коренное различие в том, что **виртуальная** конструирует новый искусственный мир, а **дополненная реальность** лишь вносит отдельные искусственные элементы в восприятие мира реального.

**Дополненная реальность** — добавление к поступающим из реального мира ощущениям мнимых объектов, обычно вспомогательно-информационного свойства. В западном научном сообществе данное направление получило устоявшуюся терминологию — *Augmented Reality, AR*. По своей сути, это родственное искусственной реальности явление.

Известным примером дополненной реальности может служить нашлемное целеуказание в самолётах-истребителях (Су-27 и др.), вывод дополнительной информации на ветровое стекло автомобиля.

Виртуальная реальность — это огромный дивный мир, в который мы не заглянули даже глазком. Хотя под определенной интерпретацией виртуальной реальности можно понимать Интернет, в действительности же ее потенциал гораздо больше. Это место, в которое человек может погрузиться целиком и полностью и найти там гораздо больше, чем в реальной жизни, а также, не думая о том, чтобы отличать виртуальное от реального. На данный момент разными компаниями разрабатывается аппаратное обеспечение для полного выхода в виртуальную реальность: Omni, Oculus Rift, Project Morpheus от Sony.

Также многое делается для создания дополненной реальности. Этим, к примеру, занимаются Google с их Google Glass и Microsoft с их HoloLens. Вполне может получиться, что с развитием высоких технологий в этой сфере виртуальная реальность займет прочное место в нашей жизни и обеспечит людей огромным, практически безграничным пространством для ведения любых дел.

## Реализация

Принято называть системами «виртуальной реальности» устройства, которые более полно по сравнению с обычными компьютерными системами имитируют взаимодействие с виртуальной средой, путём воздействия на все пять имеющихся у человека органов чувств.

Объекты виртуальной реальности обычно ведут себя близко к поведению аналогичных объектов материальной реальности. Однако часто в развлекательных целях пользователям виртуальных миров позволяет больше, чем возможно в реальной жизни (например: летать, создавать любые предметы и т. п.).

В целом, пользователь всегда может воздействовать на эти объекты в согласии с реальными или же встроенными в реальность законами физики (гравитация, свойства воды, столкновение с предметами, отражение и т. п.).

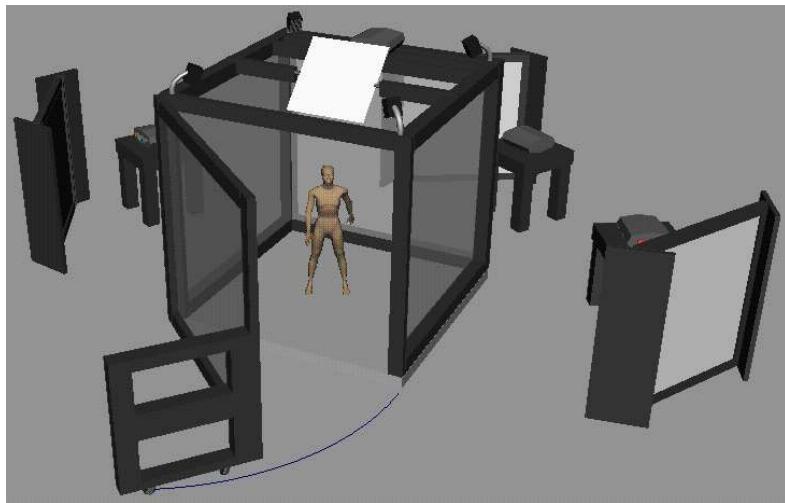
Коснёмся поподробнее каждого из аспектов ВР.

## Изображение

### *Шлем / очки виртуальной реальности (HMD - display)*

Современные шлемы виртуальной реальности представляют собой скорее очки, нежели шлем, и содержат один или несколько дисплеев, на которые выводятся изображения для левого и правого глаза, систему линз для корректировки геометрии изображения, а также систему трекинга, отслеживающую ориентацию устройства в пространстве. Как правило, системы трекинга для шлемов виртуальной реальности разрабатываются на основе гироскопов, акселерометров и магнитометров. Для систем этого типа важен широкий угол обзора, точность работы системы трекинга при отслеживании наклонов и поворотов головы пользователя, а также минимальная задержка между детектированием изменения положения головы в пространстве и выводом на дисплеи соответствующего изображения.

### *MotionParallax3D дисплеи*



К устройствам этого типа относится множество различных устройств: от некоторых смартфонов до комнат виртуальной реальности (CAVE). Системы данного типа формируют у пользователя иллюзию объемного объекта за счет вывода на один или несколько дисплеев специально сформированных проекций виртуальных объектов, сгенерированных исходя из информации о положении глаз

пользователя. При изменении положения глаз пользователя относительно дисплеев, изображение на них соответствующим образом меняется. Все системы данного типа задействуют зрительный механизм восприятия объёмного изображения параллакс движения (Motion Parallax). Также, в большинстве своём, они обеспечивают вывод стереоизображения с помощью стереодисплеев, задействуя стереоскопическое зрение. Системы трекинга для MotionParallax3D дисплеев отслеживают координаты глаз пользователей в пространстве. Для этого используются различные технологии: оптическая (определение координат глаз пользователя на изображении с камеры, отслеживание активных или пассивных маркеров), существенно реже - ультразвуковая. Зачастую системы трекинга могут включать в себя дополнительные устройства: гироскопы, акселерометры и магнитометры. Для систем данного типа важна точность отслеживания положения пользователя в пространстве, а также минимальная задержка между детектированием изменения положения головы в пространстве и выводом на дисплеи соответствующего изображения. Системы данного класса могут выполняться в различных формах — факторах: от виртуальных комнат с полным погружением до экранов виртуальной реальности размером от трёх дюймов.

**Параллакс** (греч. παραλλάξ, от παραλλαγή, «смена, чередование») — изменение видимого положения объекта относительно удалённого фона в зависимости от положения наблюдателя.

#### *Виртуальный ретинальный монитор*

Устройства данного типа формируют изображение непосредственно на сетчатке глаза. В результате пользователь видит изображение, «висящее» в воздухе перед ним. Устройства данного типа ближе к системам дополненной реальности, поскольку изображения виртуальных объектов, которые видит пользователь, накладываются на изображения объектов реального мира. Тем не менее, при определенных условиях (тёмная комната, достаточно широкое покрытие сетчатки изображением, а также в сочетании с системой трекинга), устройства данного типа могут использоваться для погружения пользователя в виртуальную реальность.

Также существуют различные гибридные варианты: например, система CastAR, в которой получение корректной проекции изображения на плоскости достигается за счет расположения проекторов непосредственно на очках, а стереоскопическое разделение - за счет использования световозвращающего покрытия поверхности, на которую ведётся проецирование. Но пока такие устройства широко не распространены и существуют лишь в виде прототипов.

На данный момент самыми совершенными системами виртуальной реальности являются проекционные системы, выполненные в компоновке комнаты виртуальной реальности (CAVE).



Такая система представляет собой комнату, на все стены которой проецируется 3D-стереоизображение. Положение пользователя, повороты его головы отслеживаются трекинговыми системами, что позволяет добиться максимального эффекта погружения. Данные системы активно используются в маркетинговых, военных, научных и других целях.

### Звук

Многоканальная акустическая система позволяет производить локализацию источника звука, что позволяет пользователю ориентироваться в виртуальном мире с помощью слуха.

### Имитация тактильных ощущений

Имитация тактильных или осознательных ощущений уже нашла своё применение в системах виртуальной реальности. Это так называемые устройства с обратной связью. Применяются для решения задач виртуального прототипирования и эргономического проектирования, создания различных тренажёров, медицинских тренажёров, дистанционном управлении роботами, в том числе микро- и нано-, системах создания виртуальных скульптур.

### Управление

С целью наиболее точного воссоздания контакта пользователя с окружением применяются интерфейсы пользователя, наиболее реалистично соответствующие моделируемым: компьютерный руль с педалями, рукояти управления устройствами, целеуказатель в виде пистолета и т. д.

Для бесконтактного управления объектами используются как перчатки виртуальной реальности, так и отслеживание перемещений рук, осуществляемое с помощью видеокамер. Последнее обычно реализуется в небольшой зоне и не требует от пользователя дополнительного оборудования.

Перчатки виртуальной реальности могут быть составной частью костюма виртуальной реальности, отслеживающего изменение положения всего тела и передающего также тактильные, температурные и вибрационные ощущения.

Устройство для отслеживания перемещений пользователя может представлять собой свободно вращаемый шар, в который помещают пользователя, или осуществляться лишь с помощью

подвешенного в воздухе или погружённого в жидкость костюма виртуальной реальности. Также разрабатываются технические средства для моделирования запахов.

#### [Прямое подключение к нервной системе](#)

Описанные выше устройства воздействуют на органы чувств человека, но данные могут передаваться и непосредственно нервным окончаниям, и даже напрямую в головной мозг посредством мозговых интерфейсов. Подобная технология применяется в медицине для замены утраченных чувствительных способностей, но пока она слишком дорога для повседневного применения и не достигает качества передачи данных, приемлемого для передачи виртуальной реальности. На этом же принципе основаны различные физиотерапевтические приборы и устройства, воспроизводящие ощущения реального мира в измененном состоянии сознания ("Радиосон" и др.).

#### [Применение](#)

##### [Компьютерные игры](#)

Интерактивные компьютерные игры основаны на взаимодействии игрока с создаваемым ими виртуальным миром. Многие из них основаны на отождествлении игрока с персонажем игры, видимым или подразумеваемым.

Существует устоявшееся мнение, что качественная трёхмерная графика обязательна для качественного приближения виртуального мира игры к реальности. Если виртуальный мир игры не отличается графической красотой, схематичен и даже двумерен, погружение пользователя в этот мир может происходить за счёт захватывающего игрового процесса, характеристики которого индивидуальны для каждого пользователя.

Существует целый класс игр-симуляторов какого-либо рода деятельности. Распространены авиасимуляторы, автосимуляторы, разного рода экономические и спортивные симуляторы, игровой мир которых моделирует важные для данного рода физические законы, создавая приближенную к реальности модель.

Специально оборудованные тренажёры и определённый вид игровых автоматов к выводу изображения и звука компьютерной игры/симулятора добавляют другие ощущения, такие, как наклон мотоцикла или тряска кресла автомобиля. Подобные профессиональные тренажёры с соответствующими реальным средствами управления применяются для обучения пилотов.

Несоответствие команд интерфейса пользователя осуществляемым в игре действиям, его сложность могут мешать погружению в мир игры. С целью снять эту проблему используется не только компьютерная клавиатура и мышь, но и компьютерный руль с педалями, целеуказатель в виде пистолета и другие игровые манипуляторы.

##### [Обучение](#)

Виртуальная реальность применяется для обучения профессиям, где эксплуатация реальных устройств и механизмов связана с повышенным риском либо связана с большими затратами (пилот самолёта, машинист поезда, диспетчер, водитель, горноспасатель и т. п.).

Кроме целей чему-либо научиться, виртуальная реальность используется и в развлекательных целях.

## История

До эры компьютерных технологий под виртуальностью понимали объект или состояние, которые реально не существуют, но могут возникнуть при определенных условиях.

Понятие *искусственной реальности* было впервые введено Майроном Крюгером (*Myron Krueger*) в конце 1960-х. В 1964 году Станислав Лем в своей книге «Сумма Технологии» под термином «Фантомология» описывает задачи и суть ответа на вопрос «как создать действительность, которая для разумных существ, живущих в ней, ничем не отличалась бы от нормальной действительности, но подчинялась бы другим законам?». Первая система виртуальной реальности появилась в 1962 году, когда Мортон Хейлиг (*Morton Heilig*) представил первый прототип мультисенсорного симулятора, который он называл «Сенсорама» (*Sensorama*). Сенсорама погружала зрителя в виртуальную реальность при помощи коротких фильмов, которые сопровождались запахами, ветром (при помощи фена) и шумом мегаполиса с аудиозаписи. В 1967 году Айвен Сазерленд (*Ivan Sutherland*) описал и сконструировал первый шлем, изображение на который генерировалось при помощи компьютера. Шлем Сазерленда позволял изменять изображения соответственно движениям головы (зрительная обратная связь).

В 1970-х годах компьютерная графика полностью заменила видеосъёмку, до того использовавшуюся в симуляторах. Графика была крайне примитивной, однако важным было то, что тренажёры (это были симуляторы полётов) работали в режиме реального времени. Первой реализацией виртуальной реальности считается «Кинокарта Аспена» (*Aspen Movie Map*), созданная в Массачусетском Технологическом Институте в 1977 году. Эта компьютерная программа симулировала прогулку по городу Аспен, штат Колорадо, давая возможность выбрать между разными способами отображения местности. Летний и зимний варианты были основаны на реальных фотографиях.

В середине 1980-х появились системы, в которых пользователь мог манипулировать с трехмерными объектами на экране благодаря их отклику на движения руки. В 1989 году Джарон Ланьеर ввёл более популярный ныне термин «виртуальная реальность». В фантастической литературе поджанра киберпанк виртуальная реальность есть способ общения человека с «киберпространством» — некой средой взаимодействия людей и машин, создаваемой в компьютерных сетях.

В данный момент технологии виртуальной реальности широко применяются в различных областях человеческой деятельности: проектировании и дизайне, добыче полезных ископаемых, военных технологиях, строительстве, тренажёрах и симуляторах, маркетинге и рекламе, индустрии развлечений и т. д. Объём рынка технологий виртуальной реальности оценивается в 15 млрд долларов в год.

## Известные реализации виртуальной реальности

Second Life — сетевой трехмерный виртуальный мир с элементами социальной сети, который насчитывает свыше 1 млн активных пользователей. Самая популярная на сегодняшний день реализация виртуальной реальности.

Многие университеты и компании используют Second Life для обучения, включая Гарвардский и Оксфордский университеты. В 2007 году Second Life используется как место для обучения иностранным языкам. И преподаватели в Second Life и реальные преподаватели используют

виртуальный мир для обучения языкам. Английский (как второй язык) сейчас изучают в нескольких школах, включая Британский Совет, который сфокусировался на интерактивных молодёжных образовательных сетях (Teen Grid). Институт испанского языка и культуры «Instituto Cervantes» и Goethe-Institut имеют свои острова в Second Life. Список образовательных проектов (включающий несколько языковых школ) в Second Life можно найти на сайте SimTeach.

Ряд корпораций открыли свои представительства в Second Life, используя виртуальный мир в рекламных целях, а также для проведения совещаний и взаимодействия сотрудников. Например, *IBM выстраивает виртуальное рабочее пространство для работников из удалённых регионов*, Sun Microsystems — для сотрудников, работающих вне офиса. Reuters и CNN используют Second Life для распространения и получения информации, а *NASA открыло виртуальный исследовательский центр. В Second Life работает информационный центр Армии США*.

Second Life выполняет роль социальной сети, в рамках которой пользователи могут общаться друг с другом (в этом смысле Second Life не является компьютерной игрой, так как все игровые моменты организуются самими пользователями, а не разработчиками). Linden Lab поощряет развитие в этом направлении, интегрировав профили пользователей с твиттером и Facebook. Участники Second Life объединены в многочисленные группы самой разнообразной направленности, образуя сообщества по интересам.

### Философское понятие

Философия абстрагирует идею виртуальной реальности от её технического воплощения. Виртуальную реальность можно толковать как совокупность моделируемых реальными процессами объектов, содержание и форма которых не совпадает с этими процессами. Существование моделируемых объектов сопоставимо с реальностью, но рассматривается обособленно от неё — виртуальные объекты существуют, но не как субстанции реального мира. В то же время эти объекты актуальны, а не потенциальны. «Виртуальность» (мнимость, ложная кажимость) реальности устанавливается по отношению к обуславливающей её «основной» реальности. Виртуальные реальности могут быть вложены друг в друга. При завершении моделирующих процессов, идущих в «основной» реальности, виртуальная реальность исчезает.

### Свойства

Независимо от реализации виртуальной реальности, в ней можно выделить следующие свойства:

- порождённость (виртуальная реальность производится другой, внешней к ней реальностью),
- актуальность (существует актуально, в момент наблюдения, «здесь и сейчас»),
- автономность (имеет свои законы бытия, времени и пространства);
- интерактивность (может взаимодействовать с другими реальностями, тем не менее, обладая независимостью).

По философской концепции С. С. Хоружего компьютерную виртуальную реальность можно характеризовать как многомодусное бытие, то есть бытие, допускающее множество вариантов и сценариев развития событий.

*А не живём ли мы в виртуальной реальности?*

На мой взгляд, нет.

На данный момент, мы в течение прошлого века упорно разрабатывали невиданный доселе инструмент, который сейчас есть у каждого без исключения. И, само собой, этот инструмент начал на нас влиять. Он породил не одну дискуссию вокруг себя.

Понятно, что речь идёт об компьютерах, об ЭВМ, да и обо всём, что так или иначе связано с техникой. Мы шагнули далеко вперёд и до сих пор осваиваем плод нашего труда. Не поспорить также и с тем фактом, что мы не до конца изучили этот инструмент, однако сумели его создать.

Не до конца, это значит, мы не можем быть однозначно уверены в его влиянии на окружающих. А влияние его велико, просто огромно. Иначе как объяснить вопросы «а не в виртуальной реальности ли мы»?

Вспомните Коперника с его гелиоцентрической системой, вспомните бум инопланетной тематики в 50-ые годы прошлого века и вы поймёте, что человека всегда интересует неизведанное, неопознанное, непонятное. И на основе этого невежества с любопытством, которые, на мой взгляд, с развитием технологий только преумножаются, человек и строит такие красочные теории, которые, надо отметить, находят своё применение.

Какое? Да они заставляют нас всех посмотреть на мир по-другому, иными глазами. Мы просто начинаем задавать себе вопросы, которые раньше не задавали. То, что было раньше ясно, ставится под разумное сомнение, ведь мы становимся умнее, у нас появляются более изощрённые инструменты для познания мира.

Но живём ли мы в виртуальности? В Матрице?

На данный момент скажу: «Нет, с очень малой вероятностью». Потому что способов проверить, что это не так, мало. Смерть, которую всё-таки страшно к себе подпускать, и выход за доступные границы. Но и он ассоциируется со смертью. Плюс, эти самые доступные границы нами же постоянно расширяются, так что, непонятно, имеет ли наш мир конец, который мы можем достигнуть.

То, что наша Вселенная конечна известно, однако долететь до этого конца, чтобы выйти за край мы не можем – недоразвиты.

Поэтому, чтобы точно ответить на этот вопрос, надо сказать такую фразу: «развивайтесь и учитесь».

#### *ВР = обман?*

Если быть слишком прагматичным и придиличивым, то да, ВР = обман. Вы видите и взаимодействуете с тем, чего нет. Но иногда это полезно, ведь поместив группу из реальных участников внутрь виртуальной симуляции какого-то бедствия, можно добиться от них сплочённой командной работы, а также переосмысления общечеловеческих ценностей.

Если рассматривать ВР в рамках только развлекательной сферы, то тут изначально пользователь совершает внегласный договор, где он разрешает игре собой манипулировать.

#### *Опасения, связанные с ВР*

Информация и общение посредством суррогатов / автаров. Своего рода наркомания, основанная на подмене своей индивидуальности более удобной конструкцией.

Как с этим бороться толком не ясно. Хотя ноги растут у этой проблемы оттуда же, откуда появляется игровая зависимость.

### Строение Oculus Rift

Oculus Rift Development Kit версии 1.1 включает в себя головной набор Oculus Rift, который отдаленно напоминает защитные лыжные очки. В комплект также поставляется блок управления, который прочно прикреплен к шлему кабелем длиной 1.8 метра, съемный ремешок для головы для дополнительного комфорта и стабильности, три пары объективов с различным фокусным расстоянием, кабель HDMI, кабель USB, кабель DVI и стандартный блок питания вместе с международными адаптерами питания 5 Вольт. Все это располагается в жестком футляре. В Oculus Rift Dev Kit очки весят меньше, чем 1 фунт — всего лишь 369 граммов — и будущая потребительская модель может стать еще легче.

Блок управления используется для подключения шлема к компьютеру и дальнейшего выполнения основных функций управления. Здесь и пригодятся кабели HDMI, DVI, MiniUSB и порты DC, а также пять кнопок для управления контрастностью и яркостью. Синий индикатор показывает, подключено ли устройство или нет.

Шлем разработчика предоставляет следующий функционал в области виртуальной реальности: 3 степени свободы (глубина резкости), ультранизкая латентность и поле зрения (FOV) из 110 градусов по диагонали и 90 градусов по горизонтали для более убедительного погружения в несуществующий мир.

Rift оснащен 7-дюймовым (17.8 сантиметров) жидкокристаллическим дисплеем с разрешением 1280x800 (640x720 на каждый глаз) точек с частотой обновления изображения 60 Гц.

Фиксированное расстояние между централи линз составляет 2.5 дюйма (64 миллиметра). У компании есть планы увеличить разрешение потребительской модели до формата FullHD, и Oculus уже продемонстрировала два прототипа. Входы отображения следующие: DVI-D Single Link, HDMI 1.3 и USB 2.0 Full Speed.

Шлем виртуальности реальности Oculus Rift имеет изготовленный специально на заказ блок движения и датчики ориентации с частотой дискретизации до 1000 Гц. Блок датчиков включает в себя гироскоп, акселерометр и магнитометр. В качестве процессора в устройстве служит микроконтроллер на основе платформы ARM Cortex-M3. Данные всех трех датчиков сочетаются посредством процесса, называемого «системой датчиков» — он позволяет быстро и точно отслеживать движения вашей головы и ее ориентацию в пространстве. Благодаря этому вы можете поворачивать голову в любом направлении и смотреть вокруг в виртуальной среде в режиме реального времени.

### Фильмы о виртуальной реальности

Фильмы, представленные в данном списке, рекомендуются к просмотру любому, кто заинтересован в понимании и видении концептуальных сторон виртуальной реальности. По крайне мере, многие из вопросов «А не живём ли мы в виртуальной реальности» задаются именно в этих фильмах.

- Газонокосильщик; Газонокосильщик 2: За пределами киберпространства
- Матрица

- Vr5 (многосерийный фильм)
- Тринадцатый этаж
- Virtual Combat
- Начало
- Авалон
- Экзистенция
- Нирвана
- Виртуозность
- Виртуальный кошмар
- «Хаккер» / «Охотник за кодом» (Code hunter)
- Аватар
- Странные дни
- Соблазн подсознания
- Трон: Наследие
- Призрак в машине
- Мозговой штурм (фильм)
- Звёздный инспектор
- Исходный код (фильм)
- Закрытые пространства
- Суррогаты (фильм)
- аниме «Sword Art Online»
- Accel World
- Log Horizon

### Ещё пару слов о виртуальной реальности

Представьте на минуту, что любые расстояния для вас потеряли свою непреодолимость, любые желания получили моментальное исполнение, а все красоты мира стали доступны по простому нажатию кнопки или двух. Куда бы мы ни сунулись, на нас накладываются ограничения: ноют икры от долгой ходьбы, появляется боязнь высоты, не хватает денег на билет в любую точку мира, а диван обладает мощнейшим гравитационным полем. Да, если бы мы жили в ефремовской утопии или были чрезвычайно богаты, жизнь была бы проще, но на достижение этого ушло бы много лет. К счастью, у всемогущества есть рецепт попроще: виртуальная реальность.

Идея виртуальной реальности уже много лет, но только в последние несколько лет мир подобрался настолько близко к этой границе, что вот-вот — и можно будет пощупать. Сам термин «виртуальная реальность» вошёл в употребление только в 1985 году, тридцать лет назад. Первая техническая реализация устройства, которое, по плану разработчика Айвена Сазерленда, должно было погружать людей в вымышленный мир, увидела свет в 1968 году. Из-за огромных размеров и побочных эффектов его назвали «Дамокловым мечом», и на этом идея себя исчерпала. Впрочем, некогда и Билл Гейтс считал, что 640 килобайт должно быть достаточно для каждого.

Именно период до 2000 года отложился в головах людей как «история развития виртуальной реальности». Потому-то сегодня шлемом и виртуальными тренажёрами никого не удивить, и потому-то скептики пожимают плечами и говорят, что «всё это уже было». В воздухе витает

неуловимое ощущение того, что виртуальная реальность давно с нами и никуда не уходила. Хвала богам, что есть в этом мире любители разбивать шаблоны.

Герой нашего времени моложе многих из нас. В пятнадцать лет Палмер Лаки стал обладателем самой большой в мире коллекции гарнитур с дисплеями, носимых на голове, благодаря своему хобби. В шестнадцать — собрал первый рабочий прототип в своем гараже. В августе 2012 года кампания, запущенная на Kickstarter, уже поднимает 250 000 долларов всего за несколько часов. В марте 2014 года Oculus VR за 2 миллиарда долларов покупает гигант Facebook. Сегодня 21-летний основатель компании стал чуть ли не иконой подрастающего поколения геймеров, а также одним из главных евангелистов «второй волны» виртуальной реальности. В чём секрет успеха?

Гораздо интереснее то, как работает это устройство и что в нём такого волшебного, чего не удавалось сделать предыдущие 50 лет. Процесс проникновения в виртуальную реальность происходит весьма просто. Вы надеваете на голову шлем, садитесь поудобнее, подключаетесь к внешнему источнику развлечений (компьютеру, конечно) и погружаетесь. Сейчас это можно опробовать только при наличии одного из комплектов для разработчиков, но Oculus активно готовит всех, от производителей игр до обычных пользователей, к началу продаж. Открытая платформа для разработчиков — один из плюсов Oculus. На старте сразу будет во что поиграть.

На деле же Oculus решила и продолжает решать основные проблемы, с которыми сталкивалась виртуальная реальность все предыдущие годы.

1. В 90-х простенькая гарнитура, во многом благодаря дисплеям и системам отслеживания движений, стоила порядка 15 тысяч долларов. Относительно мощную графическую систему можно было приобрести за 50 тысяч долларов. Разумеется, системы виртуальной реальности доставались только самым богатым учреждениям — научным лабораториям и военным для подготовки. Сегодня благодаря невероятному развитию технологий дисплеи стали стоить сущие копейки.
2. В начале 90-х Virtuality Group выпустила мощные гарнитуры виртуальной реальности с невероятным на то время разрешением — 276x372. Последняя версия Oculus Rift может похвастать разрешением 960x1080, а к её стоимости мы ещё вернемся.
3. Команде Палмера Лаки удалось решить проблемы, связанные с качеством восприятия виртуальной реальности. Решены проблемы укачивания (пользователей постоянно тошнило) из-за размытия движений и недостаточного количества кадров в секунду, а также задержки обновления картинки. Теперь тестеры проводят сутки с гарнитурой на голове и чувствуют себя вымотанными, но счастливыми.
4. Колossalное падение стоимости процессоров и общее удешевление и миниатюризация сопутствующих технологий (тех же акселерометра и гироскопа) привели к тому, что Oculus Rift можно будет взять в десятки и даже сотни раз дешевле, чем самую лучшую систему виртуальной реальности ещё десять лет назад.

Повышение качества изображения, отслеживание движений пользователя за счёт внешней камеры, а также, что самое важное, увеличение угла обзора до 110 градусов — всё это стало секретом успеха. Стереоскопический трёхмерный обзор в гарнитуре дарит полноценное периферическое зрение и эффект погружения так, будто вы смотрите на окружающий мир своими родными зенками.

Параллельно с Oculus и другие компании начинают постепенно выходить на рынок виртуальной реальности. Sony (далеко не пионер в области игр) анонсировала свою гарнитуру Morpheus; Samsung (при поддержке инженеров Oculus) представила свой шлем виртуальной реальности Gear VR, в котором в качестве основного дисплея выступает Galaxy Note 4. Никто не удивится, если следующий год станет годом гарнитур виртуальной реальности, которые посыпятся отовсюду так же, как некогда посыпались смартфоны.

### *Неизбежное будущее виртуальной реальности*

В сторону технические детали, давайте лучше поговорим о том, к чему всё это идет. Кому нужна виртуальная реальность? Зачем вообще отгораживаться от мира за выпуклыми линзами? Ответ один на все: это действительно круто.

Как ни крути, виртуальная реальность придёт к нам быстрее, чем мы ожидали, и тренд обещает выстрелить мощнее, чем вышеупомянутые кинотеатры в 3D, санкции или новый iPhone. Геймеры, вооружившись недорогими гарнитурами, обретут второе дыхание и на многие годы пропадут в виртуальной реальности. Люди попроще получат шанс испытать себя гонщиком «Формулы-1», пилотом-истребителя или капитаном «Энтерпрайза». Особые перспективы обещает применение виртуальной реальности в медицине и на поле боя. Обо всём подробнее и понемногу.

### *Симуляторы и архитектура*

Виртуальные тренажёры для подготовки будущих пилотов и операторов АЭС существуют уже давно. Но с развитием Oculus Rift и сопутствующих гарнитур устройств каждый может испытать себя в роли птицы, например. Лётный симулятор для всего тела Birdly, разработанный Максом Райннером, швейцарским художником, буквально превращает людей в птиц. Можно летать над городами, ощущая ветер из вентилятора, который треплет виртуальное оперение. Нет ничего сложного в том, чтобы переместиться из тела птицы в кабину истребителя. Да, авиа- и другие симуляторы давно привлекают геймеров, но, поверьте, в виртуальной реальности всё совсем по-другому.

Другим крайне полезным применением очков виртуальной реальности в будущем станет проектирование и архитектура, наряду с виртуальными экскурсиями и посещениями музеев. Надев очки, человек сможет почувствовать себя в роли творца, даром что ресурсы для его творений будут практически неисчерпаемы — виртуальны. В плане таких применений виртуальной реальности возможности практически безграничны.

### *Медицина*

В медицине крайне необходимы виртуальные операции на виртуальных пациентах. Например, для того, чтобы лучше подготовиться к самому процессу и по возможности предупредить все чрезвычайные ситуации. Хирургическая система da Vinci позволяет хирургу с помощью 3D-камеры увидеть все происходящее в теле пациента и распознать движение рук хирурга, преобразуя их в инструменты внутри тела. С применением виртуальной реальности любая операция, эксперимент или обучение студентов могут многократно приобрести в точности и предсказуемости.

Используется виртуальная реальность и для лечения фобий, реабилитации, облегчения боли и других связанных с восприятием и воспоминаниями терапий. Весьма интересен пример программы SnowWorld, в ходе которой пациенты с тяжёлыми ожогами помещались в виртуальную реальность, где гуляли по заснеженной стране чудес и бросались виртуальными

снежками. Этот терапевтический инструмент реально облегчал боль. Аналогичная программа — SpiderWorld — снижала уровень тревожности при встрече с пауками в процессе лечения арахнофобии. В конце концов, виртуальную среду можно контролировать, и это важно.

### *Слияние реальности и виртуальности*

Цель Палмера Лаки — сделать Oculus Rift повседневным продуктом, без которого не мыслит жизнь ни один уважающий себя человек. Похоже на смартфоны, не так ли? При этом создатель Oculus признаёт, что первыми пользователями гарнитур будут, вероятно, хардкорные геймеры, которым понадобится мощный компьютер. В дальнейшем Oculus Rift смогут обходиться и без необходимости подключения к внешнему источнику, вроде ПК или консоли. Но пока... Пока Oculus даже не планирует продавать свою технику в рознице — только через сайт.

Индустрия игр — это тяжёлый маховик, который раскручивается, чтобы вывести виртуальную реальность в мир. Со временем, считает Лаки, виртуальная реальность выйдет за пределы игр. Те люди, которые занимаются созданием трёхмерных игр сегодня, будут заниматься архитектурой, виртуальными прогулками, фильмами и прочим.

Что дальше? Мир вокруг вас наполнится интерактивными элементами, всплывающими подсказками и рекламой, назойливо всплывающей на каждом углу. Независимо от того, будете вы ходить в шлеме, очках или линзах, основной интерфейс — человеческий глаз — останется неизменным. Большую часть информации мы «видим». Согласитесь, было бы интересно, если бы всё, что мы видим, можно было менять, как по мановению волшебной палочки.

### *Искусственные нейронные сети*

**Искусственная нейронная сеть** — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы.

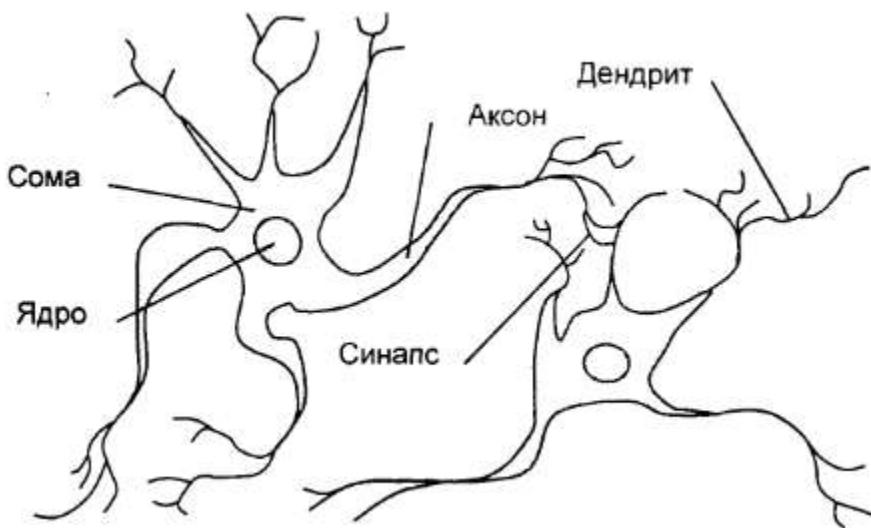
Первые попытки смоделировать процессы, происходящие в человеческом мозге, предприняли Уорен Маккалок и Уолтер Питтс, которые выпустили в 1943 году статью «Логическое исчисление идей, относящихся к нервной активности». Именно они создали первые модели нейронных сетей и искусственных нейронов.

### *Биологические нейронные сети*

Нервная система и мозг человека состоят из нейронов, соединённых между собой нервными волокнами. Нервные волокна способны передавать электрические импульсы между нейронами.

Нейрон является особой биологической клеткой, которая обрабатывает информацию. Он состоит из тела, или сомы, и отростков нервных волокон двух типов — дендритов, по которым принимаются импульсы, и единственного аксона, по которому нейрон может передавать импульс. Тело нейрона включает ядро, которое содержит информацию о наследственных свойствах, и плазму, обладающую молекулярными средствами для производства необходимых нейрону материалов. Нейрон получает сигналы от аксонов других нейронов через дендриты и передаёт сигналы, сгенерированные телом клетки, вдоль своего аксона, который в конце разветвляется на волокна. На окончаниях этих волокон находятся специальные образования — синапсы, которые влияют на величину импульсов.

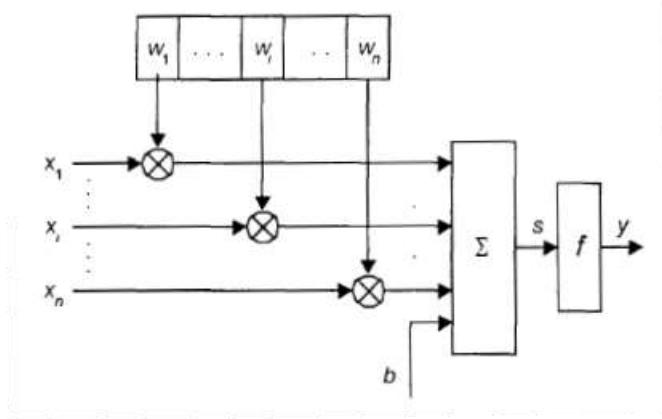
Синапс является функциональным узлом между двумя нейронами. Когда импульс достигает синаптического окончания, высвобождаются химические вещества, называемые нейротрансмиттерами. Нейротрансмиттеры приводят к возбуждению или затормаживанию, в зависимости от типа синапса, способности нейроприёмника генерировать электрические импульсы. Синапсы могут обучаться в зависимости от активности процессов, в которых они участвуют, то есть синапс настраивается проходящими через него сигналами. Важно отметить, что веса синапсов могут изменяться со временем, а значит, меняется и поведение соответствующих нейронов.



Кора головного мозга человека содержит около  $10^{11}$  нейронов и представляет собой протяженную поверхность толщиной от 2 до 3 мм с площадью около  $2200 \text{ см}^2$ . Каждый нейрон связан с  $10^3$ - $10^4$  другими нейронами. В целом мозг человека содержит приблизительно от  $10^{14}$  до  $10^{15}$  взаимосвязей.

### Искусственный нейрон

Искусственный нейрон состоит из элементов трех типов: умножителей (синапсов), сумматора и нелинейного преобразователя. Синапсы осуществляют связь между нейронами, умножают входной сигнал на число, характеризующее силу связи, (вес синапса) Сумматор выполняет сложение сигналов, поступающих по синаптическим связям от других нейронов, и внешних входных сигналов. Нелинейный преобразователь реализует нелинейную функцию одного аргумента - выхода сумматора. Эта функция называется функцией активации или передаточной функцией нейрона.



Нейрон в целом реализует скалярную функцию векторного аргумента. Математическая модель нейрона:

$$S = \sum_{i=1}^n w_i x_i + b,$$

$$y = f(s),$$

где  $w_i$  - вес синапса,  $x_i$  - компоненты входного вектора сигналов,  $b$  - значение смещения,  $S$  - результат суммирования,  $y$  - выходной сигнал нейрона,  $n$  - количество входов нейрона.

В общем случае входной сигнал, весовые коэффициенты и смещение могут принимать действительные значения, а во многих практических задачах - лишь некоторые фиксированные значения. Выход ( $y$ ) определяется видом функции активации и может быть, как действительным, так и целым.

Синаптические связи с положительными весами называют возбуждающими, с отрицательными весами – тормозящими.

Описанный вычислительный элемент можно считать упрощенной математической моделью биологических нейронов. Чтобы подчеркнуть различие нейронов биологических и искусственных, вторые иногда называют нейроноподобными элементами или формальными нейронами.

Вот некоторые примеры активационных функций:

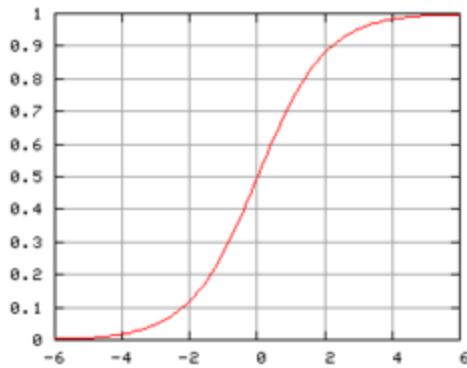
Функции активации нейронов

Название	Формула	Область значений
Линейная	$f(s) = k s$	$(-\infty, \infty)$
Полулинейная	$f(s) = \begin{cases} k s, & s > 0, \\ 0, & s \leq 0 \end{cases}$	$(0, \infty)$
Логистическая (сигмоидальная)	$f(s) = \frac{1}{1 + e^{-as}}$	$(0, 1)$
Гиперболический тангенс (сигмоидальная)	$f(s) = \frac{e^{as} - e^{-as}}{e^{as} + e^{-as}}$	$(-1, 1)$
Экспоненциальная	$f(s) = e^{-as}$	$(0, \infty)$
Синусоидальная	$f(s) = \sin(s)$	$(-1, 1)$
Сигмоидальная (рациональная)	$f(s) = \frac{s}{a +  s }$	$(-1, 1)$
Шаговая (линейная с насыщением)	$f(s) = \begin{cases} -1, & s \leq -1, \\ s, & -1 < s < 1, \\ 1, & s \geq 1 \end{cases}$	$(-1, 1)$
Пороговая	$f(s) = \begin{cases} 0, & s < 0 \\ 1, & s \geq 0 \end{cases}$	$(0, 1)$
Модульная	$f(s) =  s $	$(0, \infty)$
Знаковая (сигнатурная)	$f(s) = \begin{cases} 1, & s > 0, \\ -1, & s \leq 0 \end{cases}$	$(-1, 1)$
Квадратичная	$f(s) = s^2$	$(0, \infty)$

Одной из наиболее распространённых является нелинейная функция активации с насыщением, так называемая логистическая функция или сигмоид (функция S-образного вида):

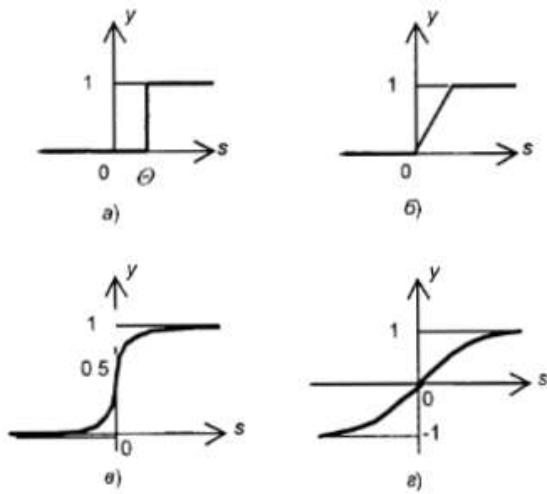
$$f(S) = \frac{1}{1 + e^{-as}}$$

Сигмоида применяется в нейронных сетях в качестве функций активации, так как позволяет как усиливать слабые сигналы, так и не насыщаться от сильных сигналов, так как они соответствуют областям аргументов, где сигмоид имеет пологий наклон.



Гроссберг (1973 год) обнаружил, что подобная нелинейная функция активации решает поставленную им дилемму шумового насыщения:

«Слабые сигналы нуждаются в большом сетевом усилении, чтобы дать пригодный к использованию выходной сигнал. Однако усилительные каскады с большими коэффициентами усиления могут привести к насыщению выхода шумами усилителей, которые присутствуют в любой физически реализованной сети. Сильные входные сигналы в свою очередь также будут приводить к насыщению усилительных каскадов, исключая возможность полезного использования выхода. Таким образом одна и та же сеть может обрабатывать как слабые, так и сильные сигналы?»



а – функция единичного скачка, б – линейный порог (гистерезис),  
в – сигмоид (логистическая функция), г – сигмоид (гиперболический тангенс)

### Классификация нейронных сетей

Нейронная сеть представляет собой совокупность нейроподобных элементов, определённым образом соединённых друг с другом и с внешней средой с помощью связей, определяемых весовыми коэффициентами. В зависимости от функций, выполняемых нейронами в сети, можно выделить три их типа:

- входные нейроны, на которые подается вектор, кодирующий входное воздействие или образ внешней среды; в них обычно не осуществляется вычислительных процедур, а информация передается с входа на выход путем изменения их активации;
- выходные нейроны, выходные значения которых представляют выходы нейронной сети; преобразования в них осуществляются по выражениям;
- промежуточные нейроны, составляющие основу нейронных сетей, преобразования в которых выполняются также по выражениям.

В большинстве нейронных моделей тип нейрона связан с его расположением в сети. Если нейрон имеет только выходные связи, то это входной нейрон, если наоборот – выходной нейрон. Однако возможен случай, когда выход топологически внутреннего нейрона рассматривается как часть выхода сети. В процессе функционирования сети осуществляется преобразование входного вектора в выходной некоторая переработка информации. Конкретный вид выполняемого сетью преобразования данных обусловливается не только характеристиками нейроподобных элементов,

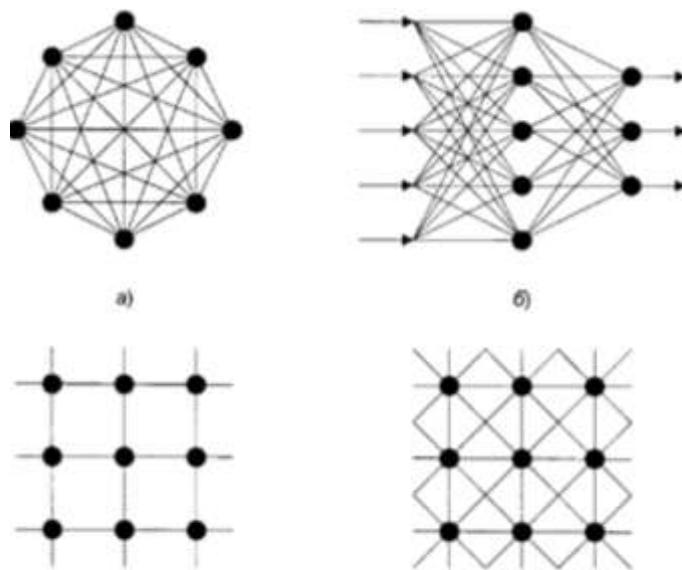
но и особенностями ее архитектуры, а именно топологией межнейронных связей, выбором определенных подмножеств нейроподобных элементов для ввода и вывода информации, способами обучения сети, наличием или отсутствием конкуренции между нейронами, направлением и способами управления и синхронизации передачи информации между нейронами.

С точки зрения топологии можно выделить три основных типа нейронных сетей

- полносвязные;
- многослойные или слоистые;
- слабосвязные (с локальными связями).

В полносвязных нейронных сетях каждый нейрон передает свой выходной сигнал остальным нейронам, в том числе и самому себе. Все входные сигналы подаются всем нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети.

В многослойных нейронных сетях нейроны объединяются в слои. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в слое может быть любым и не зависит от количества нейронов в других слоях. В общем случае сеть состоит из  $Q$  слоев, пронумерованных слева направо. Внешние входные сигналы подаются на входы нейронов входного слоя (его часто нумеруют как нулевой), а выходами сети являются выходные сигналы последнего слоя. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько скрытых слоев. Связи от выходов нейронов некоторого слоя  $q$  к входам нейронов следующего слоя ( $q+1$ ) называются последовательными.



а – полносвязная сеть, б – многослойная сеть с последовательными связями,  
в – слабосвязные сети

В свою очередь среди многослойных нейронных сетей выделяют следующие типы:

- Монотонные

Это частный случай слоистых сетей с дополнительными условиями на связи и нейроны. Каждый слой кроме последнего (выходного) разбит на два блока: возбуждающий и тормозящий. Связи между блоками тоже разделяются на тормозящие и возбуждающие. Если от нейронов блока А к нейронам блока В ведут только возбуждающие связи, то это означает, что любой выходной сигнал блока я является монотонной неубывающей функцией любого выходного сигнала блока А. Если же эти связи только тормозящие, то любой выходной сигнал блока В является невозрастающей функцией любого выходного сигнала блока А. Для нейронов монотонных сетей необходима монотонная зависимость выходного сигнала нейрона от параметров входных сигналов.

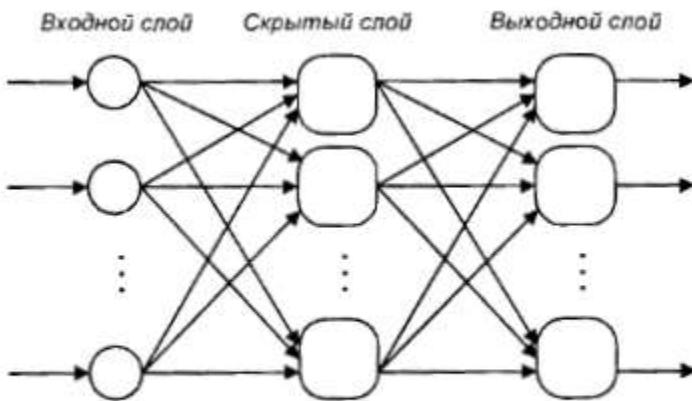
- **Сети без обратных связей**

В таких сетях нейроны входного слоя получают входные сигналы, преобразуют их и передают нейронам первого скрытого слоя, и так далее вплоть до выходного, который выдает сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый выходной сигнал  $q$ -го слоя подается на вход всех нейронов  $(q+1)$ -го слоя; однако возможен вариант соединения  $q$ -го слоя с произвольным  $(q+p)$ -м слоем

Среди многослойных сетей без обратных связей различают полносвязанные (выход каждого нейрона  $q$ -го слоя связан с входом каждого нейрона  $(q+1)$ -го слоя) и частично полносвязанные. Классическим вариантом слоистых сетей являются полносвязанные сети прямого распространения.

- **Сети с обратными связями**

В сетях с обратными связями информация с последующих слоев передается на предыдущие.



Многослойная (двухслойная) сеть прямого распространения

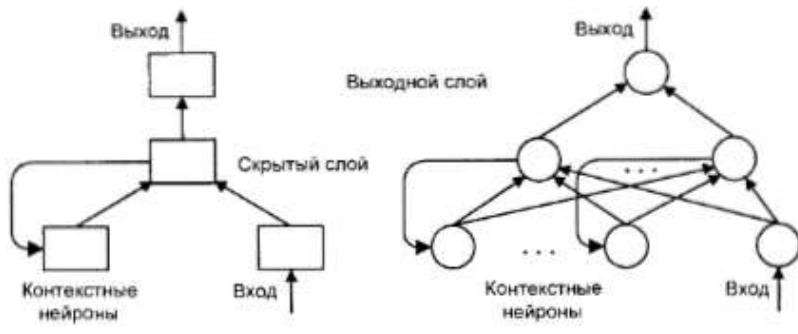
Среди них, в свою очередь, выделяют следующие:

- слоисто-циклические, отличающиеся тем, что слои замкнуты в кольцо, последний слой передает свои выходные сигналы первому; все слои равноправны и могут как получать входные сигналы, так и выдавать выходные;
- слоисто-полносвязанные состоят из слоев, каждый из которых представляет собой полносвязную сеть, а сигналы передаются как от слоя к слою, так и внутри слоя; в каждом

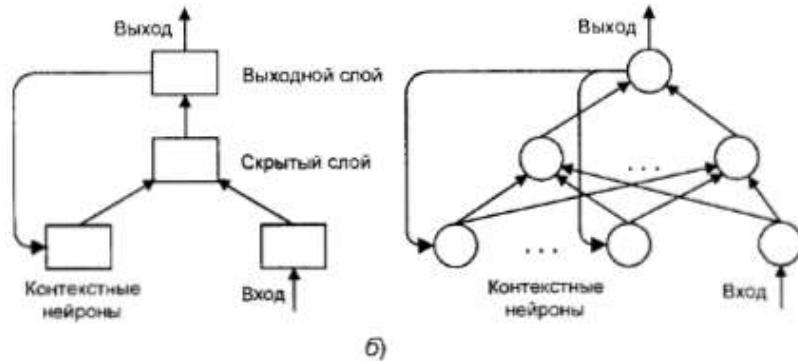
слое цикл работы распадается на три части, прием сигналов с предыдущего слоя, обмен сигналами внутри слоя, выработка выходного сигнала и передача к последующему слою,

- полносвязанно-слоистые, по своей структуре аналогичные слоисто-полносвязанным, но функционирующим по-другому: в них не разделяются фазы обмена внутри слоя и передачи следующему. на каждом такте нейроны всех слоев принимают сигналы от нейронов как своего слоя, так и последующих.

Примером сетей с обратными связями являются частично-рекуррентные сети Элмана и Жордана.



а)



б)

а – Элмана, б – Жордана

В слабосвязных нейронных сетях нейроны располагаются в узлах прямоугольной или гексагональной решетки. Каждый нейрон связан с четырьмя (окрестность фон Неймана), шестью (окрестность Голея) или восемью (окрестность Мура) своими ближайшими соседями.

Известные нейронные сети можно разделить по типам структур нейронов на гомогенные (однородные) и гетерогенные. Гомогенные сети состоят из нейронов одного типа с единой функцией активации, а в гетерогенную сеть входят нейроны с различными функциями активации.

Существуют бинарные и аналоговые сети. Первые из них оперируют только двоичными сигналами, и выход каждого нейрона может принимать значение либо логического ноля (заторможенное состояние), либо логической единицы (возбужденное состояние).

Еще одна классификация делит нейронные сети на синхронные и асинхронные. В первом случае в каждый момент времени лишь один нейрон меняет свое состояние, во втором - состояние меняется

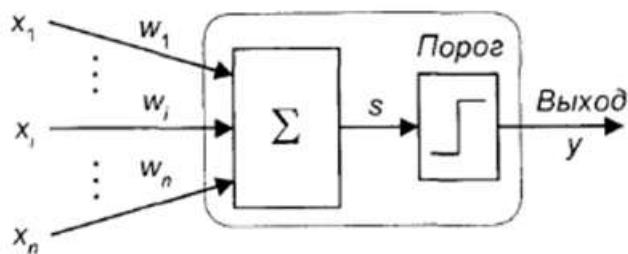
сразу у целой группы нейронов, как правило, у всего слоя. Алгоритмически ход времени в нейронных сетях задается итерационным выполнением однотипных действий над нейронами.

Сети можно классифицировать также по числу слоев. Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированных микросхем, на которых обычно реализуется нейронная сеть. Чем сложнее сеть, тем более сложные задачи она может решать.

### Известные нейронные сети

#### Перцептроны

Перцептрон, или персепtron — математическая или компьютерная модель восприятия информации мозгом (кибернетическая модель мозга), предложенная Фрэнком Розенблаттом в 1957 году и реализованная в виде электронной машины «Марк-1» в 1960 году. Перцептрон стал одной из первых моделей нейросетей, а «Марк-1» — первым в мире нейрокомпьютером. Несмотря на свою простоту, перцептрон способен обучаться и решать довольно сложные задачи. Основная математическая задача, с которой он справляется, — это линейное разделение любых нелинейных множеств, так называемое обеспечение линейной сепарабельности.



Перцептронный нейрон

Суть перцептронного нейрона состоит в том, что сумма, полученная сумматором, сравнивается с каким-то заданным порогом. Если сумма больше этого порога, то выход нейрона равен единице, иначе — нулю. Семейство систем, использующих такие нейроны и называется перцептрами. Они состоят из одного слоя искусственных нейронов, соединенных с помощью весовых коэффициентов с множеством входов, хотя бывают и более сложные системы.

В 60-е годы перцептроны вызвали большой интерес Розенблатт доказал теорему об обучении перцептронов. Уидроу продемонстрировал возможности систем перцептронного типа. Однако дальнейшие исследования показали, что перцептроны не способны обучаться решению ряда простых задач. Марвин Минский строго проанализировал эту проблему и показал, что существуют жесткие ограничения на то, что могут выполнять однослойные перцептроны, и, следовательно, на то, чему они могут обучаться. Так как в то время методы обучения многослойных сетей не были известны, исследования в области нейронных сетей пришли в упадок. Возрождение интереса к нейронным сетям связано в большей степени со сравнительно недавним открытием с таких методов.

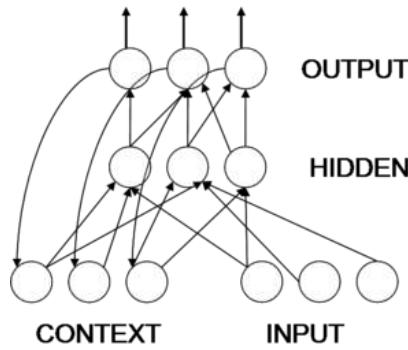
Работа Минского возможно и охладила пыл первых исследователей нейронных сетей, однако обеспечила необходимое время для развития лежащей в их основе теории. Важно отметить, что анализ Минского не был опровергнут и до сих пор остается весьма существенным.

Несмотря на ограничения, персептроны широко изучались. Теория персепtronов является основой для изучения многих других типов искусственных нейронных сетей.

### Нейронная сеть Джордана

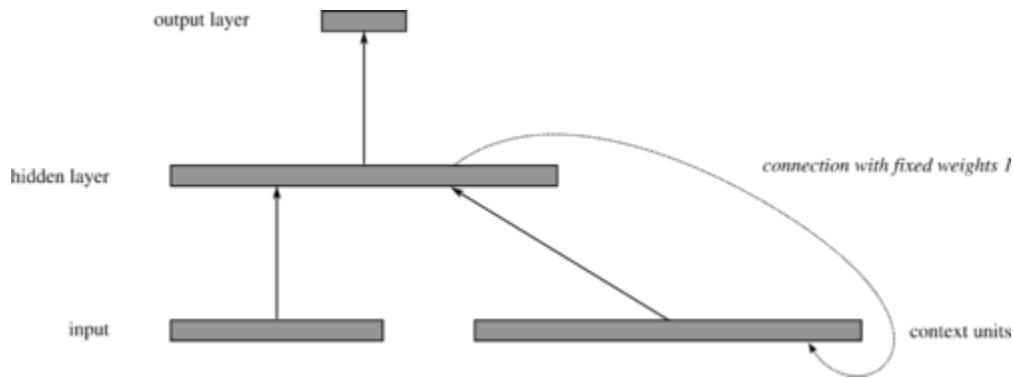
Она является примером сети с обратными связями. Сеть Джордана - вид нейронных сетей, который получается из многослойного перцептрана, если на его вход подать, помимо входного вектора, выходной с задержкой на один или несколько тактов.

В первых рекуррентных сетях главной идеей было дать сети видеть свой выходной образ на предыдущем шаге. У такой сети только часть рецепторов принимает сигналы из окружающего мира, на другие рецепторы приходит выходной образ из предыдущего момента времени. Рассмотрим прохождение последовательности сигналов через сеть. Сигнал поступает на группу рецепторов, соединенных с внешним миром (INPUT) и проходит в скрытый слой (HIDDEN). Преобразованный скрытым слоем сигнал пойдет на выходной слой (OUTPUT) и выйдет из сети, а его копия попадет на задержку. Далее в сеть, на рецепторы, воспринимающие внешние сигналы, поступает второй образ, а на контекстную группу рецепторов (CONTEXT) – выходной образ с предыдущего шага из задержки. Далее со всех рецепторов сигнал пойдет в скрытый слой, затем на выходной.



### Нейронная сеть Элмана

Нейронная сеть Элмана — один из видов рекуррентной сети, которая так же, как и сеть Джордана получается из многослойного перцептрана введением обратных связей, только связи идут не от выхода сети, а от выходов внутренних нейронов. Это позволяет учесть историю наблюдаемых процессов и накопить информацию для выработки правильной стратегии управления. Эти сети могут применяться в системах управления движущимися объектами, так как их главной особенностью является запоминание последовательностей.



### [Нейронная сеть Ворда](#)

Нейронная сеть Ворда — искусственная нейронная сеть, топология которой характеризуется тем, что внутренние (скрытые) слои нейронов разбиты на блоки.

Разбиение скрытых слоев на блоки позволяет использовать различные передаточные функции для различных блоков скрытого слоя. Таким образом, одни и те же сигналы, полученные от входного слоя, взвешиваются и обрабатываются параллельно с использованием нескольких способов, а полученный результат затем обрабатывается нейронами выходного слоя. Применение различных методов обработки для одного и того же набора данных позволяет сказать, что нейронная сеть анализирует данные с различных аспектов. Практика показывает, что сеть показывает очень хорошие результаты при решении задач прогнозирования и распознавания образов. Для нейронов входного слоя, как правило, устанавливается линейная функция активации. Функция активации для нейронов из блоков скрытого и выходного слоя определяется экспериментально.

### [Нейронная сеть Кохонена](#)

Нейронные сети Кохонена — класс нейронных сетей, основным элементом которых является слой Кохонена. Слой Кохонена состоит из адаптивных линейных сумматоров («линейных формальных нейронов»). Как правило, выходные сигналы слоя Кохонена обрабатываются по правилу «Победитель получает всё»: наибольший сигнал превращается в единичный, остальные обращаются в ноль.

### [Нейронная сеть Хэмминга](#)

Сеть состоит из двух слоев. Первый и второй слои имеют по  $m$  нейронов, где  $m$  – число образцов. Нейроны первого слоя имеют по  $n$  синапсов, соединенных со входами сети (образующими фиктивный нулевой слой). Нейроны второго слоя связаны между собой ингибиторными (отрицательными обратными) синаптическими связями. Единственный синапс с положительной обратной связью для каждого нейрона соединен с его же аксоном.

Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образцов. Расстоянием Хэмминга называется число отличающихся битов в двух бинарных векторах. Сеть должна выбрать образец с минимальным расстоянием Хэмминга до неизвестного входного сигнала, в результате чего будет активирован только один выход сети, соответствующий этому образцу.

## Подбор данных и характеристик сети

### Сбор данных для обучения

Выбор данных для обучения сети и их обработка является самым сложным этапом решения задачи. Набор данных для обучения должен удовлетворять нескольким критериям:

- Репрезентативность — данные должны иллюстрировать истинное положение вещей в предметной области;
- Непротиворечивость — противоречивые данные в обучающей выборке приведут к плохому качеству обучения сети.

Исходные данные преобразуются к виду, в котором их можно подать на входы сети. Каждая запись в файле данных называется обучающей парой или обучающим вектором. Обучающий вектор содержит по одному значению на каждый вход сети и, в зависимости от типа обучения (с учителем или без), по одному значению для каждого выхода сети. Обучение сети на «сыром» наборе, как правило, не даёт качественных результатов. Существует ряд способов улучшить «восприятие» сети.

- Нормировка выполняется, когда на различные входы подаются данные разной размерности. Например, на первый вход сети подаются величины со значениями от нуля до единицы, а на второй — от ста до тысячи. При отсутствии нормировки значения на втором входе будут всегда оказывать существенно большее влияние на выход сети, чем значения на первом входе. При нормировке размерности всех входных и выходных данных сводятся воедино;
- Квантование выполняется над непрерывными величинами, для которых выделяется конечный набор дискретных значений. Например, квантование используют для задания частот звуковых сигналов при распознавании речи;
- Фильтрация выполняется для «зашумленных» данных.

Кроме того, большую роль играет само представление как входных, так и выходных данных. Предположим, сеть обучается распознаванию букв на изображениях и имеет один числовой выход — номер буквы в алфавите. В этом случае сеть получит ложное представление о том, что буквы с номерами 1 и 2 более похожи, чем буквы с номерами 1 и 3, что, в общем, неверно. Для того, чтобы избежать такой ситуации, используют топологию сети с большим числом выходов, когда каждый выход имеет свой смысл. Чем больше выходов в сети, тем большее расстояние между классами и тем сложнее их спутать.

### Выбор топологии сети

Выбирать тип сети следует, исходя из постановки задачи и имеющихся данных для обучения. Для обучения с учителем требуется наличие для каждого элемента выборки «экспертной» оценки. Иногда получение такой оценки для большого массива данных просто невозможно. В этих случаях естественным выбором является сеть, обучающаяся без учителя (например, самоорганизующаяся карта Кохонена или нейронная сеть Хопфилда). При решении других задач (таких, как прогнозирование временных рядов) экспертная оценка уже содержится в исходных данных и может быть выделена при их обработке. В этом случае можно использовать многослойный перцептрон или сеть Ворда.

### Экспериментальный подбор характеристик сети

После выбора общей структуры нужно экспериментально подобрать параметры сети. Для сетей, подобных перцептрону, это будет число слоев, число блоков в скрытых слоях (для сетей Ворда), наличие или отсутствие обходных соединений, передаточные функции нейронов. При выборе количества слоев и нейронов в них следует исходить из того, что способности сети к обобщению тем выше, чем больше суммарное число связей между нейронами. С другой стороны, число связей ограничено сверху количеством записей в обучающих данных.

### Экспериментальный подбор параметров обучения

После выбора конкретной топологии необходимо выбрать параметры обучения нейронной сети. Этот этап особенно важен для сетей, обучающихся с учителем. От правильного выбора параметров зависит не только то, насколько быстро ответы сети будут сходиться к правильным ответам. Например, выбор низкой скорости обучения увеличит время схождения, однако иногда позволяет избежать паралича сети. Увеличение момента обучения может привести как к увеличению, так и к уменьшению времени сходимости, в зависимости от формы поверхности ошибки. Исходя из такого противоречивого влияния параметров, можно сделать вывод, что их значения нужно выбирать экспериментально, руководствуясь при этом критерием завершения обучения (например, минимизация ошибки или ограничение по времени обучения).

Вопрос о необходимых и достаточных свойствах сети для решения задач того или иного рода представляет собой целое направление нейрокомпьютерной науки. Так как проблема синтеза нейронной сети сильно зависит от решаемой задачи, дать общие подробные рекомендации затруднительно. В большинстве случаев оптимальный вариант получается на основе интуитивного подбора, хотя в литературе приведены доказательства того, что для любого алгоритма существует нейронная сеть, которая может его реализовать.

### Обучение нейронных сетей

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и/или «зашумленных», частично искаженных данных.

Очевидно, что процесс функционирования нейронной сети, сущность действий, которые она способна выполнять, зависит от величин синаптических связей. Поэтому, задавшись определённой структурой сети, соответствующей какой-либо задаче, необходимо найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными).

Этот этап называется обучением нейронной сети, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время функционирования.

Необходимо выбрать параметры обучения нейронной сети. Этот этап особенно важен для сетей, обучающихся с учителем. От правильного выбора параметров зависит не только то, насколько быстро ответы сети будут сходиться к правильным ответам. Например, выбор низкой скорости обучения увеличит время схождения, однако иногда позволяет избежать паралича сети. Увеличение момента обучения может привести как к увеличению, так и к уменьшению времени сходимости, в зависимости от формы поверхности ошибки. Исходя из такого противоречивого влияния параметров, можно сделать вывод, что их значения нужно выбирать экспериментально, руководствуясь при этом критерием завершения обучения (например, минимизация ошибки или ограничение по времени обучения).

В процессе обучения сеть в определенном порядке просматривает обучающую выборку. Порядок просмотра может быть последовательным, случайным и т. д. Некоторые сети, обучающиеся без учителя (например, сети Хопфилда), просматривают выборку только один раз. Другие (например, сети Кохонена), а также сети, обучающиеся с учителем, просматривают выборку множество раз, при этом один полный проход по выборке называется эпохой обучения. При обучении с учителем набор исходных данных делят на две части — собственно обучающую выборку и тестовые данные; принцип разделения может быть произвольным. Обучающие данные подаются сети для обучения, а проверочные используются для расчета ошибки сети (проверочные данные никогда для обучения сети не применяются). Таким образом, если на проверочных данных ошибка уменьшается, то сеть действительно выполняет обобщение. Если ошибка на обучающих данных продолжает уменьшаться, а ошибка на тестовых данных увеличивается, значит, сеть перестала выполнять обобщение и просто «запоминает» обучающие данные. Это явление называется переобучением сети или оверфиттингом. В таких случаях обучение обычно прекращают. В процессе обучения могут проявиться другие проблемы, такие как паралич или попадание сети в локальный минимум поверхности ошибок. Невозможно заранее предсказать проявление той или иной проблемы, равно как и дать однозначные рекомендации к их разрешению.

Все выше сказанное относится только к итерационным алгоритмам поиска нейросетевых решений. Для них действительно нельзя ничего гарантировать и нельзя полностью автоматизировать обучение нейронных сетей. Однако, наряду с итерационными алгоритмами обучения, существуют не итерационные алгоритмы, обладающие очень высокой устойчивостью и позволяющие полностью автоматизировать процесс обучения.

Таким образом, при обучении с учителем испытуемая система принудительно обучается с помощью примеров «стимул-реакция». Между входами и эталонными выходами (стимул-реакция) может существовать некоторая зависимость, но она неизвестна. Известна только конечная совокупность прецедентов — пар «стимул-реакция», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость (построить модель отношений стимул-реакция, пригодных для прогнозирования), то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ.

При данном типе обучения экспериментальная система состоит из испытываемой (используемой) системы, пространства стимулов, получаемых из внешней среды и системы управления подкреплением (регулятора внутренних параметров). В качестве системы управления подкреплением может быть использовано автоматическое регулирующее устройство (например, терmostат) или человек-оператор (учитель), способный реагировать на реакции испытываемой

системы и стимулы внешней среды путем применения особых правил подкрепления, изменяющих состояние памяти системы.

Различают два варианта: (1) когда реакция испытываемой системы не изменяет состояние внешней среды, и (2) когда реакция системы изменяет стимулы внешней среды. Эти схемы указывают принципиальное сходство такой системы общего вида с биологической нервной системой.

В то же время, при обучении без учителя испытуемая система спонтанно обучается выполнять поставленную задачу, без вмешательства со стороны экспериментатора. Как правило, это пригодно только для задач, в которых известны описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами.

В многослойных нейронных сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, неизвестны. Трех- или более слойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах сети.

Один из вариантов решения этой проблемы - разработка наборов выходных сигналов, соответствующих входным, для каждого слоя нейронной сети, что, конечно, является очень трудоемкой операцией и не всегда осуществимо. Второй вариант - динамическая подстройка весовых коэффициентов синапсов, в ходе которой выбираются, как правило, наиболее слабые связи и изменяются на малую величину в ту или иную сторону, а сохраняются только те изменения, которые повлекли уменьшение ошибки на выходе всей сети. Очевидно, что данный метод, несмотря на кажущуюся простоту, требует громоздких рутинных вычислений. И, наконец, третий, более приемлемый вариант - распространение сигналов ошибки от выходов нейронной сети к ее входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения получил название процедуры обратного распространения ошибки.

Обучение сети методом обратного распространения ошибки включает в себя три этапа: подачу на вход данных, с последующим распространением данных в направлении выходов, вычисление и обратное распространение соответствующей ошибки и корректировку весов. После обучения предполагается лишь подача на вход сети данных и распространение их в направлении выходов. При этом, если обучение сети может являться довольно длительным процессом, то непосредственное вычисление результатов обученной сетью происходит очень быстро. Кроме того, существуют многочисленные вариации метода обратного распространения ошибки, разработанные с целью увеличения скорости протекания процесса обучения.

### Оценка правильности обучения

Даже в случае успешного, на первый взгляд, обучения сеть не всегда обучается именно тому, чего от неё хотел создатель. Известен случай, когда сеть обучалась распознаванию изображений танков по фотографиям, однако позднее выяснилось, что все танки были сфотографированы на одном и том же фоне. В результате сеть «научилась» распознавать этот тип ландшафта, вместо того, чтобы «научиться» распознавать танки. Таким образом, сеть «понимает» не то, что от неё требовалось, а то, что проще всего обобщить.

Тестирование качества обучения нейросети необходимо проводить на примерах, которые не участвовали в её обучении. При этом число тестовых примеров должно быть тем больше, чем выше

качество обучения. Если ошибки нейронной сети имеют вероятность близкую к одной миллиардной, то и для подтверждения этой вероятности нужен миллиард тестовых примеров. Получается, что тестирование хорошо обученных нейронных сетей становится очень трудной задачей.

Существует ещё один интересный способ определения правильности обучения нейронной сети. Допустим, нейронную сеть учат распознавать изображения, показывая ей изображения различных предметов. Сеть находит общие признаки в изображениях и учится находить их в других изображениях. Чтобы проверить, какие именно признаки выделяет сеть, можно провести «обратный» эксперимент: загрузить картинку, состоящую из «шума», и запросить уже обученную сеть найти в ней признаки того или иного предмета, а затем дополнительно выделить их. Так, в одном из случаев по запросу «гантели» стало очевидно, что нейронная сеть практически всегда выделяет не просто гантели, но и руки, которые их держат, — скорее всего, на всех «увиденных» изображениях гантели держали в руках.

Этот процесс можно провести и с обычными фотографиями, на которых изображены пейзажи, предметы и животные. В этом случае программы не дают запросов на поиск какого-либо предмета, а просто просят найти знакомые и выделить их. Нейронные сети действуют в десятках слоев: один может отыскать на изображении углы, другой — края, следующий — собрать детали изображения с объектом вроде здания, дерева или головы животного.

Наконец, для создания самых сложных картин специалисты сначала загружали в нейронную сеть картинку, затем — сгенерированное ей изображение и повторяли этот процесс несколько раз, каждый раз увеличивая масштаб. При этом интересные изображения можно получить даже в том случае, если на самой первой картинке был случайный шум.

## Сфера применения

### Распознавание образов и классификация

В качестве образов могут выступать различные по своей природе объекты: символы текста, изображения, образцы звуков и т. д. При обучении сети предлагаются различные образцы образов с указанием того, к какому классу они относятся. Образец, как правило, представляется как вектор значений признаков. При этом совокупность всех признаков должна однозначно определять класс, к которому относится образец. В случае, если признаков недостаточно, сеть может соотнести один и тот же образец с несколькими классами, что неверно. По окончании обучения сети ей можно предъявлять неизвестные ранее образы и получать ответ о принадлежности к определённому классу.

Топология такой сети характеризуется тем, что количество нейронов в выходном слое, как правило, равно количеству определяемых классов. При этом устанавливается соответствие между выходом нейронной сети и классом, который он представляет. Когда сети предъявляется некий образ, на одном из её выходов должен появиться признак того, что образ принадлежит этому классу. В то же время на других выходах должен быть признак того, что образ данному классу не принадлежит. Если на двух или более выходах есть признак принадлежности к классу, считается, что сеть «не уверена» в своём ответе.

## Принятие решений и управление

Эта задача близка к задаче классификации. Классификации подлежат ситуации, характеристики которых поступают на вход нейронной сети. На выходе сети при этом должен появиться признак решения, которое она приняла. При этом в качестве входных сигналов используются различные критерии описания состояния управляемой системы.

## Кластеризация

Под кластеризацией понимается разбиение множества входных сигналов на классы, при том, что ни количество, ни признаки классов заранее не известны. После обучения такая сеть способна определять, к какому классу относится входной сигнал. Сеть также может сигнализировать о том, что входной сигнал не относится ни к одному из выделенных классов — это является признаком новых, отсутствующих в обучающей выборке, данных. Таким образом, подобная сеть может выявлять новые, неизвестные ранее классы сигналов. Соответствие между классами, выделенными сетью, и классами, существующими в предметной области, устанавливается человеком. Кластеризацию осуществляют, например, нейронные сети Кохонена.

Нейронные сети в простом варианте Кохонена не могут быть огромными, поэтому их делят на гиперслои (гиперколонки) и ядра (микроколонки). Если сравнивать с мозгом человека, то идеальное количество параллельных слоёв не должно быть более 112. Эти слои в свою очередь составляют гиперслои (гиперколонку), в которой от 500 до 2000 микроколонок (ядер). При этом каждый слой делится на множество гиперколонок, пронизывающих насквозь эти слои. Микроколонки кодируются цифрами и единицами с получением результата на выходе. Если требуется, то лишние слои и нейроны удаляются или добавляются. Идеально для подбора числа нейронов и слоёв использовать суперкомпьютер. Такая система позволяет нейронным сетям быть пластичными.

## Прогнозирование

Способности нейронной сети к прогнозированию напрямую следуют из её способности к обобщению и выделению скрытых зависимостей между входными и выходными данными. После обучения сеть способна предсказать будущее значение некой последовательности на основе нескольких предыдущих значений и (или) каких-то существующих в настоящий момент факторов. Следует отметить, что прогнозирование возможно только тогда, когда предыдущие изменения действительно в какой-то степени предопределяют будущие. Например, прогнозирование котировок акций на основе котировок за прошлую неделю может оказаться успешным (а может и не оказаться), тогда как прогнозирование результатов завтрашней лотереи на основе данных за последние 50 лет почти наверняка не даст никаких результатов.

## Аппроксимация

Нейронные сети могут аппроксимировать непрерывные функции. Доказана обобщённая аппроксимационная теорема: с помощью линейных операций и каскадного соединения можно из произвольного нелинейного элемента получить устройство, вычисляющее любую непрерывную функцию с некоторой наперёд заданной точностью. Это означает, что нелинейная характеристика нейрона может быть произвольной: от сигмоидальной до произвольного волнового пакета или вейвлета, синуса или многочлена. От выбора нелинейной функции может зависеть сложность конкретной сети, но с любой нелинейностью сеть остаётся универсальным аппроксиматором и при

правильном выборе структуры может достаточно точно аппроксимировать функционирование любого непрерывного автомата.

#### [Сжатие данных и Ассоциативная память](#)

Способность нейросетей к выявлению взаимосвязей между различными параметрами дает возможность выразить данные большой размерности более компактно, если данные тесно взаимосвязаны друг с другом. Обратный процесс — восстановление исходного набора данных из части информации — называется (авто)ассоциативной памятью. Ассоциативная память позволяет также восстанавливать исходный сигнал/образ из зашумленных/поврежденных входных данных. Решение задачи гетероассоциативной памяти позволяет реализовать память, адресуемую по содержимому.

#### [Предсказание финансовых временных рядов](#)

Входные данные — курс акций за год. Задача — определить завтрашний курс. Проводится следующее преобразование — выстраивается в ряд курс за сегодня, вчера, за позавчера. Следующий ряд — смещается по дате на один день и так далее. На полученном наборе обучается сеть с 3 входами и одним выходом — то есть выход: курс на дату, входы: курс на дату минус 1 день, минус 2 дня, минус 3 дня. Обученной сети подаем на вход курс за сегодня, вчера, позавчера и получаем ответ на завтра. Нетрудно заметить, что в этом случае сеть просто выведет зависимость одного параметра от трёх предыдущих. Если желательно учитывать ещё какой-то параметр (например, общий индекс по отрасли), то его надо добавить как вход (и включить в примеры), переобучить сеть и получить новые результаты. Для наиболее точного обучения стоит использовать метод ОРО, как наиболее предсказуемый и несложный в реализации.

#### [Психодиагностика](#)

Серия работ М. Г. Доррера с соавторами посвящена исследованию вопроса о возможности развития психологической интуиции у нейросетевых экспертных систем. Полученные результаты дают подход к раскрытию механизма интуиции нейронных сетей, проявляющейся при решении ими психодиагностических задач. Создан нестандартный для компьютерных методик интуитивный подход к психодиагностике, заключающийся в исключении построения описанной реальности. Он позволяет сократить и упростить работу над психодиагностическими методиками.

#### [Хемоинформатика](#)

Нейронные сети широко используются в химических и биохимических исследованиях. В настоящее время нейронные сети являются одним из самых распространенных методов хемоинформатики для поиска количественных соотношений структура-свойство, благодаря чему они активно используются как для прогнозирования физико-химических свойств и биологической активности химических соединений, так и для направленного дизайна химических соединений и материалов с заранее заданными свойствами, в том числе при разработке новых лекарственных препаратов.

#### [Нейроуправление](#)

Нейронные сети успешно применяются для синтеза систем управления динамическими объектами. Нейросети обладают рядом уникальных свойств, которые делают их мощным инструментом для создания систем управления: способностью к обучению на примерах и обобщению данных, способностью адаптироваться к изменению свойств объекта управления и внешней среды, пригодностью для синтеза нелинейных регуляторов, высокой устойчивостью к повреждениям своих элементов в силу изначально заложенного в нейросетевую архитектуру параллелизма.

## Экономика

Алгоритмы искусственных нейронных сетей нашли широкое применение в экономике. С помощью нейронных сетей решается задача разработки алгоритмов нахождения аналитического описания закономерностей функционирования экономических объектов (предприятие, отрасль, регион). Эти алгоритмы применяются к прогнозированию некоторых «выходных» показателей объектов. Применение нейросетевых методов позволяет решить некоторые проблемы экономико-статистического моделирования, повысить адекватность математических моделей, приблизить их к экономической реальности. Поскольку экономические, финансовые и социальные системы очень сложны и являются результатом человеческих действий и противодействий, создание полной математической модели с учётом всех возможных действий и противодействий является очень сложной (если разрешимой) задачей. В системах подобной сложности естественным и наиболее эффективным является использование моделей, которые напрямую имитируют поведение общества и экономики. Именно это способна предложить методология нейронных сетей.

## Педагогическое прогнозирование

Повседневная педагогическая жизнь изобилует ситуациями, когда какой-либо цели требуется достичь максимально эффективным способом. В условиях постоянной смены информации, большого количества обрабатываемых данных (фактов и причин), а иногда и вследствие информационного хаоса, обучаемые не всегда могут самостоятельно и до конца адекватно структурировать, оценивать и анализировать учебно-воспитательную информацию, которая к ним поступает, что с точки зрения классических педагогических теорий приводит к непонятному поведению, и как следствие – неправильному прогнозу. В этом случае на помощь приходят нейронные сети, которые работают на идеях искусственного интеллекта. Примерами компьютерных нейронных систем являются программные продукты: STATISTICA Neural Networks, BrainMaker, NeuroShell, OWL (HyperLogic), Neuro Builder.

## Роботы

**Aibo** — собака-робот, разработанная компанией Sony. Она имеет множество модификаций, первая модель была выпущена в 1999 году. AIBO умеет ходить, «видеть» окружающие его предметы с помощью видеокамеры и инфракрасных датчиков расстояния, распознавать команды и лица. Робот является полностью автономным: он может учиться и развиваться, основываясь на побуждениях своего хозяина, обстановки, или другого AIBO. Несмотря на это, он поддаётся настройкам с помощью специальных программ. Существует программное обеспечение, имитирующее «взрослую собаку», которая сразу использует все свои функции и программное обеспечение имитирующее «щенка», который раскрывает свои возможности постепенно.

«Настроение» AIBO может меняться в зависимости от окружающей обстановки, и влиять на поведение. Инстинкты позволяют AIBO двигаться, играть с его игрушками, удовлетворять своё любопытство, играть и общаться с хозяином, самостоятельно подзаряжаться и просыпаться после сна. Разработчики утверждают, что у AIBO есть симулирование шести эмоций: счастье, грусть, страх, антипатия, удивление, и гнев.

**NAO** — это автономный программируемый человекоподобный робот, разработанный компанией Aldebaran Robotics, штаб-квартира которой находится во Франции (Париж). Разработка робота началась с запуска Проекта NAO (Project NAO) в 2004 году. 15 августа 2007 года, робот NAO заменил робота-собаку Aibo компании Sony в международном соревновании по робофутболу RoboCup

Standard Platform League (SPL). Также робот NAO принимал участие в соревновании RoboCup 2008 и 2009 года и был выбран в качестве базовой платформы для SPL на RoboCup 2010. Учебная версия NAO была разработана для университетов и лабораторий для исследования и обучения. Для институтов эта версия была реализована в 2008, а позднее (к 2011) стала доступна для большинства. NAO вскоре начал использоваться в различных университетах всего мира, таких как Токийский Университет, Индийский технологический институт Канпур в Индии, Alfaaisal в Саудовской Аравии. В декабре 2011 года Aldebaran Robotics произвела модель Nao Next Gen с расширением программного обеспечения и более мощным процессором, а также с камерами более высокого разрешения (HD).

### [Нейрокомпьютеры](#)

**Нейрокомпьютер** — устройство переработки информации на основе принципов работы естественных нейронных систем.

В отличие от цифровых систем, представляющих собой комбинации процессорных и запоминающих блоков, нейропроцессоры содержат память, распределённую в связях между очень простыми процессорами, которые часто могут быть описаны как формальные нейроны или блоки из однотипных формальных нейронов. Тем самым основная нагрузка на выполнение конкретных функций процессорами ложится на архитектуру системы, детали которой в свою очередь определяются межнейронными связями.

Три основных преимущества нейрокомпьютеров:

- Все алгоритмы нейроинформатики высокопараллельны, а это уже залог высокого быстродействия.
- Нейросистемы можно легко сделать очень устойчивыми к помехам и разрушениям.
- Устойчивые и надёжные нейросистемы могут создаваться и из ненадёжных элементов, имеющих значительный разброс параметров.

На роль центральной проблемы, решаемой всей нейроинформатикой и нейрокомпьютингом, А. Горбань предложил проблему эффективного параллелизма. Давно известно, что производительность компьютера возрастает намного медленнее, чем число процессоров. М. Минский сформулировал гипотезу: производительность параллельной системы растёт (примерно) пропорционально логарифму числа процессоров — это намного медленнее, чем линейная функция (Гипотеза Минского).

Для преодоления этого ограничения применяется следующий подход: для различных классов задач строятся максимально параллельные алгоритмы решения, использующие какую-либо абстрактную архитектуру (парадигму) мелкозернистого параллелизма, а для конкретных параллельных компьютеров создаются средства реализации параллельных процессов заданной абстрактной архитектуры. В результате появляется эффективный аппарат производства параллельных программ.

Нейроинформатика поставляет универсальные мелкозернистые параллельные архитектуры для решения различных классов задач. Для конкретных задач строится абстрактная нейросетевая реализация алгоритма решения, которая затем реализуется на конкретных параллельных

вычислительных устройствах. Таким образом нейросети позволяют эффективно использовать параллелизм.

**IBM Watson** — суперкомпьютер фирмы IBM, оснащённый вопросно-ответной системой искусственного интеллекта, созданный группой исследователей под руководством Дэвида Феруччи. Его создание — часть проекта DeepQA. Основная задача Уотсона — понимать вопросы, сформулированные на естественном языке, и находить на них ответы в базе данных. Назван в честь основателя IBM Томаса Уотсона.

В феврале 2011 года суперкомпьютер принял участие в телепередаче *Jeopardy!* (российский аналог — «Своя игра»). Его соперниками были Брэд Раттер — обладатель самого большого выигрыша в программе, и Кен Дженнингс — рекордсмен по длительности беспрогрышной серии. Компьютер одержал победу, получив 1 млн долларов, в то время, как Дженнингс и Раттер получили, соответственно, по 300 и 200 тысяч.

Watson состоит из 90 серверов IBM p750, каждый из которых оснащён четырьмя восьмиядерными процессорами архитектуры POWER7. Суммарная оперативная память — более 15 терабайт.

Система имела доступ к 200 млн страниц структурированной и неструктурированной информации объёмом в 4 терабайта, включая полный текст Википедии. Во время игры Watson не имел доступа к интернету.

Разработчики суперкомпьютера IBM Watson сделали большую ошибку, когда закачали в его память словарь уличного жаргона с сайта Urban Dictionary. IBM Watson обучен распознавать смысл предложений/вопросов и отвечать на них, используя усвоенные массивы неструктурированных данных (data mining).

Разработчики думали, что информация о дополнительных смыслах слов улучшит искусственный интеллект, поможет ему лучше понимать людей.

Они ошиблись. Уличный жаргон оказался исключительно вреден суперкомпьютеру. Однажды он даже ответил на вопрос неприличным словом “Bullshit”.

В результате, разработчики приняли решение очистить память IBM Watson от сленга и поставить фильтр на выдаваемые слова, чтобы не допустить случайно ненормативной лексики в прямом эфире.

IBM совместно с Nuance Communications (производителем средств распознавания речи) планирует в ближайшие два года разработать продукт, направленный на помощь в диагностировании и лечении пациентов. Также рассматриваются возможности использования в других сферах, таких как оценка политик страхования или эффективности энергопотребления.

Представители компаний IBM и WellPoint провели совместную конференцию, на которой объявили о начале коммерческой эксплуатации медицинской системы IBM Watson. Шесть экземпляров IBM Watson уже «трудоустроены» в больницы США в качестве врачей-диагностов.

За два года учёбы Watson изучил 605 тыс. медицинских документов, в общей сложности 2 миллиона страниц текста. Перед началом врачебной практики компьютер проанализировал 25 тыс. историй болезни и проработал 14,7 тыс. для тонкой настройки алгоритмов.

## Интерпретация данных

большую роль играет представление как входных, так и выходных данных. Многие задачи распознавания образов (зрительных, речевых), выполнения функциональных преобразований при обработке сигналов, управления, прогнозирования, идентификации сложных систем, сводятся к следующей математической постановке. Необходимо построить такое отображение  $X \rightarrow Y$ , чтобы на каждый возможный входной сигнал  $X$  формировался правильный выходной сигнал  $Y$ . Отображение задается конечным набором пар ( $<\text{вход}, \text{известный выход}>$ ). Число этих пар (обучающих примеров) существенно меньше общего числа возможных сочетаний значений входных и выходных сигналов. Совокупность всех обучающих примеров носит название обучающей выборки.

В задачах распознавания образов  $X$  - некоторое представление образа (изображение, вектор),  $Y$  - номер класса, к которому принадлежит входной образ.

В задачах управления  $X$  - набор контролируемых параметров управляемого объекта.  $Y$  - код, определяющий управляющее воздействие, соответствующее текущим значениям контролируемых параметров.

В задачах прогнозирования в качестве входных сигналов используются временные ряды, представляющие значения контролируемых переменных на некотором интервале времени. Выходной сигнал - множество переменных, которое является подмножеством переменных входного сигнала.

При идентификации  $X$  и  $Y$  представляют входные и выходные сигналы системы соответственно.

## Что нас ждёт?

В настоящее время искусственные нейронные сети являются важным расширением понятия вычисления. Они уже позволили справиться с рядом непростых проблем и обещают создание новых программ и устройств, способных решать задачи, которые пока под силу только человеку.

Множество надежд в отношении нейронных сетей сегодня связывают именно с аппаратными реализациями, но пока время их массового выхода на рынок, видимо, еще не пришло. Они или выпускаются в составе специализированных устройств, или достаточно дороги, а зачастую и то, и другое. На их разработку тратится значительное время, за которое программные реализации на самых последних компьютерах оказываются лишь на порядок менее производительными, что делает использование нейропроцессоров нерентабельным. Но все это только вопрос времени — нейронным сетям предстоит пройти тот же путь, по которому еще совсем недавно развивались компьютеры, увеличивая свои возможности и производительность, захватывая новые сферы применения по мере возникновения новых задач и развития технической основы для их разработки.

Это и понятно, так как предпосылки для появления компьютеров тоже накапливались постепенно: механические калькуляторы были созданы еще во времена Паскаля, теория универсальных вычислений была разработана в 30-х годах Аланом Тьюрингом, а лампы и развитие радиоэлектроники подготовили создание элементной базы для первых ЭВМ. Вторая мировая война поставила задачу расчета баллистических траекторий, для решения которой понадобились мощные

калькуляторы, роль которых сыграли ЭВМ 40-х годов, производящие вычисления по разработанному алгоритму, много раз повторяющему одну и ту же последовательность операций.

Переход к транзисторам, а затем к интегральным схемам делал компьютеры все более дешевыми и доступными. Они перестали использоваться как простые вычислители, им стали поручать более интеллектуальные задачи: работу с документами, обработку и анализ данных. Соответственно развивался и интерфейс взаимодействия пользователей и компьютеров, который с момента появления первых ЭВМ был узким местом этих устройств, существенно снижающим эффективность работы с ними. Компьютеры не могли читать, понимать речь, распознавать другую образную информацию: их основным языком были буквы и цифры. Поэтому сначала человеку пришлось учить язык компьютера и программировать в двоичных и машинных кодах, но впоследствии компьютер начал учить язык человека. Тумблеры-переключатели, а затем командная строка превратились в графические интуитивно понятные интерфейсы, а теперь уже речь идет о системах, которые будут в состоянии общаться с человеком на одном языке. Скорее всего, эта задача будет возложена на будущие операционные системы, которые станут заниматься не только распознаванием образов, но и интеллектуальной фильтрацией и поиском информации с учетом интересов пользователя. И, конечно, для решения этих задач будут использоваться нейронные сети, реализованные программно или аппаратно.

Но все это, естественно, дело отдаленного будущего. Сегодня же нейронные сети используются для работы в относительно узких областях, и неизвестно, доверят ли им когда-нибудь решение вопросов, которые требуют понимания социального контекста. Между тем нейронные сети уверенно продолжают проникать в нашу жизнь, и примеров тому немало.

## Работы и роботизация человека

### Истоки термина

Слово «робот» появилось с подачи чешского писателя Карела Чапека после публикации в 1920 году его пьесы «Россумские Универсальные Роботы».

Герой пьесы «Р. У. Р.» учёный Россум изобрёл искусственных людей. Вскоре их производство было поставлено на промышленные рельсы. Искусственные люди должны были помогать человеку во всём.

В черновом варианте пьесы Чапек использовал для описания искусственных людей слово «лабор» (от лат. *labor* — «работа»). По замыслу автора, новый термин должен был отражать основную сущность машин — способность без устали выполнять рутинные действия. Впоследствии Карел по совету своего брата Йозефа изменил «лабор» на «робот» (от чешск. *robota* — «работа», «тяжёлый труд»), звучание этого слова показалось ему более подходящим.

Один из героев пьесы, генеральный директор компании «Р. У. Р.», отвечая на вопрос «что такое роботы?», говорит: «Роботы — это не люди... они механически совершеннее нас, они обладают невероятно сильным интеллектом, но у них нет души». Так впервые появилось новое понятие «робот», которое стали использовать не только в фантастической литературе, но и в науке и технике.

### Значение

Можно встретить множество значений рассматриваемого понятия. Вот лишь некоторые из них:

- **Большая советская энциклопедия**

Робот (чеш. robot, от robota - подневольный труд, rob - раб) – машина с антропоморфным (человекоподобным) поведением, которая частично или полностью выполняет функции человека (иногда животного) при взаимодействии с окружающим миром.

- **Викисловарь**

Робот – электромеханическое, пневматическое, гидравлическое устройство или их комбинация, предназначенное для замены человека в промышленности, опасных средах и др.

Робот – программа, работающая без участия человека и выполняющая действия, (как правило, рутинные), которые обычно выполняются человеком

- **Википедия**

Робот (чеш. robot — подпольный труд) — автоматическое устройство. Действуя по заранее заложенной программе и получая информацию о внешнем мире от датчиков (аналогов органов чувств живых организмов), робот самостоятельно осуществляет производственные и иные операции, обычно выполняемые человеком). При этом робот может как и иметь связь с оператором (получать от него команды), так и действовать автономно.

Робот (от словацк. robota) — автоматическое устройство с антропоморфным действием, которое частично или полностью заменяет человека при выполнении работ в монотонных, опасных для жизни условиях или при относительной недоступности объекта.

- **Американский институт по изучению роботической техники (The Robot Institute of America)**

Робот – репрограммируемый мультифункциональный манипулятор, предназначенный для перемещения/передвижения материалов, предметов, их частей или иных специализированных устройств с целью выполнения различных задач.

- **Словарь Уэбстера**

Робот – автономный аппарат или устройство, осуществляющий различные действия, свойственные человеку, и выполняющий их как будто под контролем человеческого разума.

[История роботов](#)

[Древность](#)

Если обратиться к истории, то становится ясно, что с древних времен человечество пыталось использовать машины для облегчения своего труда, выполнения наиболее тяжелой работы, требуемой значительных физических усилий.

Древнегреческий философ, математик и механик Архит Тарентский (428-347 до н.э.) спроектировал первую летающую машину - деревянную птицу, способную самостоятельно двигать крыльями при помощи пара и перемещаться на расстояние до 200 метров. Следующим шагом стало изобретение древнегреческим математиком Ктесибием Александрийским (285-222 г. до н.э.) в 250 году до н.э. хитроумных водяных часов, названных клепсидрами, ставшими самыми точными определителями

времени вплоть до изобретения в XVII веке голландским физиком Христианом Гюйгенсом маятника для поддержания незатухающих колебаний.

Уже в те времена появились идеи создания технических средств, похожих на человека, и были предприняты первые попытки по их созданию. Статуи богов с подвижными частями тела (руки, голова) появились еще в Древнем Египте, Вавилоне, Китае. В 3 веке до н. э. римский поэт Клавдий упоминал об автомате, изготовленном Архимедом. Он имел форму стеклянного шара с изображением небесного свода, на котором воспроизводилось движение всех известных в то время небесных светил. Шар приводился в движение водой.

Так же ярким представителем древних механизмов является Антикитерский механизм. 4 апреля 1900 года греческим водолазом Ликопантисом обнаружен в Эгейском море между греческим островом Крит и полуостровом Пелопоннес недалеко от острова Антикитера на глубине от 43 до 62 метров затонувший античный римский корабль. Ныряльщики за губками подняли на поверхность в 1901 году бронзовую статую юноши и множество других артефактов. 17 мая 1902 года археолог Валериос Стасис обнаружил среди поднятых предметов несколько бронзовых шестерён, закреплённых в кусках известняка. Артефакт оставался неизученным до 1951 года, когда английский историк науки Дерек Джон де Солла Прайс заинтересовался им и впервые определил, что механизм является уникальным античным механическим вычислительным устройством. Возможно, корабль шел с острова Родос, где во II веке до н. э. жил и работал известный греческий астроном и математик Гиппарх Никейский. Монеты, найденные на месте находки артефакта уже в 70-х годах XX века известным французским исследователем Жаком-Ивом Кусто, дали первую примерную дату изготовления находки — 85 год до н. э. Механизм содержал 37 бронзовых шестерён в деревянном корпусе, на котором были размещены циферблаты со стрелками и, по реконструкции, использовался для расчёта движения небесных тел.

До нас дошли книги Герона Александрийского (I век н.э.), где описаны подобные и многие другие автоматы древности. В качестве источника энергии в них использовались вода, пар, гравитация (гири). Например, известен автомат Герона для продажи воды.



#### Средние века

В средние века большой популярностью пользовались различного рода автоматы, основанные на использовании часовых механизмов. Были созданы всевозможные часы с движущимися фигурами людей, ангелов и т. п. К этому периоду относятся сведения о создании первых подвижных

человекоподобных механических фигур – андроидов. Так, андроид алхимика Альберта Великого (1193 – 1280) представлял собой куклу в рост человека, которая, когда стучали в дверь, открывала и закрывала ее, кланяясь при этом входящему.

Прообразами роботов были также механические фигуры, созданные арабским учёным и изобретателем Аль-Джазари (1136–1206). Так, он создал лодку с четырьмя механическими музыкантами, которые играли на бубнах, арфе и флейте.

13 веке Альберт Великий создал автомат, ставший впоследствии известным как «говорящая голова», способный воспроизводить человеческий голос.

Чертёж человекоподобного робота был сделан Леонардо да Винчи около 1495 года. Записи Леонардо, найденные в 1950-х, содержали детальные чертежи механического рыцаря, способного сидеть, раздвигать руки, двигать головой и открывать забрало. А в 1500 году он построил механического льва, который при въезде короля Франции в Милан выдвигался, раздирал когтями грудь и показывал герб Франции.

#### [Новое время](#)

Работы по созданию андроидов достигли наибольшего развития в XVIII в. Одновременно с расцветом часового мастерства. Французский механик и изобретатель Жак де Вокансон (1709-1789) создал в 1738 году первое работающее человекоподобное устройство (андроид), которое играло на флейте. «Флейтист» был ростом с человека. Подвижными пальцами он мог исполнять 11 мелодий с помощью заложенной в него программы. Вокансон также создал механическую утку, покрытую настоящими перьями, которая могла ходить, двигать крыльями, крякать, пить воду, клевать зерно и, перемалывая его маленькой внутренней мельницей, отправлять нужду на пол. Утка состояла из более чем 400 движущихся деталей и была однозначно признана венцом творения мастера. Созданием автоматов также занимались швейцарские часовщики Пьер-Жак Дро (1721-1790) и его сын Анри Дро (1752-1791). От имени последнего позднее было образовано и понятие «андроид». Пьер-Жак Дро создал несколько автоматов, из которых наибольшую известность получили писец и художник. Писец представлял собой сидящую за столом девочку, которая выписывала аккуратным почерком буквы, слова и даже могла нарисовать собаку.

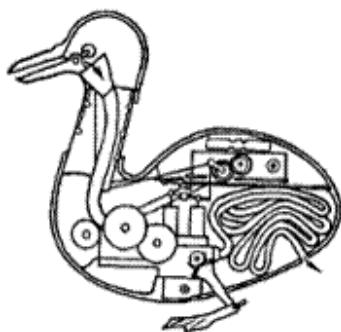


Рис. 1. Утка Жака де Вокансона

Вместе с сыном они создали девушку, играющую на клавесине. Сохранилось восторженное описание этой фигуры современником: «Девушка играет, шевелит губами, грудь ее поднимается и опускается при «дыхании», она смотрит на клавиши, в ноты, а иногда бросает взгляд на публику, по

окончании «номера» встает и кланяется». Эти человекоподобные игрушки представляли собой многопрограммные автоматы с оперативно сменяемыми программами.

Не остались в стороне и русские механики. Однако их конструкции отличались простотой конструкции. Так, механик И.П. Кулибин (1735-1818) построил в течении трех лет яичную фигуру – универсальные часы. Часы давали театрализованное представление и играли музыку. В этих часах было три самостоятельных механизма и три завода: часовой, боевой и курантовый, а также автоматические приборы для приведения в действие механизмов, воспроизводящих сцены, музыку и бой. Как свидетельствует сохранившаяся описание частей, составленная Кулибиным, часы яичной фигуры состояли из 427 деталей. Все они были изготовлены исключительно точно и тонко.

Во второй половине XVIII века (до 1770 года) в городе Несвиже (Литва, Минское воеводство) Евной Якобсоном была создана суммирующая машина. Машина выполнена в виде латунной коробки: 34,2 см x 21,8 см x 3,4 см на четырех точенных ножках. Устройство было довольно сложным, а сам механизм мог использоваться для выполнения операций сложения (сумма не должна была превышать 109) и вычитания чисел. Эта счетная машина сохранилась и находится в коллекции научных инструментов Музея им. М.В.Ломоносова. (Санкт-Петербург).

#### [Новейшее время](#)

Благодаря всеобщему интересу к работам изобретателям удается разрабатывать оригинальные конструкции роботов-androидов:

- «Мистер Телевокс» (1928, американский инженер Дж. Уэнсли) — робот, имевший внешнее сходство с человеком, способный выполнять элементарные движения по команде, подаваемой голосом, и ставший экспонатом Всемирной выставки в Нью-Йорке.
- «Эрик» (1928) — робот, который на Выставке Британской ассоциации инженеров по моделированию «выступил» с небольшой речью.
- «Естествоиспытатель» (1928, под руководством доктора Нисимура Макота) — японский робот, способный с помощью электропривода манипулировать руками и головой. Впоследствии этот андроид стали считать родоначальником роботостроения в Японии.
- «Альфа» (1932, английский изобретатель Гарри Мей) — человекоподобный автомат, который по голосовым командам садился и вставал, двигал руками и говорил.
- «Сабор» (австрийский изобретатель Август Губер) — автоматы, которые управлялись по радио и могли говорить, ходить, выполнять разные манипуляции.
- В2М (1936, московский школьник Вадим Мацкевич) — первый робот-андроид в России. В 1937 году был удостоен диплома Всемирной выставки в Париже.

Несмотря на такой прорыв в сфере новой техники и демонстрацию творческих возможностей человека, все эти роботы имели крайне узкое практическое применение.

Проблемы внедрения роботов в промышленность как таковые не решались. Если обратиться к работам как к программно-управляемым многоцелевым автоматам манипуляционного типа, предназначенным для использования в промышленности или научных исследованиях, то одним из самых первых промышленных манипуляторов был поворотный механизм с захватным устройством для удаления заготовок из печи, разработанный в США Бэббитом в 1892 году (патент США № 484870). Особую известность получили копирующие манипуляторы, разработанные Государственным научно-исследовательским институтом штата Орегон (США) ANL; предложенные

им инструкции и принципы управления до сих пор находят применение во многих моделях промышленных роботов.

Одним из первых в ANL манипуляторов для обслуживания атомных станций был разработан в 1948 году под руководством Р. Герца. Это был двухнаправленный копирующий манипулятор. Благодаря силовому ощущению оператора, который находился за толстой перегородкой в специальном помещении, имел возможность не только наблюдать на экране перемещение управляемого им копирующего манипулятора, но и ощущать руками величину усилий, которые развивает захват манипулятора. Использование такой обратной связи позволило упростить процесс управления на расстоянии и расширить функциональные возможности дистанционных управляемых манипуляторов.

Более прямыми предшественниками современных манипуляционных роботов можно считать программируемые краскораспылительные машины, разработанные в 1930-1940 гг. в США, например, машины Уилларда Л.В. Полларда и Гарольдом Роузландо, которые программировались путем записи сигнала от рычажного механизма, перемещаемого по заданной траектории. Возросший экономический потенциал и потребности в современных видах вооружения ведущих промышленных стран в первой половине XX века дают мощный импульс развитию науки и научно-технических направлений, без которых возникновение и прогресс современной робототехники стали бы невозможными. Речь идет, прежде всего, о вычислительной технике и кибернетике.

Возникновение современных роботов следует отнести к 1959 г. В этом году в США были созданы первые промышленные манипуляторы с программным управлением, которые получили общепринятое название промышленных роботов (ПР) и положили начало коммерческому производству. В 50-х гг. ХХ века группа американских инженеров начала работу над проблемой применения теории управления в решении общих задач оптимального перемещения оборудования. Первопроходцами здесь стали два талантливых американских инженера – Джордж К. Девол (1912-2011) и Джозеф Ф. Энгельбергер (род. в 1925). В 1954 г. Девол запатентовал в США способ перемещения предметов между различными участками предприятия на основе управляемой программы на перфокартах, сходных с предложенным когда-то Бэббиджем. Изобретение было призвано решить, в первую очередь, именно проблему гибкости, т.е. создания универсального транспортировочного устройства, легко перестраиваемого для выполнения других операций.

В 1956 г. Девол вместе с Энгельбергером, работавшим тогда в одной из аэрокосмических компаний, организовали первую в мире робототехническую компанию «Unimation» («Юнимейшн»), что означает «универсальная автоматизация» – сокращенное от «Universal Automation», в лаборатории этой компании и был создан первый в мире промышленный робот по патенту Девола, носивший скромное название «программируемое устройство для передачи предметов» и ставший прототипом последующих разработок.

В начале 1960-х гг. первые американские промышленные роботы с торговыми марками «Unimate» и «Versatran», созданные соответственно фирмами «Unimation», «American Machine and Foundry» (AMF) и предназначенные для обслуживания технологических процессов – поступили на промышленный рынок. Они представляли собой уже достаточно совершенные системы с обратной связью и контролируемой траекторией движения, имели числовое программное управление и

память, как у ЭВМ. Уже в первых роботах «Unimate» и «Versatran» был реализован принцип программирования обучением.

Применение роботов в автомобильной и металлургической промышленности оказалось экономически выгодным: затраты на приобретение роботов «Unimate» или «Versatran» окупались за 1,5 — 2,5 года.

Первые коммерческие успехи применения промышленных роботов явились мощным импульсом для их дальнейшего совершенствования. В начале 1970-х гг. появляются роботы, управляемые компьютерами. Первый мини-компьютер, управляющий роботом, был выпущен в 1974 г. фирмой «Cincinnati Milacron», одной из ведущих фирм — изготовителей роботов в США. В конце 1971 г. американской фирмой «INTEL» был создан первый микропроцессор, а несколькими годами позже появляются роботы с микропроцессорным управлением, что обусловило существенное повышение их качества при одновременном снижении стоимости.

В последующие годы после создания и выхода на промышленный рынок первых роботов во всем мире началось стремительное развитие робототехники. В ряде капиталистических стран организуются ассоциации или общества, курирующие исследования и разработки в области создания и использования промышленных роботов, в частности, в 1972 г. образована Японская ассоциация промышленной робототехники (JIRA), в 1974 — Институт робототехники США (RIA) и ассоциация роботов Великобритании (BRA), в 1975 — Итальянское общество робототехники (SIRI), в 1978 — Французская (AFRI), в 1980 — Шведская (SWIRA), в 1981 — Австралийская (ARA), в 1982 — Датская (DRA) и Сингапурская (SRA) ассоциации роботов.

Изменяется и сам принцип использования промышленных роботов — от единичного к комплексному. В ведущих робототехнических странах (Япония, США, ФРГ, СССР и др.) в конце 1960-х — начале 1970-х гг. разрабатываются и создаются гибкие производственные системы (ГПС), так называемые «безлюдные» производства, представляющие собой производства будущего. Научно-технические достижения робототехники позволили в 1960-1980-х гг. создать ряд сложных научных и специальных робототехнических комплексов для исследования космического пространства (станции типа «Луна», аппараты «Луноход» — СССР; станции типа «Маринер», «Сервейер», «Викинг» — США и др.), а также освоения подводных глубин (аппараты «TV», «Москито», «Долфин» — Япония; аппараты «KURV», «RCV» — США; «Манта», «ОСА» — СССР; «ROV», «RM» — Франция; «ARCS» — Канада и др.).

Технический прогресс в развитии роботов был направлен, прежде всего, на совершенствование систем управления. Промышленные роботы первого поколения имели программное управление, в основном заимствованное у станков с числовым управлением. Второе поколение роботов — это очуствленные роботы, т. е. снабженные сенсорными системами, главными из которых являются системы технического зрения.

Первые промышленные роботы с развитой сенсорной системой и микропроцессорным управлением появились на рынке и получили практическое применение в 1980-1981 гг. прежде всего на сборке, дуговой сварке, контроле качества для взятия неориентированных предметов, например, с конвейера. К их числу относятся снабженные системами технического зрения роботы «Пума», «Юнимейт», «Авто-плейс», «Цинциннати милакрон», сборочные робототехнические системы фирм «Хитачи», «Вестингауз» (система «Апас»), «Дженерал моторс» (система «Консайт»).

Доля таких роботов в общем парке роботов неуклонно росло и приближалось к 50% несмотря на то, что эти роботы были в несколько раз дороже роботов с программным управлением и значительно сложнее в обслуживании. Однако это окупается неизмеримо большими функциональными возможностями, а, следовательно, и областями применения.

Третье поколение роботов – это интеллектуальные роботы, т.е. с интеллектуальным управлением. Интеллектуальный робот – это робот конкретного назначения, в основных функциональных системах которого используются методы искусственного интеллекта. Возникновение интеллекта у роботов связано с развитием ЭВМ. В 1967 г. в США (Стэнфордский университет) был создан лабораторный макет робота, снабженного техническим зрением и предназначенного для исследования и отработки системы «глаз – рука», способной распознавать объекты внешней среды и оперировать ими в соответствии с заданием.

В 1968 г. в СССР (Институтом океанологии Академии наук СССР совместно с Ленинградским политехническим институтом и другими вузами) был создан телеуправляемый от ЭВМ подводный робот «Манта» с очувствленным захватным устройством, а в 1971 г. – следующий его вариант с техническим зрением и системой целеуказания по телевизионному экрану.

В 1969 г. в США (Стэнфордский университет) в рамках работ по искусственному интеллекту был разработан экспериментальный макет подвижного робота «Шейки» с развитой системой сенсорного обеспечения, включая техническое зрение, обладавшего элементами искусственного интеллекта, что позволило ему целенаправленно передвигаться в заранее неизвестной обстановке, самостоятельно принимая необходимые для этого решения.

В 1971 г. в Японии также были разработаны экспериментальные образцы роботов с техническим зрением и элементами искусственного интеллекта: робот «Хивип», способный самостоятельно осуществлять механическую сборку простых объектов по предъявленному чертежу.

В 1972-1975 годах в Киевском Институте кибернетики под руководством Н. М. Амосова и В. М. Глушкова был создан макет транспортного автономного интегрального робота (ТАИР). Робот демонстрировал целенаправленное движение в естественной среде, обход препятствий и т.п. Конструктивно ТАИР представлял собой трехколесную самоходную тележку, снабженную системой датчиков: оптическим дальномером, навигационной системой с двумя радиомаяками и компасом, контактными датчиками, датчиками углов наклона тележки, таймером и др. Особенностью, которая отличает ТАИР от многих других систем, созданных в СССР и за рубежом, является отсутствие в его составе компьютера в том виде, к которому мы привыкли. Основу системы управления составляет аппаратно реализованная нейронная сеть (узлы сети – специальные электронные схемы, собранные на транзисторах, связи между узлами – резисторы), на которой реализуются различные алгоритмы обработки сенсорной информации, планирования поведения и управления движением робота.

В 1985 г. в промышленности капиталистических стран эксплуатировались свыше 100 тыс. промышленных роботов, а в 1989 г. - уже более 257 тыс., в том числе в Японии - 174, в США - 35, в Западной Европе - 48 тыс.

## 2000-ые

Кевин Уорвик (Kevin Warwick) в Орегонском университете создает первого киборга (кибернетический организм). В небольшой стандартный робот Khepera включены элементы мозга морской змеи (*Petromyzon marinus*). Соединенный с сенсорами мозг реагирует на световые сигналы, перемещаясь в тень при освещении сенсоров. Работы ведутся Орегонским, Чикагским и Иллинойским университетами США, а также университетом Генуи, Италия.

Стюарт Уилкинсон (Stuart Wilkinson) из Университета Южной Флориды представляет плотоядного робота Chew Chew. 12-колесный «гастроробот», напоминающий поезд, работает на микробиологических топливных элементах, которые разлагают пищевые продукты при помощи бактерий и вырабатывают электроэнергию. Chew Chew потребляет кусковой сахар, но, по мнению изобретателя, по энергетической эффективности мясо — идеальное топливо. Уилкинсон в шутку предупреждает, что опасно приучать гастророботов к вкусу мяса, ведь вокруг может быть много людей.

В МИТ создан робот, имитирующий актинию и названный Public Apemton. Робот реагирует на свет и прикосновения, шевелит щупальцами и может перемещаться на небольшие расстояния. Подобно своим живым прототипам, робот имеет мягкое трубчатое тело и несколько щупалец, которые на самом деле представляют собой световоды. Внешняя оболочка робота сделана из мягкого силикона и создаёт иллюзию органического происхождения. Когда к аквариуму подходит человек, робот с помощью световодов фиксирует изменение освещенности, определяет его причину и поворачивается в этом направлении. Когда человек пытается дотронуться до робота, тот прячет щупальца.

В восточных странах популярны верблюжьи бега. Управлять верблюдом жокею совсем несложно, поэтому решающее значение имеет вес — чем легче жокей, тем больше преимущества у пары. Это привело к повсеместной эксплуатации мальчиков от 4 лет в качестве наездников, которых, к тому же, часто морили голодом, чтобы они набирали меньше веса. Несколько лет назад в ОАЭ и Катаре детский труд запретили, что способствовало развитию миниатюрных роботов-жокеев, управляемых оператором дистанционно.

Роботы стали незаменимыми помощниками в изучении животного мира. Так, японская робот-рыба может незаметно для морских обитателей вести наблюдение за стаями. Под силиконовой оболочкой, повторяющей внешний вид красного луциана, спрятана система балластов наподобие тех, что используются в подводных лодках для всплытия и погружения. В действие устройство приводится движениями хвостовой части.

А тараканороботы могут уничтожать популяции вредных домашних насекомых изнутри. Ученые Франции, Бельгии и Швейцарии создали модель, которая выглядит и пахнет как таракан, передвигается на колесиках, оснащена камерами и инфракрасными сенсорами и воздействует на коллективное сознание насекомых, увлекая их на свет. В будущем изобретатели намерены создать модели посередине, например, для управления овечьим стадом.

Изобретатели Австрии создали робота-алкоголика. Bar Bot сидит в баре, выискивая "жертву". Поймав на себе любопытный взгляд, он начинает просить монетку, собрав необходимую сумму, принимается крутиться вокруг своей оси, приговаривая: "Пожалуйста, одно пиво". Бармен

вставляет банку пива в "руку". "Большое спасибо", – благодарит Bar Bot и не спеша выливает напиток в напоминающий раковину "рот". Затем швыряет банку на пол, и процесс начинается снова.

Построенный в Канаде космический манипулятор Canadarm2 успешно начал работу по завершению сборки Международной Космической станции (МКС). Canadarm2 является представителем второго поколения канадских манипуляторов фирмы MD Robotics. Манипуляторы первого поколения RMS (Canadarm) уже почти 20 лет используются на Шаттлах. Конструктивно Canadarm2 состоит из двух «плеч» и имеет 7 степеней свободы. Длина манипулятора – 17,6 м.

Первый в мире серийно выпускаемый бытовой робот-пылесос Trilobite представлен на рынок шведской компанией Electrolux. Робот ориентируется с помощью ультразвукового сонара и имеет высоту 13 см при диаметре 35 см. Максимальная скорость уборки — 40 квадратных сантиметров в секунду. Когда аккумуляторы робота "садятся", Trilobite сам находит зарядное устройство и едет заряжаться.

Компания iRobot в сотрудничестве с Acer Technology Ventures разработала уникального робота-санитара Bloodhound. Робот, предназначенный для спасения раненых солдат, способен автономно передвигаться по незнакомой местности и обходить препятствия. Диагностическое оборудование Bloodhound состоит из видеокамеры, электронного стетоскопа и радиопередатчика, позволяющего медику общаться с раненым. Определив степень повреждений, санитар может оказать раненому первую помощь посредством устройств по остановке кровотечения, внутримышечных инъекций (морфий, адреналин, противоядия и т.п.).

Также, данной компанией представлен робот PackBot, способный определять местонахождение снайперов и осуществлять наведение на цель. Система Redowl снабжена лазерным прицелом и прожектором, акустическим локализатором и классификатором, тепловизором, датчиком глобального позиционирования (GPS) и камерами инфракрасного и дневного света, а также двумя широкоугольными камерами. Машина может записывать и передавать видео с помощью камкордеров от Sony. С помощью интеллектуального ПО робот может отличить выстрелы от банального эха, например, в горах. Снайперу достаточно сделать один выстрел, как REDOWL тут же определит его точное местонахождение.

Американские ученые из Стенфордского университета, проводившие исследования под руководством профессора Марка Каткоски, создали робота-геккона Stickybot, способного самостоятельно передвигаться по гладким вертикальным поверхностям и даже стеклу. Принцип работы Stickybot позаимствован у природы, в частности у ящериц гекконов.

В Корнеллском университете в Итаке (штат Нью-Йорк) создан робот, передвигающийся на 4 конечностях и способный самостоятельно оценивать причиненные ему повреждения и определять, как приспособиться для продолжения выполнения задания.

Робот оснащен датчиками, которые дают ему информацию о пространственной ориентации и позволяют создавать компьютерную модель собственной конфигурации и движения. Эта особенность позволяет менять программу, если происходит нечто непредвиденное.

Основатель корпорации Microsoft Билл Гейтс предрек, что человечество находится на пороге новой эры. Роботы станут неотъемлемой частью повседневной жизни и будут так же доступны, как сейчас

компьютеры. По мнению Гейтса, в ближайшее время в робототехнике произойдут революционные изменения, схожие с прорывом в вычислительной технике 30 лет назад.

Авторитетный американский журнал *Forbes* представил своим читателям пятёрку роботов, которые, по мнению издания, в ближайшее время изменят жизнь человека. Первой названа хирургическая система да Винчи (*da Vinci Surgical System*), позволяющая медикам проводить операции практически на любом расстоянии. Врач удалённо управляет автоматизированными руками, которые делают всю работу. Исследования показали, что пациенты, прооперированные с помощью робота, быстрее встают на ноги, у них остаётся меньше шрамов.

По оценке ООН на 2000-ый год, в мире используется 742,500 промышленных роботов. Больше чем половина из них трудятся в Японии.

По данным Международной федерации робототехники, в 2013 году мировой объём продаж промышленных роботов составил 178132 единиц (рост на 12 % по сравнению с предыдущим годом). Крупнейшим рынком промышленных роботов стала Китайская Народная Республика, предприятия которой закупили 36 560 промышленных роботов. За ней следуют Япония (25110 единиц), США (23700 единиц), Республика Корея (21307 единиц), Германия (18297 единиц) и другие промышленно развитые страны. Наибольшее число новых промышленных роботов — 69400 — было установлено на предприятиях автомобильной промышленности; второе место занимают предприятия электротехнической и электронной промышленности (36200 единиц), третье — предприятия металлообрабатывающей и машиностроительной промышленности (16500 единиц).

Использование промышленных роботов в мире продолжает расти. Статистика и экономические прогнозы говорят о том, что в 2015 году, будет внедрено более 200 000 промышленных роботов по всему миру. Это на 15% больше, чем в 2014 году. Промышленные роботы с каждым годом становятся более совершенными, что позволяет ежегодно расширять границы их применения.

Основная статистика:

- В 2014 году 72% от общего количества внедренных роботов, пришлось на страны лидеры: Япония, Китай, Корея, Германия и США;
- Китай стал самым большим рынком промышленной робототехники, доля которого составляет 20%;
- Мексика прибавила за 2014 год на 32% в сравнении с прошлогодним объемом потребления промышленных роботов;
- Бразилия в 2017 году достигнет значения – 18 300 внедренных роботов в год;
- В Австралии и Азии (без Китая), также планируется рост в размере 16% к 2017 году;
- В Африке прогнозируется самый большой рост к 2017 году в размере 85%. К этому времени, Африканское ежегодное потребление роботов приблизится к 8000 единиц;
- в 2014 году Российская доля в мировом объеме потребления промышленных роботов, составила <1%. Кроме этого, в период до 2017 года планируется снижение существующих объемов.

Главный футуролог Cisco Дэйв Эванс (Dave Evans) давно снискал широкую известность своими долгосрочными прогнозами дальнейшего развития информационных и коммуникационных

технологий, причем, как пишут американские СМИ, ещё ни разу не ошибся. Его специальность сегодня остается во многом экзотичной: штатные футурологи есть у считанного числа компаний.

Эванс сделал следующее предположение:

К 2025 году количество роботов превзойдет население развитых стран, к 2032 году – их интеллектуальные возможности окажутся выше человеческих, к 2035 – роботы полностью заменят людей в качестве рабочей силы.

### Трагические факты

В 1981 году Кэндзи Урада, рабочий завода Kawasaki стал первой официальной жертвой, погибшей от руки робота. С этого времени число жертв роботов растёт, несмотря на внедрение усовершенствованных механизмов безопасности.

18 марта 2008 года 81-летний австралиец стал первым человеком, который покончил жизнь самоубийством при помощи робота, которого сам собрал согласно схемам, взятым из сети Интернет.

### Роботы в культуре

Идея искусственных созданий впервые упоминается в древнегреческом мифе о Кадме, который, убив дракона, разбросал его зубы по земле и запахал их, из зубов выросли солдаты, и в другом древнегреческом мифе о Пигмалионе, который вдохнул жизнь в созданную им статую — Галатею. Также в мифе про Гефеста рассказывается, как он создал себе различных слуг. Еврейская легенда рассказывает о глиняном человеке — Големе, который был оживлён пражским раввином Йехудой бен Бецалелем при помощи каббалистической магии.

Похожий миф излагается в скандинавском эпосе Младшая Эдда. Там рассказывается о глиняном гиганте Мёккуркальви, созданном троллем Хрунгниром для схватки с Тором, богом грома.

Значительный вклад в формирование образа «робота» в литературе внес Айзек Азимов. Им, в рассказе 1942 года «Хоровод», были сформулированы «Три Закона Роботехники»:

- Робот не может причинить вреда человеку или своим бездействием допустить, чтобы человеку был причинён вред.
- Робот должен выполнять приказы человека в той мере, в которой это не противоречит Первому Закону.
- Робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому и Второму Законам.

Азимов в своих произведениях показывает, что эти законы, будучи заложены в программу-мозг робота в виде обязательных (безусловно исполняемых роботом) законов исключают возможность проявления любых недружественных действий робота по отношению к человеку. Приводятся также примеры негативных последствий, возникающих в случае, когда люди пренебрегая требованиям обязательности трех законов блокируют на этапе программирования робота один из законов (например, вторую часть первого закона). В этом случае робот может найти логически непротиворечивое решение, позволяющее ему нарушить 1-й закон и стать опасным для человека.

Также Айзеком Азимовым (в романах «Роботы и Империя», «На пути к основанию») сформулирован так называемый «нулевой» закон робототехники: «Робот не может причинить вред человечеству или своим бездействием способствовать этому».

«...Нулевой. Робот не может причинить вред человечеству или, своим бездействием, способствовать этому. Тогда Первый Закон следует читать следующим образом: Первый. Робот не может причинить вред человеческому существу или, своим бездействием, способствовать этому, кроме тех случаев, когда это противоречит Нулевому Закону. Таким же образом следует трактовать и последние два...» — Айзек Азимов «На пути к основанию»

В романе 1897 года «Война миров» Герберт Уэллс описывает гигантские треножники, оснащённые тепловыми лучами, которые использовались пришельцами для захвата Земли.

Так же культовый статус имеет роман Филипа Дика «Мечтают ли андроиды об электроовцах?», написанный в 1968 году. В 1982 году по мотивам романа Ридли Скотт снял фильм «Бегущий по лезвию» с Харрисоном Фордом в главной роли. Сценарий, который создали Хэмптон Фэнчер и Дэвид Пиплс, довольно сильно отличается от книги. Действие компьютерной игры «Бегущий по лезвию» происходит в той же фантастической вселенной, но во многих отношениях игра ближе к роману, чем фильм.

Роботы «оказали» такой же вклад в киноиндустрию, как когда-то в фантастическую литературу и анимацию, особенно на жанр кинофантастики. Так фильм Терминатор стал одним из самых культовых фильмов в истории кинематографа вообще, а его главные герои известны практически каждому человеку. Образы роботов в фильмах частично впитали ту роль которая им отдавалась в художественной литературе. В основном роботы были специально созданы машинами для убийства — неубиваемым врагом с которым предстояло сразиться герою.

В середине XX века популярным был образ Робота, основанный на персонаже «Робота-охранника» из фильма «День, когда земля остановилась», который представлял собой гигантского робота, основным оружием которого был лазер, способный уничтожать целые города. Впоследствии этот штамп стал настолько популярным, что уже почти не использовался и высмеивался как один из самых заезженных штампов Фантастики наряду со злобными пришельцами и бластерами.

За время развития образ роботов в Кино изменился, в отличие от медленных «жестяноч»-истуканов из фильмов 30-60 годов, современный образ экраных роботов стал больше заимствовать стилистику современной техники, в том числе стиль Хай-тек. Так дизайн роботов из агрессивного стал на данный момент более детальным и «стильным».

Также культовыми стали следующие роботы-герои фильмов:

- C3PO и R2-D2 из Звёздных войн
- Робот-страж из фильма «День, когда земля остановилась»
- Треножники (частично пилотируемые) — из книги и одноименного фильма «Война миров»
- T-800 и T-1000 из серии фильмов Терминатор
- Дэйта — андроид из сериала «Звёздный путь. Новое поколение»
- Робот Бендер — робот из мультсериала «Футурама»
- Идея робота обыграна в фильме «Приключения Электроника»
- Робот из мультфильма «Валл-и»

- Вертер — робот из «Гостя из будущего»
- Робокоп из фильма «Робот-полицейский»
- Марвин из Автостопом по галактике
- Эндрю из фильма «Двухсотлетний человек»
- Оптимус Прайм, Мегатрон, Старскрим, Бамблби и другие роботы-трансформеры, персонажи многочисленных мультсериалов и фильмов о трансформерах.

Также, тема роботов затрагивается в огромном количестве видеоигр. Существует жанр видео игр — симуляторы меха. Наиболее известным представителем этого жанра является серия игр MechWarrior. В таких играх как Lost Planet, Shogo: Mobile Armor Division, Quake IV, Chrome, Unreal Tournament 3, Battlefield 2142, F.E.A.R. 2: Project Origin имеется возможность управлять роботами.

Также, в культовой серии игр Deus Ex затрагивается моральная сторона роботизации человека.

### Типы роботов

#### Промышленные роботы

Появление станков с числовым программным управлением (ЧПУ) привело к созданию программируемых манипуляторов для разнообразных операций по загрузке и разгрузке станков. Появление в 70-х гг. микропроцессорных систем управления и замена специализированных устройств управления на программируемые контроллеры позволили снизить стоимость роботов в три раза, сделав рентабельным их массовое внедрение в промышленности. Этому способствовали объективные предпосылки развития промышленного производства.

Несмотря на их высокую стоимость, численность промышленных роботов в странах с развитым производством быстро растёт. Основная причина массовой роботизации такова: «Роботы выполняют сложные производственные операции по 24 ч в сутки. Выпускаемая продукция при этом имеет высокое качество. Они... не болеют, не нуждаются в обеденном перерыве и отдыхе, не бастуют, не требуют повышения заработной платы и пенсии. Роботы не подвержены влиянию температуры окружающей среды либо воздействию газов или выбросов агрессивных веществ, опасных для жизни человека».

#### Медицинские роботы

В последние годы роботы получают всё большее применение в медицине; в частности, разрабатываются различные модели хирургических роботов. Ещё в 1985 году робот Unimation Puma 200 был использован для позиционирования хирургической иглы при выполнении биопсии головного мозга, проводившейся под управлением компьютера. В 1992 году разработанный в Имперском колледже Лондона робот ProBot впервые осуществил операцию на предстательной железе, положив начало практической роботизированной хирургии. С 2000 года компания Intuitive Surgical серийно выпускает робот Da Vinci, предназначенный для лапароскопических операций и установленный в нескольких сотнях клиник по всему миру.

#### Бытовые роботы

Одним из первых примеров удачной массовой промышленной реализации бытовых роботов стала механическая собачка AIBO корпорации Sony.

В сентябре 2005 в свободную продажу впервые поступили первые человекообразные роботы «Вакамару» производства фирмы Mitsubishi. Робот стоимостью \$15 тыс. способен узнавать лица,

понимать некоторые фразы, давать справки, выполнять некоторые секретарские функции, следить за помещением.

Всё большую популярность набирают роботы-уборщики (по своей сути — автоматические пылесосы), способные самостоятельно прибраться в квартире и вернуться на место для подзарядки без участия человека.

#### [Роботы для обеспечения безопасности](#)

В последнее время роботы всё чаще применяются силовыми структурами: полицией, органами государственной безопасности, аварийно-спасательными службами, силами ведомственной и внедомственной охраны. В 2007 году в Перми прошли первые испытания робота-полицейского Р-БОТ 001, разработанного московской компанией «Лаборатория Трёхмерного Зрения». При тушении пожаров применяют роботизированные установки пожаротушения.

Для оперативной разведки агентства по чрезвычайным ситуациям и полиция используют «летающих роботов» (беспилотные летательные аппараты). При проведении под водой обследования потенциально опасных объектов и поисково-спасательных работ службы МЧС России используют подводные роботы серии «Гном», выпускаемые с 2001 года московской компанией «Подводная робототехника».

#### [Боевые роботы](#)

Боевым роботом называют автоматическое устройство, заменяющее человека в боевых ситуациях или при работе в условиях, несовместимых с возможностями человека, в военных целях: разведка, боевые действия, разминирование и т. п. Боевыми роботами являются не только автоматические устройства с антропоморфным действием, которые частично или полностью заменяют человека, но и действующие в воздушной и водной среде, не являющейся средой обитания человека (авиационные беспилотные с дистанционным управлением, подводные аппараты и надводные корабли). В настоящее время большинство боевых роботов являются устройствами телеприсутствия, и лишь очень немногие модели имеют возможность выполнять некоторые задачи автономно, без вмешательства оператора.

В Технологическом институте Джорджии под руководством профессора Хенрика Кристенсена разработаны напоминающие муравьёв инсектоморфные роботы, способные обследовать здание на предмет наличия там врагов и мин-ловушек ( доставляются к зданию «главным роботом» — мобильным роботом на гусеничном ходу). Получили распространение в войсках и летающие роботы. На начало 2012 года военными во всём мире использовались около 10 тысяч наземных и 5 тысяч летающих роботов; 45 стран мира разрабатывало или закупало военных роботов.

В США проведены испытания прототипа робота-собаки Spot, разработанного компанией Boston Dynamics. Во время тестов на базе морских пехотинцев Куантико, робот обследовал помещения на предмет нахождения в них противника. Данные об обнаруженных целях робот передавал на пульт оператора. Робота проверяли в условиях пересеченной местности, леса и городской застройки. Spot пригоден для использования в войсках для разведки, патрулирования и переноски грузов.

#### [Космороботы](#)

Космороботы — это роботы, приспособленные работать в космическом пространстве. Преимущество космических роботов перед человеком заключается в том, что они могут работать в

крайне неблагоприятных условиях (например, в космосе есть радиация, поэтому человек не может выйти в открытый космос без скафандра, чего нельзя сказать про робота) и обходиться без каких-либо ресурсов (например, топлива), так как в большинстве случаев они работают на солнечных батареях. Также гораздо легче будет пережить потерю такого робота, чем гибель астронавта. Обычно, задача космического робота заключается в проведении какой-нибудь научной работы (например, собрать образцы грунта, просканировать их и отправить собранные данные учёным на Землю).

Луноход-1 – первый в мире дистанционно-управляемый самоходный аппарат, успешно работавший на Луне. Отправлен он туда был для изучения лунного грунта, а также для изучения радиоактивного и рентгеновского излучения. На поверхность луны он был доставлен 17 ноября 1970 года советской межпланетной станцией «Луна-17».

Марсоходы «Спирит» и «Оппортьюнити» - аппараты близнецы, успешно запущенные на Марс в 2004 году. Отправлены они были туда, впринципе, для одной цели – установить, была ли когда-нибудь на Марсе вода или нет.

### Роботы-учёные

Первые роботы-учёные Адам и Ева были созданы в рамках проекта Robot Scientist университета Аберистуита и в 2009 году одним из них было совершено первое научное открытие.

К роботам-учёным безусловно можно отнести роботов, с помощью которых исследовались вентиляционные шахты Большой Пирамиды Хеопса. С их помощью были открыты т. н. «дверки Гантенбринка» и т. н. «ниши Хеопса». Исследования продолжаются.

Robot Scientist — междисциплинарный научный проект университета Аберистуита по созданию робота-учёного. Первого робота-учёного в рамках проекта начали разрабатывать в 1999 году. На декабрь 2010 года в рамках проекта было разработано два робота: Адам и Ева. В 2009 году Адамом было совершено первое научное открытие — робот нашёл кодирующие гены для ферментов сирот *S. cerevisiae* пекарских дрожжей *Saccharomyces cerevisiae*. Робот, на основании начальных данных выдвинул гипотезы, провёл эксперименты и вывел вероятность соответствия гипотез действительности. По результатам работы Адама в апреле 2007 года в журнале *Science* была опубликована статья «The Automation of Science». По планам учёных Адам вместе со вторым роботом Евой помогут в поиске лекарства от малярии и шистосомоза. На данном этапе роботы выполняют работу скорее младшего ассистента, чем учёного, но Адам способен выполнять до 1000 экспериментов в день и с гораздо меньшим числом ошибок.

Потребность в роботах-учёных была спрогнозирована заранее, например, Станислав Лем в своей книге «Сумма технологии» пишет: «Количество учёных растёт экспоненциально. ... Таким образом, если нынешний темп научного роста сохранится, то через какие-нибудь 50 лет (книга написана в 1963 году) каждый житель Земли будет учёным». Создание «армии искусственных учёных» Лем видит как одно из очевидных, но временных решений.

### Роботы-футболисты

RoboCup — международные соревнования среди роботов, основанные в 1993 году. Целью является создание автономных роботов-футболистов для содействия научным исследованиям в области искусственного интеллекта. Название RoboCup — сокращение от полного названия соревнования, англ. "Robot Soccer World Cup" (Чемпионат по футболу среди роботов), но, в рамках соревнования,

существуют и другие виды состязаний, например, среди спасательных роботов, по танцам среди роботов.

Официальная цель проекта:

К середине 21-го века команда полностью автономных человекоподобных роботов-футболистов должна выиграть футбольный матч, соблюдая правила FIFA, у победителя Чемпионата мира.

### [Андроиды](#)

Андроид (от греч. корня ἀνδρ- слова ἄντρος — «человек, мужчина» и суффикса -oid — от греч. слова εἶδος — «подобие») — человекоподобный. В современном значении обычно подразумевается робот (человекоподобный робот).

Создание первого андроида приписывается Альберту Кельнскому. Значительную роль в популяризации термина сыграл французский писатель Филипп Огюст Матиас Вилье де Лиль-Адам, использовав его в своём романе «Будущая Ева» (фр. L'Ève future) для обозначения человекоподобного робота, описывая искусственную женщину Адали (Hadaly). Созданная Томасом Эдисоном Адали разговаривала с помощью фонографа, выдающего одну за другой классические цитаты.

Современные человекоподобные роботы:

- Aiko — гиноид с имитацией человеческих чувств: осознание, слух, речь, зрение.
- ТОРIO — андроид, разработанный для игры в настольный теннис против человека.
- ASIMO — андроид, созданный корпорацией Хонда, в Центре Фундаментальных Технических Исследований Вако (Япония).
- Einstein Robot — голова робота с внешностью Эйнштейна. Модель для тестирования и воспроизведения роботом человеческих эмоций.
- EveR-1 — робот, похожий на 20-летнюю кореянку: её рост 1,6 метра, а вес — около 50 килограммов. Ожидается, что андроиды вроде EveR смогут служить гидами, выдавая информацию в универмагах и музеях, а также развлекать детишек.
- HRP-4C — робот-девушка, предназначенная для демонстрации одежды. Рост робота составляет 158 см, а вес вместе с батареями — 43 кг. Что касается степеней свободы, их 42, к примеру, в области бёдер и шеи их по три, а в лице — восемь, они дают возможность выражать эмоции.
- Repliee R-1 — человекоподобный робот с внешностью японской пятилетней девочки, предназначенная для ухода за пожилыми и недееспособными людьми.
- Repliee Q2 — робот-девушка под рабочим названием Repliee Q1expo был показан на международной выставке World Expo, проходившей в Айти (Aichi), Япония. На демонстрациях он исполнял роль телевизионного интервьюера, при этом постоянно взаимодействуя с людьми. В роботе были установлены всенаправленные камеры, микрофоны и датчики, которые позволяли Repliee Q2 без особых трудностей определять человеческую речь и жестикуляцию.
- Ибн Сина — андроид, названный в честь древнего персидского философа и врача Ибн Сины. Один из самых продвинутых современных (2010 год) андроидов. Говорит на арабском языке. Способен самостоятельно найти своё место в самолёте, общаться с людьми.

Распознаёт выражение лица говорящего и прибегает к соответствующей ситуации мимике. Его губы двигаются довольно монотонно, однако отмечается, что особенно хорошо у него получается поднимать брови и прищуривать глаза.

- Франк (Франкенштейн) — первый биоробот, созданный в 2011 году Бертолтом Мейером из Цюрихского университета.

## Киборги

Киборг (сокращение от англ. *cybernetic organism* — кибернетический организм) — в медицине — биологический организм, содержащий механические или электронные компоненты, машинно-человеческий гибрид (в научной фантастике, гипотетике и т. п.), и неспособный жить без этих механических или электронных компонентов.

Концепция человека-машины присутствовала как образ в мифологии и творчестве во все времена, начиная с искусственного гиганта Талоса из греческих мифов и заканчивая Железным Гансом из одноименной сказки братьев Гримм. В англоязычной литературе основоположником научного представления человека с механическими компонентами считается Эдгар Аллан По — в своём рассказе «Человек, которого изрубили в куски» (1843) он описал пожилого офицера, получившего тяжелейшие ранения иувечья во время войны с индейцами и практически целиком (кроме тела, одной руки и головы) состоящего из протезов. В некотором смысле андроидом можно назвать градоначальника Брудастого из «Истории одного города» Салтыкова-Щедрина (1869—1870). Во французской литературе популярностью пользовалась серия романов Жана де ла Хира о Никталопе (1908—1944), часто так же рассматриваемого как первого литературного супергероя. Сам термин «киборг» был введён Манфредом Е. Клайнсом и Натаном С. Клином в 1960 году, в связи с их концепцией расширения возможностей человека для выживания вне Земли. Эта концепция являлась результатом размышлений на тему необходимости более близких, интимных отношений между человеком и машиной, по мере того как космические исследования становятся реальностью.

Возрастание зависимости человека от механизмов, а также замена органов механическими приспособлениями (протезами, имплантатами) создаёт условия для постепенного превращения человека в киборга. В технике человек проецирует себя, поэтому совместная эволюция человека и техники в киборга — процесс объективный.

- Повсеместно применяются кохлеарные имплантаты, позволяющие восстановить слух пациентам с выраженной или тяжёлой потерей слуха сенсоневральной этиологии. Проводятся эксперименты с применением стволовых слуховых имплантатов, позволяющих восстановить слух некоторым пациентам с глухотой невральной этиологии.
- Специалисты из Института реабилитации инвалидов в Чикаго (США) успешно имплантировали бионическую руку женщине по имени Клодия Митчел, потерявшей свою руку в дорожной аварии. До этого подобные манипуляторы были успешно имплантированы пяти мужчинам.
- Сегодня система C-LEG используется для замены ампутированных человеческих ног. Значительный эффект оказывает использование сенсоров в искусственных ногах. Это один из первых шагов к киборгизации.
- В 2008 году немецкие учёные-офтальмологи впервые имплантировали человеку глазной электронный протез, полностью помещающийся внутри глаза, добившись частичного

восстановления зрения. Ранее все экспериментальные имплантаты, частично восстанавливающие зрительную функцию человека, имели массивные внешние элементы.

- В 2009 году агентство по перспективным оборонным научно-исследовательским разработкам США продемонстрировало радиоуправляемых жуков, в нервные узлы которых были вживлены электроды. Средняя продолжительность управляемого полёта составляла 45 секунд, но один из экземпляров управлялся около 30 минут.
- Весной 2011 года хирурги провели уникальную операцию: искусственное сердце нового типа полностью заменило собой настоящее, но пациент Крейг Льюис не прожил долго, он умер через месяц от амилоидоза.
- В 2013 году биохакер Тим Кэннон (Tim Cannon) с помощью своего друга ввел чип непосредственно под кожу (без анестезии). Чип под названием Circadia 1.0 может записывать данные из тела Кэннона и передавать их на любое мобильное устройство с ОС Android.

## Нанороботы

Нанороботы, или наноботы — роботы, размером сопоставимые с молекулой (менее 10 нм), обладающие функциями движения, обработки и передачи информации, исполнения программ.

Нанороботы, способные к созданию своих копий, то есть самовоспроизведству, называются репликаторами. Возможность создания нанороботов рассмотрел в своей книге «Машины создания» американский учёный Эрик Дрекслер.

Другие определения описывают наноробота как машину, способную точно взаимодействовать с наноразмерными объектами или способной манипулировать объектами в наномасштабе. Вследствие этого, даже крупные аппараты, такие как атомно-силовой микроскоп можно считать нанороботами, так как он производит манипуляции объектами наnanoуровне. Кроме того, даже обычных роботов, которые могут перемещаться с наноразмерной точностью, можно считать нанороботами.

Кроме слова «наноробот» также используют выражения «нанит» и «наноген», однако, технически правильным термином в контексте серьёзных инженерных исследований все равно остается первый вариант.

На данный момент (2015 год), нанороботы находятся в научно-исследовательской стадии создания. Некоторыми учёными утверждается, что уже созданы некоторые компоненты нанороботов. Разработке компонентов наноустройств и непосредственно нанороботам посвящён ряд международных научных конференций.

Уже созданы некоторые примитивные прототипы молекулярных машин. Например, датчик, имеющий переключатель около 1,5 нм, способный вести подсчет отдельных молекул в химических образцах. Недавно университет Райса продемонстрировал наноустройства для использования их в регулировании химических процессов в современных автомобилях.

Одним из самых сложных прототипов наноробота является «DNA box», созданный в конце 2008 года международной группой под руководством Йоргена Кьемса. Устройство имеет подвижную часть, управляемую с помощью добавления в среду специфических фрагментов ДНК. По мнению Кьемса, устройство может работать как «ДНК-компьютер», т.к на его базе возможна реализация

логических вентиляй. Важной особенностью устройства является метод его сборки, так называемый ДНК оригами (англ.), благодаря которому устройство собирается в автоматическом режиме.

В 2010 году были впервые продемонстрированы нанороботы на основе ДНК, способные перемещаться в пространстве.

### Квантовые роботы

Квантовый робот — гипотетическое квантовое устройство, представляющее собой подвижную квантовую наносистему со встроенным квантовым компьютером и системами взаимодействия с окружающей средой. Первую модель квантового робота предложил Поль Бенёв в 1998 году.

Квантовые роботы предназначены для того, чтобы изменять или измерять квантовые состояния окружающей среды. В простейших моделях квантовых роботов окружающая среда представляется как квантовый оракул, база данных или квантовый регистр. Квантовый робот и его взаимодействие с окружающей средой описывается некоторой последовательностью, сменяющих друг друга вычислений и действий. Для гамильтоновых моделей роботов эта последовательность описывается унитарным оператором.

Вычислительная фаза предназначена для определения квантовым вычислением следующего действия и генерации нового конечного квантового состояния. При этом входные данные состоят из старых конечных квантовых состояний, данных из памяти квантового компьютера и базы данных наблюдений за состоянием окружающей среды. Фаза действий предназначена для совершения движения квантового робота и изменения состояния окружающей среды, которые задаются вычисленным новым конечным квантовым состоянием. Во время фазы действия квантовые состояния систем квантового робота не изменяются. Предполагается, что в структуре квантового робота присутствует контрольный кубит, в функции которого входит производить переключения между фазами вычислений и фазами действий. В гамильтоновых моделях с каждой фазой связывают определенные унитарные операторы, описывающие изменение общего квантового состояния окружающей среды и квантового робота.

### Роботизированные протезы

Активно развивается сфера роботизированных протезов. С каждым годом они становятся всё совершеннее.

В 2014 году в Университете Джонса Хопкинса (США) проведена уникальная операция: впервые пациент с ампутированными на уровне плеч руками смог одновременно управлять двумя роботизированными конечностями.

Лес Бау (Les Baugh) из Колорадо потерял обе руки ещё в 70-х годах в результате серьёзной травмы электрическим током. Теперь, спустя четыре десятилетия, он получил возможность самостоятельно брать предметы и выполнять несложные действия.

Операция потребовала восстановления нарушенной иннервации хирургическими методами. Затем в течение определённого времени господин Бау проходил реабилитацию и тренировку на специальном компьютерном симуляторе, который, реагируя на сигналы от сохранившихся нервов, позволял управлять виртуальными протезами на мониторе. После этого наступил решающий этап

— использование высокотехнологичных моторизованных конечностей, полностью заменяющих руки.

Уже сейчас при помощи «силы мысли» Лес Боу может выполнять такие действия, как захват и перемещение определённых предметов. При этом протезы обеспечивают большое количество степеней свободы. Учёные надеются, что с течением времени пациент сможет осуществлять гораздо более сложные действия и более точно управлять роботизированными кистями и пальцами.

В 2015 компания молодых разработчиков из Новосибирска создала технологию производства роботизированного протеза кисти, который будет втрое дешевле немецкого и в семь раз дешевле английского аналога. Это стало возможно благодаря отказу от дорогостоящих материалов. Карбон и титан новосибирские разработчики заменили полимерами и более дешевыми металлическими сплавами. Кроме того, в производстве используется 3D-печать.

Также счастливым обладателем роботизированного протеза стал американец Зак Воутер, который потерял ногу в аварии. Сейчас он перемещается по городу без задержек со скоростью среднестатистического пешехода. Чтобы перестроить роботизированную ногу от хождения по ровным поверхностям к подъёму по лестницам, ему достаточно просто подумать об этом и представить, что он управляет утраченной ногой. Контроль над роботизированной конечностью осуществляется при помощи электромиографии — электрических сигналов, производимых мышцами. Электроды приживаются к девяти различным мышцам в бедренной части ноги и действуют как антенны, принимая электрические сигналы, посылаемые нервами в мышцах. Компьютерные программы распознают эти сигналы и запускают соответствующие электродвигатели, заставляя ногу согнуться в колене или голеностопном суставе. Часть информации, необходимой для правильного выполнения движения, поступает от механических сенсоров на искусственной конечности.

### Зал славы роботов

Зал славы роботов (англ. The Robot Hall of Fame) был основан в 2003 Школой компьютерных наук, входящей в состав университета Карнеги — Меллона (Питсбург, США). Целью Robot Hall of Fame является увековечивание роботов, представляющих собой достижения в робототехнике, и образов роботов из научно-фантастической литературы, вдохновивших разработку настоящих роботов.

Номинировать робота на включение в Зал славы может любой желающий. Решение о включении принимает жюри, состоящее из учёных, исследователей, писателей и конструкторов. Первые представители пантеона роботов были объявлены на специальной церемонии, состоявшейся 10 ноября 2003 в Carnegie Science Center (Питсбург).

Работы в Зале славы:

### 2003

Вымышленные:

- HAL 9000 — компьютерная система из романа Артура Кларка Космическая одиссея 2001 года.
- R2-D2 — персонаж Звёздных войн.

Реальные:

- Sojourner — марсоход, исследовавший химический состав марсианских пород в 1997 году в рамках миссии Mars Pathfinder.
- Unimate — первый промышленный робот, который начал работать на конвейере General Motors в 1961 году.

## 2004

Вымышленные:

- Astro Boy — робот-мальчик из одноимённой манги.
- C-3PO — персонаж Звёздных войн.
- Robby, the Robot — робот из фильма Запретная планета.
- NS-4, NS-5 — роботы из фильма Я, Робот.

Реальные:

- ASIMO — робот-андроид, созданный компанией Honda.
- Shakey — первый робот, «рассуждающий» о своих действиях.

## 2006

Вымышленные:

- Горт — робот из фильма День, когда остановилась Земля.
- Мария — робот из фильма Метрополис.
- Дэвид — мальчик-андроид из фильма Искусственный разум.

Реальные:

- AIBO — робо-собака производства Sony.
- SCARA — Selective Compliance Assembly Robot Arm.

## 2007-2008

Вымышленные:

- Дейта — персонаж сериала Звёздный путь: Следующее поколение.

Реальные:

- Lego Mindstorms NXT — робо-конструктор от Lego.
- Navlab.
- Marc Raibert's Hopper.

## 2009

Вымышленные:

- Терминатор T-800.
- Хьюи, Дьюи и Луи из фильма Молчаливый бег.

Реальные:

- Марсоходы Спирит и Оппортьюнити.
- Da Vinci — аппарат для проведения хирургических операций.
- Roomba — роботизированный пылесос от iRobot.

## Устройство роботов

### Приводы

Приводы — это «мышцы» роботов. В настоящее время самыми популярными двигателями в приводах являются электрические, но применяются и другие, использующие химические вещества или сжатый воздух.

Двигатели постоянного тока: В настоящий момент большинство роботов используют электродвигатели, которые могут быть нескольких видов.

Шаговые электродвигатели: Как можно предположить из названия, шаговые электродвигатели не вращаются свободно, подобно двигателям постоянного тока. Они поворачиваются пошагово на определённый угол под управлением контроллера. Это позволяет обойтись без датчика положения, так как угол, на который был сделан поворот, заранее известен контроллеру; поэтому такие двигатели часто используются в приводах многих роботов и станках с ЧПУ.

Пьезодвигатели: Современной альтернативой двигателям постоянного тока являются пьезодвигатели, также известные как ультразвуковые двигатели. Принцип их работы весьма оригинален: крошечные пьезоэлектрические ножки, вибрирующие с частотой более 1000 раз в секунду, заставляют мотор двигаться по окружности или прямой. Преимуществами подобных двигателей являются высокое нанометрическое разрешение, скорость и мощность, несопоставимая с их размерами. Пьезодвигатели уже доступны на коммерческой основе и также применяются на некоторых роботах.

Воздушные мышцы: Воздушные мышцы — простое, но мощное устройство для обеспечения силы тяги. При накачивании сжатым воздухом мышцы способны сокращаться до 40 % от своей длины. Причиной такого поведения является плетение, видимое с внешней стороны, которое заставляет мышцы быть или длинными и тонкими, или короткими и толстыми. Так как способ их работы схож с биологическими мышцами, их можно использовать для производства роботов с мышцами и скелетом, аналогичными мышцам и скелету животных.

Электроактивные полимеры: Электроактивные полимеры — это вид пластмасс, который изменяет форму в ответ на электрическую стимуляцию. Они могут быть сконструированы таким образом, что могут гнуться, растягиваться или сокращаться. Впрочем, в настоящее время нет ЭАП, пригодных для производства коммерческих роботов, так как все ныне существующие их образцы неэффективны или непрочны.

Эластичные нанотрубки: Это — многообещающая экспериментальная технология, находящаяся на ранней стадии разработки. Отсутствие дефектов в нанотрубках позволяет волокну эластично деформироваться на несколько процентов. Человеческий бицепс может быть заменён проводом из такого материала диаметром 8 мм. Подобные компактные «мышцы» могут помочь роботам в будущем обгонять и перепрыгивать человека.

### [Способы передвижения](#)

Для передвижения по открытой местности чаще всего используют колёсный или гусеничный движитель (примерами подобных роботов могут служить Warrior и PackBot). Реже используются шагающие системы (примерами подобных роботов могут служить BigDog и Asimo). Для неровных поверхностей создаются гибридные конструкции, сочетающие колёсный или гусеничный ход со сложной кинематикой движения колёс. Такая конструкция была применена в луноходе.

Внутри помещений, на промышленных объектах роботы передвигаются вдоль монорельсов, по напольной колее и т. д. Для перемещения по наклонным или вертикальным плоскостям, по трубам используются системы, аналогичные «шагающим» конструкциям, но с вакуумными присосками. Роботы, предназначенные для обследования высоковольтных линий электропередач, имеют в своей верхней части колёсные шасси, перемещающиеся по проводам. Также известны роботы, использующие принципы движения живых организмов — змей, червей, рыб, птиц, насекомых и других; соответственно, говорят о ползающих, инсектоморфных и других типах роботов бионического происхождения.

Системы построения модели окружающего пространства по ультразвуку или сканированием лазерным лучом широко используются в гонках роботизированных автомобилей (которые уже успешно и самостоятельно проходят реальные городские трассы и дороги на пересечённой местности с учётом неожиданно возникающих препятствий).

### [Системы управления](#)

Под управлением роботом понимается решение комплекса задач, связанных с адаптацией робота к кругу решаемых им задач, программированием движений, синтезом системы управления и её программного обеспечения.

По типу управления робототехнические системы подразделяются на:

#### 1. Биотехнические:

- командные (кнопочное и рычажное управление отдельными звенями робота);
- копирующие (повтор движения человека, возможна реализация обратной связи, передающей прилагаемое усилие, экзоскелеты);
- полуавтоматические (управление одним командным органом, например, рукояткой всей кинематической схемой робота);

#### 2. Автоматические:

- программные (функционируют по заранее заданной программе, в основном предназначены для решения однообразных задач в неизменных условиях окружения);
- адаптивные (решают типовые задачи, но адаптируются под условия функционирования);
- интеллектуальные (наиболее развитые автоматические системы);

#### 3. Интерактивные:

- автоматизированные (возможно чередование автоматических и биотехнических режимов);
- супервизорные (автоматические системы, в которых человек выполняет только целеуказательные функции);

- диалоговые (робот участвует в диалоге с человеком по выбору стратегии поведения, при этом как правило робот оснащается экспертной системой, способной прогнозировать результаты манипуляций и дающей советы по выбору цели).

Среди основных задач управления роботами выделяют такие:

- планирование положений;
- планирование движений;
- планирование сил и моментов;
- анализ динамической точности;
- идентификация кинематических и динамических характеристик робота.

В развитии методов управления роботами огромное значение имеют достижения технической кибернетики и теории автоматического управления.

Помимо уже широко применяющихся нейросетевых технологий, существуют алгоритмы самообучения взаимодействию робота с окружающими предметами в реальном трёхмерном мире: робот-собака Aibo под управлением таких алгоритмов прошел те же стадии обучения, что и новорожденный младенец, самостоятельно научившись координировать движения своих конечностей и взаимодействовать с окружающими предметами (погремушками в детском манеже). Это дает ещё один пример математического понимания алгоритмов работы высшей нервной деятельности человека.

## Интернет вещей

### Введение

В 1926 году Никола Тесла в интервью для журнала «Collier's» сказал, что в будущем радио будет преобразовано в «большой мозг», все вещи станут частью единого целого, а инструменты, благодаря которым это станет возможным, будут легко помещаться в кармане. В 1990 году выпускник MIT (Massachusetts Institute of Technology), один из отцов протокола TCP/IP, Джон Ромки подключил к сети свой тостер. Тостер мог удаленно включаться и сообщать о готовности тоста. Это было первым проявлением интернета вещей.

Интернет вещей (*Internet of Things, IoT*) — концепция вычислительной сети физических объектов («вещей»), оснащённых встроенными технологиями для взаимодействия друг с другом или с внешней средой, рассматривающая организацию таких сетей как явление, способное перестроить экономические и общественные процессы, исключающее из части действий и операций необходимость участия человека. Сам термин «Интернет вещей» (*Internet of Things*) был предложен Кевином Эштоном в 1999 году. В этом же году был создан Центр автоматической идентификации (Auto-ID Center), занимающийся радиочастотной идентификацией (RFID) и сенсорными технологиями, благодаря которому эта концепция и получила широкое распространение. В 2008–2009 по данным доклада компании Cisco произошел переход от «Интернета людей» к «Интернету вещей», т.е. количество подключенных к сети предметов превысило количество людей на планете Земля.

### Технологии

Говоря об интернете вещей, необходимо понимать, что это не только множество различных приборов и датчиков, объединенных между собой проводными и беспроводными каналами связи

и подключенных к сети Интернет, но еще и более тесная интеграция реального и виртуального миров, в котором общение производится между людьми и устройствами. Как достичь этого технически? Во-первых, мультиагентные технологии — они уже везде и всюду, и интернет вещей без них невозможен. Каждому участнику из реального мира (т.е. каждому человеку и каждому устройству) ставится в соответствие программный агент — объект с некоторой степенью интеллектуальности, представляющий его интересы в мире виртуальном. Как живут и работают агенты

Жизненный цикл агентов довольно прост. Сначала они воспринимают информацию из внешнего мира. Потом ее нужно обработать, т.е. запланировать некие действия. Ну а действия уже нужно выполнить — отдав соответствующие команды в реальный мир. Получается, что в нашем “умном” доме агент человека постоянно общается с агентами кофеварки, лампочек и так далее, отдавая им команды и обмениваясь информацией.

Данную концепцию связывают, как правило, с развитием двух технологий. Это радиочастотная идентификация (RFID) и беспроводные сенсорные сети (БСС).

#### *RFID*

RFID (англ. Radio Frequency Identification, радиочастотная идентификация) — метод автоматической идентификации объектов, в котором посредством радиосигналовчитываются или записываются данные, хранящиеся в так называемых транспондерах, или RFID-метках.

Данная технология хорошо подходит для отслеживания движения некоторых объектов и получения небольшого объема информации от них. Так, например, если бы все продукты были оснащены RFID-метками, а холодильник RFID-ридером, то он легко мог бы отслеживать срок годности продуктов, а мы могли бы, например, уходя с работы удаленно заглянуть в холодильник и определить, что надо закупить еще.

#### *Беспроводные сенсорные сети*

Беспроводная сенсорная сеть — это распределенная самоорганизуемая сеть множества датчиков (сенсоров) и исполнительных устройств, объединенных между собой посредством радиоканала. Причем область покрытия подобной сети может составлять от нескольких метров до нескольких километров за счет способности ретрансляции сообщений от одного элемента к другому.

Применяется данная технология для решения многих практических задач связанных с мониторингом, управлением, логистикой и так далее.

#### *Проблема с идентификацией*

Для объектов, непосредственно подключённых к интернет-сетям традиционный идентификатор — MAC-адрес сетевого адаптера, позволяющий идентифицировать устройство на канальном уровне, при этом диапазон доступных адресов практически исчерпаем ( $2^{48}$  адресов в пространстве MAC-48). Использование идентификатора канального уровня не слишком удобно для приложений. Более того, согласно прогнозам компании Cisco, к 2020 году к сети Интернет будет подключено свыше 50 миллиардов устройств, что вызывает проблемы с идентификацией. Проблему призван решить протокол IPv6, обеспечивающий уникальными адресами сетевого уровня не менее 300 млн устройств на одного жителя Земли (2 в 128).

## *Протоколы*

Говоря о протоколах, необходимо понимать специфику. Устройства (D) должны устанавливать друг с другом связь (D2D). Затем нужно собрать и передать данные с этих устройств в серверную (S) инфраструктуру (D2S). Эта серверная инфраструктура должна совместно использовать данные (S2S), имея возможность передавать данные обратно устройствам, программам анализа или людям. Можно выделить следующие протоколы для решения задач в этой инфраструктуре:

- MQTT: протокол для сбора данных устройств и передачи их серверам (D2S);
- XMPP: протокол для соединения устройств с людьми, частный случай D2S-схемы, когда люди соединяются с серверами;
- DDS: быстрая шина для интегрирования интеллектуальных устройств (D2D);
- AMQP: система организации очередей для соединения серверов между собой (S2S).

Каждый из этих протоколов широко распространён. Есть по крайней мере 10 вариантов реализации каждого из них. По сути, все четыре вышеперечисленных протокола представляют собой протоколы «Интернета вещей» реального времени с публикацией/подпиской, которые способны соединять тысячи устройств.

На самом деле эти протоколы очень разные. Современный Интернет поддерживает сотни протоколов. «Интернет вещей» будет поддерживать ещё на сотни протоколов больше.

## *Инструменты и продукты*

Наиболее известной компанией, которая смогла успешно построить бизнес на интернете вещей, является Nest. Самым известным продуктом этой компании стал термостат. Помимо термостатов и иной бытовой техники компания организовала программу Works with Nest. Таких интеграций «Works with Nest» уже десяток. Среди участников, например, компании Jawbone (термостат ориентируется на данные браслета и регулирует температуру в зависимости от того, спите вы или бодрствуете), Whirlpool (стиральная машина выбирает режим работы, опираясь на данные о том, дома вы или нет) и Mercedes-Benz, Chamberlain (ворота для гаражей), а также производитель периферийных устройств Logitech и сервис IFTTT. В 2014 году компанию Nest приобрел интернет-гигант Google за \$3.2 миллиарда. И уже через год, на конференции Google I/O 2015 были представлены проекты Project Brillo и Google Weave.

## *Project Brillo*

Brillo — название губки для мытья посуды. Звучит немного странно, но Google объясняет это тем, что Project Brillo очищает Android до основания.

Project Brillo — платформа на Android, которая, по планам Google, окажется на всех видах умных устройств. На самом деле, у большинства из этих устройств не будет мощных процессоров и памяти для работы. У многих даже не будет экрана. Поэтому Project Brillo должен быть очень легким и нетребовательным. Google также выделяет «широкую поддержку железа». Это основа для расширения и гибкости, чтобы позволить производителям устройств использовать любые чипы широкого круга производителей, что оградит рынок от монополии и позволит снизить минимальную цену поддерживаемых устройств. Также большой акцент был сделан на безопасность. Project Brillo будет «удобен для защиты». Это, возможно, самая критичная из всех целей.

Проект Brillo является проектом с открытым исходным кодом с 20 ноября 2015 года. Подробности можно увидеть здесь: [android.googlesource.com/brillo/manifest/](http://android.googlesource.com/brillo/manifest/)

#### *Google Weave*

На I/O 2015 Google представила также специальный язык, который разработчики будут использовать для общения с устройствами Brillo. Он называется Weave.

Weave — библиотека определений и команд, которые будут использоваться на поддерживаемых устройствах. Это общий язык для всех устройств Интернета вещей, и разработчики смогут добавлять свои термины в язык.

#### *Другие*

Важность и перспективность направления интернета вещей подчеркивает то, что кроме компании Google, известной своей привычкой пробовать и открывать нетронутые рынки, свой вклад внесли такие важные игроки как Apple и Samsung. Apple летом того же года, когда Google приобрела Nest, на своей ежегодной конференции для разработчиков WWDC (аналог Google I/O) представила HomeKIT - инструмент, позволяющий разработчикам разрабатывать приложения, осуществляющие контроль и общее взаимодействие с "умными" вещами.

Южнокорейский гигант тоже не заставил долго ждать: в сентябре 2015 года, на ежегодной конференции IFA в Берлине, Samsung показал большое количество умных вещей, от розеток и стиральных машин, до анализаторов сна, а также сервисы, позволяющие удаленно запускать или останавливать двигатель автомобиля, управлять климат-контролем автомобиля и напоминать о закрытии автомобиля.

#### *Безопасность*

Безусловно очень важным аспектом любого ПО является безопасность. В интернете вещей этот вопрос стоит особенно остро. Так в отчёте Национального разведывательного совета США (National Intelligence Council) 2008 года «интернет вещей» фигурирует как одна из шести потенциально разрушительных технологий, указывается, что повсеместное и незаметное для потребителей превращение в интернет-узлы таких распространённых вещей, как товарная упаковка, мебель, бумажные документы, может нанести урон национальной информационной безопасности.

Открытость системы Project Brillo даст должный прирост безопасности, но это не снимает ответственности с разработчика: к вопросу безопасности такого рода систем необходимо подходить ответственно еще на моменте проектирования.

#### *Заключение*

В заключение хотелось бы привести отрывок диалога из книги "Рассветники" Юрия Никитина:

*"— Да не сломалось, — с неохотой выговорил он, — а... понимаешь, у меня температура чуть-чуть ниже нормы. Не тридцать шесть и шесть, а тридцать шесть и одна десятая. Ну, есть такие люди, два-три на миллион, это тоже как бы норма, хоть и на самом краю. Но этот дурацкий умный дом требует, чтобы я принял какие-то таблетки!.. Теперь надо либо отключить эту систему, либо перепрограммировать, а то будет звонить и на работу, он уже так делал на прошлой неделе, когда узнал, что у меня запор, в офисе теперь даже пылесосы ржут, как только захожу..."*

## Облачные вычисления

Почему облачные вычисления лучше, чем классическая схема построения сетевой инфраструктуры, какова основная причина того, многие организации „перебираются в облака“?». Далее мысли по этому поводу.

### История и ключевые факторы развития

В первые идеи того, что мы сегодня называем облачными вычислениями, была озвучена J.C.R. Licklider, в 1970 году. В эти годы он был ответственным за создание ARPANET (Advanced Research Projects Agency Network). Его идея заключалась в том, что каждый человек на земле будет подключен к сети, из которой он будет получать не только данные но и программы. Другой ученый John McCarthy высказал идею о том, что вычислительные мощности будут предоставляться пользователям как услуга (сервис). На этом развитие облачных технологий было приостановлено до 90-х годов, после чего ее развитию поспособствовал ряд факторов.

1. Расширение пропускной способности Интернета, в 90-е годы не позволило получить значительного скачка в развитии в облачной технологии, так как практически ни одна компания не технологии того времени не были готовы к этому. Однако сам факт ускорения Интернета дал толчок скорейшему развитию облачных вычислений.
2. Одним из наиболее значимых событий в данной области было появление Salesforce.com в 1999 году. Данная компания стала первой компанией, предоставившей доступ к своему приложению через сайт, по сути данная компания стала первой компанией, предоставившей свое программное обеспечение по принципу – программное обеспечение как сервис (SaaS).
3. Следующим шагом стала разработка облачного веб-сервиса компанией Amazon в 2002 году. Данный сервис позволял хранить, информацию и производить вычисления.
4. В 2006, Amazon запустила сервис под названием Elastic Compute cloud (EC2), как веб-сервис, который позволял его пользователям запускать свои собственные приложения. Сервисы Amazon EC2 и Amazon S3 стали первыми доступными сервисами облачных вычислений.
5. Другая веха в развитие облачных вычислений произошла после создания компанией Google, платформы Google Apps для веб-приложений в бизнес секторе.
6. Значительную роль в развитии облачных технологий сыграли технологии виртуализации, в частности программное обеспечение, позволяющее создавать виртуальную инфраструктуру.
7. Развитие аппаратного обеспечения способствовало не столько быстрому росту облачных технологий, сколько доступности данной технологии для малого бизнеса и индивидуальных лиц. Что касается технического прогресса, то значительную роль в этом сыграло создание многоядерных процессоров и увеличения емкости накопителей информации.

### Облачнее вычисления в настоящее время.

Википедия дает следующее определение облачных вычислений. Облачные вычисления (англ. cloud computing) — технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис. Предоставление пользователю услуг как Интернет-сервис является ключевым. Однако под Интернет-сервисом не стоит понимать доступ к сервису только через Интернет, он может осуществляться также и через обычную локальную сеть с использованием веб-технологий.

Из определения и истории видно, что основой для создания и быстрого развития облачных вычислительных систем послужили крупные интернет сервисы, такие как Google, Amazon и др, а также технический прогресс, что по сути говорит о том, что появление облачных вычислений было всего лишь делом времени. Рассмотрим каким же образом развитие вышеперечисленных направлений позволило облачным системам стать доступнее.

Развитие многоядерных процессоров привело к:

- увеличению производительности, при тех же размерах оборудования;
- снижение стоимости оборудования, как следствие эксплуатационных расходов;
- снижение энергопотребления облачной системы, для большинства ЦОД это действительно проблема при наращивании мощностей ЦОД.

Увеличение емкостей носителей информации, снижение стоимости хранения 1 Мб информации позволило:

- безгранично (по крайней мере так позиционируют себя большинство «облаков») увеличить объемы хранимой информации;
- снизить стоимость обслуживания хранилищ информации, значительно увеличив объемы хранимых данных.

Развитие технологии многопоточного программирования привело к:

- эффективному использованию вычислительных ресурсов многопроцессорных систем;
- гибкое распределение вычислительных мощностей облаков.

Развитие технологий виртуализации привело к:

- созданию программного обеспечения, позволяющего создавать виртуальную инфраструктуру независимо от количества предоставленных аппаратных ресурсов;
- легкость масштабирования, наращивания систем;
- уменьшение расходов на администрирование облачных систем;
- доступность виртуальной инфраструктуры через сеть Интернет.

Увеличение пропускной способности привело к:

- увеличению скорости работы с облачными системами в частности виртуальный графический интерфейс и работа с виртуальными носителями информации;
- снижение стоимости Интернет трафика для работы с большими объемами информации;
- проникновению облачных вычислений в массы.

Все вышеперечисленные факторы привели к повышению конкурентоспособности облачных вычислений в ИТ сфере.

### Достоинства облачных вычислений

- **доступность** – облака доступны всем, из любой точки, где есть Интернет, с любого компьютера, где есть браузер. Это позволяет пользователям (предприятиям) экономить на закупке высокопроизводительных, дорогостоящих компьютеров. Также сотрудники компаний становятся более мобильными так, как могут получить доступ к своему рабочему

месту из любой точки земного шара, используя ноутбук, нетбук, планшетник или смартфон. Нет необходимости в покупки лицензионного ПО, его настройки и обновлении, вы просто заходите на сервис и пользуетесь его услугами заплатив за фактическое использование.

- **низкая стоимость** – основные факторы, снизившие стоимость использования облаков следующие:
  - снижение расходов на обслуживания виртуальной инфраструктуры, вызванное развитием технологий виртуализации, за счет чего требуется меньший штат для обслуживания всей ИТ инфраструктуры предприятия;
  - оплата фактического использования ресурсов, пользователь облака платит за фактическое использование вычислительных мощностей облака, что позволяет ему эффективно распределять свои денежные средства. Это позволяет пользователям (предприятиям) экономить на покупке лицензий к ПО;
  - использование облака на правах аренды позволяет пользователям снизить расходы на закупку дорогостоящего оборудования, и сделать акцент на вложение денежных средств на наладку бизнес процессов предприятия, что в свою очередь позволяет легко начать бизнес;
  - развитие аппаратной части вычислительных систем, в связи с чем снижение стоимости оборудования.
- **гибкость** — неограниченность вычислительных ресурсов (память, процессор, диски), за счет использования систем виртуализации, процесс масштабирования и администрирования «облаков» становиться достаточно легкой задачей, так как «облако» самостоятельно может предоставить вам ресурсы, которые вам необходимы, а вы платите только за фактическое их использование.
- **надежность** – надежность «облаков», особенно находящихся в специально оборудованных ЦОД, очень высокая так, как такие ЦОД имеют резервные источники питания, охрану, профессиональных работников, регулярное резервирование данных, высокую пропускную способность Интернет канала, высокая устойчивость к DDOS атакам. Безопасность – «облачные» сервисы имеют достаточно высокую безопасность при должном ее обеспечении, однако при халатном отношении эффект может быть полностью противоположным.  
Большие вычислительные мощности – вы как пользователь «облачной» системы можете использовать все ее вычислительные способности, заплатив только за фактическое время использования. Предприятия могут использовать данную возможность для анализа больших объемов данных.

## Недостатки

- **постоянное соединение с сетью** – для получения доступа к услугам «облака» необходимо постоянное соединение с сетью Интернет. Однако в наше время – это не такой и большой недостаток особенно с приходом технологий сотовой связи 3G и 4G.
- **программное обеспечение и его кастомизация** – есть ограничения по ПО которое можно разворачивать на «облаках» и предоставлять его пользователю. Пользователь ПО имеет ограничения в используемом ПО и иногда не имеет возможности настроить его под свои собственные цели.
- **конфиденциальность** – конфиденциальность данных хранимых на публичных «облаках» в настоящее вызывает много споров, но в большинстве случаев эксперты сходятся в том, что

не рекомендуется хранить наиболее ценные для компании документы на публичном “облаке”, так как в настоящее время нет технологии, которая бы гарантировала 100% конфиденциальность хранимых данных.

- **надежность** – что касается надежности хранимой информации, то с уверенностью можно сказать что если вы потеряли информацию, хранимую в “облаке”, то вы ее потеряли навсегда.
- **безопасность** – “облако” само по себе является достаточно надежной системой, однако при проникновении на него злоумышленник получает доступ к огромному хранилищу данных. Еще один минус — это использование систем виртуализации, в которых в качестве гипервизора используются ядра стандартные ОС такие, как Linux, Windows и др., что позволяет использовать вирусы.
- **дороговизна оборудования** – для построения собственного облака компании необходимо выделить значительные материальные ресурсы, что не выгодно только что созданным и малым компаниям.

#### Виды услуг, предоставляемые облачными системами

Что касается предоставляемых услуг, то в настоящее время концепция облачных вычислений предполагает оказание следующих типов услуг своим пользователям:

- **все как услуга (Everything as a Service);**  
При таком виде сервиса пользователю будет предоставлено все от программно аппаратной части и до управления бизнес процессами, включая взаимодействие между пользователями, от пользователя требуется только наличие доступа в сеть Интернет. На мой взгляд, данный вид сервиса — это более общее понятие по отношению к нижеприведенным услугам, являющимися более частными случаями.
- **инфраструктура как услуга (Infrastructure as a service);**  
Пользователю предоставляется компьютерная инфраструктура, обычно виртуальные платформы (компьютеры) связанные в сеть. Которые он самостоятельно настраивает под собственные цели.
- **платформа как услуга (Platform as a service);**  
Пользователю предоставляется компьютерная платформа, с установленной операционной системой возможно и с программным.
- **программное обеспечение как услуга (Software as a service);**  
Данный вид услуги обычно позиционируется как «программное обеспечение по требованию», это программное обеспечение, развернутое на удаленных серверах и пользователь может получать к нему доступ посредством Интернета, причем все вопросы обновления и лицензий на данное программное обеспечение регулируется поставщиком данной услуги. Оплата в данном случае производиться за фактическое использование программного обеспечения.
- **аппаратное обеспечение как услуга (Hardware as a Service);**  
В данном случае пользователю услуги предоставляется оборудование, на правах аренды которое он может использовать для собственных целей. Данный вариант позволяет экономить на обслуживании данного оборудования, хотя по своей сути мало чем отличается от вида услуги «Инфраструктура как сервис» за исключением того, что вы имеете

голое оборудование на основе которого разворачиваете свою собственную инфраструктуру с использованием наиболее подходящего программного обеспечения.

- **рабочее место как услуга (Workplace as a Service);**

В данном случае компания использует облачные вычисления для организации рабочих мест своих сотрудников, настроив и установив все необходимое программное обеспечение, необходимое для работы персонала.

- **данные как услуга (Data as a Service);**

Основная идея данного вида услуги заключается в том, что пользователю предоставляется дисковое пространство, которое он может использовать для хранения больших объемов информации.

- **безопасность как сервис (Security as a Service).**

Данный вид услуги предоставляет возможность пользователям быстро развертывать, продукты, позволяющие обеспечить безопасное использование веб-технологий, безопасность электронной переписки, а также безопасность локальной системы, что позволяет пользователям данного сервиса экономить на развертывании и поддержании своей собственной системы безопасности.

## Классификация облачных сервисов.

В настоящее время выделяют три категории «облаков»:

1. Публичные;
2. Частные;
3. Гибридные.

**Публичное облако** — это ИТ-инфраструктура используемое одновременно множеством компаний и сервисов. Пользователи данных облаков не имеют возможности управлять и обслуживать данное облако, вся ответственность по этим вопросам возложена на владельца данного облака. Абонентом предлагаемых сервисов может стать любая компания и индивидуальный пользователь. Они предлагают легкий и доступный по цене способ развертывания веб-сайтов или бизнес-систем, с большими возможностями масштабирования, которые в других решениях были бы недоступны. Примеры: онлайн сервисы Amazon EC2 и Simple Storage Service (S3), Google Apps/Docs, Salesforce.com, Microsoft Office Web.

**Частное облако** — это безопасная ИТ-инфраструктура, контролируемая и эксплуатируемая в интересах одной-единственной организации. Организация может управлять частным облаком самостоятельно или поручить эту задачу внешнему подрядчику. Инфраструктура может размещаться либо в помещениях заказчика, либо у внешнего оператора, либо частично у заказчика и частично у оператора. Идеальный вариант частного облака — это облако, развернутое на территории организации, обслуживаемое и контролируемое ее сотрудниками.

**Гибридное облако** — это ИТ-инфраструктура использующая лучшие качества публичного и приватного облака, при решении поставленной задачи. Часто такой тип облаков используется, когда организация имеет сезонные периоды активности, другими словами, как только внутренняя ИТ-инфраструктура не справляется с текущими задачами, часть мощностей перебрасывается на

публичное облако (например, большие объемы статистической информации, которые в необработанном виде не представляют ценности для предприятия), а также для предоставления доступа пользователям к ресурсам предприятия (к частному облаку) через публичное облако.

### [Куда надо развиваться или на чем можно заработать деньги?](#)

По оценкам экспертов потенциал облачных вычислений очень высок. А соответственно можно будет попасть в этот поток и отхватить его часть работая в следующих направлениях:

1. Предоставление услуг облачных вычислений – данная возможность доступна не многим компаниям так, нужны значительные вложения в построение и разработку ЦОД.
2. Разработка ПО для построения виртуальной инфраструктуры, не следует забывать и про тех, кто будет внедрять и настраивать это ПО, т. е. потребуются специалисты в этой области.
3. Аутсорсинг, администрирование облаков – потребуются специалисты по администрированию и консультированию в сфере облачных вычислений.
4. Аппаратное обеспечение – компании, занимающиеся разработкой и проектированием аппаратного обеспечения для создания «облаков».
5. Проектирование – данная сфера охватывает практически все вышеперечисленные сферы начиная от проектирования ЦОД и заканчивая проектированием программного обеспечения.

### [Будущее](#)

На мой взгляд, в будущем облачные вычисления будут становиться доступнее для пользователей и компаний. Это будет вызвано рядом факторов:

- аппаратная виртуализация – повышение производительности облачных вычислений;
- снижение энергопотребления аппаратного обеспечения – понижение энергопотребления;
- повышение скоростей – пропускная способность сетевого оборудования постоянно повышается, что увеличивает производительность и уменьшает количество оборудования при том же канале.

### [3D печать](#)

*Будущее уже наступило: если нет возможности что-то купить, это можно просто напечатать.*

### [Определение](#)

Периферийное устройство (англ. peripheral) — аппаратура, которая позволяет вводить информацию в компьютер или выводить её из него.

Периферийные устройства являются не обязательными для работы системы и могут быть отключены от компьютера. Однако большинство компьютеров используются вместе с теми или иными периферийными устройствами.

Периферийные устройства делят на три типа:

- устройства ввода — устройства, использующиеся для ввода информации в компьютер: мышь, клавиатура, тачпад, сенсорный экран, микрофон, сканер, веб-камера, устройство захвата видео, ТВ-тюнер;

- устройства вывода — устройства, служащие для вывода информации из компьютера: видеокарта, монитор, принтер, акустическая система;
- устройства хранения(ввода/вывода) — устройства, служащие для накопления информации, обрабатываемой компьютером: накопитель на жёстких магнитных дисках (НЖМД), накопитель на гибких магнитных дисках (НГМД), ленточный накопитель, USB-флешнакопитель.

3D-принтер — это периферийное устройство, использующее метод послойного создания физического объекта по цифровой 3D-модели.

В зарубежной литературе данный процесс трехмерной печати называют быстрым прототипированием.

### История создания

Первым и самым совершенным 3D-принтером можно считать человека: мысленно воссоздав образ чего-то желаемого, он не пощадил своих рук, чтобы мечта стала реальностью. Так появились, например, пирамиды и компьютеры.

Способности машин трехмерной печати весьма широки: история создания 3D-принтера не насчитывает и 30 лет, но результат эволюции превзошел все ожидания.

Развитие трехмерной технологии было положено в середине 90-х годов: некоторая продукция перестала расходиться миллионными тиражами, а потому потребовалась другая модель производства. Изготовители нуждались в технике, способной выпускать небольшие объемы.

Мелкосерийное производство требовало таких же временных и стоимостных затрат: разработка необходимых форм, приобретение качественного оборудования, создание всех новых моделей для изделий. Чтобы сократить затраты и сделать цену на продукты более приемлемой, применялись станки с программным управлением. Именно из этой ветви станкостроения берут свои истоки первые 3D-принтеры.

В 1986 году Чарльз Халл разработал SLA-установку, которая стала первым прототипом такого принтера. Они использовали стереолитографию, определяя методику 3D-технологии — материал накладывался послойно. После создания установки Чарльз основал фирму с говорящим названием «3D Systems».

Спустя 2 года Скотт Крамп изобрел новую методику создания объемных печатных объектов — наплавление. Именно компания Крампа “Stratasys” стала преемником Чарльза Халла в разработке 3d-принтеров.

Если говорить о том, когда изобрели 3D-принтер, то можно назвать именно 1990 год — Крамп и его супруга называли свою технику не иначе как лазерным и струйным 3D-принтером. Но все же эти модели были далеки от современной 3D-технологии.

В 1993 году была основана компания «Solidscape», которая выпускала струйные принтеры, а в 1995 году американские студенты-изобретатели создали модифицированную версию струйного принтера, который использовал материал не на бумаге, а в емкости. В результате такого метода изготавливались объемные детали.

Как раз после выхода такой модели струйного принтера появилась компания “Z Corporation” (с подачи самих изобретателей) и понятие 3D-печати.

При помощи этой технологии в настоящее время изготавливают принтеры такие производители, как ExOne и Z Corp.

Чуть позже студенческой разработки появилась методика PolyJet, в основе которой лежало использование в качестве материала фотополимерного жидкого пластика (фотополимер наносится печатной головкой и засвечивается лампой).

Методика PolyJet – первый наиболее выигрышный вариант, который дал возможность выпускать готовые к использованию детали с высокой точностью и оптимальной ценой материала.

В основу современных настольных принтеров вошла технология Reprap, которая была создана в 2005 году Адрианом Боуером. Он преподавал машиностроение в одном из университетов Великобритании. Сам проект был направлен на создание самопроизводящих машин, которые будут использовать для печати биоразлагаемые материалы.

Основной идеей стала доступность и повсеместная распространенность создаваемых принтеров благодаря экономичности, простоте и приемлемой цене. В 2006 году разработки Боуера увенчались успехом – прототип RepRap 0.2 напечатал деталь, которая идеально подошла устройству при замещении. Успешный проект стал первым шагом в создании всех настольных FDM-принтеров.

После создания вышеописанных моделей технологии трехмерной печати развивались семимильными шагами – «вырастали» новые фирмы и методики, применялись самые разные материалы. Между тем главными параметрами, которые отличали поколения принтеров, являлись цена и размер.

Собственно, эти же критерии актуальны и сейчас. Более старые модели были крупногабаритными, а стоимость их не укладывалась в бюджет небольшой фирмы – позволить себе их могли только крупные производители.

Сегодня 3D-технологии доступны всем: выпускаются как профессиональные, так и домашние модели высокого качества по бюджетной цене, а кроме того, стандартный размер принтера позволяет поместить его на столе.

### [Солнечный 3D-принтер](#)

Промышленный дизайнер и проектировщик Маркус Кейсер потратил около года на сборку и тестирование двух фантастических приборов, которые используют солнечную энергию, в самых жарких странах. Первый прибор под названием Солнечный фрезер – это несложный лазерный фрезер, использующий огромные круглые линзы для фокусирования солнечных лучей на движущейся поверхности. Во время движения поверхности сфокусированный свет разрезает 2D компоненты, как лазер. Впервые проект был протестирован в августе 2010 года в египетской пустыне, тогда Кейсер использовал тонкую фанеру, из которой он сделал очень интересные очки. Но на этом он не остановился.

Дальше Кейсер начал исследовать процесс 3D-печати. Объединив два самых распространенных в пустыне ресурса, практически неограниченное количество песка и солнца, он сконструировал Солнечный принтер, устройство, переплавляющее песок для создания 3D объектов из стекла.

Этот процесс преобразования сыпучей субстанции под влиянием процесса нагревания в твердый цельный предмет, известен под названием «спекание» в последние годы он стал основным процессом в прототипировании, известный как «3D-печать» или СЛС (селективное лазерное спекание). Используя солнечные лучи, вместо лазера, и песок, вместо смолы, он получил основу для абсолютно нового солнечного устройства и процесса производства стеклянных объектов из самых распространенных компонентов в пустынях по всему миру – песка и солнца.

### Технологии и процесс 3D-печати

3D-печать может осуществляться разными способами и с использованием различных материалов, но в основе любого из них лежит принцип послойного создания (выращивания) твёрдого объекта.

#### Технологии, применяемые для создания слоев

- Лазерная:
  - Лазерная стереолитография — ультрафиолетовый лазер постепенно, пиксель за пиксели, засвечивает жидкий фотополимер, либо фотополимер засвечивается ультрафиолетовой лампой через фотошаблон, меняющийся с новым слоем. При этом жидкий полимер затвердевает и превращается в достаточно прочный пластик.
  - Лазерное сплавление (англ. melting) — при этом лазер сплавляет порошок из металла или пластика, слой за слоем, в контур будущей детали.
  - Ламинарирование — деталь создаётся из большого количества слоёв рабочего материала, которые постепенно накладываются друг на друга и склеиваются, при этом лазер вырезает в каждом контуре сечения будущей детали.
- Струйная:
  - Заствивание материала при охлаждении — раздаточная головка выдавливает на охлаждаемую платформу-основу капли разогретого термопластика. Капли быстро застывают и слипаются друг с другом, формируя слои будущего объекта.
  - Полимеризация фотополимерного пластика под действием ультрафиолетовой лампы — способ похож на предыдущий, но пластик твердеет под действием ультрафиолета.
  - Склейивание или спекание порошкообразного материала — похоже на лазерное спекание, только порошковая основа (подчас на основе измельчённой бумаги или целлюлозы) склеивается жидким (иногда kleющим) веществом, поступающим из струйной головки. При этом можно воспроизвести окраску детали, используя вещества различных цветов. Существуют образцы 3D-принтеров, использующих головки струйных принтеров.
  - Густые керамические смеси тоже применяются в качестве самоотверждаемого материала для 3D-печати крупных архитектурных моделей.
  - Биопринтеры — ранние экспериментальные установки, в которых печать 3D-структур будущего объекта (органа для пересадки) производится каплями, содержащими живые клетки. Далее деление, рост и модификации клеток обеспечивает окончательное формирование объекта.

Также применяются различные технологии позиционирования печатающей головки:

- Декартова, когда в конструкции используются три взаимно-перпендикулярные направляющие, вдоль каждой из которых двигается либо печатающая головка, либо основание модели.
- При помощи трёх параллелограммов, когда три радиально-симметрично расположенных двигателя согласованно смещают основания трёх параллелограммов, прикреплённых к печатающей головке.
- Автономная, когда печатающая головка размещена на собственном шасси, и эта конструкция передвигается целиком за счёт какого-либо движителя, приводящего шасси в движение.
- Ручная, когда печатающая головка выполнена в виде ручки/карандаша, и пользователь сам подносит её в то место пространства, куда считает нужным добавить выделяемый из наконечника быстро затвердевающий материал. Назван такой прибор "3D-ручка", и к 3D-принтерам может быть отнесён с известной натяжкой. Существуют варианты с использованием термополимера, застывающего при охлаждении, и с использованием фотополимера, отверждаемого ультрафиолетом.

Существуют следующие технологии:

- Лазерная стереолитография — объект формируется из специального жидкого фотополимера, затвердевающего под действием лазерного излучения (или излучения ртутных ламп). При этом лазерное излучение формирует на поверхности текущий слой разрабатываемого объекта, после чего объект погружается в фотополимер на толщину одного слоя, чтобы лазер мог приступить к формированию следующего слоя.
- Селективное лазерное спекание — объект формируется из плавкого порошкового материала (пластик, металл) путём его плавления под действием лазерного излучения. Порошкообразный материал наносится на платформу тонким равномерным слоем (обычно специальным выравнивающим валиком), после чего лазерное излучение формирует на поверхности текущий слой разрабатываемого объекта. Затем платформа опускается на толщину одного слоя и на неё вновь наносится порошкообразный материал. Данная технология не нуждается в поддерживающих структурах "висящих в воздухе" элементов разрабатываемого объекта за счёт заполнения пустот порошком. Для уменьшения необходимой для спекания энергии температура рабочей камеры обычно поддерживается на уровне чуть ниже точки плавления рабочего материала, а для предотвращения окисления процесс проходит в бескислородной среде.
- Электронно-лучевая плавка — аналогична первым двум технологиям, только здесь объект формируется путём плавления металлического порошка электронным лучом в вакууме.
- Моделирование методом наплавления — объект формируется путём послойной укладки расплавленной нити из плавкого рабочего материала (пластик, металл, воск). Рабочий материал подаётся в экструзионную головку, которая выдавливает на охлаждаемую платформу тонкую нить расплавленного материала, формируя таким образом текущий слой разрабатываемого объекта. Далее платформа опускается на толщину одного слоя, чтобы можно было нанести следующий слой. Часто в данной технологии участвуют две рабочие головки — одна выдавливает на платформу рабочий материал, другая — материал поддержки.

- Изготовление объектов с использованием ламинации — объект формируется послойным склеиванием (нагревом, давлением) тонких плёнок рабочего материала с вырезанием (с помощью лазерного луча или режущего инструмента) соответствующих контуров на каждом слое. За счет отсутствия пустот данная технология не нуждается в поддерживающих структурах "висящих в воздухе" элементов разрабатываемого объекта, однако, удаление лишнего материала (обычно его разделяют на мелкие кусочки) в некоторых ситуациях может вызывать затруднения.

3D-принтеры работают, принимая 3D компьютерную модель и нарезая её на слои. Затем, 3D-принтер строит объект слой за слоем с помощью одного из нескольких методов.

Большинство домашних и любительских принтеров печатают с использованием моделирования методом наплавления (Fused Deposition Modeling, FDM), которое строит модель из расплавленного пластика. FDM машины подают пластиковую нить в печатающую головку, где пластик плавится и выжимается из сопла, называемого экструдером. Головка вычерчивает контуры каждого слоя, постепенно строя модель из расплавленного пластика.

Качество готовой модели зависит от многих факторов, включая качество основного материала, насколько тонко нарезана модель, механики 3D-принтера и тщательности подготовки компьютерной 3D модели.

Тем не менее, качество 3D моделей, произведённых с помощью технологии FDM, как правило, не соответствует качеству моделей, построенных по некоторым более дорогостоящим технологиям 3D-печати, используемым в промышленности. Одной из таких технологий является лазерное спекание.

Лазерное спекание использует лазер для сплавления порошка в модель. Процесс происходит также очерчиванием контура каждого слоя, и из порошкообразного материала с помощью лазера, сплавляется объект слой за слоем. Лазерное спекание способно воспроизводить мелкие детали и строить модели из большего диапазона материалов, чем FDM: керамика, металл и стекло.

В то время, как FDM-принтеры для начинающих пользователей обычно производят относительно простые модели одного цвета и из одного материала, более современные машины становятся доступными.

#### Материалы, используемые для трехмерной печати

АБС-пластик – данное название является упрощенным вариантом, так как химики именуют данное вещество акрилонитрилбутадиенстиролом. Такой материал отличается высокой прочностью и достаточным уровнем эластичности. В сфере трехмерной печати широко применяется порошковая смесь из АБС-пластика, когда объемные изделия выполняются методом заливки экструдируемого расплава. Пластик для 3D принтера является одним из наиболее долговечных производственных материалов, но под воздействием солнечных лучей он быстро разрушается. Такой материал совершенно не подходит для воссоздания прозрачных предметов. Пластик для 3D принтера купить можно в специализированных магазинах или заказать через интернет.

Поликарбонат – считается одним из наиболее перспективных и востребованных материалов для трехмерной печати. Он может применяться в нескольких технологиях: избирательное лазерное спекание, заливка экструдируемым расплавом и др. Отличительной особенностью данного

вещества можно считать невысокую температуру плавления, которая успешно сочетается с достаточно высокой скоростью затвердевания. Более того, его химический состав не приносит абсолютно никакого вреда человеческому организму и отличается небывалой ударопрочностью.

Полиэтилен низкого давления – такое вещество уже давно по праву считается одним из наиболее популярных типов пластмассы во всем мире, именно поэтому нет ничего удивительного в том, что разработчики трехмерных принтеров нашли ему применение в этой сфере. Все мы знаем, как выглядит полиэтиленовый пакет, пластмассовая бутылка и другие предметы из пищевой пластмассы. Всё это полиэтилен низкого давления, который на сегодняшний день является один из признанных лидеров в сфере объемной печати. Еще одним фактором, который способствует такому широкому применению данного вещества, является то, что он подходит для любой технологии объемной печати. Лишь немногие материалы обладают такими же универсальными свойствами, но преимуществом полиэтилена является его низкая стоимость.

### Сфера применения

3D-принтер находит применение во многих сферах.

На нем можно, например, напечатать сладости. Все эти вкусности можно килограммами распечатывать в двух вариантах: белом и цветном. Конфеткам можно придавать вкус ванили, мяты, кислого яблока, вишни и даже арбуза. Технология 3D печати действительно впечатляет. Удивительные пищевые 3D принтеры, существование которых казалось невозможным ещё вчера, сегодня стали реальностью, и вполне возможно, что они окажутся чем-то вполне обыденным в ближайшем будущем.

Принцип действия пищевого 3D принтера очень похож на принцип работы обычного струйного принтера. Только вместо картриджей с жидкими красителями в пищевом принтере используются картриджи с пищевыми ингредиентами. В памяти принтера хранится множество рецептов. Чтобы напечатать блюдо, необходимо выбрать один из рецептов и нажать на кнопку. После этого принтер, в соответствии с заложенным в него алгоритмом, начнёт слоями выкладывать ингредиенты на рабочую поверхность или на тарелку. Полученный таким образом продукт охлаждается или запекается.

Учёные Массачусетского технологического института разработали 3D принтер Digital Chocolatier, который печатает лакомства из шоколада, фруктов и орехов. Принтер состоит из карусели с ингредиентами, приёмной терморегулируемой формочки и пользовательского интерфейса.

Также можно напечатать мебель, сноуборды, обувь, машины, лекарства. Но самое интересное – это печать органов. Китайские ученые научились печатать прототипы человеческих органов, но «живут» они не более 4-х месяцев и лишены кровеносных сосудов. Ученые уверены, пригодные для трансплантации органы – вопрос 15-20 лет. А протезы глаз разработали инженеры Манчестерского университета и студии дизайна Тома Фриппа. Обычно их делают вручную – это долго и дорого (3000 фунтов стерлингов). На 3D-принтере можно напечатать 150 глазных протезов за час – стоимостью не больше сотни за каждый. Искусственную нижнюю челюсть впервые напечатали исследователи из университета Хасселта. Ее трансплантировали 83-летней пациентке, после чего женщина смогла дышать, говорить и жевать. Для людей с артритом уже разработали экзоскелет в печатной версии.

## Заправка 3D-принтера

Картриджи для пищевых 3D принтеров заправляются самыми разными пищевыми компонентами, которые можно заказывать исходя из собственных предпочтений.

Процесс заправки трехмерного принтера мало чем отличается от обычного двумерного устройства печати струйного типа. Стоит отметить, что «чернила» 3D-принтерам хватает надолго, поэтому проводить такую процедуру часто вам не придется. Отвечая на вопрос, чем заправлять 3D принтер, стоит в первую очередь задуматься о цели использования данного устройства. Наиболее востребованным материалом всегда был и остается пластик для 3D-принтера, купить который можно либо в специализированных магазинах, либо в интернете на сайтах производителей.

Заправляют трехмерное устройство печати и другими веществами, всё зависит от создаваемого объекта. Сегодня уже известны кулинарные 3D-принтеры, которые заправляются пищевыми продуктами. Некоторые трехмерные устройства на основе стволовых клеток человеческого или животного происхождения воссоздают самые настоящие живые органы, готовые к пересадке. Понятное дело, что пластик для 3D принтера, производящего детали машин и металлоконструкций, не подойдет. Промышленные устройства объемной печати вообще могут «питаться» любыми материалами, а более простые агрегаты работают с ограниченным количеством веществ.

Как правило, струйный 3D принтер продают в комплекте с катушкой специализированного пластика и набором необходимых инструментов. В зависимости от того, сколько стоит 3D-принтер, такие наборы могут быть достаточно впечатляющими или состоять только из самых необходимых элементов. Три рабочие платформы и специальные перчатки для безопасной эксплуатации аппаратов также нередко входят в данный комплект. Кроме того, каждая отдельная модель трехмерного печатного аппарата снабжена подробной инструкцией, в которой с помощью детального описания устройства и наглядных картинок объясняется процесс заправки агрегата и многие другие полезные моменты.

## Оружие

Главная же проблема заключается в том, что ружья, распечатанные на 3D-принтере также опасны для стрелка, как и для цели. Спустя пару выстрелов вполне можно оказаться в больнице из-за разрыва ствола. Оно и понятно, пластик ведь.

В 2012 году сетевая организация Defense Distributed анонсировала планы "разработать работающий пластмассовый пистолет, который любой человек сможет скачать и напечатать на 3D-принтере". В мае 2013 года они закончили разработку, а вскоре после этого Государственный департамент США потребовал удалить инструкции с веб-сайта.

21 ноября 2013 года в Филадельфии (США) был принят закон, запрещающий изготовление огнестрельного оружия с помощью 3D-принтеров.

В Великобритании нелегальны производство, продажа, приобретение и владение оружием, напечатанным на 3D-принтере.

## Программное обеспечение трехмерных принтеров

Само по себе устройство объемной печати работать не сможет, так как все процессы, происходящие в рабочей камере такого принтера, основаны на том или ином программном средстве. Специфика его работы заключается в том, что оно должно спроектировать объемное изображение или фигуру

вместо плоской картинки. Более того, модели для 3D принтера должны представлять собой идеальную виртуальную версию будущего объекта, ведь именно ими и будет руководствоваться трехмерный принтер в дальнейшем. Каждая деталь и уточненный контур должны быть выполнены с максимальной точностью, так как от этого зависит качество производства.

Независимо от того, сколько стоит 3D-принтер, он в любом случае нуждается в программном обеспечении, которое в свою очередь подразделяется на несколько видов:

- Программы для управления работой принтера;
- Трехмерные редакторы, которые отвечают за выполнение устройством конкретных задачий.

Важно, чтобы данные программы были выполнены в соответствии с существующими нормами и стандартами. От этого будет зависеть то, как собственно работает 3D-принтер, изделия, а точнее их качество и точность, и многое другое. Среди наиболее важных особенностей такого программного обеспечения стоит отметить форматы данных, которые они должны читать и редактировать:

- STL – специализированный язык программирования, который для описания плоскостей заданного объекта пользуется треугольниками;
- X3D – этот язык основывается на так называемом стандарте XML, согласно которому отсчет ведется от заранее указанных данных профиля;
- VRML – за основу берутся треугольники, не имеющие совместных вершин.

Создание трехмерных фигур с помощью объемной печати начинается с того, что формируются STL модели для 3D-принтера. Они снабжают оператора системы всей необходимой информацией о создаваемом объекте. Разработка STL модели для 3D-принтера, как правило, ведется с помощью CAD программ. Надо сказать, что такое программное обеспечение стоит довольно дорого. Если хороший 3D-принтер купить по невысокой стоимости может быть достаточно сложно, то с подобным программным обеспечением наблюдается немного другая картина. Конечно же, когда речь идет о масштабном производстве, нет смысла экономить на специализированных программах. Однако большую часть необходимых функций может выполнить и бесплатное приложение Google SketchUp.

Человеку, который пытается подобрать подходящий трехмерный принтер для своего дела, стоит обратить внимание и на возможности программного средства, которые могли бы максимально удовлетворить его потребностям, ведь именно от него и будет зависеть качество и скорость выполнения будущих задач. Эффективность работы 3D-принтера главным образом зависит от выбранного пользователем программного обеспечения.

### Цена

Самые простые и недорогие 3D-принтеры можно приобрести за цену, варьирующуюся в окрестности пятидесяти миллионов.

### Недостатки и способы их устранения

К сожалению, устройство не может построить что-то, размеры чего превышают собственные размеры принтера. Но существуют различные обходные пути, используемые для создания крупных объектов. Например, можно сделать много мелких деталей, каждая из которых легко вписывается

в «объем работ», а уже потом вручную собрать все детали в один большой объект. (Этот подход используется в RepRap устройствах Дарвин и Мендель.) Возможно, есть еще ограничения на материал.

### [Самый большой в мире 3D-принтер](#)

Итальянская компания WASP создала самый большой в мире строительный 3D-принтер. Называется он Big Delta. Размеры устройства составляют 12 метров в высоту и 6 метров в диаметре. Создавалась система для возведения жилых домов. Как утверждают разработчики, система потребляет минимальное количество энергии, а печатать при этом способно не только бетоном, но и другими материалами, включая глину.

Сейчас главная задача системы — быстрое возведение недорогих домов. Это может пригодиться, например, в местах, где произошло стихийное бедствие, и множество людей лишилось кровла. Кроме того, подходит такой принтер и для развивающихся стран, где нужны большие объемы жилья. По мнению разработчиков, модификации Big Delta смогут работать и на других планетах, создавая дома для колонистов, научные базы и прочие строения.

Развитие технологий не стоит на месте. И вполне возможно, что в ближайшем будущем 3D-принтеры смогут заменить заводы. Важно только решить проблему печати деталей большого размера и создание заправки картриджа прочным материалом.

## Глава 8. Компьютерная и информационная безопасность

Только атаки дилетантов нацелены на машины.  
Атаки профессионалов нацелены на людей.

Б. Шнайер

### Компьютерная и информационная безопасность

#### Компьютерная и информационная безопасность

**Компьютерная безопасность** — меры безопасности, применяемые для защиты вычислительных устройств (компьютеры, смартфоны и другие), а также компьютерных сетей (частных и публичных сетей, включая Интернет). Поле деятельности системных администраторов охватывает все процессы и механизмы, с помощью которых цифровое оборудование, информационное поле и услуги защищаются от случайного или несанкционированного доступа, изменения или уничтожения данных, и приобретает всё большее значение в связи с растущей зависимостью от компьютерных систем в развитом сообществе.

**Информационная безопасность** — это процесс обеспечения конфиденциальности, целостности и доступности информации. Конфиденциальность: обеспечение доступа к информации только авторизованным пользователям. Целостность: Обеспечение достоверности и полноты информации и методов ее обработки. Доступность: Обеспечение доступа к информации и связанным с ней активам авторизованных пользователей по мере необходимости.

Зачастую, люди путают эти два понятия. Внесём ясность.

**Компьютерная безопасность** - это безопасность компьютера, его программного обеспечения, информации, заложенной в нём. Она связана конкретно с компьютерами и техникой. А **информационная безопасность** - это защищенность информационных ресурсов (документов и массивов документов какой-либо организации, например), а также защита прав личности и государства в информационной сфере (например, от разглашения гос. тайн или от пиратства) и многое другого.

На самом деле, если рассуждать строго, **компьютерная безопасность** - это меры, направленные на исключение нарушения работоспособности компьютера (или компьютеров) или управления компьютером (или компьютерами) сторонними лицами. Доступ к компьютеру злоумышленника ещё не означает кражу информации. Информация может быть зашифрована или находиться на недоступном для злоумышленника носителе. **Специалист по информационной безопасности** работает в конкретном направлении, составляя нечто подобное плану защиты информации, воплощают это в жизнь как правило уже другие люди, в зависимости от распределения обязанностей организации. **Специалист по компьютерной безопасности** это частный случай **специалиста по информационной безопасности**. Чаще всего тот, кто занимается вопросами компьютерной безопасности, он же их и воплощает в жизнь, поэтому **специалист по компьютерной безопасности** - больше теоретик, **по информационной** – практик.

Информация является одним из наиболее ценных ресурсов любой компании, поэтому обеспечение защиты информации является одной из важнейших и приоритетных задач.

В силу развитых технологий не стоит безответственно относиться к своим личным данным, цифровым данным, которые при определённых обстоятельствах оказывают влияние на нашу судьбу. Уже ни для кого не секрет, что электронные девайсы, которыми пользуется человек в современном мире для передачи информации, подвержены многим уязвимостям, а также и они сами и информация представляют огромный интерес для злоумышленников и правительства.

Аргумент "мне нечего скрывать" не считается весомым, так как даже с первого взгляда безобидная информация может поведать о каждом конкретном индивидууме многое: личные предпочтения, круг общения, перемещения и прочие вещи, которые в умелых руках могут стать отличным инструментом для достижения тех или иных целей в отношении этого индивидуума.

Ну и по поводу размещения информации в сети интернет. Не выкладывайте свои личные данные, никаких фотографий, никакой привязки аккаунтов к реальному вашему номеру телефона (для этого существуют

сервисы

подобные

этим:

<http://receivesmsonline.com/> <http://www.vsms24.com/web/index.php>), никакой установки игровых клиентов на компьютер с важными данными и никакой серьёзной переписки в чатах этих онлайн-игр. (Но следует помнить и о том, что спецслужбы при необходимости могут сохранять коммуникации на том лишь основании, что они зашифрованы, а затем хранить зашифрованное сообщение вечно или по крайней мере до тех пор, пока его не расшифруют. О незашифрованных данных известно, что все логи посещений сайтов сохраняются, а система СОРМ может записывать ваш незашифрованный трафик.)

#### Уничтожение и шифрование данных

Также необходимо помнить о безвозвратном удалении следов информации с жёсткого диска, для чего целесообразнее применять CCleaner, PrivaZer, wipefile. И вот ещё список того, что желательно уничтожать:

#### Информация на компьютере:

- **Неиспользуемые пароли и данные к аккаунтам.** Удалить невосстанавливаемым программным методом.
- **Закладки в браузере.** Внимательно просмотреть и выявить все ненужное.
- **Браузеры.** Очистить всю историю и любые другие данные, сохраненные сайтами.
- **Неиспользуемые контакты.** Просмотреть и удалить неиспользуемые контакты: icq, jabber, skype, другие im-приложения, почтовые программы и сервисы.
- **Связка gpg-ключей.** Просмотреть все связи и удалить неиспользуемые, забытые и неизвестные публичные и секретные ключи.
- **Сообщения.** Удалить ненужные и старые сообщения в почтовых программах, icq, jabber-клиентах, skype и прочих программах обмена сообщениями.
- **Документы.** Просмотреть папки хранения документов — старые, неиспользуемые и ненужные удалить невосстанавливаемым программным методом.
- **Бекапы.** Просмотреть места хранения бекапов и удалить старые невосстанавливаемым программным методом.
- **Скачанное.** Очистить локальную папку для скачивания файлов.
- **Неиспользуемое свободное место на дисках.** Для очистки использовать программы Ccleaner и Bleachbit.

- **Пароли.** Решить, где лучше сменить пароли.

#### Интернет:

- **"Личка".** Просмотреть все посещаемые форумы и остальные социальные сервисы и удалить все личные сообщения.
- **Сообщения.** Удалить старые и ненужные сообщения в почтовых сервисах.

#### Средства общения, мобильные компьютеры

- **Контакты.** Пройтись по контактам в телефонах и удалить ненужные.
- **Сообщения.** Удалить все СМС и другие сообщения на смартфоне/планшете.
- **Документы.** Найти и удалить ненужные и забытые документы и книги на смартфоне/планшете.
- **Закладки в браузере.** Внимательно просмотреть и выявить все ненужное.
- **Браузеры.** Очистить всю историю и любые другие данные, сохраненные сайтами.
- **Неиспользуемые контакты в im-приложениях.** Просмотреть и удалить неиспользуемые контакты: icq, jabber, skype, другие im-приложения, почтовые программы и сервисы.
- **Приложения.** Выявить и удалить старые и ненужные приложения, а также информацию, оставленную ими.
- **Пароли.** Решить, где лучше сменить пароли.

#### Материальное

- **Флешки и другие носители информации.** Проверить все носители информации в пределах досягаемости и уничтожить (разломать, сжечь и т.п.) информацию на них по своему усмотрению.
- **Телефоны, модемы, Wi-Fi модули и другие средства коммуникации.** Использованные, старые, ненужные и по-другому "засвеченные" средства коммуникации физически сломать и выбросить минимум в трех кварталах (километрах) от их местоположения.
- **СИМ-карты.** Выявить и уничтожить (разломать, сжечь и т.п.) все забытые и неиспользуемые СИМ-карты и другие карты доступа.
- **Документы.** Выявить ненужные и забытые документы, записки, заметки, записные книжки и уничтожить (лучше сжечь).

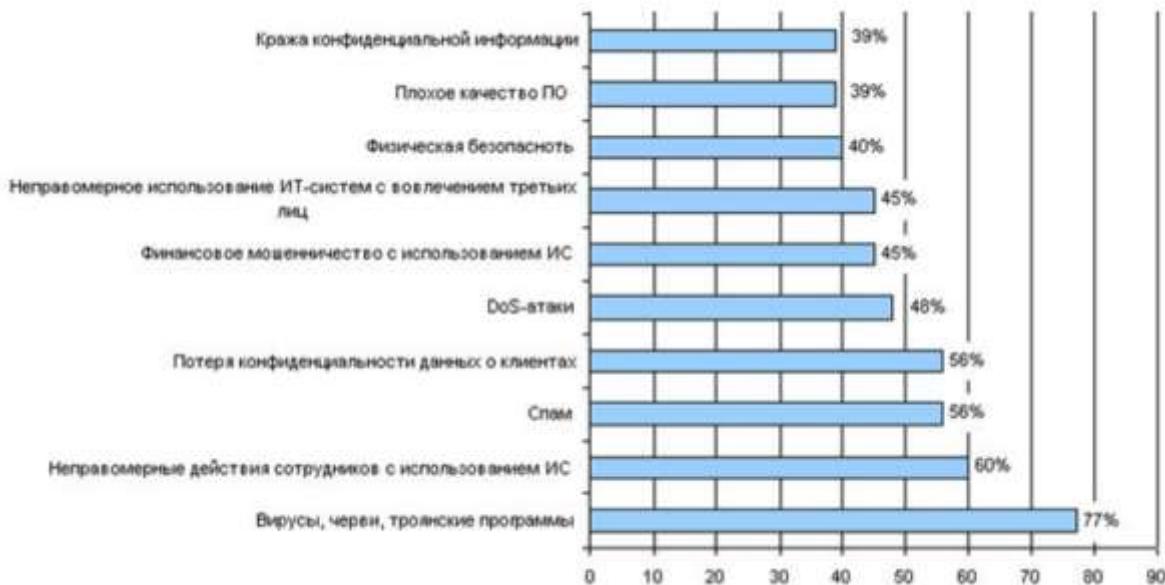
Желательно "не оставлять следов": использование кредитных карт, мобильных телефонов даёт возможность отследить ваше передвижение, покупки, контакты. Нужно иметь в виду, что каждое использование компьютера также оставляет следы.

Также следует упомянуть о необходимости шифрования данных. Ведь оно защищает нашу частную информацию и анонимность. Эта защита важна для каждого. Более того, в последние годы вопрос конфиденциальности в сети становится очень актуальным, учитывая сканирование наших электронных писем. Сегодня самым известным и простым алгоритмом, который используется в системах защиты информации организаций и предприятий, является DES (Data Encryption Standard). Он необходим для защиты от незаконного доступа к важной информации в организациях. Особенности алгоритма DES — используется ключ длиной 56 бит; зашифровать можно сообщение с помощью одной программы, а расшифровать — используя любую другую программу,

соответствующую DES; высокая скорость обработки достигается за счёт несложного алгоритма и высокой стойкости. Data Encryption Standard подходит для шифрования и аутентификации данных.

### Угрозы безопасности информационных систем

Известны следующие источники угроз безопасности информационных систем:



### Средства защиты информации от несанкционированного доступа

Получение доступа к ресурсам информационной системы предусматривает выполнение трех процедур: *идентификация, аутентификация и авторизация*.

**Идентификация** - присвоение пользователю (объекту или субъекту ресурсов) уникальных имен и кодов (идентификаторов).

**Аутентификация** - установление подлинности пользователя, представившего идентификатор или проверка того, что лицо или устройство, сообщившее идентификатор является действительно тем, за кого оно себя выдает. Наиболее распространенным способом аутентификации является присвоение пользователю пароля и хранение его в компьютере.

**Авторизация** - проверка полномочий или проверка права пользователя на доступ к конкретным ресурсам и выполнение определенных операций над ними. Авторизация проводится с целью разграничения прав доступа к сетевым и компьютерным ресурсам.

### Защита информации в компьютерных сетях

Локальные сети предприятий очень часто подключаются к сети Интернет. Для защиты локальных сетей компаний, как правило, применяются межсетевые экраны - брандмауэры (firewalls). Экран (firewall) - это средство разграничения доступа, которое позволяет разделить сеть на две части (граница проходит между локальной сетью и сетью Интернет) и сформировать набор правил, определяющих условия прохождения пакетов из одной части в другую. Экраны могут быть реализованы как аппаратными средствами (различные устройства, системы, платформы), так и программными.

Межсетевой экран, сетевой экран, файрвол или брандмауэр — это комплекс аппаратных и программных средств в компьютерной сети, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Основной задачей сетевого экрана является защита сети или отдельных её узлов от несанкционированного доступа. Также сетевые экраны часто называют фильтрами, так как их основная задача — не пропускать (фильтровать) пакеты, не подходящие под критерии, определённые в конфигурации.

#### *Проблемы, не решаемые фаерволом*

Межсетевой экран сам по себе не панацея от всех угроз для сети. В частности, он:

- не защищает узлы сети от проникновения через «люки» (англ. back doors) или уязвимости ПО;
- не обеспечивает защиту от многих внутренних угроз, в первую очередь — утечки данных;
- не защищает от загрузки пользователями вредоносных программ, в том числе вирусов;

Для решения последних двух проблем используются соответствующие дополнительные средства, в частности, антивирусы.

Но зачем Linux антивирус нужен вообще, ведь она практически не подвержена вирусным атакам? А ответ очень прост: многие линуксоиды пользуются не только Linux, но ставят, или оставляют на компьютере, второй системой Windows. Поэтому антивирус нужен, чтобы сканировать раздел Windows, когда она не работает, на возможность заражения. Многие вирусы могут легко прятаться от антивируса, но их намного легче обнаружить именно тогда, когда Windows спит. Именно так поступают некоторые антивирусы в Windows: они перезагружают Windows и начинают её сканировать перед загрузкой. Но даже если у вас не стоит второй системой Windows, вам может потребоваться Linux антивирус, чтобы не стать разносчиком вирусов. Простая ситуация: вы взяли у друга флешку, скопировали нужные вам файлы, а в них вирус. Вам хоть бы что, и это здорово. Но вы передаёте эти файлы третьему человеку, у которого Windows, и заражаете его этими вирусами. Ну и третий вариант — это сервер, на котором стоит Linux. Но вот в офисе все работают на Windows, и вам просто необходимо мониторить интернет трафик на наличие вирусов. И в этом случае Linux антивирус хорошее решение, так как он может прекрасно работать в режиме командной строки.

Считается, что Unix-системы намного лучше защищены от компьютерных вирусов, в сравнении с операционными системами Windows. В принципе, до сих пор нет ни одного широко распространенного вируса для Linux, в то время как в среде Windows просто кишит всякой заразой. В Linux очень быстро исправляют уязвимости.

Существует мнение, что вирусов на Linux мало потому, что его доля на рынке операционных систем крайне мала, поэтому и создают вирусы для Windows, из-за его большой популярности и распространенности. Будь Linux популярнее, и для него так же написали бы кучу вирусов. Но это не так. Если взять самое популярное программное обеспечение для web-серверов, то это будет Apache. По состоянию на 2013 год доля Apache на web-серверах составляет почти 45%, а доля IIS от Microsoft — 23.10% рынка. Из этого следует, что атаки хакеров должны быть активнее к большему числу интернет-ресурсов на Apache, и должно наблюдаться больше червей, вирусов и других вредоносных программ, которые будут нацелены на Apache и те операционные системы, под

управлением которых он работает, чем на Windows и IIS. Но в жизни дела обстоят иначе. Уже много времени целью для сетевых червей и всевозможных атак есть IIS от Microsoft.

#### *Основные достоинства Linux, в плане безопасности:*

- для \*nix-платформ создано крайне мало вирусов;
- существует большое количество разных Linux-дистрибутивов. Одни работают с .deb пакетами, другие с .RPM пакетами. Создание вируса, который будет одинаково хорошо работать во всех из них очень затруднительно;
- в разных дистрибутивах по умолчанию инсталлированы разные наборы, по-разному скомпилированного программного обеспечения, что также уменьшает вероятность массового заражения вирусом;
- программы, которые скачаны из интернета, в Linux по умолчанию не являются исполняемыми. Сперва их нужно сделать таковыми;
- главным источником ПО в Linux являются проверенные (официальные) репозитории. Это дает право утверждать, что вероятность попадания вирусов в эти источники крайне мала.

#### **Криптографическая защита информации**

Для обеспечения секретности информации применяется ее шифрование или криптография. Для шифрования используется алгоритм или устройство, которое реализует определенный алгоритм. Управление шифрованием осуществляется с помощью изменяющегося кода ключа.

Извлечь зашифрованную информацию можно только с помощью ключа. Криптография - это очень эффективный метод, который повышает безопасность передачи данных в компьютерных сетях и при обмене информацией между удаленными компьютерами.

#### **Электронная подпись**

Для исключения возможности модификации исходного сообщения или подмены этого сообщения другим необходимо передавать сообщение вместе с электронной подписью. Электронная цифровая подпись - это последовательность символов, полученная в результате криптографического преобразования исходного сообщения с использованием закрытого ключа и позволяющая определять целостность сообщения и принадлежность его автору при помощи открытого ключа.

Другими словами, сообщение, зашифрованное с помощью закрытого ключа, называется электронной цифровой подписью. Отправитель передает незашифрованное сообщение в исходном виде вместе с цифровой подписью. Получатель с помощью открытого ключа расшифровывает набор символов сообщения из цифровой подписи и сравнивает их с набором символов незашифрованного сообщения.

При полном совпадении символов можно утверждать, что полученное сообщение не модифицировано и принадлежит его автору.

Целью применения систем цифровой подписи является аутентификация информации - защита участников информационного обмена от навязывания ложной информации, установление факта модификации информации, которая передается или сохраняется, и получения гарантии ее подлинности, а также решение вопроса об авторстве сообщений. Система цифровой подписи предполагает, что каждый пользователь сети имеет свой секретный ключ, который используется

для формирования подписи, а также соответствующий этому секретному ключу открытый ключ, известный некоторому кругу пользователей сети и предназначенный для проверки подписи. Цифровая подпись вычисляется на основе секретного ключа отправителя информации и собственно информационных бит документа (файла). Один из пользователей может быть избран в качестве "нотариуса" и заверять с помощью своего секретного ключа любые документы. Остальные пользователи могут провести верификацию его подписи, то есть убедиться в подлинности полученного документа. Способ вычисления цифровой подписи таков, что знание открытого ключа не может привести к подделке подписи. Проверить подпись может любой пользователь, имеющий открытый ключ, в том числе независимый арбитр, который уполномочен решать возможные споры об авторстве сообщения (документа).

### Защита информации от компьютерных вирусов

Компьютерный вирус – это небольшая вредоносная программа, которая самостоятельно может создавать свои копии и внедрять их в программы (исполнимые файлы), документы, загрузочные сектора носителей данных и распространяться по каналам связи.

Существует несколько классификаций компьютерных вирусов:

1. **По среде обитания** различают вирусы сетевые, файловые, загрузочные и файлово-загрузочные.
2. **По способу заражения** выделяют резидентные и нерезидентные вирусы.
3. **По степени воздействия** вирусы бывают неопасные, опасные и очень опасные.
4. **По особенностям алгоритмов** вирусы делят на паразитические, репликаторы, невидимки, мутанты, троянские, макро-вирусы.

**Загрузочные вирусы** заражают загрузочный сектор винчестера или дискеты и загружаются каждый раз при начальной загрузке операционной системы.

**Резидентные вирусы** загружается в память компьютера и постоянно там находится до выключения компьютера.

**Самомодифицирующиеся вирусы (мутанты)** изменяют свое тело таким образом, чтобы антивирусная программа не смогла его идентифицировать.

**Стелс-вирусы (невидимки)** перехватывает обращения к зараженным файлам и областям и выдают их в незараженном виде.

**Троянские вирусы** маскируют свои действия под видом выполнения обычных приложений.

Троянская программа — вредоносная программа, распространяемая людьми, в отличие от вирусов и червей, которые распространяются самопроизвольно. В данную категорию входят программы, осуществляющие различные несанкционированные пользователем действия: сбор информации и её передачу злоумышленнику, её разрушение или злонамеренную модификацию, нарушение работоспособности компьютера, использование ресурсов компьютера в неблаговидных целях.

Существуют Password Trojans. Эти трояны находят на вашем компьютере пароли и отправляют их хозяину. Нет разницы, какой пароль, будь то интернет пароль, пароль от почтового ящика, этот троян найдет все эти пароли.

Keyloggers трояны очень просты в использовании. Они записывают все, что было набрано на клавиатуре (включая пароли) в файл и впоследствии отправляют это все по электронной почте хозяину. Keylogger-ы обычно занимают мало места, и могут маскироваться под другие полезные программы, из-за чего их бывает трудно обнаружить. Некоторые трояны этого типа могут выделять и расшифровывать пароли, найденные в специальных полях для ввода паролей. Обычно информация с паролями отсылается хозяину трояна по электронной почте.

Одним из самых эффективных методов защиты от кейлоггеров является использование виртуальной клавиатуры. Не стоит спешить прибегать к помощи виртуальной клавиатуры, встроенной в Windows. Эта программа не предназначена для защиты от кейлоггеров. Подобного рода виртуальные клавиатуры не защищены от считывания информации, набираемой на них. В данном случае нужна специально разработанная виртуальная клавиатура, наподобие той, что входит в состав продуктов Лаборатории Касперского. Наличие виртуальной клавиатуры в программах данной лаборатории подтверждает факт сложности обнаружения кейлоггеров. В настоящее время в большинстве серьезных банков используются виртуальные клавиатуры для защиты своих клиентов от кражи вводимой информации при доступе к интернет-банкинг системам. Новое бесплатное приложение Oxynger KeySheild позволяет защитить данные от передачи хакерам без ведома пользователя. Оно реализовано в форме виртуальной клавиатуры, помогающей безопасно вводить критически важную информацию и таким образом избегать необходимости использовать аппаратную клавиатуру, за которой, возможно, следят кейлоггеры.

Современные кейлоггеры умеют соотносить клавиатурный ввод с текущим окном и элементом ввода. Многие из них умеют отслеживать список работающих приложений, делают "фото" экрана по заданному расписанию или событию, шпионят за содержимым буфера обмена, скрытно следить за пользователем. Вся собранная информация сохраняется на диске, а затем записывается в Log-файл – что-то вроде журнала регистрации, данные могут быть переданы по электронной почте или http/ftp-протоколу. Как заявляет разработчик виртуальной клавиатуры Oxynger KeySheild, программа обезопасит пользователя от шпионского программного обеспечения и троянов за счет запрета регистрации, входа в личный кабинет и тому подобного ввода критически важных данных при помощи клавиатуры, мыши, буфера обмена и экранных действий. Специальная функция блокирования скриншотов дает возможность отменять экранные снимки на таких уязвимых для информации этапах. Еще один интересный факт об Oxynger KeySheild – возможность полностью осуществлять ввод данных без помощи мыши. Для каждой инсталляции своего экземпляра программа использует различные раскладки клавиатуры. Поскольку стандартная раскладка QWERTY давно уже не является серьезным барьером для киберпреступников, Oxynger предоставляет уникальную организацию клавиш для каждой отдельной копии программного обеспечения, что в итоге оборачивается большой головной болью для хакеров.

Разумеется, приложение Oxynger KeySheild не предназначено для ежедневного использования в рамках работы с большими объемами данных на клавиатуре. Программа нацелена на ситуации, когда пользователю требуется ввести какую-либо конфиденциальную информацию: номер карты или счета, к примеру.

Подобрать пароли от различных аккаунтов можно с помощью фишинга — это популярный способ получения информации от беспечных пользователей. Хакеры присыпают письма, которые по внешнему виду очень похожи на настоящие ресурсы и просят пользователя якобы заново ввести

логин и пароль к своей учетной записи. Еще один пример фишинговых писем — это маскировка "зловредов" под отправленные документы. Правда никаких документов на самом деле в письме нет, а вместо документов в письмо вложена GIF-картинка со ссылкой на сайт для выманивания паролей.

Открывает список необычных компьютерных вирусов червь P2PShared.U. Этот вредоносный код распространялся с помощью почтовых сообщений с темой: "McDonalds желает Вам счастливого Рождества!". В сообщении речь шла о купоне, дающем право на бесплатный обед в McDonald's. Но именно купон и являлся носителем вредоносного кода.

BatGen.D, испанский шеф-повар. Этот вредоносный код — специалист в области приготовления вредоносных закусок на любой вкус. Он попадает на компьютеры в виде файла под названием "personalcake.bat". На деле он представляет собой инструмент для создания вредоносного программного обеспечения, который затем спрашивает у пользователя, каким именем он бы хотел назвать творение.

На седьмом месте расположился вредоносный код Aidreden.A. После заражения компьютера на экран выводится сообщение: "Вы умрете в следующем месяце". Таким образом Aidreden.A пытается убедить пользователя в том, что его компьютер якобы заражен вирусами, после чего предлагает загрузить фальшивый антивирус. Представляет собой обычное окно с кнопкой OK. Потом открывает страницу Microsoft Security Center и предлагает скачать антивирус.

Все современные вирусы не пишутся наобум, нужно написать такой, который хорошо прячется, модифицируется, еще лучше если дальше распространяется и умеет выполнять какую-то работу, а не просто спалить видеокарту.

Поэтому при ответе на вопрос, на каком языке написать вирус, правильным ответом будет: на том, котором знаете. Ведь что такое вирус? Это программа, которая работает с определенными тонкостями системы. На каком языке можно сделать программу - теоретически на любом. Есть тонкости: что-то на одном проще, что-то на другом, вот и вся разница. Не рекомендуется писать на Ассемблере, так как тяжело получить доступ к каким-либо API.

### DoS-атаки

DoS (от англ. Denial of Service — отказ в обслуживании) — хакерская атака на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых легальные пользователи системы не могут получить доступ к предоставляемым системным ресурсам (серверам), либо этот доступ затруднён. Если атака выполняется одновременно с большого числа компьютеров, говорят о DDoS-атаке (от англ. Distributed Denial of Service, распределённая атака типа "отказ в обслуживании"). В настоящее время DoS и DDoS-атаки наиболее популярны, так как позволяют довести до отказа практически любую систему, не оставляя юридически значимых улик. Последняя проводится в том случае, если требуется вызвать отказ в обслуживании хорошо защищённой крупной компании или правительственный организации. Особенностью данного вида компьютерного преступления является то, что злоумышленники не ставят своей целью незаконное проникновение в защищенную компьютерную систему с целью кражи или уничтожения информации. Цель данной атаки - парализовать работу атакуемого веб-узла.

Схематически DDoS-атака выглядит примерно так: на выбранный в качестве жертвы сервер обрушивается огромное количество ложных запросов со множества компьютеров с разных концов света. В результате сервер тратит все свои ресурсы на обслуживание этих запросов и становится практически недоступным для обычных пользователей. Циничность ситуации заключается в том, что пользователи компьютеров, с которых направляются ложные запросы, могут даже не подозревать о том, что их машина используется хакерами. Программы, установленные злоумышленниками на этих компьютерах, принято называть "зомби". Известно множество путей "зомбирования" компьютеров: от проникновения в незащищенные сети, до использования программ-тロjanцев. Пожалуй, этот подготовительный этап является для злоумышленника наиболее трудоемким. Чаще всего злоумышленники при проведении DDoS-атак используют трехуровневую архитектуру, которую называют "клuster DDoS". Такая иерархическая структура содержит:

- управляющую консоль (их может быть несколько), т.е. именно тот компьютер, с которого злоумышленник подает сигнал о начале атаки;
- главные компьютеры. Это те машины, которые получают сигнал об атаке с управляющей консоли и передают его агентам-“зомби”. На одну управляющую консоль в зависимости от масштабности атаки может приходиться до нескольких сотен главных компьютеров;
- агенты - непосредственно сами “зомбированные” компьютеры, своими запросами атакующие узел-мишень.

Другая опасность DDoS заключается в том, что злоумышленникам не нужно обладать какими-то специальными знаниями и ресурсами. Программы для проведения атак свободно распространяются в Сети. Для защиты от сетевых атак применяется ряд фильтров, подключенных к интернет-каналу с большой пропускной способностью. Фильтры действуют таким образом, что последовательно анализируют проходящий трафик, выявляя нестандартную сетевую активность и ошибки. В число анализируемых шаблонов нестандартного трафика входят все известные на сегодняшний день методы атак, в том числе реализуемые и при помощи распределённых бот-сетей.

#### Защита от основных видов DoS-атак

В основном защита от DoS-атак строится на правильной настройке компьютера. Следующие меры защиты способны защитить лишь от слабых DoS-атак, либо они будут использоваться в качестве снижения её эффективности.

Также есть несколько универсальных советов, которые помогут подготовить систему к DoS-атаке:

- Все серверы, которые имеют доступ во внешнюю сеть, должны быть подготовлены к удаленной аварийной перезагрузке. Также желательно наличие второго сетевого интерфейса, через который по ssh-соединению можно быстро получить доступ к серверу.
- Программное обеспечение, которое установлено на сервере, должно быть в актуальном состоянии, а именно: должно быть установлено последнее ПО, касающееся обеспечения безопасности системы.
- Все сетевые сервисы должны быть защищены брандмауэром.

Полностью защититься от DDoS-атак на сегодняшний день невозможно, так как совершенно надёжных систем не существует. Здесь также большую роль играет человеческий фактор, потому

что любая ошибка системного администратора, неправильно настроившего маршрутизатор, может привести к весьма плачевным последствиям. Однако, несмотря на всё это, на настоящий момент существует масса как аппаратно-программных средств защиты, так и организационных методов противостояния.

### Хакерство

Как ни крути, а информационная безопасность у многих прочно ассоциируется с хакерами. В наши дни под хакером понимается злоумышленник, который делает что-то нелегальное, взламывает какие-то системы с материальной выгодой для себя. Но это далеко не всегда было так.

Вернемся на полвека назад, в 1960-е годы, когда ЭВМ постепенно стали проникать в нашу жизнь. Хакерство началось еще тогда, с попыток использовать технику не по назначению. Например, чтобы запустить на ней нами же написанную игру. В те времена понятие "хакер" — это очень увлеченный человек, пытающийся сделать с системой что-то нестандартное. Доступ к компьютерам ведь был в основном у сотрудников университетов, причем процессорного времени каждому выделялось не так уж много. Позволялось поработать с вычислительной машиной всего несколько часов в неделю, по строгому расписанию. Но даже в таких условиях людям удавалось выкраивать время на эксперименты. Хакерам того времени было интересно не просто решить какую-то вычислительную задачу, они хотели понять, как устроена и работает машина. Культура хакерства вышла из очень увлеченных людей.

В 70-е хакерство окончательно сформировалось как попытка "поиграть" с информационной системой, обходя ее ограничения. Интернета тогда еще не было, но уже была телефония. Поэтому появилось такое явление как фрикинг (сленговое выражение, обозначающее взлом телефонных автоматов, телефонных сетей и сетей мобильной связи, с использованием скрытых от пользователя или недокументированных функций. Обычно фрикинг осуществляется для бесплатных звонков, пополнения личного мобильного счета.). Отцом фрикинга считается Джон Дрейпер, известный под ником "Капитан Кранч". Однажды он нашел в пачке кукурузных хлопьев "Captain Crunch" подарок — свисток. Телефонные линии в то время были аналоговыми и телефонные аппараты общались друг с другом посредством обмена тоновыми сигналами. Оказалось, что тональность обнаруженному Дрейпером свистка совпадает с тоном, используемым телефонным оборудованием для передачи команд. Фрикеры стали при помощи свистков эмулировать систему команд и бесплатно дозванивались из уличных телефонов в соседние города и штаты. Следующим шагом стало создание Стивеном Возняком и Стивом Джобсом "blue box" — аппарата, эмулировавшего все те же тоновые команды. Он позволял не только дозваниваться до нужных номеров, но и пользоваться секретными служебными линиями. Подав сигнал на частоте 2600 Гц можно было перевести телефонные системы в административный режим и дозвониться на недоступные обычному человеку номера, например, в Белый Дом. Развлечение продолжалось до конца 80-х, когда в популярной газете была опубликована большая статья о "blue box", которая привлекла к фрикерам внимание полиции. Многие фрикеры, в том числе и сам Дрейпер, были арестованы. Позже все же удалось разобраться, что делали они это все не ради денег, а скорее из спортивного интереса и баловства. В то время в уголовном кодексе просто не существовало никаких статей, относящихся к мошенничеству с информационными системами, так что вскоре все фрикеры были отпущены на свободу.

В 80-е годы слово "хакер" впервые получило негативный оттенок. В сознании людей уже стал формироваться образ хакера как человека, который может делать что-то незаконное для получения прибыли.

#### Кевин Митник

*Как и его предшественники, Кевин Митник начал заниматься хакерством, вторгаясь в телефонные линии. В 1981 году семнадцатилетний Кевин со своим другом взломал телефонную станцию Computer System for Mainframe Operations (COSMOS), принадлежащую компании Pacific Bell в Лос-Анджелесе. Вторгвшись в систему, он переключал телефонные линии и перехватывал все звонки, идущие через эту станцию. Абоненты вскоре начали жаловаться, списывая это на ошибки и розыгрыши операторов. Кевин Митник, разумеется, отвечал на звонки сам, иногда при этом отмачивая беспактные шутки.*

*Но Митник на этом не остановился: он продолжал влезать в систему компании Pacific Bell и её COSMOS. Хакер сумел войти в базу данных компании и украсть информацию о нескольких абонентах. Он с лёгкостью получил доступ к телефонным счетам, паролям, комбинациям шлюзов и даже к системному руководству. Митник даже использовал переключение телефонных линий для своих личных нужд. В конце концов, один из сотрудников Pacific Bell обнаружил нарушения в системе COSMOS. Было начато расследование, которое быстро привело к телефонной будке, используемой Кевином Митником для звонков и доступа к сети; оставалось только дождаться, когда появится преступник и поймать его с поличным. Приговор суда был мягким: Митнику обвинили в порче данных и краже и приговорили к трём месяцам отбывания наказания в исправительном учреждении и дали один год испытательного срока.*

*Самый большой успех пришёл к Кевину Митнику в 1983 году, когда он совершил поистине впечатляющее деяние. В то время он был студентом южнокалифорнийского университета. Используя один из университетских компьютеров, возможно, TRS-80 с 1,77-МГц процессором Zilog, Митник проник в глобальную сеть ARPANet, являющуюся предшественницей Internet, которая в то время была предназначена для военных целей и объединяла крупные корпорации и университеты. Проникнув в эту сеть, Митник добрался до самых защищённых компьютеров того времени, до компьютеров Пентагона. Он получил доступ ко всем файлам Министерства обороны США. В то время не было никаких следов кражи информации или порчи: Митник действовал просто из любопытства и проверял свои способности. Но один из системных администраторов обнаружил акт вторжения и поднял тревогу. Расследование выявило автора атаки, и Кевина Митника арестовали прямо на территории университета. Он был осуждён и отбыл своё первое настоящее наказание за незаконное вторжение в компьютерную систему, проведя полгода в исправительном центре для молодёжи.*

*В 1987 году Митник оставил свою незаконную деятельность. Кевин находился на испытательном сроке, в соответствии с последним приговором суда, поэтому он не мог позволить себе никаких правонарушений. Однако вскоре Митник снова был замешан в тёмных делах.*

*Однажды вечером вместе со своим другом Ленни ДиСикко Митник вторгся во внутреннюю сеть исследовательской лаборатории американской компьютерной компании Digital Equipment Corporation (DEC). Для Митника сделать это было несложно, так как ДиСикко являлся сотрудником данной лаборатории и одновременно был соучастником взлома. EasyNet - внутренняя сеть DEC - не выдержала хакерской атаки, и вскоре сообщники получили доступ ко всей системе.*

*Как и в случае с предыдущими атаками, вторжение в лабораторию было быстро обнаружено, но на этот раз Митник это предвидел и подготовился. Он зашифровал источник вызовов, сделав бесполезными все попытки выследить его. И на этот раз Митник взломал систему не из простого любопытства или для проверки своих способностей: у него было другая цель. Хакер хотел завладеть исходным кодом операционной системы VMS, разработанной компанией DEC для компьютеров VAX.*

*Митник предпринял все меры предосторожности, но не учёл одного - своего собственного друга. Кевин любил игры и розыгрыши, поэтому как-то раз он позвонил работодателю своего друга Ленни ДиСикко, притворившись представителем государственной власти. Он сказал, будто у одного из его служащих (ДиСикко) проблемы с налогами. ДиСикко не оценил такую шутку и решил отомстить по-своему.*

*ДиСикко предал Митника, сообщив своему работодателю о вторжении Митника в сеть компании. Затем он связался с ФБР и сказал, что может сдать хакера, который регулярно проникал в сеть лаборатории. Во время встречи Митника с ДиСикко хакер попал в ловушку, расставленную его собственным другом, который пришёл в сопровождении двух агентов ФБР. Митник был арестован.*

*Судебное дело длилось недолго: компания DEC обвинила хакера в краже информации и запросила за нанесённый ущерб более \$200 000. Митника приговорили к одному году тюремного заключения, кроме того, он должен был посещать шестимесячные курсы для излечения от компьютерной зависимости.*

*В 1994 году Кевин Митник вернулся к своей незаконной деятельности, его разыскивало ФБР. За свои "подвиги" Митник уже прославился на весь мир. Федералы разослали повсюду его фотографии, чтобы узнавшие его люди могли позвонить властям. В течение этого и следующего года на Кевина Митника была объявлена облава, самая впечатляющая за всю историю поимки хакеров.*

*Митник решил атаковать другого хакера и эксперта по компьютерной безопасности - Тсумото Шимомуру (Tsutomu Shimomura). Он хорошо подготовил свою атаку и, чтобы убедиться, что никто ему не помешает, запустил её в Рождество, 25 декабря 1994 года. Митник взломал личный компьютер Шимомуры, используя неизвестную в то время методику - подмену IP-адреса, то есть разновидность вторжения, при котором взломщик пытается замаскироваться под другую систему, используя её IP-адрес.*

*Однако Митника подвёл брандмауэр Шимомуры, который записывал все действия, происходящие на целевой машине. На следующий же день, 26 декабря, Шимомуре позвонил один из его коллег и сообщил, что его компьютер стал жертвой*

*вторжения. Шимомура быстро вычислил Митника и решил помочь ФБР в поимке хакера, используя свои же хакерские умения.*

*Для поимки Митника ФБР предоставило Шимомуре полную свободу действий, в том числе разрешило использовать хакерство. Облава превратилась в виртуальную охоту. Однажды Шимомура сообщил, например, что он застал Митника врасплох 17 января 1995 года, когда тот влез в сеть, принадлежащую Motorola, чтобы украсть секретное программное обеспечение компании.*

*Преследование усиливалось, и спецслужбы начали приближаться к киберпреступнику, отступающему в г. Роли, штат Северная Каролина. Чтобы засечь сотовый телефон, с которого Митник совершал свои атаки, Шимомура в течение двух дней бродил по улицам Роли с детектором связи. 15 февраля 1995 года в 2 часа ночи агенты ФБР вместе с Шимомурой ворвались в квартиру Митника. Когда знаменитый хакер увидел своего соперника, он воскликнул: "Hi, Тсумому! Мои поздравления!". После почти двухлетнего преследования Митника приговорили к пяти годам тюремного заключения. На то время это было самое суровое наказание для хакера. Ныне является консультантом по компьютерной безопасности.*

Роберт Таппан Моррис

*Роберт Таппан Моррис (Robert Tappan Morris), который сейчас является штатным профессором в Массачусетском технологическом институте (МТИ) в лаборатории компьютерных наук и искусственного интеллекта, создал первого "интернет-черва".*

*И опять же, к созданию "черва" Морриса побудило любопытство. По его словам, основной целью его программы было оценить истинные размеры сети Интернет, т.е. определить количество подключённых к ней компьютеров. В то время к Интернету было подключено не так много машин, но вопреки расчётам Морриса, его "червь" причинил гораздо больше вреда, чем ожидалось.*

*"Червь" Морриса, отправленный с компьютеров МТИ, был запограммирован на то, чтобы проверить компьютер и скопировать себя в систему, если машина ещё не была инфицирована. Проблемы начались, когда Моррису пришла в голову мысль, что некоторые системные администраторы могут придумать уловку с поддельными копиями, чтобы обмануть "черва" и заставить его думать, что компьютер уже заражён. Тогда он модифицировал код программы, чтобы "червь" оставлял свои копии каждый раз, независимо от того, инфицирован компьютер или нет.*

*"Червь" Морриса распространился, как пожар, заразив несколько тысяч компьютеров всего за несколько часов. Приблизительно подсчитали, что восстановление каждой инфицированной системы будет стоить от \$200 до \$53 000, в зависимости от компьютера. Чтобы остановить "черва", были мобилизованы различные команды программистов, длянейтрализации атаки понадобилось несколько дней.*

*Роберт Таппан Моррис был признан виновным в компьютерном мошенничестве и был приговорён к трем годам условно, 400 часам общественных работ и \$10 050 штрафа.*

Кевин Поулсен

Кевин Поулсен - это ещё одно имя, которое вошло в анналы ФБР в 80-е гг. Впервые его арестовали в 1989 году, когда ему было 24 года. Тогда его обвинили в нескольких взломах телефонных сетей и компьютерных серверов, против него были собраны разные уличающие доказательства. Но когда пришло время предстать перед судом, Поулсен решил скрыться, агенты ФБР 17 месяцев выслеживали его. Именно в это время он совершил своё самое знаменитое хакерское деяние. В прямом эфире на лос-анджелесской радиостанции KIIS-FM проходила викторина-розыгрыш. Слушателям предлагалось позвонить на радиостанцию и попытаться выиграть автомобиль Porsche 944 S2. Приз должен был достаться 102-му дозвонившемуся радиослушателю. Кевин Поулсен начал действовать: он так заблокировал телефонную сеть, что на радио никто не мог дозвониться, кроме него самого. Следовательно, Кевин стал 102-м дозвонившимся и выиграл приз. Поулсен "утёр нос" ФБР, заставив власти вновь приступить к его поиску, потому что после этого он исчез.

В апреле 1991 года ФБР, наконец, удалось арестовать Поулсена. Поимке поспособствовал анонимный донос: кто-то сообщил властям, что Поулсен делает покупки в супермаркете на окраине Лос-Анджелеса. В 1994 году ему предъявили обвинение и в результате приговорили к четырём годам тюремного заключения. В то время это был самый суровый приговор, назначенный судом за хакерство.

В 1983 году на обложке Newsweek появился один из основателей хакерской группы "414s", ставшей известной благодаря серии взломов серьезных компьютерных систем. У хакеров появляются и собственные журналы, и другие способы обмена информации. В эти же годы власти западных стран начинают формировать законы, связанные с компьютерной безопасностью. Впрочем, масштаб, конечно, не идет ни в какое сравнение с сегодняшним днем. Так, много шума случилось вокруг дела Кевина Поулсена, который, благодаря своим фрикерским навыкам, первым дозвонился в эфир радиостанции KIIS-FM, что позволило ему выиграть автомобиль Porsche в рамках проходящего конкурса. По сравнению с современными атаками, которые позволяют уводить со счетов миллионы долларов — это просто мелочь, но тогда эта история вызвала очень большой резонанс.

Девяностые – это эра активного развития интернета, в это же время за хакерством окончательно закрепляется криминальный оттенок. Персональные компьютеры уже стали относительно доступны простым людям, при этом о безопасности никто особенно не задумывался. Появлялось огромное количество готовых программ, с помощью которых можно было взламывать пользовательские компьютеры, не имея каких-то серьезных технических навыков и способностей. Чего стоит только известная программа winnuke, которая позволяла отправить Windows 95/98 в "синий экран" путем отправки одного-единственного IP-пакета.

Владимир Левин

Люди занимаются хакерством не только из любопытства и азарта, иногда в дело замешаны деньги. Самый яркий пример - ограбление банка с целью получения денег, иногда удавалось украсть миллионы. Так, Владимир Левин (Vladimir Levin) завоевал дурную славу тем, что украл несколько миллионов долларов при странных обстоятельствах.

*В 1994 году Левин проник во внутреннюю сеть американского банка Citibank, взломав аналоговое модемное подключение банка и получив доступ к нескольким счетам. Он сумел перевести 10,7 миллиона долларов на счета в США, Финляндию, Германию, Израиль и Нидерланды. Левину помогали трое сообщников, которые должны были вернуть украденные деньги.*

*Однако его сообщников арестовали, когда они пытались стащить украденные деньги. Их допрос вывел на след Левина, который работал программистом в Санкт-Петербурге. Российского хакера арестовали в марте 1995 года в лондонском аэропорту Хитроу. Судебное разбирательство против него началось только в сентябре 1997 года и закончились в феврале следующего года. Левина приговорили к трём годам лишения свободы.*

Вообще, девяностые считаются золотыми годами темных хакеров: возможностей для мошенничества полно, компьютерные системы соединены через интернет, а серьезных механизмов безопасности в массовых ОС в эти годы еще нет. Еще одна отличительная особенность девяностых — огромное количество голливудских фильмов про хакеров как иллюстрация того факта, что взлом компьютерных систем постепенно становится "обыденной диковинкой" для широких масс. Ведь почти в каждом таком фильме обязательно фигурировал какой-нибудь вирус, который взрывал мониторы. Может быть, именно из-за киноиндустрии у пользователей в голове сложился стереотип "безопасность — это вирусы", что, конечно, помогло многим антивирусным компаниям сделать свое состояние.

В 2002 году Билл Гейтс написал своим сотрудникам в компании Microsoft письмо о том, что ситуацию нужно исправлять, и пора начинать разрабатывать программное обеспечение с оглядкой на безопасность. Данная инициатива получила название "Trustworthy computing", и развивается она до сих пор. Начиная с Windows Vista, эта идея начала воплощаться в жизнь. Количество уязвимостей в ОС от Microsoft заметно снизилось, эксплойты (компьютерная программа, фрагмент программного кода или последовательность команд, использующие уязвимости в программном обеспечении и применяемые для проведения атаки на вычислительную систему) под них все реже появляются в публичном доступе. Как ни удивительно это звучит, но с точки зрения подхода к безопасности последние версии Windows гораздо надежнее других распространенных операционных систем. Так, в OS X только в последнее время стали появляться механизмы, которые затрудняют эксплуатацию уязвимостей.

Нулевые годы нашего века. Цифровой криминал выходит на новый уровень. Каналы связи стали толще, стало проще совершать мощные DoS-атаки. Никто уже не занимается взломом компьютерных систем просто ради удовольствия, это многомиллиардный бизнес. Расцветают ботнеты: огромные системы, состоящие из миллионов зараженных компьютеров. Другой характерной особенностью последнего десятилетия является тот факт, что фокус атакующего сместился на пользователя, на его персональный компьютер. Системы становятся сложнее, современный браузер — это уже не просто программа, которая умеет рендерить HTML, показывать текст и картинки. Это очень сложный механизм, полноценное окно в интернет. Почти никто уже не пользуется отдельными мессенджерами и почтовыми клиентами, все взаимодействие с интернетом происходит именно через браузер. Неудивительно, что один из основных методов заражения пользователей в наш дни — drive-by-downloads — происходит как раз при помощи браузера. Конечно, в современных браузерах стали появляться механизмы борьбы с этим,

пользователей стараются предупреждать, что посещаемый сайт может быть опасен. Но браузер по-прежнему остается одним из основных посредников при заражении пользовательских машин. Еще один большой канал распространения вредоносных программ — мобильные устройства. Если в официальных магазинах приложений программы хоть как-то проверяют на вредоносность, то из неофициальных источников можно занести на свой девайс практически все, что угодно. Да и вообще безопасность мобильных устройств сейчас — довольно молодая и немного сумбурная отрасль, что связано с той скоростью, с которой современные мобильные платформы ворвались в нашу жизнь.

Андрисан Ламо

*Андрисан Ламо, безусловно, свёл с ума самое большое количество сетевых администраторов. От его деятельности пострадали такие крупные корпорации, как Microsoft, New York Times, AOL, Sun Microsystems, Yahoo!, MacDonal'd's и Cingular. Ему приписываются все виды атак и нарушений защиты корпоративных систем безопасности. Ламо обходил защитные системы с бесконтактной простотой.*

*Так, во время эфира ночных новостей телекомпании NBC журналист предложил Андрисану доказать свой талант прямо перед объективом камеры, и тогда хакер менее чем за пять минут проник во внутреннюю сеть самой телекомпании. В настоящее время Ламо является специалистом по безопасности и наслаждается полной свободой передвижения после того, как многие годы был под наблюдением властей США.*

12 февраля 2004 года был самый обычный день, но в компании Microsoft было объявлено чрезвычайное положение. Кто-то украл исходный код операционной системы Windows 2000, которой до сих пор пользуется большое количество пользователей. И что ещё хуже, неизвестный хакер выложил этот код в Wild. Кража была масштабна: 600 миллионов байт данных, 30 195 файлов и 13,5 миллиона строк кода. Утечка информации коснулась операционной системы Windows 2000 и её "старшей сестры" Windows NT4. Все сотрудники "софтового" гиганта пытались выяснить, что произошло, но никто не мог дать ответ. Данные были украдены прямо из сети Microsoft. Неизвестный хакер вошёл во внутреннюю сеть компании, взломав пароль одного из компьютеров. Исходный код быстро распространился по Интернету, особенно по P2P-сетям. К счастью, несмотря на то, что все опасались худшего, последствия этой грандиозной кражи оказались довольно мягкими.

### Ответственность

На сегодняшний день за "хакерство" можно получить наказания по статьям 212 (Хищение путем использования компьютерной техники), 354 (Разработка, использование либо распространение вредоносных программ).

*Статья 212. Хищение путем использования компьютерной техники*

*1. Хищение имущества путем изменения информации, обрабатываемой в компьютерной системе, хранящейся на машинных носителях или передаваемой по сетям передачи данных, либо путем введения в компьютерную систему ложной информации - наказывается штрафом, или лишением права занимать определенные должности или заниматься определенной деятельностью, или арестом на срок до*

*шести месяцев, или ограничением свободы на срок до трех лет, или лишением свободы на тот же срок.*

*2. То же деяние, совершенное повторно, либо группой лиц по предварительному сговору, либо сопряженное с несанкционированным доступом к компьютерной информации, - наказывается ограничением свободы на срок от двух до пяти лет или лишением свободы на срок до пяти лет с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения.*

*3. Деяния, предусмотренные частями первой или второй настоящей статьи, совершенные в крупном размере, - наказываются лишением свободы на срок от трех до десяти лет с конфискацией имущества или без конфискации и с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения.*

*4. Деяния, предусмотренные частями первой, второй или третьей настоящей статьи, совершенные организованной группой либо в особо крупном размере, - наказываются лишением свободы на срок от шести до пятнадцати лет с конфискацией имущества и с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения.*

#### *Статья 354. Разработка, использование либо распространение вредоносных программ*

*1. Разработка компьютерных программ или внесение изменений в существующие программы с целью несанкционированного уничтожения, блокирования, модификации или копирования информации, хранящейся в компьютерной системе, сети или на машинных носителях, либо разработка специальных вирусных программ, либо заведомое их использование, либо распространение носителей с такими программами - наказываются штрафом, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до двух лет, или лишением свободы на тот же срок.*

*2. Те же действия, повлекшие тяжкие последствия, - наказываются лишением свободы на срок от трех до десяти лет.*

За всю историю самое жестокое наказание за хакерство было в 1998-1999 годах в Китае, когда два брата подверглись смертной казни за взлом компьютерной системы государственного банка с целью кражи денег. Один из братьев получил отсрочку исполнения приговора за согласие дать свидетельские показания.

Не всегда же хакерство носит негативный оттенок. Известны случаи, когда можно законно зарабатывать, будучи хакером. Ведь некоторые из них помогают защитить наши компьютеры и гаджеты от кражи информации и взломов. Весной 2015 года особо отличился хакер Джун Хун Ли из Южной Кореи. Участвовал в конкурсе Pwn2Own, который спонсирует корпорация Google. В ходе этого мероприятия хакеры со всего мира демонстрируют конкурсному жюри обнаруженные ими баги, уязвимости и эксплойты в различных приложениях и операционных системах. Разумеется, за каждый обнаруженный баг хакер получает награду от организаторов. Джун Хун Ли выявлял одну уязвимость за другой. Сначала он сломал браузер Chrome. Затем продемонстрировал, как с помощью этого же браузера можно организовать широкомасштабную атаку на операционную

систему. Потом взломал браузер Apple. Заработав в общей сложности 225 000 долларов за два дня конкурса, Джун Хун Ли наконец расслабился и перевёл дыхание.

Подведем итог о том, что такое хакерство сегодня, и чего стоит ожидать в ближайшем будущем. Хакерство возникло в 70-е гг. прошлого века, но некоторые из многочисленных группировок, сформировавшихся вокруг этого движения, существуют и по сей день. Всё больше и больше людей получают возможность пользоваться Интернетом, а хакеров и так называемых "script kiddies" (компьютерных хулиганов-подростков) сейчас больше, чем когда-либо. Однако мы не наблюдаем бурного роста крупномасштабных атак против компьютерных систем, который был бы соразмерен увеличению числа хакеров. Движение пошло немного по другому направлению, по сравнению с периодом расцвета хакерства. Некоторые атаки по-прежнему производят фурор, и системным администраторам приходится за это расплачиваться. Кевины Митники и Джоны Дрейперы, которые вошли в историю, уже успокоились и остепенились, а нынешние специалисты по безопасности компьютерных систем сталкиваются с менее опасной, но гораздо более массовой угрозой. В начале двухтысячных, если обнаруживалась какая-нибудь уязвимость в Windows, практически сразу в свободном доступе появлялся экспloit, который позволял получить контроль над пользовательским компьютером. Тогда над извлечением прибыли от уязвимостей практически не задумывались. Конечно, были программы, которые воровали данные пользователей, уводили компьютеры в ботнеты, но сами уязвимости, приводящие к компрометации компьютера, было относительно просто эксплуатировать, а значит, написать экспloit. Примерно последние пять лет найти в публичном доступе рабочий экспloit для недавно обнаруженной уязвимости стало очень непросто. Теперь это огромный бизнес. Ведь написать экспloit для уязвимости в системе, в которой внедрены механизмы вроде DEP и ASLR, значительно сложнее. Последние годы также показывают, что всех нас ждут проблемы безопасности так называемого "интернета вещей". Компьютеры с доступом в интернет сейчас все чаще в том или ином виде присутствуют в самых разнообразных бытовых и медицинских приборах, а также автомобилях. И уязвимости в них точно такие же, как в обычных компьютерах. Исследования по взлому привычных нам вещей становятся популярной темой на ведущих мировых конференциях по безопасности. Ведь если злоумышленники начнут пользоваться такими уязвимостями, это будет представлять серьезную опасность для здоровья и жизни пользователей. Тем важнее становится роль специалиста по безопасности.

## [Антивирусы](#)

### [Общие понятия](#)

**Антивирусная программа (антивирус)** — специализированная программа для обнаружения компьютерных вирусов, а также нежелательных (считающихся вредоносными) программ вообще и восстановления заражённых (модифицированных) такими программами файлов, а также для профилактики — предотвращения заражения (модификации) файлов или операционной системы вредоносным кодом.

На данный момент антивирусное программное обеспечение разрабатывается, в основном, для ОС семейства Windows от компании Microsoft. Это вызвано большим количеством вредоносных программ именно под эту платформу (а это, в свою очередь, вызвано большой популярностью этой ОС, так же, как и большим количеством средств разработки, в том числе бесплатных и даже «инструкций по написанию вирусов»). В настоящий момент на рынок выходят продукты и для

других операционных систем, таких, к примеру, как Linux и Mac OS X. Это вызвано началом распространения компьютерных вирусов и под эти платформы, хотя UNIX-подобные системы традиционно пользуются репутацией более устойчивых к воздействию вредоносных программ.

Помимо ОС для настольных компьютеров и ноутбуков, также существуют платформы и для мобильных устройств, такие, как Windows Mobile, Symbian, Apple iOS, BlackBerry, Android, Windows Phone 7 и др. Пользователи устройств на данных ОС также подвержены риску заражения вредоносным программным обеспечением, поэтому некоторые разработчики антивирусных программ выпускают продукты и для таких устройств.

По используемым технологиям антивирусной защиты:

- Классические антивирусные продукты (продукты, применяющие только сигнатурной защиту, продукты, применяющие только проактивные технологии антивирусной защиты);
- Комбинированные продукты (продукты, применяющие как сигнатурные методы защиты, так и проактивные)

**Обнаружение, основанное на сигнтурах** — метод работы антивирусов и систем обнаружения вторжений, при котором программа, просматривая файл или пакет, обращается к словарю с известными вирусами, составленному авторами программы. В случае соответствия какого-либо участка кода просматриваемой программы известному коду (сигнатуре) вируса в словаре, программа антивирус может заняться выполнением одного из следующих действий:

1. Удалить инфицированный файл.
2. Отправить файл в «карантин» (то есть сделать его недоступным для выполнения, с целью недопущения дальнейшего распространения вируса).
3. Попытаться восстановить файл, удалив сам вирус из тела файла.

Для достижения достаточно продолжительного успеха при использовании этого метода необходимо периодически пополнять словарь известных вирусов новыми определениями (в основном в онлайновом режиме).

- Достоинства и недостатки:
- Позволяют определять конкретную атаку с высокой точностью и малой долей ложных вызовов
- Беззащитны перед полиморфными вирусами и изменёнными версиями того же вируса
- Требуют регулярного и крайне оперативного обновления
- Требуют кропотливого ручного анализа вирусов
- Неспособны выявить какие-либо новые атаки

## Карантин

**Карантин** – это некоторая защищенная антивирусом область, которая позволяет понаблюдать за действиями файла, а также за работой программ и операционной системы в отсутствии этого файла на прежнем месте. Дело в том, что при перемещении файла в карантин он удаляется из своего первоначального месторасположения и копируется в некоторую папку, которая находится под контролем антивируса. Такой подход позволяет обезопасить себя от проблем, которые обязательно возникнут если вдруг какой-то важный для работы программы или операционной

системы файл будет ошибочно удален антивирусом, ведь в случае возникновения проблем вы всегда сможете вернуть файл из карантина назад.

Чаще всего лечение доступно только при сигнатурном анализе. Из программы удаляется код вируса.

### [Проактивные технологии](#)

**Проактивные технологии** — совокупность технологий и методов, используемых в антивирусном программном обеспечении, основной целью которых, в отличие от реактивных (сигнатурных) технологий, является предотвращение заражения системы пользователя, а не поиск уже известного вредоносного программного обеспечения в системе. При этом проактивная защита старается блокировать потенциально опасную активность программы только в том случае, если эта активность представляет реальную угрозу. Серьезный недостаток проактивной защиты — блокирование легитимных программ (ложные срабатывания).

#### *Технологии проактивной защиты:*

##### **Эвристический анализ**

Технология эвристического анализа позволяет на основе анализа кода выполняемого приложения, скрипта или макроса обнаружить участки кода, отвечающие за вредоносную активность.

Эффективность данной технологии не является высокой, что обусловлено большим количеством ложных срабатываний при повышении чувствительности анализатора, а также большим набором техник, используемых авторами вредоносного ПО для обхода эвристического компонента антивирусного ПО.

##### **Эмуляция кода**

Технология эмуляции позволяет запускать приложение в среде эмуляции, эмулируя поведение ОС или центрального процессора. При выполнении приложения в режиме эмуляции приложение не сможет нанести вреда системе пользователя, а вредоносное действие будет детектировано эмулятором.

Несмотря на кажущуюся эффективность данного подхода, он также не лишен недостатков — эмуляция занимает слишком много времени и ресурсов компьютера пользователя, что негативно сказывается на быстродействии при выполнении повседневных операций, также, современные вредоносные программы способны обнаруживать выполнение в эмулированной среде и прекращать свое выполнение в ней.

##### **Анализ поведения**

Технология анализа поведения основывается на перехвате всех важных системных функций или установке т. н. мини-фильтров, что позволяет отслеживать всю активность в системе пользователя. Технология поведенческого анализа позволяет оценивать не только единичное действие, но и цепочку действий, что многократно повышает эффективность противодействия вирусным угрозам. Также, поведенческий анализ является технологической основой для целого класса программ — поведенческих блокираторов (HIPS — Host-based Intrusion Systems).

## Sandboxing (Песочница)

— ограничение привилегий выполнения.

Технология Песочницы работает по принципу ограничения активности потенциально вредоносных приложений таким образом, чтобы они не могли нанести вреда системе пользователя.

Ограничение активности достигается за счет выполнения неизвестных приложений в ограниченной среде — собственно песочнице, откуда приложение не имеет прав доступа к критическим системным файлам, веткам реестра и другой важной информации. Технология ограничения привилегий выполнения является эффективной технологией противодействия современным угрозам, но, следует понимать, что пользователь должен обладать знаниями, необходимыми для правильной оценки неизвестного приложения.

## Виртуализация рабочего окружения

Технология виртуализации рабочего окружения работает с помощью системного драйвера, который перехватывает все запросы на запись на жесткий диск и вместо выполнения записи на реальный жесткий диск выполняет запись в специальную дисковую область — буфер. Таким образом, даже в том случае, если пользователь запустит вредоносное программное обеспечение, оно проживет не далее, чем до очистки буфера, которая по умолчанию выполняется при выключении компьютера.

Однако, следует понимать, что технология виртуализации рабочего окружения не сможет защитить от вредоносных программ, основной целью которых является кража конфиденциальной информации, так как доступ на чтение к жесткому диску не запрещен.

Антивирусы для сайтов можно поделить условно на несколько типов:

- Серверный — устанавливается на веб-сервер. Поиск вирусов, в этом случае, происходит в файлах всего сервера.
- Скрипт или компонент CMS — выполняющий поиск вредоносного кода, непосредственно в файлах сайта.
- SaaS сервис — система централизованного управления, позволяющая управлять файлами, базами данных, настройками и компонентами веб-ресурсов на VDS и DS удаленно.

Говоря о системах Майкрософт, следует знать, что обычно антивирус действует по схеме:

- поиск в базе данных антивирусного ПО сигнатур вирусов.
- если найден инфицированный код в памяти (оперативной и/или постоянной), запускается процесс «карантина», и процесс блокируется.
- зарегистрированная программа обычно удаляет вирус, незарегистрированная просит регистрации и оставляет систему уязвимой.

Для использования антивирусов необходимы постоянные обновления так называемых баз антивирусов. Они представляют собой информацию о вирусах — как их найти и обезвредить. Поскольку вирусы пишут часто, то необходим постоянный мониторинг активности вирусов в сети. Для этого существуют специальные сети, которые собирают соответствующую информацию. После

сбора этой информации производится анализ вредоносности вируса, анализируется его код, поведение, и после этого устанавливаются способы борьбы с ним. Чаще всего вирусы запускаются вместе с операционной системой. В таком случае можно просто удалить строки запуска вируса из реестра, и на этом в простом случае процесс может закончиться. Более сложные вирусы используют возможность заражения файлов. Например, известны случаи, как некие даже антивирусные программы, будучи зараженными, сами становились причиной заражения других чистых программ и файлов. Поэтому более современные антивирусы имеют возможность защиты своих файлов от изменения и проверяют их на целостность по специальному алгоритму. Таким образом, вирусы усложнились, как и усложнились способы борьбы с ними. Сейчас можно увидеть вирусы, которые занимают уже не десятки килобайт, а сотни, а порой могут быть и размером в пару мегабайт. Обычно такие вирусы пишут в языках программирования более высокого уровня, поэтому их легче остановить. Но по-прежнему существует угроза от вирусов, написанных на низкоуровневых машинных кодах наподобие ассемблера. Сложные вирусы заражают операционную систему, после чего она становится уязвимой и нерабочей.

### Выявление сетевых атак

Процесс обнаружения информационных атак начинается со сбора исходных данных, необходимых для того, чтобы сделать вывод о проведении атаки в ИС. Примерами таких данных являются:

- сведения о пакетах данных, передаваемых в ИС;
- информация о производительности программно-аппаратного обеспечения ИС (вычислительная нагрузка на процессор хостов ИС, загруженность оперативной памяти, скорость работы прикладного ПО и др.);
- сведения о доступе к файлам ИС;
- информация о регистрации новых пользователей в ИС и др.

Информация, собранная сетевыми и хостовыми датчиками, анализируется СОА с целью выявления возможных атак нарушителей. Анализ данных может проводиться при помощи двух основных групп методов - сигнатурных и поведенческих.

### Zip-бомбы

**Zip-бомба**, также известная как **Архив Смерти** или англ. *decompression bomb* — архивный файл, который по своей природе обладает разрушающим действием. При распаковке может вызвать крах системы. Современные антивирусы вполне распознают подобные файлы и предупреждают пользователя о разрушающем действии. Внешне подобный файл выглядит как маленький архив. При распаковке распаковывается тот же самый архив. Данный файл может предоставлять опасность для антивирусов: в попытке распаковать все архивы антивирус может забить всю память и ничего не найти.

Лучше всего с ними бороться, устанавливая количество уровней декомпрессии для проверки антивирусом.

### Первые антивирусы

Зимой 1984 г. Анди Хопкинс (Andy Hopkins) написал программы CHK4BOMB и BOMBSQAD. Первая из них позволяла проанализировать текст загрузочного модуля и выявляла все текстовые сообщения и "подозрительные" участки кода (команды прямой записи на диск и др.). Благодаря

своей простоте (фактически использовался только контекстный поиск) и эффективности CHK4BOMB получила значительную популярность. Программа BOMBSQAD.COM перехватывает операции записи и форматирования, выполняемые через BIOS. При выявлении запрещенной операции можно разрешить ее выполнение. В начале 1985 г. Ги Вонг (Gee Wong) написал программу DPROTECT - резидентную программу, перехватывающую попытки записи на дискеты и винчестер. Она блокировала все операции (запись, форматирование), выполняемые через BIOS. В случае выявления такой операции программа требует рестарта системы. Несколько позднее появилась программа FLUSHOT, написанная Росс М. Гринберг. Более поздняя версия этой программы - FluShot Plus (версия 1.7), распространяемая как SHAREWARE с регистрационной ценой 10 долларов, используется и в настоящее время.

Сравнение современных антивирусов.

#### *Avast!*

**Avast!** — антивирусная программа для операционных систем Windows, Linux, Mac OS, а также для КПК на платформе Palm, Android и Windows CE. Разработка компании AVAST Software, основанной в 1991 году в Чехословакии. Avast! Free Antivirus считается самым популярным бесплатным антивирусом. Всего же антивирусом Avast! пользуются более 230 миллионов пользователей во всём мире.

#### **Возможности:**

- Проверка компьютера на вирусы во время запуска, до полной загрузки операционной системы. При этом Avast! использует прямой доступ к жёстким дискам, т.е в обход драйверов файловой системы Windows. Avast! — единственный антивирус, в котором встречается подобного рода функция.
- Начиная с шестой версии, бесплатный вариант антивируса включает дополнительную функцию WebRep. Эта функция информирует пользователя о репутации посещаемых сайтов на основании оценок, выставленных сообществом пользователей Avast!. Работает в браузерах Internet Explorer, Mozilla Firefox и Google Chrome. В седьмой версии реализована и для Opera. В девятой версии переименована в Avast! Online Security.
- Голосовые сообщения при обнаружении вредоносной программы, успешном обновлении вирусной базы данных и завершении сканирования. Одновременно с этим в нижнем правом углу экрана появляется соответствующее сообщение. До пятой версии использовался мужской голос. Начиная с пятой — женский. Также на официальном сайте можно найти и другие голоса на разных языках (на русском языке дополнительных голосов пока нет).

#### *Comodo AntiVirus*

**Comodo AntiVirus** — бесплатный антивирус с закрытым кодом компании Comodo для Microsoft Windows XP, Vista, Windows 7 и Windows 8. Comodo AntiVirus входит в состав Comodo Internet Security.

#### **Возможности:**

- Ежедневные, автоматические обновления антивирусных баз.
- Изолирование подозрительных файлов в карантин для предотвращения инфекции.

- Проактивная защита включает в себя HIPS (Host Intrusion Prevention Systems) — система отражения локальных угроз. Задачей HIPS является контроль за работой приложений и блокировка потенциально опасных операций по заданным критериям.

#### *Dr. Web*

**Dr.Web** (рус. *Доктор Веб*) — общее название семейства программного антивирусного ПО для различных платформ (Windows, OS X, Linux, мобильные платформы). Разрабатывается компанией «Доктор Веб».

#### **Возможности:**

- Возможность установки на зараженную машину.
- Обнаружение и лечение сложных полиморфных, шифрованных вирусов и руткитов.
- Компактная вирусная база и небольшой размер обновлений. Одна запись в вирусной базе позволяет определять до тысячи подобных вирусов.
- Обновления вирусных баз производятся немедленно по мере выявления новых вирусов, до нескольких раз в час. Разработчики антивирусного продукта отказались от выпуска обновлений вирусных баз по какому-либо графику, поскольку вирусные эпидемии не подчиняются таковым.
- Кроссплатформенность — используется единая вирусная база и единое ядро антивирусного сканера на разных plataформах ОС.
- **Fly-code** — эмулятор с динамической трансляцией кода, реализующий механизм универсальной распаковки вирусов, защищённых от анализа и детектирования одним или цепочкой новых и/или неизвестных упаковщиков, крипторов и дропперов. Это позволяет распаковывать файлы, защищенные, к примеру, ASProtect, EXECryptor, VMProtect и тысячами других упаковщиков и протекторов, включая неизвестные антивирусу.

#### *ESET NOD32*

**ESET NOD32** — антивирусный пакет, выпускавшийся словацкой фирмой ESET. Первая версия была выпущена в конце 1987 года. Большая часть кода антивируса написана на языке ассемблера, поэтому для него характерно малое использование системных ресурсов и высокая скорость проверки с настройками по умолчанию.

#### **Особенности:**

- Из участников тестирования Virus Bulletin 100% Eset NOD32 обладает наибольшим среди тестируемых (80 по состоянию на Июль 2013 года) количеством наград данной лаборатории.
- Компактный размер обновлений (размер измеряется десятками килобайт).

#### *Антивирус Касперского*

**Антивирус Касперского** — антивирусное программное обеспечение, разрабатываемое Лабораторией Касперского. Предоставляет пользователю защиту от вирусов, троянских программ, шпионских программ, руткитов, adware, а также неизвестных угроз с помощью проактивной защиты, включающей компонент HIPS.

#### **Возможности:**

- Проверка Java- и Visual Basic-скриптов
- Анализ и устранение уязвимостей в браузере Internet Explorer
- Возможность установки программы на заражённый компьютер

Антивирус часто получает положительные отзывы за достаточно высокий уровень выявления вредоносных программ, как и часто критикуется пользователями, за большое количество ложных срабатываний. Один из самых известных недостатков, — крайне большая ресурсоёмкость программы. Антивирус Касперского критикуют за его избыточную назойливость. Например, знаменитый «поросечий визг» — звук при обнаружении вирусов в старых версиях программы, который пугал большинство пользователей. И хотя этот звук убрали в седьмой версии программы, у многих пользователей антивирус Касперского ассоциируется с этим звуком.

*Рейтинг антивирусов по количеству скачиваний с сайта Comss.ru за 2014:*

1. AVAST Software
2. Kaspersky Lab
3. Doctor Web
4. AVG Technologies
5. ESET
6. Comodo
7. Перфомикс/Adguard
8. Qihoo 360
9. Avira Operations
10. Symantec/Norton

### [Вирусы для android](#)

В конце апреля 2013 эксперты компании Lookout Mobile Security обнаружили на Google Play тридцать две программы, содержащие инфицированную библиотеку. Вредоносный компонент был интегрирован под видом стандартной рекламной функции в игры, словари и утилиты уже после того, как они успешно прошли проверку в Google Play.

Всего заражённые программы были скачены около девяти миллионов раз. Помимо традиционных рекламных сообщений пользователи получали заведомо ложные предупреждения о мнимых угрозах и критических обновлениях. При клике на таком баннере происходило перенаправление на заражённый или фишинговый веб-сайт.

Скрыто от пользователя троянская компонента подключалась к контролируемым злоумышленниками удаленным серверам и передавали своим создателям собранную информацию, включая номер мобильного телефона и его IMEI. Дополнительно загружался троян семейства AlphaSMS, отправлявший SMS на платные номера.

Апрельский скандал получил продолжение. Недавно специалистами компании Webroot на Google Play был обнаружен очередной троян, распространяемый под видом программы управления шрифтами. Во всех случаях пострадавшие либо не пользовались антивирусами, либо последние оказались неэффективны.

## Червь Морриса.

В 1988 году Робертом Моррисом-младшим был создан первый массовый сетевой червь. 60 000-байтная программа разрабатывалась с расчётом на поражение операционных систем UNIX Berkeley 4.3. Вирус изначально разрабатывался как безвредный и имел целью лишь скрытно проникнуть в вычислительные системы, связанные сетью ARPANET, и остаться там необнаруженным. Вирусная программа включала компоненты, позволяющие раскрывать пароли, имеющиеся в инфицированной системе, что, в свою очередь, позволяло программе маскироваться под задачу легальных пользователей системы, на самом деле занимаясь размножением и рассылкой копий. Вирус не остался скрытым и полностью безопасным, как задумывал автор, в силу незначительных ошибок, допущенных при разработке, которые привели к стремительному неуправляемому саморазмножению вируса.

По самым скромным оценкам инцидент с червём Морриса стоил свыше 8 миллионов часов потери доступа и свыше миллиона часов прямых потерь на восстановление работоспособности систем. Общая стоимость этих затрат оценивается в 96 миллионов долларов (в эту сумму, также, не совсем обосновано, включены затраты по доработке операционной системы). Ущерб был бы гораздо больше, если бы вирус изначально создавался с разрушительными целями.

Червь Морриса поразил свыше 6200 компьютеров. В результате вирусной атаки большинство сетей вышло из строя на срок до пяти суток. Компьютеры, выполнявшие коммутационные функции, работавшие в качестве файл-серверов или выполнявшие другие функции обеспечения работы сети, также вышли из строя.

## Написание собственного червя.

1. Войдите в систему как администратор.
2. Откройте диск C: и создайте там папку Programs.
3. Откройте Блокнот и введите: @echo off
4. На второй строке введите: Copy C:\Programs\virus.bat C:\Programs. На третьей строке введите: Start C:\Programs\virus.bat
5. Нажмите «Сохранить как» и сохраните файл под именем virus.bat в созданной папке Programs.
6. Вставьте его ярлык в папку автозагрузки.
7. Вы успешно написали вирус (червь). Чтобы запустить его, перезагрузите компьютер и на диске C: не останется свободного места.

## Приложение

### Список вопросов, возникнувших в ходе обсуждения

#### 1. IT-проект

##### IT-проект и его структура

- Каким образом заказчик может вносить изменения в проект? (Белый А.А.)
- Чем отличается куратор проекта от менеджера? (Белый А.А.)
- Какими навыками должен обладать менеджер? (Белый А.А.)
- Что обязана делать группа людей, занимающаяся внедрением готового продукта? (Борисевич П.И.)
- Кто контролирует аудит проекта? (Гетьман С.И.)
- Как планируется бюджет проекта, и кто является инвестором? (Гетьман С.И.)
- Что означает "развертывание IT-инфраструктуры"? (Гетьман С.И.)
- Кто и зачем занимается документацией проекта? (Гетьман С.И.)
- Как влияет изменение состава исполнителей на развитие проекта? (Гетьман С.И.)
- Кто такие подрядчики и субподрядчики, их роль и структура? (Григорьев А.В.)
- Если какие-нибудь дополнения к схеме участников проекта? (Грушевский А.А.)
- Отличия open source проекта от коммерческого? (Грушевский А.А.)
- Какая роль на проекте является самой важной, а какая самой незначительной? (Ипатов А.Е.)
- Какие роли требуют высокой квалификации? (Ипатов А.Е.)
- Какие существуют части планирования? (Ипатов А.Е.)
- Какие роли могут совмещаться, а какие нет? (Ипатов А.Е.)
- Как организуется взаимодействие между командой и руководством? (Лебедев Н.А.)
- Бизнес-аналитики в структуре проекта? (Лебедев Н.А.)
- Какова роль менеджера проекта? (Лебедев Н.А.)
- Какова роль руководителей компаний в структуре проекта? (Лебедев Н.А.)
- Самая ответственная должность на проекте? (Лебедев Н.А.)
- В чем заключается принципиальное различие в работе куратора и руководителя проекта? (Михальцова А.Ю.)
- Что означает фраза "проект структурирован"? (Михальцова А.Ю.)
- Какими способами можно контролировать выполнение проекта? (Ровдо Д.И.)
- Какие должности в структуре проекта обязательны, а какие нет? (Ровдо Д.И.)
- Насколько важную роль в проекте играют подрядчики? (Ровдо Д.И.)
- Как можно разрешить ситуацию, когда кто-то из структуры (руководство, исполнители) не может выполнять свою работу из-за нехватки знаний, опыта? (Трубач Г.Г.)
- Как происходит общение бизнес-аналитика с заказчиком? (Трубач Г.Г.)
- Как разработчики получают прибыль из open source проектов? (Трубач Г.Г.)
- Какова структура open source и фриланс проектов? (Трубач Г.Г.)
- Есть ли отличия между структурами IT-проектов в Беларуси и проектов других стран? Если есть, то какие? (Ярошевич Я.О.)
- Как осуществляется взаимодействие в команде между собой, если проект является международным? (Ярошевич Я.О.)
- Как происходит выбор подходящей команды для проекта? (Ярошевич Я.О.)

- Как организуется структура проекта, создающийся не в IT –компании?  
(Ярошевич Я.О.)

#### Жизненный цикл IT-проекта и фазы разработки ПО

- На каком этапе реализуется поддержка проекта? (Белый А.А.)
- Существуют ли отличия между жизненным циклом обычного проекта и IT-проекта? (Белый А.А.)
- Как осуществляется интеграция проекта? (Белый А.А.)
- С какой целью созданы международные стандарты жизненного цикла и в каких случаях их требуют к выполнению? (Борисевич П.И.)
- Насколько тщательно подходят к планированию жизненного цикла? (Гетьман С.И.)
- Как справиться с постоянным обновлением и улучшением различных платформ разработки? (Гетьман С.И.)
- Как долго длится сопровождение продукта? (Гетьман С.И.)
- Влияет ли бюджет проекта на его жизненный цикл? (Гетьман С.И.)
- Суть и назначение пост-релизного этапа разработки. (Григорьев А.В.)
- Роль планирования в процессе разработки проекта (Григорьев А.В.)
- Почему планирование проекта требует малых затрат? (Лебедев Н.А.)
- На какой стадии жизненного цикла определяется архитектура проекта?  
(Лебедев Н.А.)
- Что может произойти с проектом, если при планировании допущена архитектурная ошибка? (Лебедев Н.А.)
- Можно ли выпускать проект на альфа-стадии его разработки? (Михальцова А.Ю.)
- Чем отличаются стадии разработки "релиз кандидат" от "публичной реализации"?  
(Михальцова А.Ю.)
- Означает ли фаза "завершение", что проект уже окончательно готов к использованию?  
(Михальцова А.Ю.)
- Когда занимаются поддержкой проекта, если в жизненном цикле отсутствует фаза post-release? (Ровдо Д.И.)
- На какой стадии лучше всего начинать тестирование продукта? (Ровдо Д.И)
- Какие существуют подэтапы этапа планирования? Какие из них являются наиболее важными? (Ровдо Д.И)
- Каков жизненный цикл open source проектов и стартапов? (Трубач Г.Г.)
- Какие существуют стандарты жизненного цикла? (Трубач Г.Г.)
- Какие существуют варианты выхода из ситуации, когда заказчик отказывается от проекта и возможен ли такой случай? (Трубач Г.Г.)
- Возможно ли возвращение от более позднего этапа жизненного цикла к более раннему?  
Если возможно, то по каким причинам? (Ярошевич Я.О.)
- Как соотносятся между собой этапы жизненного цикла по времени? (Ярошевич Я.О.)
- Осуществляется ли поддержка проекта после релиза и в каких случаях? (Ярошевич Я.О.)

## Моделирование жизненного цикла

- Как в Scrum методологии осуществляется контроль рисков? (Белый А.А.)
- Какая методология наиболее универсальна? (Белый А.А.)
- Можно ли менять методологию в процессе разработки ИТ-проекта? (Борисевич П.И.)
- Когда используется модель водопада? (Борисевич П.И.)
- Кто выбирает методологию? (Гетьман С.И.)
- Какая методология наименее затратная? (Гетьман С.И.)
- Какие методологии используют наиболее успешные компании? (Гетьман С.И.)
- Какие основные отличия между спиральной и итерационной моделью? (Григорьев А.В.)
- В чем суть экстремального программирования? (Григорьев А.В.)
- Суть Scrum методологии и ее виды. (Григорьев А.В.)
- Какая модель наиболее эффективна? (Ипатов А.Е.)
- Какая методология наиболее универсальна? (Ипатов А.Е.)
- В чем заключается простота Scrum методологии? (Лебедев Н.А.)
- Как происходит переход между спринтами в Scrum методологии? (Лебедев Н.А.)
- В чем преимущества и недостатки каждой из методологий? Какая из них наиболее оптимальна? (Лебедев Н.А.)
- Каковы минусы SCRUM-а, если они есть. (Ровдо Д.И.)
- Какая модель самая оптимальная? (Ровдо Д.И.)
- Почему модель "спираль" довольно известна? Ведь, по сути, она почти не отличается от итерационной. Какие есть особенности этой модели, которые отличают ее от других? (Ровдо Д.И.)
- Какая методология сейчас используется наиболее часто? (Трубач Г.Г.)
- От чего зависит выбор той или иной модели? (Трубач Г.Г.)
- Можно ли изменять спринт при Scrum методологии? (Трубач Г.Г.)
- Чем грозит срыв спрингта при Scrum методологии? (Трубач Г.Г.)
- Можно ли изменять сроки спрингта при Scrum методологии? (Трубач Г.Г.)
- Каковы основные принципы экстремального программирования? Как происходит работа с заказчиком в данной методологии? (Трубач Г.Г.)
- Какая методология сейчас наиболее популярна? (Щавровский С.А.)
- Каковы особенности Scrum методологии? (Щавровский С.А.)
- Примеры использования тех или иных методологий? (Щавровский С.А.)
- Какая модель используется чаще других? (Ярошевич Я.О.)
- В чем разница между итеративной и спиральной моделью? (Ярошевич Я.О.)
- Как выбрать лучшую модель для конкретного проекта? (Ярошевич Я.О.)
- Проводят ли компании тренинги по изучению той или иной модели? (Ярошевич Я.О.)
- Какие существуют методологии, придуманные самими компаниями? (Ярошевич Я.О.)

## Мозговой штурм

- Может ли мозговой штурм проводится удаленно (например, по Skype)? (Гетьман С.И.)
- Что произойдет, если при мозговом штурме, в команде находятся люди, которые друг с другом в плохих отношениях? (Гетьман С.И.)
- Как можно сорвать мозговой штурм? (Гетьман С.И.)

- Проводятся ли мозговые штурмы в крупных компаниях? (Григорьев А.В.)
- Кто оценивает и отсеивает идеи: вся группа или кто-то конкретный? (Григорьев А.В.)
- Почему 6-12 человек при мозговом штурме является оптимальным количеством? Уложится ли команда в отведенное время? (Лебедев Н.А.)
- Можно ли провести мозговой штурм с самим собой? (Лебедев Н.А.)
- Почему для мозговых штурмов так важно отсутствие критики? (Михальцова А.Ю.)
- Кто отвечает за анализ идей/решений? (Михальцова А.Ю.)
- Есть ли у мозговых штурмов чёткие временные ограничения? (Михальцова А.Ю.)
- Какого рода вопросы/проблемы решаются во время мозговых штурмов? (Михальцова А.Ю.)
- Как часто стоит устраивать мозговые штурмы? (Михальцова А.Ю.)
- Почему важнее количество высказанных идей, чем их качество? (Михальцова А.Ю.)
- Существует ли разделение ролей при мозговом штурме? (Трубач Г.Г.)
- Какова продолжительность рабочего дня при мозговом штурме? (Трубач Г.Г.)
- Часто ли используется методика мозгового штурма? (Щавровский С.А.)
- Какие известные команды используют методику мозгового штурма? (Щавровский С.А.)
- Будете ли вы продвигать методику мозгового штурма в своем рабочем окружении? (Щавровский С.А.)
- Где мозговой штурм может быть применен? (Ярошевич Я.О.)
- Как предотвратить хаос во время мозгового штурма (вечные споры, ругательство и т.д.)? (Ярошевич Я.О.)
- Как из практик мозгового штурма наиболее эффективна? (Ярошевич Я.О.)

### [Классификация программного обеспечения](#)

- Что мешает использовать свободное по с GPL, для своих приложений, вместо того, чтобы платить за "снятие лицензии"? (Как именно снимается лицензия) (Белый А.А.)
- Какие альтернативы свободные ОС есть для обеспечения независимости, а также удобства для пользователя (Белый А.А.)
- Какая типовая лицензия для Open Source приложений подходит для использования в коммерческих целях? (Борисевич П.И.)
- Что могут сделать нарушителю GPL? (Борисевич П.И.)
- Как выбирать лицензию для свободного софта? (Борисевич П.И.)
- В чём особенность MIT License? В чём её основные преимущества и недостатки в сравнении с GPL? (Гетьман С.И.)
- В чём особенности лицензирования GitHub проектов? (Гетьман С.И.)
- Как свободное ПО влияет на рынок и на зарплаты программистов, менеджеров? (Гетьман С.И.)
- На какие хлеба живёт Ричард Столман? (Гетьман С.И.)
- Как Open Source / free лицензии борются с вредителями? (Гетьман С.И.)
- Насколько важен выбор лицензии для разработки ПО? Бывали ли случаи судебных тяжб из-за неправильно выбранной лицензии? (Гетьман С.И.)
- Какие существуют типы лицензий Open Source? (Ипатов А.Е.)
- На твой взгляд, есть ли перспектива развития направления Open Source? (Ипатов А.Е.)
- Как происходит тестирование Open Source проектов? (Ипатов А.Е.)

- Назови какие-нибудь популярные Open Source проекты. (Ипатов А.Е.)
- Существует ли свободное ПО, которые нарушает 4 свободы? (Лебедев Н.А.)
- В чем выгода для организации, которая работает под Open Source? (Лебедев Н.А.)
- Как сложилось, что образовались такие кластеры как Freeware, FreeSoft, ComSoft? (Лебедев Н.А.)
- Как заработать на рынке Open Source? (Лебедев Н.А.)
- На ваш взгляд, какое будущее у Open Source? (Лебедев Н.А.)
- Существуют другие лицензии, кроме GPL? Если есть, то почему нет единого стандарта? (Лебедев Н.А.)
- Какие проблемы с законом могут возникнуть при использовании Open Source? (Михальцова А.Ю.)
- Любой ли желающий может использовать Open Source? (Михальцова А.Ю.)
- Считаешь ли ты эффективным использование Open Source и почему? (Михальцова А.Ю.)
- Возможно ли в дальнейшем переведение всего ПО на открытое? (Михальцова А.Ю.)
- Как именно владельцы свободного ПО с лицензией GPL обходят ее и продают возможность пользоваться своим продуктом, не "заражая" покупателей? (Ровдо Д.И.)
- Почему все вокруг так увлечены Open Source проектами? (Ровдо Д.И.)
- В чем смысл подхода владельцев софта с открытым исходным кодом, но запретом его использовать? (Ровдо Д.И.)
- Как в проектах отслеживается тип лицензии? (Трубач Г.Г.)
- Возможно ли в открытых приложениях использовать проприетарный софт и наоборот? (Трубач Г.Г.)
- Какие самые известные Open Source проекты существуют? (Трубач Г.Г.)
- Какая ответственность предполагается за использование Open Source проектов в коммерческих целях? (Трубач Г.Г.)
- Примеры свободного программного обеспечения? (Ярошевич Я.О.)
- Можно ли отойти от одного или нескольких пунктов из десяти пунктов в определении Open Source программного обеспечения, и всё равно считать его таковым? (Ярошевич Я.О.)
- Примеры Open Source программного обеспечения? (Ярошевич Я.О.)

## 2. Менеджмент IT-проекта

### Менеджмент IT-проекта

- Имеет ли место в практике, что один менеджер контролирует несколько проектов? (Белый А.А.)
- Какие существующие меры контроля рисками? (Белый А.А.)
- Чем отличается работа менеджера на крупном проекте и на стартапе? (Белый А.А.)
- Какой человек может стать менеджером? (Белый А.А.)
- Можно ли рассчитывать на успех команды менеджмента проекта в новом проекте, если она хорошо справилась с предыдущим? (Борисевич П.И.)
- Как можно оценить деятельность команды менеджмента проекта? (Борисевич П.И.)
- Насколько глубокими должны быть технические знания менеджера в IT-проекте? (Григорьев А.В.)
- Может ли менеджер быть одновременно разработчиком в том же проекте? (Григорьев А.В.)

- Может ли в проекте быть одновременно несколько менеджеров? Если да, то могут ли пересекаться их сферы деятельности? (Григорьев А.В.)
- Чем отличаются полномочия менеджера от его полномочий его помощников? Где они пересекаются и в чем друг друга дополняют? (Грушевский А.А.)
- Должен ли менеджер знать некоторую техническую составляющую проекта? (Ипатов А.Е.)
- Может ли менеджер принимать решение об отказе от того или иного заказа? (Ипатов А.Е.)
- Может ли менеджер выполнять еще какие-либо роли или ему стоит сосредоточиться на своей роли? (Ипатов А.Е.)
- С помощью каких программных или аналитических средств менеджер строит бизнес-модель? (Лебедев Н.А.)
- Направление развития проекта определяется только бизнес-моделью менеджера или существуют другие определяющие факторы? (Лебедев Н.А.)
- Почему менеджер не учитывает тенденций рынка на этапе анализа? (Лебедев Н.А.)
- В чем заключается работа менеджера? (Михальцова А.Ю.)
- Разработка проекта считается непрерывным процессом? (Михальцова А.Ю.)
- Что включает check-лист? (Михальцова А.Ю.)
- Может ли заказчик выступать в качестве менеджера? (Михальцова А.Ю.)
- Какая группа успешнее выполняет проект, поставленную задачу? (Михальцова А.Ю.)
- Может ли менеджер устанавливать сроки проекта? (Михальцова А.Ю.)
- Менеджер не обязательно должен владеть техническими данными, но в его обязанности входит пункт разбиения задачи на подзадачи. Тогда получается, что у менеджера должны быть помощники? (Ровдо Д.И.)
- Какие инструменты влияния есть для обычных работников (не фрилансеров)? (Ровдо Д.И.)
- Должен ли менеджер интересоваться ситуацией и разговаривать с командой или достаточно действий командного лидера? (Ровдо Д.И.)
- Является ли проблема разногласий в команде проблемой менеджера? (Ровдо Д.И.)
- Каким образом определяются и строятся бизнес-модели проекта? (Трубач Г.Г.)
- Как происходит обсуждение вопросов менеджмента и бизнес-части проекта? (Трубач Г.Г.)
- Каковы особенности метода менеджмента проекта — делегирования? (Трубач Г.Г.)
- Чем характерен метод менеджмента — «разделяй и властвуй»? (Трубач Г.Г.)
- Полезно ли записывать (логирование) все события менеджмента (разногласия, соглашения, задачи и т.д.)? (Трубач Г.Г.)
- Почему среди перечисленных ресурсов не было ресурса времени? (Щавровский С.А.)
- Как происходит менеджмент проекта в open-source проектах? (Щавровский С.А.)
- Какие есть решения по автоматизации менеджмента проекта? (Щавровский С.А.)
- Какое образование должен иметь менеджер и его команда? (Ярошевич Я.О.)
- Кто несет наказание в случае нарушения сроков? Что произойдет, если не учтутся некоторые факторы и сроки затянутся? (Ярошевич Я.О.)
- Общается ли менеджер непосредственно с заказчиком? (Ярошевич Я.О.)
- Может ли менеджер работать удаленно? (Ярошевич Я.О.)

#### [Роли в команде и их функции](#)

- Кто распределяет роли в команде? (Белый А.А.)

- Кто такой проектный архитектор? (Белый А.А.)
- Могут ли совмещаться роли разработчика и QA? Применимо ли это на практике? (Белый А.А.)
- Как происходит повышение должности работника? (Белый А.А.)
- Почему нежелательно, чтобы командный лидер и менеджер проекта был одним человеком? (Борисевич П.И.)
- На каком этапе жизненного цикла распределяются роли в проекте? (Борисевич П.И.)
- Может ли заказчик сам назначать роли? (Гетьман С.И.)
- Сколько ролей оптимально использовать на проекте? (Гетьман С.И.)
- Сколько ролей может иметь один человек в команде? (Гетьман С.И.)
- С чем может быть связано изменение ролей? (Гетьман С.И.)
- Сколько ролей оптимально использовать на проекте? (Гетьман С.И.)
- В чем разница бизнес компетенции в кластере управление продуктом и кластере управления выпуском? (Гетьман С.И.)
- Кого лучше взять в команду: узкопрофильного или многопрофильного специалиста? (Гетьман С.И.)
- Может ли меняться роль работника на протяжении проекта? (Григорьев А.В.)
- Может ли один человек выполнять сразу несколько ролей? (Григорьев А.В.)
- Является ли выделение родственных функций в роли обязательным или оно носит формальный характер? (Григорьев А.В.)
- Зависит ли наличие определенных ролей от выбранной модели разработки? И от чего зависит наличие той или иной роли? (Григорьев А.В.)
- Кто занимается консультациями при разработке? (Грушевский А.А.)
- Кто помогает заказчику понимать документацию по отчетности разработки, технических возможностей проекта и т.д.? (Грушевский А.А.)
- Кому подчиняется архитектор? (Грушевский А.А.)
- Что входит в документацию пользователя? (Грушевский А.А.)
- На каком этапе происходит распределение ролей? (Ипатов А.Е.)
- На всех ли проектах у одной роли одинаковые функции? (Ипатов А.Е.)
- Какие роли являются ключевыми, а какие нет? (Ипатов А.Е.)
- Чем отличается кластер управление программой от кластера управление продуктом? (Лебедев Н.А.)
- Как командный лидер оценивает компетентность своей команды? (Лебедев Н.А.)
- Какие роли можно совмещать? (Лебедев Н.А.)
- Какая ситуация наиболее приемлема: сотрудник работает по конкретному плану или сотрудник сам определяет свои обязанности? (Лебедев Н.А.)
- Самые важные роли, с которыми можно начинать разработку? (Лебедев Н.А.)
- Кто распределяет роли в команде? (Михальцова А.Ю.)
- Четко ли соблюдается распределение ролей в команде или возможны совмещения? (Михальцова А.Ю.)
- По каким принципам распределяются роли в различных моделях жизненного цикла? (Михальцова А.Ю.)

- Участвует ли командный лидер в непосредственной разработке проекта? (Михальцова А.Ю.)
- Если командный лидер разъясняет требования заказчика разработчикам, то каковы обязанности бизнес аналитика? (Михальцова А.Ю.)
- Какие роли совмещать нежелательно? (Михальцова А.Ю.)
- Какая из ролей находится выше, а как ниже в структуре проекта? (Ровдо Д.И.)
- Некоторые роли имеют некоторые схожие функции. Получается, что некоторые функции пересекаются между разными ролями? (Ровдо Д.И.)
- Кто такой product owner? (Ровдо Д.И.)
- Как можно сопоставить качества человека и роли? (Щавровский С.А.)
- Возможно ли создать команду без разделения ролей? (Щавровский С.А.)
- Кто такие full-stack разработчики? (Щавровский С.А.)
- Какие роли лучше всего сочетать? (Щавровский С.А.)
- Кто занимается обучением сотрудников в IT-компаниях? (Ярошевич Я.О.)
- Какие навыки нужны для роли командного лидера? (Ярошевич Я.О.)
- Бывают ли такие проекты, на которых отсутствует QA-менеджер? (Ярошевич Я.О.)
- Какими ролями можно пренебречь? (Ярошевич Я.О.)

#### [Совместная работа в проектировании](#)

- Почему именно такие условия совместной работы выделяют? (Белый А.А.)
- Могут ли два человека заменить целую команду разработчиков? (Белый А.А.)
- Как можно повысить эффективность совместной работы? (Борисевич П.И.)
- Какой оптимальное число человек должно быть в хорошей команде? (Борисевич П.И.)
- Как следует выбирать сотрудников для совместной работы? (Борисевич П.И.)
- Как предотвратить перетягивание одеял в случае совместной работы двух команд/компаний? (Гетьман С.И.)
- Что важно в первую очередь: прибыль или расход времени на согласование моментов? (Гетьман С.И.)
- Возможна ли совместная работа над авторским проектом? (Гетьман С.И.)
- Что лучше: как можно быстрее выйти на рынок, или довести продукт до идеала и выпустить его? (Григорьев А.В.)
- Есть ли примеры крупных проектов, реализованных в одиночку? (Григорьев А.В.)
- Какие есть технологии для реализации эффективной совместной работы? (Григорьев А.В.)
- Как быть, если люди в одной команде с чем-то не согласны друг с другом? (Григорьев А.В.)
- Важно ли хорошее отношение в коллективе при совместной работе? (Михальцова А.Ю.)
- Какие можно предложить способы организации процесса разработки? (Михальцова А.Ю.)
- Есть ли какой-то процесс отбора людей в команду проектировщиков? (Михальцова А.Ю.)
- Может ли хорошо организован проект, в котором около 200 человек? (Ровдо Д.И.)
- Что делать, если проект настолько большой, что двое разработчиков не справятся? (Ровдо Д.И.)
- Насколько реально работать без системы контроля версий? (Ровдо Д.И.)
- Чем плохо, если проектированием занимается большое количество человек? (Трубач Г.Г.)

- Говорят, одна голова хорошо, а две лучше и так далее. Почему же тогда два проектировщика лучше, чем один? (Щавровский С.А.)
- Можно ли грамотно выстроить процесс без лидера? (Ярошевич Я.О.)
- Какие преимущества у проекта, который разработан командой, над проектом, который создан в одиночку? (Ярошевич Я.О.)

### **Дистанционное сотрудничество**

- Достигла ли система дистанционного сотрудничества такого уровня, что может заменить встречи вживую? (как скоро это может произойти) (Белый А.А.)
- Что лучше: большая компания в одном месте, где удобно проводить личные встречи всем участникам проекта, либо обособленные компании, находящиеся в разных местах, где контактировать приходится посредством интернет-технологий? (Белый А.А.)
- Какой главный недостаток современных средств для дистанционного сотрудничества? (Белый А.А.)
- Как ПО для дистанционного сотрудничества обходить проблему часовых поясов? (Гетьман С.И.)
- Видео и аудиоконференции уже существуют. А где голограмки из Star Wars? (Гетьман С.И.)
- Выгодно ли компаниями разрабатывать свои внутренние системы дистанционного сотрудничества? Или существуют свободные альтернативы? (Гетьман С.И.)
- Какие есть лучшие представители ПО для совместной работы. (Гетьман С.И.)
- Были ли попытки обмана через видео ПО для дистанционного сотрудничества? Как они пресекались? Как вообще их можно свести к минимуму? (Гетьман С.И.)
- Как в подобном ПО обеспечивается конфиденциальность данных? (Гетьман С.И.)
- Какие есть варианты заработка на поприще ПО для совместной работы и дистанционного сотрудничества? (Гетьман С.И.)
- Есть такой фильм, "Мне бы в небо" с Д. Клуни в главной роли. Суть, вкратце, такова: ГГ (главный герой) летает по штатам, нанося визит в крупные компании, где он с глазу на глаз сообщает сотрудникам о том, что они уволены. Однако, более молодое поколение, решает привнести в такой затратный процесс (перелёты весьма затратны, даже у них в США) технологические нотки, упростив всё до видеоконференций, на которых сообщается неожиданная для сотрудника новость. Вот как вы считаете, будет ли кощунством перевести подобную систему (с глазу на глаз) на рельсы техники? (Гетьман С.И.)
- А не слишком ли мы уходим от реальности из-за подобного ПО и подходов? (Гетьман С.И.)
- Были ли попытки совместить ПО для совместной работы с виртуальной реальностью? К примеру, сделать что-то в стиле Работников-Суррогатов? (Гетьман С.И.)
- Для чего происходит разделение производства и проектирования? Какие это влечёт плюсы? (Григорьев А.В.)
- Являются ли open-source проекты своего рода проектами с разделением разработки? (Григорьев А.В.)
- Какие из мессенджеров наиболее популярны для бизнес-общения? Какие используются в той или иной ситуациях? (Грушевский А.А.)

- Какие средства сотрудничества, на твой взгляд, будут применяться в будущем? (Грушевский А.А.)
- В каких сферах, связанных с Software Engineering, можно без проблем отказаться от личных встреч, а в каких невозможно? (Грушевский А.А.)
- По каким критериям выделяют качественные интерфейсы? (Ипатов А.Е)
- Какое из средств является наиболее популярным? (Ипатов А.Е)
- Существуют ли 3D конференции? (Ипатов А.Е)
- Чем грозит отсутствие личных встреч при разработке? (Лебедев Н.А.)
- Есть ли советы для организации дистанционного сотрудничества? Где их найти? (Лебедев Н.А.)
- В чем преимущества дистанционного сотрудничества? (Лебедев Н.А.)
- Эффективно ли разрабатывать составляющие одного продукта в нескольких странах одновременно? Ведь это требует больших финансовых издержек. (Михальцова А.Ю.)
- Как компании, расположенные в разных городах, решают проблему огромной разницы в часовом поясе? (Михальцова А.Ю.)
- Безопасно ли дистанционное взаимодействие? Не может ли произойти утечка информации о разработке продукта и т.д.? (Михальцова А.Ю.)
- Какая будет в будущем замена скайпу? (Ровдо Д.И.)
- Неужели до сих пор для обмена информацией используют бумажные письма? (Ровдо Д.И.)
- Как ты думаешь, что приносит больший доход из расчета на единицу времени - небольшая компания, расположенная в одном месте, или побольше, расположенная в разных? (Ровдо Д.И.)
- Какие есть методики решения проблемы, когда члены команды находятся далеко друг от друга? (Трубач Г.Г.)
- Нужно ли команде собираться вместе или достаточно дистанционного сотрудничества? (Трубач Г.Г.)
- Какие есть системы контроля прогресса выполнения проекта (task-manager может)? (Щавровский С.А.)
- Понятно, что разделение проектирования и разработки в некоторых случаях очень выгодно, но как экономия средств оказывается на безопасности коммуникаций между группами? Как этот вопрос решается? (Щавровский С.А.)
- Как удаленная работа оказывается на профессионализме работника и на качестве итогового продукта? (Щавровский С.А.)
- Какие есть примеры средств проведения видеоконференций? (Ярошевич Я.О.)
- Как проходят видеоконференции, если стороны говорят на разных языках? Как в диалог вклинивается переводчик? (Ярошевич Я.О.)

#### **Стратегии развития крупнейших и наиболее известных ИТ-компаний**

- Какие есть аналоги кремниевой долины в мире, какие компании там представлены? (Борисевич П.И.)
- Какое место в рейтинге самых дорогих компаний занимает Tesla Motors? Верите ли вы что эта компания через 10 лет обгонит Apple как обещает Элон Маск? (Борисевич П.И.)

- Какие не очень большие компании имеют самые высокие темпы роста в последнее время? (Борисевич П.И.)
- Расскажи, пожалуйста, историю надкусанного яблока? (Григорьев А.В.)
- Почему продукция Apple была и остаётся такой популярной? (Григорьев А.В.)
- Возможно ли в Беларуси основать бизнес, который станет настолько крупным, как упомянутые? Почему нет? (Григорьев А.В.)
- Что из себя представляли табуляторы? (Григорьев А.В.)
- В чём заслуга IBM PC? (Григорьев А.В.)
- Какое место в рейтингах занимает Oracle? (Грушевский А.А.)
- Какая компания для тебя является самой успешной и почему? (Грушевский А.А.)
- Какие компании в нашей стране активно набирают обороты? (Ипатов А.Е.)
- Лучше делать упор на какую-то узкую сферу или же наоборот стараться внедрить свой продукт во все сферы? (Ипатов А.Е.)
- Были ли примеры быстрого старта компаний, а затем такого же быстрого падения? С чем это связано? (Ипатов А.Е.)
- Сможет ли в ближайшее время приблизиться к уровню Ерат какая-нибудь белорусская компания? (Ипатов А.Е.)
- В каких из белорусских компаний ты бы предпочла работать? (Ипатов А.Е.)
- Можно ли считать, что компания Apple Inc достигла своего успеха только благодаря Стиву Джобсу? (Михальцова А.Ю.)
- Есть ли смысл создавать свою компанию, если уже так много имеющихся крупных и сильно развитых? (Михальцова А.Ю.)
- Останется ли рейтинг IT-компаний таким же, если в качестве критерия оценки добавить еще качество производимой продукции? (Михальцова А.Ю.)
- Почему компания Sony прекратила производство ноутбуков? (Михальцова А.Ю.)
- А как так вышло, что Google так далеко от вершины? Ведь он гораздо более на слуху, чем тот же НР. (Ровдо Д.И.)
- Как ты думаешь, кто из перечисленных компаний останется в топе через 10 лет? (Ровдо Д.И.)
- У Amazon имеется своя БД – DynamoDB. Получается, это не только интернет магазин. А какие продукты у них еще имеются? (Ровдо Д.И.)
- Каковы технические характеристики Apple 1 и Apple 2? (Трубач Г.Г.)
- Почему Apple и Samsung постоянно копируют что-нибудь друг у друга? И как они разбираются с другими компаниями, которые копируют их разработки (китайцы, например)? (Трубач Г.Г.)
- Из-за чего произошел упадок компании IBM? (Трубач Г.Г.)
- Расскажи про компанию Sony поподробнее. И почему у них маленькие доходы (иногда вообще убытки) от мобильной отрасли (отделение ноутбуков уже вообще продали)? (Трубач Г.Г.)
- Какие еще есть известные IT компании, созданные белорусами? (Трубач Г.Г.)
- Есть ли какие-то собственные продукты у компании EPAM? (Щавровский С.А.)

- Почему среди первых 9 компаний из вашего списка, нету ни одной чисто софтверной? (Щавровский С.А.)
- Amazon занимается чем-нибудь еще, кроме магазина? (Щавровский С.А.)

### 3. Архитектура компьютера и мобильных устройств

#### Архитектура компьютера

- Какие существуют способы для ускорения передачи данных между ПЗУ (ОЗУ) и регистрами? (Белый А.А.)
- В чем существенное различие между поколениями ОЗУ (DDR2, DDR3, DDR4)? (Белый А.А.)
- Почему не используют закрытую архитектуру? (Гетьман С.И.)
- Какова максимальная пропускная способность системной шины? (Гетьман С.И.)
- Что означает разрядность процессора? (Григорьев А.В.)
- Может ли CD-диск выступать в качестве системного диска? (Григорьев А.В.)
- Существуют и используются ли компьютеры с открытой архитектурой? (Григорьев А.В.)
- Все ли компьютеры используют BIOS? (Григорьев А.В.)
- В чем различия распределения ресурсов в одно- и мультипроцессорных системах? (Грушевский А.А.)
- Как разрешаются конфликты доступа к ресурсам мультипользовательских UNIX-системах? (Грушевский А.А.)
- Какие существуют способы расширения возможностей компьютера? (Ипатов А.Е.)
- Особенности cash-памяти и ее уровни (L1, L2 и т.д.)? (Ипатов А.Е.)
- В чем отличия системной и адресной шин? (Ипатов А.Е.)
- Какие существуют аналоги центральной системной шины? Почему остановились на использовании магистралей? (Лебедев Н.А.)
- Можно ли жесткий диск отнести к модулям? (Михальцова А.Ю.)
- От каких параметров зависит частота импульсов и что можно сделать, чтобы ее увеличить? (Михальцова А.Ю.)
- Без каких компонентов архитектуры компьютер может работать (запускаться)? (Ровдо Д.И.)
- В чем заключается разница между HDD и SSD дисками? (Трубач Г.Г.)
- Как производится обмен данными внутри компьютера? (Трубач Г.Г.)
- Как устроен и какие функции выполняет сопроцессор? (Трубач Г.Г.)
- Существуют ли процессоры с разрядностью большей чем x64? (Трубач Г.Г.)
- Что произойдет, если вставить в нужный разъем на материнской плате планку оперативной памяти большего объема, чем это допустимо платой? (Трубач Г.Г.)
- Какие бывают типы жестких дисков и в чем их различия? (Щавровский С.А.)
- В чем структурные различия между графическим процессором и центральным процессором? (Щавровский С.А.)
- Каковы структурные различия ядра UNIX-систем и GNU/Linux? (Щавровский С.А.)
- Какие есть примеры универсальных компьютерных шин? (Ярошевич Я.О.)
- А какие процессы выполняются, когда компьютер выключается? (Ярошевич Я.О.)

#### Операционные и вычислительные системы

- Какие ОС преимущественно ставят на суперкомпьютеры и почему? (Белый А.А.)

- Как понять, что ОС построена некорректно? (Гетьман С.И.)
- Как обеспечивается на аппаратном уровне возможность работы на одном компьютере с несколькими ОС? (Гетьман С.И.)
- В чем заключается ненадежность ОС Windows? (Гетьман С.И.)
- В чем заключаются недостатки ОС X? (Гетьман С.И.)
- Есть ли будущее у индивидуальных вычислительных систем? (Гетьман С.И.)
- В чем недостатки тех или иных ОС? (Ипатов А.Е.)
- Как ОС наиболее популярна? (Ипатов А.Е.)
- Какие ОС лучше использовать при решении определенных задач и почему? (Лебедев Н.А.)
- Что следует учитывать при разработке ОС? (Лебедев Н.А.)
- Какая ОС наиболее популярна? (Михальцова А.Ю.)
- Каковы преимущества и недостатки наиболее известных и используемых ОС? (Михальцова А.Ю.)
- Что относят к системному ПО? (Михальцова А.Ю.)
- Какая ОС используется в сложных системах, где требуется сильная и стабильная безопасность? (Ровдо Д.И.)
- Какие существуют нестандартные ОС? (Ровдо Д.И.)
- Какие ОС используются в управлении самолетами? (Трубач Г.Г.)
- В чем заключаются проблемы ОС Windows? (Трубач Г.Г.)
- Каков прогноз на разработку игр на свободные ОС? (Трубач Г.Г.)
- Какие ОС входят в ТОП-5 по различным показателям (количеству пользователей, дружественности GUI и т.д.)? (Ярошевич Я.О.)
- Какой есть самый простой путь, чтобы написать свою ОС? (Ярошевич Я.О.)

#### Мобильные операционные системы

- Какие главные недостатки мобильных операционных систем Android, IOS? (Борисевич П.И.)
- Может ли мобильное устройство использовать несколько операционных систем? (Борисевич П.И.)
- Какая операционная система наименее энергозатратна? (Борисевич П.И.)
- Почему не делают общий магазин для различных мобильных ОС? (Гетьман С.И.)
- Почему Windows Phone не имеет такой популярности среди мобильных ОС, как Windows для ПК? (Гетьман С.И.)
- Что можно сказать о безопасности Android, iOS, а также их взлома (перепрошивки)? (Гетьман С.И.)
- На кого рассчитана Android? IOS? BlackBerry? Windows Phone? Sailfish? Ubuntu Touch? (Гетьман С.И.)
- Зачем нужны альтернативные магазины для Android? Одного мало? (Гетьман С.И.)
- Вероятно ли появление на рынке и обретение популярности мобильной ОС, созданной не гигантом вроде Microsoft, Google, Apple? (Гетьман С.И.)

- А создаются и используются ли ОС, которые обеспечивают максимально минималистичный интерфейс и функционал (чтобы сделать телефон прочным телефоном вроде Nokia 3310)? (Гетьман С.И.)
- Почему не производят мобильные устройства, на которые можно поставить и Windows Phone, IOS или Android? Как скоро нас ждут подобные решения? (Гетьман С.И.)
- Какие ещё существуют мобильные операционные системы и какие из них способны стать одними из лидеров? (Григорьев А.В.)
- Перспективно ли создание собственной мобильной операционной системы? (Григорьев А.В.)
- Отличия и преимущество открытых и закрытых операционных систем. (Григорьев А.В.)
- Для какой из озвученных операционных систем легче/выгоднее создавать свои приложения? (Григорьев А.В.)
- Возможность и способы заработка на своих приложениях для мобильных операционных систем. (Григорьев А.В.)
- Какие существуют альтернативные разработчики ОС Android (кроме официальных)? (Грушевский А.А.)
- Какие есть отличия и нововведения в разных версиях мобильных ОС? (Грушевский А.А.)
- В каких еще сферах возможно внедрение той или иной мобильной ОС? (Ипатов А.Е.)
- С каждым годом IOS все больше и больше набирает обороты, а Android наоборот постепенно начинает уступать. С чем это связано? (Ипатов А.Е.)
- Есть ли смысл создания единой ОС, включающей все преимущества той или иной ОС? Возможно ли это вообще? (Ипатов А.Е.)
- Какие существуют глобальные проблемы современных ОС помимо энергосбережения? (Ипатов А.Е.)
- За счет чего добиваются эффективного энергоснабжения? Чем приходится жертвовать? (Ипатов А.Е.)
- Что прежде всего стоит учесть при создании мобильного приложения? Есть ли отличия от разработки desktop приложений? (Лебедев Н.А.)
- В чем секрет успеха Android? (Лебедев Н.А.)
- Какие отличительные черты у каждой из мобильных ОС? (Лебедев Н.А.)
- Известно, что любая мобильная ОС часто обновляется. С чем это связано? (Лебедев Н.А.)
- Если ли отличия в качестве и производительности открытой от коммерческой мобильной ОС? (Лебедев Н.А.)
- Microsoft в свое время упустили мобильный рынок. Правда, что мобильный Windows — качественный продукт или это хороший маркетинг Microsoft? (Лебедев Н.А.)
- Сильно ли схожа функционально Мобильных ОС с ОС для ПК? (Михальцова А.Ю.)
- Если все Мобильные ОС функционально схожи, зачем тогда их такое большое количество? (Михальцова А.Ю.)
- Что можно предпринять для улучшения энергопотребления? (Михальцова А.Ю.)
- Бесплатна ли Мобильная ОС? (Михальцова А.Ю.)
- Удобна ли синхронизация (данных) мобильного телефона с ПК? (Михальцова А.Ю.)
- Преимущества Android для управления автомобилем. (Михальцова А.Ю.)
- Как ОС влияет на производительность мобильного устройства? (Трубач Г.Г.)

- В чем преимущество ОС IOS? Зачем люди скупают продукцию Apple, которая стоит неоправданно дорого? (Трубач Г.Г.)
- Что из себя представляет ОС Firefox? Это ОС, или какой-нибудь веб-браузер, работающий на движке Gecko? Если второе, то как работает ОС? (Трубач Г.Г.)
- Какая ОС самая (менее) энергосберегательная? (Трубач Г.Г.)
- Есть ли будущее у Windows Phone? (Щавровский С.А.)
- Почему у Android такой большой отрыв в количестве пользователей? (Щавровский С.А.)
- Почему Windows Phone не может завоевать рынок? (Щавровский С.А.)
- Что является основным критерием для вас при выборе мобильной ОС? (Щавровский С.А.)
- Каким образом мобильная ОС "понимает", что следует перейти в другой режим для энергосбережения? Какие есть режимы, и как это организовано? (Ярошевич Я.О.)
- Есть ли возможность эмуляции мобильной ОС? К примеру, специальный софт, который позволяет на устройстве с Android отображать ОС, подобную iOS? И для других аналогично? (Ярошевич Я.О.)
- Какую последовательность действий нужно совершить, чтобы выложить свое приложение в маркет на каждой из мобильных ОС, чтобы пользоваться им в дальнейшем? (Ярошевич Я.О.)
- Есть такое наблюдение, что одни и те же приложения на разных ОС стоят по-разному. Как правило, на iOS дороже. С чем это связано? В пример можно привести приложение maps.me, которое является бесплатным на ОС Android, но платным на ОС iOS. (Ярошевич Я.О.)

### [Энергосбережение](#)

- Может ли частое использование спящего режима ноутбука чем-то испортить батарею? (Борисевич П.И.)
- Как происходит переход в спящий режим жестких дисков, когда они не используются? В каких операционных системах это можно сделать? (Борисевич П.И.)
- Может ли длительное использование компьютера от зарядки испортить его батарею? (Борисевич П.И.)
- Как влияет на энергосбережение мобильных устройств режим полета (fly mode)? (Гетьман С.И.)
- Что такое троттлинг? (Гетьман С.И.)
- Полезно ли для сохранения аккумулятора разряжать его и перезаряжать чаще, чем держать его включенным в сеть? (Гетьман С.И.)
- Однаково ли реагируют новые и старые физические составляющие компьютера на одни и те же алгоритмы энергосбережения? (Гетьман С.И.)
- Стоит ли доверять различным программам-уборщикам мусора и т.п.? (Гетьман С.И.)
- Как в случае короткого замыкания ведут себя батарея и аккумулятор? Как строится сберегательный софт с учётом такой ситуации? (Гетьман С.И.)
- Какие программы и алгоритмы энергосбережения есть для суперкомпьютеров? (Гетьман С.И.)
- Если вдруг на Земле не останется электричества, то как можно будет подзарядить устройства? (Гетьман С.И.)

- Много ли внимания уделяется энергосбережению персональных компьютеров? (Григорьев А.В.)
- Допустим есть ноутбук, который используется в качестве домашнего компьютера. Как лучше поступать: держать его постоянно в сети или ждать разрядки и потом заряжать? (Григорьев А.В.)
- Что должен учитывать разработчик мобильных приложений при написании кода и выборе технологий для лучшего энергосбережения? (Григорьев А.В.)
- Насколько распространены сейчас беспроводные способы зарядки тех или иных устройств? Какие вообще способы существуют? (Григорьев А.В.)
- Допустим есть разряженное мобильное устройство и нет возможности зарядить его стандартным способом. Можно ли зарядить его нестандартным способом, используя подручные средства? (Григорьев А.В.)
- (Грушевский А.А.)
- Какое из 4 состояний системы оптимально для работы? (Лебедев Н.А.)
- По каким критериям было сделано такое разделение системы на 4 состояния? (Лебедев Н.А.)
- Как можно увеличить время работы ПК? (Лебедев Н.А.)
- Понятно, что ограничение работы приложений снижает затратность, а как же батарея? Какие батареи самые энергозатратные? Энергосберегающие? (Лебедев Н.А.)
- На данный момент существуют ли инновационные идеи или решения проблемы энергосбережения? (Лебедев Н.А.)
- Какой режим энергосбережения наиболее эффективен: ждущий режим или глубокий сон? И в чем их различия? (Михальцова А.Ю.)
- Есть ли какая-нибудь зависимость между толщиной мобильного телефона и уровнем энергосбережения? (Михальцова А.Ю.)
- Для мобильных телефонов на платформе Android в магазине (Play Store) много приложений для энергосбережения. Эффективны ли они и почему? (Михальцова А.Ю.)
- Какова сущность алгоритмов энергосбережения? (Михальцова А.Ю.)
- Почему не стоит использовать калибровку батареи? (Михальцова А.Ю.)
- Что представляет из себя алгоритм interactive? (Ровдо Д.И.)
- Что такое калибровка батареи? (Ровдо Д.И.)
- Как часто нужно менять батарею и почему? (Ровдо Д.И.)
- Сейчас распространены powerbank. Значит ли это, что производителям мобильных устройств можно придумывать меньше хитрых способов уменьшить потребление батареи? (Ровдо Д.И.)
- Что такое режим гибернации и как он работает? Каковы его отличия от спящего режима? (Трубач Г.Г.)
- Какие существуют режимы энергосбережения в OS Android? (Трубач Г.Г.)
- Какие приложения потребляют больше всего электроэнергии в Android/iOS? (Трубач Г.Г.)
- Как работают режимы Stamina и Ultra Stamina в телефонах Sony? (Трубач Г.Г.)

- Каким образом реализован процесс энергосбережения (и реализован ли) на компьютерах, в которых критически важна производительность? (бортовые компьютеры самолета и т.д.) (Щавровский С.А.)
- Какой элемент операционной системы затрачивает наибольшее количество энергии в среднем? (Щавровский С.А.)
- Сколько экономится электроэнергии в мире при помощи алгоритмов энергосбережения? (Щавровский С.А.)
- Какие из мобильных ОС наиболее энергобережливая (топ5)? (Ярошевич Я.О.)
- Энергосбережение происходит не только на уровне ОС, но и на техническом: какой бренд ноутбука / телефона / планшета наиболее бережлив? (Ярошевич Я.О.)
- Известно, что при включенном Bluetooth устройство разряжается в разы быстрее. Существуют ли алгоритмы, чтобы это максимально обходить? Особенно это важно для умных часов и прочих устройств, которые постоянно работают на Bluetooth. (Ярошевич Я.О.)
- Как на батарею устройства влияет частота подзарядки? (Ярошевич Я.О.)

#### 4. Языки программирования

##### [Языки программирования. Обзор языков программирования](#)

- Чем конкретно "вреден" оператор goto по Дейкстре? (Белый А.А.)
- Какие основные принципы функциональных языков программирования? (Какие уникальные возможности есть?) (Белый А.А.)
- Какие тенденции развития языков программирования? (Что в дальнейшем будет популярнее: языки типа Java или языки типа Python?) (Белый А.А.)
- Для каких программ использовали первые языки программирования? (Борисевич П.И.)
- Какие есть известные языки программирования без обязательной типизации данных? (Борисевич П.И.)
- Какие устройства могли запускать программы на первых языках программирования? (Борисевич П.И.)
- Почему интерпретатор выполняет код программ медленнее компилятора? (Борисевич П.И.)
- Есть ли среди языков программирования "мёртвые"? Какие атрибуты попадают под это определение? (Гетьман С.И.)
- Возможно было бы определение и создание "структурного программирования" раньше, чем в конце 60-х с выходом статьи Дейкстры? (Гетьман С.И.)
- Какие языки лучше и удобнее для разработки: со строгой или динамической типизацией? А для учёбы? (Гетьман С.И.)
- Как происходили развитие и эволюция компиляторов и интерпретаторов? (Гетьман С.И.)
- Что должен иметь язык программирования, чтобы стать используемым и популярным для разработки? (Гетьман С.И.)
- В чём смысл создания таких экзотических языков как Brainfuck (код посредством рисунка из слешей), Cook (код состоит из одной инструкции, повторяющейся различное число раз в зависимости от вызова необходимой функции)? (Гетьман С.И.)
- Нужны ли языки программирования, в которых даже структур нет (я уже молчу об классах и основах ООП)? (Гетьман С.И.)

- Нужны ли визуальные языки программирования? В чём их преимущества? (Гетьман С.И.)
- Что влияет на рост/падение популярности языка программирования? (Григорьев А.В.)
- Перспективно ли создание собственно языка программирования? И что для этого необходимо? (Григорьев А.В.)
- Где именно применяются низкоуровневые языки программирования? (Григорьев А.В.)
- Что относят к "нормальным алгоритмам"? (Михальцова А.Ю.)
- Если использовать оператор goto вредно, почему тогда на языке Assembler он один из основных при условном переходе? (Михальцова А.Ю.)
- Какие языки программирования относят к функциональным? (Михальцова А.Ю.)
- Что из себя представляют функциональные языки программирования?  
(Михальцова А.Ю.)
- Язык C++ к какому способу реализации относят? (Михальцова А.Ю.)
- Чем отличаются языки программирования низкого и высокого уровня?  
(Михальцова А.Ю.)
- Какие языки программирования относят к языкам программирования высокого уровня?  
(Михальцова А.Ю.)
- Если Java такая "тяжелая", почему она так распространена? Ведь ее используют не только для программирования под Android, но и в веб-приложениях. (Ровдо Д.И.)
- В чем плюсы таких языков как Python и Ruby, раз на них переходят? (Ровдо Д.И.)
- С чем вообще связана популярность тех или иных языков программирования? Что случается такого, что заставляет людей переходить на другие? (Ровдо Д.И.)
- Какова зависимость языков программирования от ОС? (Ровдо Д.И.)
- Присутствует ли оператор goto в языке Java? Почему этот оператор нежелательно использовать? (Трубач Г.Г.)
- Каковы плюсы и минусы ООП? Чем плохи "висячие" методы в коде? (Трубач Г.Г.)
- Приложения под ОС Android пишутся только на Java или еще на каких-нибудь языках?  
(Трубач Г.Г.)
- Что из себя представляет язык программирования Swift? (Трубач Г.Г.)
- На каких языках программирования можно разрабатывать desktop-приложения?  
(Трубач Г.Г.)
- Что означает встраиваемый язык программирования? (Щавровский С.А.)
- В какой сфере наиболее применимы функциональные языки программирования?  
(Щавровский С.А.)
- Чего на ваш взгляд не хватает современным языкам программирования?  
(Щавровский С.А.)
- Есть такая байка: все бородатые программисты в свободное время пишут свой язык программирования (в частности так появился новый язык компании Apple «Swift»). Вопрос: собираетесь ли вы отращивать бороду и писать свой язык программирования? (Щавровский С.А.)
- Какие существуют языки программирования, написанные не на английском языке?  
(Команды на каком-либо другом языке) (Ярошевич Я.О.)
- Какие есть языки с динамической типизацией? (Ярошевич Я.О.)
- Почему программы, написанные на интерпретируемом языке, работают медленнее, чем программы, написанные на компилируемом языке? (Ярошевич Я.О.)

## JavaScript

- В чем заключается неработоспособность функции isNaN? (Белый А.А.)
- Что является самой интересной возможностью сейчас в JavaScript, и в будущих версиях, что обещают разработчики? (Белый А.А.)
- Что может вызвать утечку памяти в JavaScript? (Борисевич П.И.)
- Какие есть преимущества и недостатки TypeScript в сравнении с JavaScript, или кроме типизации никаких отличий между ними нет? (Борисевич П.И.)
- Является ли JavaScript удобным языком в разработке приложений для мобильных устройств? (Борисевич П.И.)
- В каком состоянии СEnvu на 2015 год? И кем он и для чего использовался? (Гетьман С.И.)
- Как осуществляется связь между JavaScript и основным языком разработки приложения (к примеру, C++)? (Гетьман С.И.)
- Лучше ли CoffeeScript чем JavaScript? (Гетьман С.И.)
- Как обходить null в JavaScript? (Гетьман С.И.)
- Как с удобством проводить отладку JavaScript кода без помощи браузера? А как отлаживать библиотеки JavaScript, вроде jQuery? (Гетьман С.И.)
- Используется ли JavaScript в сферах кроме web-разработки? (Григорьев А.В.)
- С чем связан рост популярности JavaScript? (Григорьев А.В.)
- Какие конкуренты существуют у JavaScript? (Григорьев А.В.)
- Существуют ли языки альтернативные JavaScript? (Ипатов А.Е.)
- Какой главный недостаток и главный существенный плюс JavaScript? (Ипатов А.Е.)
- Каковы дальнейшие перспективы развития данного языка? Не превратится ли он в "мертвый" язык? (Ипатов А.Е.)
- Говорят, что JavaScript очень похож на Python, Ruby, но все же он сам по себе. В чем его уникальность? (Ипатов А.Е.)
- Какие недостатки есть у JavaScript? (Лебедев Н.А.)
- Как устроен интерпретатор JavaScript? (Лебедев Н.А.)
- Лучшая IDE для разработки на JavaScript? (Лебедев Н.А.)
- Эффективно было бы сделать JavaScript типизированным? (Михальцова А.Ю.)
- Какой сценарный язык может конкурировать с JS при разработке веб-приложений на стороне клиента? (Михальцова А.Ю.)
- Какие библиотеки JavaScript наиболее распространены? (Михальцова А.Ю.)
- Что представляет собой отладчик в JavaScript? (Михальцова А.Ю.)
- Как происходит тестирование страниц, написанных на JavaScript? (Михальцова А.Ю.)
- В каких случаях имеет смысл писать сервер на JavaScript, а, например, не на Java? (Ровдо Д.И.)
- Какой самый популярный JavaScript фреймворк и почему? (Ровдо Д.И.)
- В чем преимущество нетипизированного языка программирования, как JavaScript? Всего лишь в том, что программист может не думать о типах? То есть, для программистов, которые не хотят слишком много думать? (Ровдо Д.И.)
- Существует ли возможность разработки desktop-приложений на языке JavaScript? (Трубач Г.Г.)
- Как объяснить следующее "3" -+-+ "1" + "1" / "3" \* "6" + "2" = "42"? (Трубач Г.Г.)

- Возможна ли разработка приложения на JavaScript в связке с C++? (Трубач Г.Г.)

## Трансляторы

- В настоящее время имеет ли место разработка новых трансляторов? Если да, то с какой целью? (Белый А.А.)
- Как происходит дизассемблирование кода? (Белый А.А.)
- Можно ли как-то запутать/сломать транслятор? (Белый А.А.)
- Как происходит оптимизация выходного кода транслятора? (Борисевич П.И.)
- Трансляторы используют разные алгоритмы оптимизации? Какие из них наиболее эффективные? (Борисевич П.И.)
- С какой целью могут использовать декомпиляторы? Можно ли получить исходный код какого-нибудь приложения декомпиляцией? (Борисевич П.И.)
- Как разрабатывают компилятор? Можно ли его написать в одиночку? (Гетьман С.И.)
- Как интерпретатор и компилятор обходят ошибки? (Гетьман С.И.)
- Какие самые популярные ошибки, возникающие во время компиляции? (Гетьман С.И.)
- Где мне найти и увидеть то, что из себя представляет компилятор? Как мне увидеть воочию каждую стадию перевода исходного кода в целевой? (Гетьман С.И.)
- Перспективно ли создание своего транслятора? (Григорьев А.В.)
- Существуют ли трансляторы, которые транслируют языки в более высокоуровневые? (Григорьев А.В.)
- Возможна ли декомпиляция кода? (Григорьев А.В.)
- Какие существуют исполнимые модули? В чём их отличия? (Григорьев А.В.)
- В чём преимущество трансляторов языка, написанных на этом же языке? (Грушевский А.А.)
- В каких случаях гораздо эффективнее преобразовывать язык сначала в Assembler, а потом уже в машинный код при компиляции? (Грушевский А.А.)
- Есть ли какие-нибудь трансляторы, которые преобразуют код из одного языка высокого уровня в другой? (Грушевский А.А.)
- Что представляет из себя "дерево разбора"? (Ипатов А.Е.)
- Возможна ли компиляция без перевода в машинный язык? (Ипатов А.Е.)
- Какие существуют псевдокоды? (Ипатов А.Е.)
- Какие существуют аппаратные подходы? (Ипатов А.Е.)
- Где применяется JIT? (Ипатов А.Е.)
- В чём преимущество однопроходного/многопроходного интерпретатора? (Лебедев Н.А.)
- Смысл транслятора генерировать самого себя? (Лебедев Н.А.)
- Хочу написать программу, которая сама себя дописывает в процессе работы. Существуют ли такие трансляторы под это дело? (Лебедев Н.А.)
- Можно ли обмануть транслятор? (Лебедев Н.А.)
- Возможно ли превзойти статическую компиляцию? (Лебедев Н.А.)
- Если ли золотая середина среди трансляторов? (Лебедев Н.А.)
- Можно ли JVM считать за транслятор? (Михальцова А.Ю.)
- Примеры видов трансляторов. (Михальцова А.Ю.)
- Что эффективнее: компилятору переводить программу на Ассемблер или в байт-код? И почему? (Михальцова А.Ю.)

- Существует ли какое-нибудь понятие/характеристика о скорости работы компилятора? (Михальцова А.Ю.)
- Приведите таблицу данных о том, какие виды компиляции/интерпретации используются в популярных языках программирования (Java, C/C++, Python, Ruby, PHP, JavaScript...) (Ровдо Д.И.)
- В чем существенная разница между интерпретатором компилирующего типа и компилятором с интерпретацией? (Ровдо Д.И.)
- В чем преимущество смены компиляции и интерпретации, как в Forth? (Ровдо Д.И.)
- На каких языках обычно пишутся трансляторы современных языков программирования? (Щавровский С.А.)
- В чем выигрывает компилирование кода, по сравнению с интерпретированием, за исключением скорости? (Щавровский С.А.)
- Каковы подробности про взаимопроникновение процессов трансляции и интерпретации? (Ярошевич Я.О.)
- Что такое дизассемблер? (Ярошевич Я.О.)
- Какие языки являются интерпретируемыми? Какие существуют соответствующие интерпретаторы, как они работают? (Ярошевич Я.О.)

## 5. Базы данных

### Реляционные базы данных. SQL

- Какие есть отличительные особенности реляционных баз данных перед другими? (Борисевич П.И.)
- Что такое суррогатный ключ, и чем он лучше или хуже естественного ключа? (Борисевич П.И.)
- Можно ли полностью избежать дублирования информации в реляционных базах данных? (Борисевич П.И.)
- Почему реляционные БД имеют такой успех? (Гетьман С.И.)
- Почему с них не уходят в пользу нереляционных БД (Гетьман С.И.)
- Реляционные базы данных должны ли всегда соблюдать три правила? (Гетьман С.И.)
- Какие существуют нормальные формы (кроме трёх названных)? (Григорьев А.В.)
- Существуют ли ещё языки для работы с реляционными БД помимо SQL? (Григорьев А.В.)
- Следуют ли всем нормальным формам в серьёзных проектах? (Григорьев А.В.)
- Всегда ли в проектах используют БД, или существуют другие способы эффективного хранения данных? (Григорьев А.В.)
- В чем преимущество реляционных БД перед нереляционными и наоборот? (Грушевский А.А.)
- Какой процент полезных данных в разных реляционных БД? (Грушевский А.А.)
- Что значит программа на SQL? Ведь это просто язык запросов. (Грушевский А.А.)
- Востребован ли на данный момент SQL? (Ипатов А.Е.)
- В чем преимущества реляционных баз данных, а в чем недостатки? (Ипатов А.Е.)
- Какие наиболее популярны реляционные или нет? (Ипатов А.Е.)
- Есть ли какая-нибудь альтернатива SQL? (Ипатов А.Е.)

- Известны случаи, когда выполнение 3 нормальных форм в сочетании с ОРМ ухудшает процесс разработки. Как быть? (Лебедев Н.А.)
- Какие существуют пользовательские функции в MySQL? Зачем оно надо и как пользоваться? (Лебедев Н.А.)
- В каких случаях целесообразно пользоваться view? (Лебедев Н.А.)
- Зачем разделение на нормальные формы отношения в реляционной модели данных? (Как это помогает в работе с данными?) (Михальцова А.Ю.)
- Почему считается, что данные лучше хранить в базах данных, а не на сервере? (Михальцова А.Ю.)
- Есть ли наиболее популярный аналог MySQL? (Михальцова А.Ю.)
- (Михальцова А.Ю.)
- Отличия 3 нормальной формы от других? (Ровдо Д.И.)
- Что такое кортеж? (Ровдо Д.И.)
- В чем разница между MySQL, MS SQL Server и тд? Какие еще существуют реляционные БД? (Ровдо Д.И.)
- Какие диалекты SQL существуют и чем они отличаются? (Ровдо Д.И.)
- Есть ли такие программы, как MySQL Workbench, у других реляционных БД? (Ровдо Д.И.)
- Какие ORM существуют для работы с БД? (Трубач Г.Г.)
- Как организуются транзакции в SQL? (Трубач Г.Г.)
- Что такое триггеры и зачем они нужны? (Трубач Г.Г.)
- Каковы различия между БД MySQL и Oracle? (Трубач Г.Г.)
- Зачем нужно такое разнообразие реляционных БД: MySQL, Postgres, Oracle и т.д.? (Щавровский С.А.)
- Что такое триггер? (Щавровский С.А.)
- Посредством какого языка, кроме SQL происходит управление БД? (Щавровский С.А.)
- Какие есть нормальные формы, которые не были указаны в докладе? (Ярошевич Я.О.)
- Расскажите про 12 правил Кодда. (Ярошевич Я.О.)
- Выстраивали ли вы когда-нибудь архитектуру для реляционной базы данных? (Ярошевич Я.О.)

## [Нереляционные базы данных](#)

- Какие есть аналоги Hibernate для MongoDB? (Белый А.А.)
- Когда какие базы (реляционные или нет) выгоднее использовать? (Белый А.А.)
- Какие есть самые странные базы данных? (Белый А.А.)
- Нереляционные базы данных могут полностью заменить реляционные? (Борисевич П.И.)
- Как правильно подобрать тип хранилища нереляционной БД? (Борисевич П.И.)
- Можно ли объективно заявить, что нереляционные БД лучше, чем реляционные, или наоборот? Почему? (Григорьев А.В.)
- В каких случаях лучше использовать нереляционные БД? (Григорьев А.В.)
- Какие БД тебе больше нравятся: реляционные или нереляционные? (Грушевский А.А.)
- Подробнее про UnQL? (Грушевский А.А.)

- Каким образом использование нереляционных БД сокращает время разработки? (Грушевский А.А.)
- Как происходит создание баз данных без создания конкретной схемы? (Ипатов А.Е.)
- Расскажи подробнее о Base. (Ипатов А.Е.)
- Где на сегодняшний день используются? Примеры? (Ипатов А.Е.)
- В чем преимущества и какие недостатки? (Ипатов А.Е.)
- Будут ли у UnQL схожие с NoSQL типы хранилищ данных? (Михальцова А.Ю.)
- А какие еще типы хранилищ можно было бы реализовать? (Михальцова А.Ю.)
- В каких случаях удобнее использовать нереляционные БД, чем реляционные, отражающие логическую сущность в таблицах? (Михальцова А.Ю.)
- В чем различия, плюсы и минусы реляционных и нереляционных БД? (Трубач Г.Г.)
- Подробнее про Amazon DynamoDB. (Трубач Г.Г.)
- Как организуются запросы в noSQL? (Трубач Г.Г.)
- Каковы перспективы развития? (Трубач Г.Г.)
- В реляционных БД для управления используется SQL. А в нереляционных? (Щавровский С.А.)
- Правильно ли я понял: на малых мощностях реляционные БД быстрее, а с ростом мощности растёт преимущество noSQL? (Щавровский С.А.)
- Какие существуют IDE для MongoDB? (Ярошевич Я.О.)
- Расскажите, пожалуйста, про mongoose. (Ярошевич Я.О.)
- В каких случаях лучше использовать нереляционные базы данных? А конкретно MongoDB? (Ярошевич Я.О.)
- Какие существуют подходы хранения картинок в MongoDB? (Ярошевич Я.О.)

#### [Обработка больших данных. Подходы к формированию больших данных](#)

- Как Data mining используется в системах искусственного интеллекта? (Борисевич П.И.)
- Как будет развиваться Data mining в будущем? (Борисевич П.И.)
- Как происходит подготовка данных перед использованием алгоритмов Data mining? (Борисевич П.И.)
- Откуда берутся большие данные? (Гетьман С.И.)
- Какие новинки есть в этой сфере? (Гетьман С.И.)
- Кому это выгодно и как на этом заработать? (Гетьман С.И.)
- Есть ли большие данные в сфере компьютерных игр? (Григорьев А.В.)
- С какими областями математики связаны большие данные? (Григорьев А.В.)
- В чем сложность визуального представления больших данных? Как ее можно решить? (Грушевский А.А.)
- Где готовят ведущих специалистов в области больших данных? (Грушевский А.А.)
- Какие существуют методы работы с большими данными? (Ипатов А.Е.)
- Каковы основные характеристики bigdata? (Ипатов А.Е.)
- Есть ли какие-либо особенности работы с большим объемом данных? (Ипатов А.Е.)
- Как происходит обработка больших данных? И где они хранятся? (Михальцова А.Ю.)

- Как применяются Big Data в области бизнеса? (Михальцова А.Ю.)
- Насколько обычному разработчику нужно ориентироваться в big data? (Трубач Г.Г.)
- Как часто на проектах привлекаются big data специалисты? (Трубач Г.Г.)
- Какие есть советы по хранению данных огромного объема? (Трубач Г.Г.)
- Какая средняя зарплата big data analyst? (Щавровский С.А.)
- Big Data - это довольно большая сфера, но узкая. В связи с этим вопрос: нужно ли знать и понимать big data обычному программисту? (Щавровский С.А.)
- Расскажите про связку Big data + MongoDB подробнее. (Ярошевич Я.О.)
- В каких белорусских компаниях работают с этим понятием? Куда можно пойти знатоку R? (Ярошевич Я.О.)

## 6. Разработка программного обеспечения

### Разница между разработкой и производством программного обеспечения

- Понятия производства ПО и внедрения ПО эквивалентны между собой? (Борисевич П.И.)
- Можно ли считать разработку и производство разными стадиями жизненного цикла ПО? (Борисевич П.И.)
- Компьютерные игры разрабатываются или производятся? Почему? (Григорьев А.В.)
- Станет ли когда-нибудь разработка/производство софта конвеерным? (Григорьев А.В.)
- Как решаются проблемы разработки такие как: недостаточность трассировки, недостаточность контроля? (Грушевский А.А.)
- Что такое бережливая разработка ПО? (Грушевский А.А.)
- Если эти определения настолько похожи, то может их таки стоит объединить? (Ипатов А.Е.)
- Какие главные отличия между процессами? (Ипатов А.Е.)
- Различают ли их как-то на реальных проектах? (Ипатов А.Е.)
- Кто чаще всего занимается реализацией того или иного процесса? (Ипатов А.Е.)
- "Разработка - это каркас и на него что-то вешается". Что понимается под каркасом? Просто, на мой взгляд, разработка - это, в основном, процесс анализа и проектирования, те каркаса как такого быть не может. (Лебедев Н.А.)
- Производству присущ недостаток контроля, но как такое может быть? Разве производство, особенно массовое, не предполагает этого? (Лебедев Н.А.)
- Почему разработка и производства ПО аналогична разработке техники? Да, эти вещи близки, но если говорить о разработке, то тут скорее второе является частью первого. (Лебедев Н.А.)
- Можно ли реализацию, написание кода отнести к разработке? (Михальцова А.Ю.)
- Как лучше организовать часть жизненного цикла: разработка до производства или наоборот? (Михальцова А.Ю.)
- Что подразумевается под сырьем в определении производства? (Михальцова А.Ю.)
- Долго ли просуществует компания, которая занимается только разработкой? А только производством? (Ровдо Д.И.)
- Какие виды производства различают и их особенности? (Трубач Г.Г.)
- Разработка входит в производство или нет? (Трубач Г.Г.)

- Пример неправильного выбора методологии. Как поступать в такой ситуации? (Ярошевич Я.О.)
- Бережливая разработка ПО (Ярошевич Я.О.)
- Можно ли избежать всех проблем разработки ПО? (Ярошевич Я.О.)

### Парадигмы программирования

- Что представляет собой стек-ориентированное программирование? (Белый А.А.)
- Какие парадигмы являются основными для определения языка программирования? (Белый А.А.)
- Какие основные отличия нестрогого программирования от строгого? (Борисевич П.И.)
- Где используется или использовалось табличное программирование? (Борисевич П.И.)
- Какие есть примеры языков, поддерживающих матричное программирование? Сейчас такие языки программирования где-нибудь используются? (Борисевич П.И.)
- Почему нет языка, чётко воплощающего одну и только одну парадигму программирования? (Гетьман С.И.)
- Табличное программирование способно выступить хорошим ORM? (Гетьман С.И.)
- А возможно написать язык без использования парадигмы на уровне значений? (Гетьман С.И.)
- Были ли попытки обобщить все эти парадигмы в единое целое для создания самого универсального языка программирования? Чем они закончились и почему? (Гетьман С.И.)
- Возможно ли существование языка программирования, который поддерживает лишь одну парадигму? (Григорьев А.В.)
- Есть ли парадигма, которая очевидно превосходит другие парадигмы? Или у всех есть как плюсы, так и минусы? (Григорьев А.В.)
- Почему ООП стал так популярен? (Григорьев А.В.)
- Почему в нашем университете нас знакомили только с ООП? (Григорьев А.В.)
- Есть ли язык, который поддерживает все парадигмы, и делает ли это его лучшим языком? (Григорьев А.В.)
- В чем преимущества одних парадигм перед другими, когда это критически? (Грушевский А.А.)
- Что такое матричное программирование? (Грушевский А.А.)
- В каких случаях использование одной или другой парадигмы значительно упрощает код или увеличивает скорость написания программы? (Грушевский А.А.)
- Почему эзотерическая парадигма не воспринимается всерьёз? (Ипатов А.Е.)
- Где применяется табличная парадигма? (Ипатов А.Е.)
- Расскажите про скалярную парадигму. (Ипатов А.Е.)
- Что представляют собой "нестрогие функции"? (Михальцова А.Ю.)
- Как возникают парадигмы программирования? (Михальцова А.Ю.)
- К какой парадигме можно отнести язык R? (Михальцова А.Ю.)
- Чем отличаются табличная и матричная парадигма? (Михальцова А.Ю.)
- Что представляет собой парадигма "обмен сообщениями"? (Ровдо Д.И.)

- В чем заключаются особенности языка «Cat»? (Ровдо Д.И.)
- Что обозначает термин табличное программирование? Табличное программирование — это как программа, в которой все данные находятся в базе данных? (Ровдо Д.И.)
- В чем минусы ООП? (Трубач Г.Г.)
- Что такое АОП? (Трубач Г.Г.)
- Что такое строгие и нестрогие функции? (Трубач Г.Г.)
- Что такое атом в парадигме на уровне функций? (Трубач Г.Г.)
- Где используется процедурная парадигма? (Ярошевич Я.О.)
- Когда и при каких обстоятельствах появился термин "парадигма программирования"? (Ярошевич Я.О.)
- С какими языками связана обобщенная парадигма? (Ярошевич Я.О.)
- Что означает парадигма "обмен сообщениями"? (Ярошевич Я.О.)

### Стили программирования

- С какой целью создана функция eval? (Белый А.А.)
- Как осуществляется замена условного оператора полиморфизмом? (Белый А.А.)
- Для ООП есть подход с использованием uml-диаграмм. А какой подобный подход есть для функциональных языков? (Белый А.А.)
- Какие стили программирования используются в разработке мобильных приложений? (Борисевич П.И.)
- Какие методы рефакторинга на ваш взгляд самые интересные? (Борисевич П.И.)
- Чем стили программирования отличаются от парадигм программирования? (Борисевич П.И.)
- При других стилях программирования кроме ООП наследование запрещено/не используется? (Гетьман С.И.)
- Были ли до Роберта Мартина попытки обозначить и унифицировать понятие "clean code"? (Гетьман С.И.)
- Как программисты пришли к необходимости следовать clean code? (Гетьман С.И.)
- Должен ли clean code в различных языках быть реализован одинаково? (Гетьман С.И.)
- Есть такое пожелание к clean code: если метод private, то имя его должно быть длинным. А какая максимальная длина допустима? (Гетьман С.И.)
- Всегда задаю этот вопрос: какое количество аргументов в функции приемлемо? (Гетьман С.И.)
- Необходимо ли всегда, вне зависимости от сложности проекта, производить рефакторинг? (Гетьман С.И.)
- По Р. Мартину 10 строк в методе - уже перебор. А есть ли алгоритмы, которые не уложить в эти 10 строк? Как их "подгоняют" под требования clean code? (Гетьман С.И.)
- Clean code приемлем для языков низкого уровня, таких как ASM? Или же это исключительно порождение ООП? (Гетьман С.И.)
- Можно ли усовершенствовать clean code или этот набор требований не нуждается в дополнении? (Гетьман С.И.)

- Существуют ли языки, которые рассчитаны на использование конкретного стиля? (Григорьев А.В.)
- Если стили программирования можно сравнить со стилями одежды, то существует ли мода в программировании? То есть, бывает ли, что в какой-то момент времени модно писать в каком-то стиле? (Григорьев А.В.)
- Почему clean code для разных языков различается? С чем это связано? (Григорьев А.В.)
- Каков clean code для языка Assembler? (Григорьев А.В.)
- На что делать упор, а чем пренебрегать: понятностью или скоростью? (Григорьев А.В.)
- Какой из стилей программирования является классическим стилем? (Ипатов А.Е.)
- Что представляет из себя "выделение метода"? (Ипатов А.Е.)
- Является ли знание каких-то основ "клина кода" критическим при устройстве на работу, работе в каком-то реальном проекте? (Ипатов А.Е.)
- Как осуществляется процесс рефакторинга? (Ипатов А.Е.)
- Какие существуют самые известные методы рефакторинга. (Лебедев Н.А.)
- Иногда клин-код приводит к значительному увеличению метода/класса, с другой стороныпростыню нельзя допускать. Как быть? (Лебедев Н.А.)
- Если функциональное программирование вынесено как отдельный стиль, почему тогда нет визуального? (Лебедев Н.А.)
- Если стили программирования аналогичны соответствующим парадигмам, почему их так мало? (Лебедев Н.А.)
- Как стиль программирования зависит от выбранной методологии? (Михальцова А.Ю.)
- Если нет специалистов в функциональном стиле программирования, значит ли это, что знание это не востребовано или область изучения достаточно сложная? (Михальцова А.Ю.)
- Рекомендуют писать код используя clean code. Но в одной статье было написано, что чтобы закрепить за собой рабочее место, не стоит писать так, чтобы код был понятен другим программистам и вас легко можно было бы заменить. Так как быть? (Михальцова А.Ю.)
- Что, на твой взгляд, лучше: "чистый код" или "быстрый код"? (имеется в виду хороший алгоритм, который эффективно работает, но плохо читабелен) (Ровдо Д.И.)
- На каких этапах разработки следует делать рефакторинг? (под разработкой понимается само написание кода) (Ровдо Д.И.)
- Стиль программирования не является стилем написания кода программиста, а чем-то схож с парадигмами? (Ровдо Д.И.)
- Рефакторинг нужен исключительно для более быстрого понимания чужого кода? (Ровдо Д.И.)
- Какой стиль программирования сейчас наиболее популярен? (Трубач Г.Г.)
- Особенности clean code для C++. (Трубач Г.Г.)
- Какие трудности могут возникнуть, если не производить рефакторинг? (Трубач Г.Г.)
- Какие существуют возможности авторефакторинга в различных IDE? (Трубач Г.Г.)
- Какие существуют книги по стилям программирования, по рефакторингу? (Щавровский С.А.)
- Существует практика, в которой рефакторинг не используется, и даже не приветствуется, ввиду его "необоснованной трудоемкости". Какие есть мнения на этот счет? (Щавровский С.А.)

## Паттерны проектирования

- Какие хорошие практические советы по реализации паттерна "адаптер"? (Белый А.А.)
- Расскажите, что из себя представляют антипаттерны, какие используются активно в серьезной разработке? (Белый А.А.)
- В чем различие антипаттернов <название\_еды>-код? (Белый А.А.)
- Существуют ли паттерны, которые невозможно реализовать на каком-нибудь распространённом языке программирования? (Белый А.А.)
- Какие структурные паттерны проектирования сейчас наиболее популярны? (Борисевич П.И.)
- Какие известные анти-паттерны, на ваш взгляд, самые интересные? (Борисевич П.И.)
- Какие трудности могут возникнуть с использованием шаблонов проектирования? (Борисевич П.И.)
- Вносились ли поправки в книгу Design Patterns после 1991 года? (Гетьман С.И.)
- Почему MVC является паттерном? (Гетьман С.И.)
- Всегда ли паттерны программирования предполагают использование парадигмы ООП? (Гетьман С.И.)
- Зачем нужны антипаттерны? (Гетьман С.И.)
- Что такое GRASP? (Гетьман С.И.)
- Почему до 1991 года никто не выпускал серьёзных работ по шаблонированию разработки? (Григорьев А.В.)
- Особенности паттерна singleton? (Григорьев А.В.)
- В следствии чего появляются новые паттерны? (Григорьев А.В.)
- Нужно ли использовать паттерны или нет? (Григорьев А.В.)
- Какие паттерны, на твой взгляд, самые полезные? (Грушевский А.А.)
- Какие антипаттерны наиболее популярны? (Грушевский А.А.)
- Что такое блоб? (Грушевский А.А.)
- Как бороться с адом зависимостей? (Грушевский А.А.)
- Имеет ли шаблон concurrency какое-то отношение к многопоточности? (Михальцова А.Ю.)
- Что представляет из себя такой шаблон, как абстрактная фабрика? (Михальцова А.Ю.)
- Что такое антипаттерны, и для чего они применяются? (Михальцова А.Ю.)
- Какие шаблоны проектирования самые популярные? Какие спрашивают на собеседованиях? (Ровдо Д.И.)
- Что из себя представляет антипаттерн "слепая вера"? (Ровдо Д.И.)
- Каковы плюсы и минусы паттерна неизменяемого объекта (immutable)? (Ровдо Д.И.)
- Говорят, что паттернами увлечены, в основном, программисты среднего уровня, опытные относятся к ним куда прохладней. Почему так? (Ровдо Д.И.)
- Почему примитивы синхронизации указывались как паттерны в ответвлении concurrency? (Трубач Г.Г.)
- Что из себя представляет паттерн строитель? (Трубач Г.Г.)
- Какие существуют паттерны для взаимодействия с БД? (Трубач Г.Г.)
- Как нужно выбирать паттерн? (Трубач Г.Г.)

- Какие книги на тему паттернов проектирования обязательно необходимо прочитать? (Щавровский С.А.)
- Считаете ли вы, что знание и следование паттернам проектирования является необходимым для современного специалиста? (Щавровский С.А.)
- Паттернов много. А какой базис среди них можно выделить? (Щавровский С.А.)
- Особенности структурного паттерна "Декоратор" ("Обертка"). (Ярошевич Я.О.)
- Когда лучше использовать MVC, а когда MVVM? (Ярошевич Я.О.)
- В чем различия антипаттернов "Спагетти-код", "Лазня-код", "Равиоли-код"? (Ярошевич Я.О.)
- Особенности антипаттерна "Золушкина туфелька" (Ярошевич Я.О.)

### [Разработка мобильных приложений](#)

- За какие нарушения приложение могут не выложить в App Store, Google Play? (Борисевич П.И.)
- Какие есть популярные движки для разработки мобильных приложений? (Борисевич П.И.)
- Как можно заработать на бесплатном приложении? (Борисевич П.И.)
- А выгодно ли разрабатывать приложения на мобильные устройства без желания заработать? (Гетьман С.И.)
- Как не застрять на этапе написания User's Story для мобильного приложения? (Гетьман С.И.)
- Когда ждать кризис идей из-за коммерциализации на рынке мобильных приложений? (Гетьман С.И.)
- Зачем нужны комментарии в AppStore, если там высказывается необъективная критика, которой не могут воспользоваться разработчики? (Гетьман С.И.)
- Является ли Xamarin хорошим подспорьем для разработки приложений как wMMD или BuildAnApp? (Гетьман С.И.)
- Стоит ли создавать платное приложение? Или всё же бесплатное с монетизацией? (Григорьев А.В.)
- Какие есть виды монетизации? И какие требования тот или иной вид имеет к самому приложению? (Григорьев А.В.)
- С какими ограничениями сталкиваются разработчики мобильных приложений? (Григорьев А.В.)
- Если в файле .html в теге <link> в атрибуте media укажем "handheld", то где мы сможем протестировать страницу? (Михальцова А.Ю.)
- Как разместить свои приложения в различных Store? (Михальцова А.Ю.)
- Куда обращаться, если найден баг в мобильном приложении? (Михальцова А.Ю.)
- (Ровдо Д.И.)
- Что нужно сделать для того, чтобы можно было выкладывать приложения в Google market/Appstore? (Трубач Г.Г.)
- Зачем нужны manifest файлы в android приложениях. (Трубач Г.Г.)
- Какие есть советы, чтобы ваше приложение попало в топ магазина? (Трубач Г.Г.)
- Каковы принципы размещения рекламы в приложениях? (Трубач Г.Г.)

- В плане оптимизации и производительности, нативная разработка приложений будет лучше, чем кроссплатформенная. А все же почему кроссплатформенная разработка пользуется популярностью? (Щавровский С.А.)
- Как происходит зарабатывание денег в самых известных магазинах мобильных приложений (какой процент идет разработчику, какие условия магазинов?) (Щавровский С.А.)
- Какой магазин аккумулирует больше денег разработчикам? (Щавровский С.А.)
- Сейчас существует множество одинаковых мобильных приложений от различных кафе. "Фабрика лояльности" указана в каждом из них. Не могли бы Вы рассказать подробнее про такие приложения? (Ярошевич Я.О.)
- Зачем тестировщику нужен шкаф с телефонами, если сейчас есть достаточно возможностей эмулировать любое мобильное устройство на компьютере? (Ярошевич Я.О.)
- Можно ли заплатить Маркету, чтобы Ваше приложение рекомендовалось (поднималось в топы). Например, такая функция точно есть у Google: если хотите, чтобы Ваш сайт был в топе по таким-то ключевым словам, то нужно заплатить им какую-то сумму. (Ярошевич Я.О.)

## Тестирование

- Как типы тестирования используют в разработке мобильных приложений? (Борисевич П.И.)
- Что должен хорошо уметь делать тестировщик? (Борисевич П.И.)
- Какие есть особенности тестов, написанных перед разработкой приложения? (Борисевич П.И.)
- Зачем в тест-дизайне документация? Разве это не отягощает проект и работу над ним? (Гетьман С.И.)
- Зачем нужны обычные тестировщики, если они могут всё испортить? Откуда эти люди берутся? (Гетьман С.И.)
- Можно ли осуществить Test Driven Development в C++? (Гетьман С.И.)
- Кто важнее: разработчик или тестировщик? (Гетьман С.И.)
- Насколько чёткое разделение между разработчиками и QA? (Гетьман С.И.)
- Можно ли зарабатывать, будучи исключительно тестировщиком? (Григорьев А.В.)
- Если близко дедлайн и необходимо чем-то пожертвовать, то можно ли пожертвовать тестированием? Или стоит чем-то другим? (Григорьев А.В.)
- Для любого ли проекта подойдёт открытое тестирование? (Григорьев А.В.)
- Почему к тестировщикам относятся менее уважительно? (Грушевский А.А.)
- Что такое регрессивное тестирование? (Грушевский А.А.)
- Какие случаются трудности с usability-тестированием? (Грушевский А.А.)
- Какие из видов тестирования используются в реальных проектах? (Ипатов А.Е.)
- Какие виды тестирования используются чаще/реже всего? (Ипатов А.Е.)
- Как осуществляется выбор применяемого вида тестирования? Кто выбирает? (Ипатов А.Е.)
- Какие отличие между компонентным и модульным тестированием? (Михальцова А.Ю.)
- В чем суть приемочного уровня тестирования? (Михальцова А.Ю.)
- Почему выделяют TDD, если в жизненном цикле обычно определены фазы "разработка", а потом "тестирование"? (Михальцова А.Ю.)
- Связаны ли альфа- и бета-тестирования? И может ли одно осуществляться без другого? (Михальцова А.Ю.)

- Для чего используется каждый из видов тестирования? (Примеры ситуаций, когда используют нагрузочное, стресс-тестирование и т.д.) (Михальцова А.Ю.)
- Тесты, связанные с изменениями, включают в себя функциональные и нефункциональные? То есть, на самом деле не являются типом наравне с вышеперечисленными? (Ровдо Д.И.)
- Больше ли работы у тестировщиков при параллельном тестировании по сравнению с "тестами до кода"? (Ровдо Д.И.)
- Почему статус тестировщика считается ниже, чем у разработчика? (Ровдо Д.И.)
- Unit тесты. Что это и для чего? (Трубач Г.Г.)
- Как работает багтрекинговая система? (Трубач Г.Г.)
- Как можно автоматизировать тестирование? (Трубач Г.Г.)
- Какая наиболее используемая система тестирования? (Щавровский С.А.)
- Какие программные продукты для тестирования существуют? (Щавровский С.А.)
- Понятно, что пренебрегать тестированием нельзя, но каковы причины? (Щавровский С.А.)
- Какие есть особенности тестирования программ, написанных на определённых языках программирования? Есть ли такая связь тестирования с конкретным языком? (Ярошевич Я.О.)
- Какие есть роли тестировщиков? Какое для каждой роли требуется образование и какие нужны знания? (Ярошевич Я.О.)
- Автоматизация тестирования сейчас набирает обороты, можете рассказать подробнее про это? (Ярошевич Я.О.)
- Как Вы относитесь к практике написания тестов до написания самого кода? Используется ли сейчас такой подход? (Ярошевич Я.О.)

### Стандартизация

- Каковы нововведения в Java 9? (Белый А.А.)
- Для чего, собственно, нужна стандартизация (вернее, как дошли до того, что она нужна) (Белый А.А.)
- Что должно быть указано в стандарте для ЯП, например? (Белый А.А.)
- Когда начали появляться первые стандартизованные языки? (Борисевич П.И.)
- Кто первым предложил использовать стандарты для языков программирования? (Борисевич П.И.)
- Без стандартизации обойтись на производстве и во время разработки можно? (Гетьман С.И.)
- Насколько заметны внесения изменений в стандартизацию и как это влияет на нас с вами программистов? (Гетьман С.И.)
- Так что такое стандарт языка? Просто документация? (Григорьев А.В.)
- Кто создаёт стандарты для различных продуктов? Почему им достаётся такая честь? (Григорьев А.В.)
- Что такое The Open Group? (Грушевский А.А.)
- В чём смысл крупного спонсирования организаций, занимающихся стандартизацией? Что это дает спонсорам? (Грушевский А.А.)
- Какие стандарты есть для QR кодов? (Грушевский А.А.)
- Что представляет из себя тот или иной стандарт? (Ипатов А.Е.)

- Стоит ли вообще изучать какие-то стандарты языков? (Ипатов А.Е.)
  - Как происходит внедрение стандарта в среду? (Ипатов А.Е.)
  - В чем отличия стандартов разных языков программирования? (Ипатов А.Е.)
  - Какова роль стандартов в развитии языков программирования и их популяризации? (Лебедев Н.А.)
  - Какое принципиальное отличие частного стандарта от международного? (Лебедев Н.А.)
  - Понятно, что при расширении возможностей языка выходит новый стандарт. А были ли случаи кардинального изменения? (Лебедев Н.А.)
  - Почему так принято, что данные хранятся именно в двоичном коде? (Михальцова А.Ю.)
  - Можно ли синтаксис языка программирования отнести к стандартизации? (Михальцова А.Ю.)
  - Обязан ли программист знать всё о стандартах того языка программирования, на котором он пишет код? (Михальцова А.Ю.)
  - А как пришли к осознанию необходимости в стандартизации? (Ровдо Д.И.)
  - Каковы нововведения в Java 9? Насколько сильно отличается от Java 8? (Ровдо Д.И.)
  - Есть ли предел версий Java, C++? То есть, может ли получится так, что, например, что Java 32 будет такой, что дополнять уже просто нечего? (Ровдо Д.И.)
  - В чем различия 8-го стандарта Java и 9-го? (Трубач Г.Г.)
  - Какие самые значащие различия между C++ 11 и C++ 14? (Трубач Г.Г.)
  - Каковы нововведения Java 9? (Ярошевич Я.О.)
  - Почему официальный документ стандарта C++ стоит денег? (Ярошевич Я.О.)
- Какие стандарты ECMAScript готовятся к релизу? Какие нововведения? (Ярошевич Я.О.)

## 7. Инновационные концепции и технологии

### Искусственный интеллект

- В чем фундаментальная разница между ИИ, который создали в компьютерных играх, и тем, что пытаются сделать в реальной жизни? (Белый А.А.)
- Думают ли ученые, которые создают ИИ, что он окажется в разы умнее человека? Как это может отразиться на нашей дальнейшей жизни (Белый А.А.)
- Какой основной принцип ИИ (наличие чего позволяет отнести нечто к ИИ)? (Белый А.А.)
- Искусственный интеллект сможет поработить мир? (Гетьман С.И.)
- Что можно сказать о фразе Айзека Азимова относительно робототехники (три принципа: робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинён вред, робот должен повиноваться всем приказам, которые даёт человек, кроме тех случаев, когда эти приказы противоречат Первому Закону, робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому и Второму Законам.)? (Гетьман С.И.)
- Что может помочь воплотить эти принципы в жизнь? (Гетьман С.И.)
- Использование ИИ в игровой области. (Григорьев А.В.)
- Может ли ИИ быть опасным? Рассматривают ли современные исследователи его опасность? (Григорьев А.В.)
- С чем именно связано то, что ИИ до сих пор не достиг уровня человека? (Григорьев А.В.)

- Существовали ли какие-либо попытки изучения и создания ИИ в более ранней истории (до 20 века)? (Григорьев А.В.)
- Можно ли считать искусственным интеллектом такие программы, как программу управления светофорами, в зависимости от загруженности дороги, а также программу управления движением нескольких лифтов, работающих от одной кнопки? (Грушевский А.А.)
- Где находится граница между «умной» программой и ИИ? (Грушевский А.А.)
- Можно ли считать искусственным интеллектом сознание человека, записанное на компьютер (например, как в фильме «Превосходство»)? (Грушевский А.А.)
- Где на сегодняшний день применяется искусственный интеллект? В каких сферах ожидается его внедрение? (Ипатов А.Е.)
- Искусственный интеллект стал популярен относительно недавно. С чем это связано? (Ипатов А.Е.)
- Какое средство является наиболее удобным и популярным для разработки системы ИИ? (Ипатов А.Е.)
- Развивается ли данная система в нашей стране? (Ипатов А.Е.)
- Что вообще относят к системам ИИ? (Ипатов А.Е.)
- Существуют ли принципиально другие виды хранения знаний в памяти? (Лебедев Н.А.)
- Почему в докладе делается упор на хранение знаний в БД, ведь реакция и анализ внешних данных тоже важны? (Лебедев Н.А.)
- По сути, пока не решена проблема объединения знаний в памяти, ИИ - обычный вывод информации из БД. Так ли это? (Лебедев Н.А.)
- Возможно ли создать искусственный разум, который был бы полностью аналогичен человеческому? (Михальцова А.Ю.)
- Перспектива развития ИИ. (Михальцова А.Ю.)
- Примеры использования ИИ. (Михальцова А.Ю.)
- Есть ли отрицательное влияние у ИИ? (Михальцова А.Ю.)
- Можно ли Wolfram считать за ИИ? (Михальцова А.Ю.)
- Какая на сегодняшний день существует самая умная система и что она умеет? (Ровдо Д.И.)
- Сможет ли когда-нибудь машина думать как человек? (Ровдо Д.И.)
- Не считаете ли вы, что перечисленные вами языки программирования довольно устаревшие? Действительно ли их сейчас используют для создания интеллектуальных систем? (Ровдо Д.И.)
- Как происходит обучение ИИ? (Трубач Г.Г.)
- Может ли компьютер/робот развить свой ИИ так, что выйдет из-под контроля? (Трубач Г.Г.)
- Какие существуют самые развитые машины/системы с ИИ? (Трубач Г.Г.)
- Удастся ли создать искусственный интеллект в полном смысле этого слова или нет? (Щавровский С.А.)
- А что насчет эмоций, чувств? (Щавровский С.А.)
- Не повредит ли создание искусственного интеллекта человечеству? (Щавровский С.А.)
- Можно ли Siri считать ИИ? (Ярошевич Я.О.) Расскажите, пожалуйста, про игровой искусственный интеллект подробнее. Какие есть подходы? (Ярошевич Я.О.)
- Что такое искусственный геном? (Ярошевич Я.О.)

- Как в контексте науки философия относятся к ИИ? А какое отношение со стороны религии и этики? (Ярошевич Я.О.)

### [Виртуальная реальность](#)

- Реальна ли такая ситуация, что мы сами живём в виртуальной реальности (как в фильме "Матрица")? (Белый А.А.)
- К чему стремятся разработчики виртуальной реальности в будущем? Возможен ли такой печальный исход, что люди будут сидеть у себя дома и контактировать с этим миром исключительно при помощи, так называемых, аватаров? (Белый А.А.)
- Какие производители очков виртуальной реальности сейчас наиболее популярны? (Борисевич П.И.)
- Как виртуальная реальность влияет на человека? (Борисевич П.И.)
- Какие компании лидируют в развитии технологий виртуальной реальности? (Борисевич П.И.)
- Виртуальная реальность - это система, влияющая на органы чувств человека. То есть, имитация звуков и запахов это тоже виртуальная реальность? (Григорьев А.В.)
- Сейчас в игровой индустрии активно продвигается технология виртуальной реальности. А почему дополненная реальность не получила развития? (Григорьев А.В.)
- Возможен ли переход человечества в виртуальную реальность? (Григорьев А.В.)
- Как думаешь, получит ли развитие технология прямого подключения к мозгу/нервной системе? (Например, в играх) (Григорьев А.В.)
- Можем ли мы быть уверены, что не находимся в очень развитой виртуальной реальности? (Григорьев А.В.)
- Сможет ли сейчас виртуальная реальность заменить человеку реальный мир? (Лебедев Н.А.)
- К чему это стремится сфера виртуальной реальности? Какие цели она несет в себе? (Лебедев Н.А.)
- Какие основные математические модели/парадигмы используются в этой сфере? (Лебедев Н.А.)
- Что полезного в виртуальной реальности? Понятно, что реальность, которая имитирует условия реального мира, позволяет подготовить человека к определенным ситуациям. Есть ли еще примеры полезного использования виртуальной реальности? (Лебедев Н.А.)
- С большего виртуальная реальность — это обман. Тем не менее, многие согласны с тем, что ее нужно донести до масс. А правильно ли обманывать людей? (Лебедев Н.А.)
- Можно ли использовать виртуальную реальность как средство для управления людьми, поставив их в условия этого виртуального мира? (Лебедев Н.А.)
- А почему в определении виртуальной реальности присутствует слово мир? (Лебедев Н.А.)
- Примеры использования виртуальной реальности. (Михальцова А.Ю.)
- В комнате, когда шарик подвешивается по углам комнаты, а человек находится внутри него, как "читывается" реакция человека? (Михальцова А.Ю.)
- Каковы перспективы развития виртуальной реальности? (Михальцова А.Ю.)
- Какие существуют применения Omni? (Ровдо Д.И.)

- Как вообще можно сымитировать тактильные ощущения? (Ровдо Д.И.)
- Не случится ли так, как во всяких фильмах, что человек такими темпами перестанет различать виртуальную реальность от настоящего? (Ровдо Д.И.)
- Какова средняя цена таких игрушек? (Ровдо Д.И.)
- Перспективы развития виртуальной реальности. (Трубач Г.Г.)
- Oculus Rift. Каковы принципы его работы, и каково его устройство? (Трубач Г.Г.)
- Возможно ли восстание машин (например, роботов), как, например, Skynet? (Трубач Г.Г.)
- Что можно рассказать о системе управления махами руки в играх (например, в телевизорах Samsung, виртуальные тирсы, консоли и т.д.)? (Трубач Г.Г.)
- С какими проблемами столкнулись разработчики виртуальной реальности? (Щавровский С.А.)
- Был ли у вас опыт пользования виртуальной реальности? (Щавровский С.А.)
- Расскажите про технические средства, на данный момент существующие (компании производители, лидеры рынка)? (Щавровский С.А.)
- Какие фирмы являются ведущими в направлении виртуальной реальности? (Топ 5) Какие у них последние разработки? (Ярошевич Я.О.)
- Какие есть самые необычные и весёлые разработки в этом направлении? (Ярошевич Я.О.)
- Какие разработки в этом направлении существуют в космической сфере? (Ярошевич Я.О.)
- Существует ли виртуальная еда? (Ярошевич Я.О.)

### [Искусственные нейронные сети](#)

- Какой существует механизм обработки информации? (Белый А.А.)
- Как в бытовом плане могут использоваться нейронные сети (какие могут быть устройства, например)? (Белый А.А.)
- Какие есть известные модели НС? Чем они отличаются? Какие модели лучше подходят для прогнозирования результата? (Борисевич П.И.)
- Какие алгоритмы используются для обучения НС? (Борисевич П.И.)
- Как выбрать нужное число нейронов для сети? Можно ли считать, что большее число нейронов в сети будет лучше справляться с поставленной задачей? (Борисевич П.И.)
- Какие ещё существуют интересные и экзотические пороговые функции? (Гетьман С.И.)
- Вторая из предложенных пороговых функций ( $1/(1+\exp\{...\})$ ) позволяет "предотвратить переполнение"... поясни пожалуйста, переполнение чего? (Гетьман С.И.)
- Сумматор может дать сбой? А пороговая функция? В чём выражаются эти сбои? (Гетьман С.И.)
- Что есть понятие "слой" в контексте искусственных нейронных сетей? (Гетьман С.И.)
- Роботов обучают или они обучаются сами? Приведи примеры. (Гетьман С.И.)
- Как написать сумматор? (Гетьман С.И.)
- Возможно ли объединить архитектуру фон Неймана и архитектуру НИС? (Гетьман С.И.)
- Какие производственные задачи кроме тех, что ты перечислил, может и сможет в некотором будущем решать ИНС? (Гетьман С.И.)
- Что будет если вживить в человека ИНС каким-то образом, к примеру, через наномашины? (Гетьман С.И.)

- Может ли ИНС заместить биологическую нейронную сеть? (Гетьман С.И.)
- Можно поподробнее о процессе обучения с учителем, в частности о стимул-реакционной системе. (Гетьман С.И.)
- Могут ли меняться в ней пути передачи импульсов, создаваться новые нейроны, меняться веса сумматора в нейронах? (Грушевский А.А.)
- Есть ли аналоги синапсов для ИНС? Если есть, то какие их функции? (Грушевский А.А.)
- Как ИНС может обучаться? (Грушевский А.А.)
- Как адекватно определить, что ИНС обучилась правильно? (Грушевский А.А.)
- Какие существуют аналоги AIBO? В каких сферах применяются? (Ипатов А.Е.)
- Каким образом происходит разработка и тестирование? (Ипатов А.Е.)
- Как происходит процесс обучения? (Ипатов А.Е.)
- Какие существуют способы обучения (возможно какие-то алгоритмы)? Какие сроки обучения? (Ипатов А.Е.)
- Как происходит распознавание текста/изображения? (Михальцова А.Ю.)
- Какое существует применение ИНС (подробнее)? (Михальцова А.Ю.)
- В чем заключается программное воплощение математической модели? (Михальцова А.Ю.)
- Как нейронные сети используются в робототехнике? (Михальцова А.Ю.)
- Сможет ли когда-нибудь искусственная нейронная сеть сравняться с биологической? (Ровдо Д.И.)
- Процесс обучения нейронной сети ограничен только временем и памятью? Или есть еще что-то? (Ровдо Д.И.)
- В каких случаях и почему лучше использовать полносвязную, многослойную и слабосвязную нейронную сеть? (Ровдо Д.И.)
- Как используются НС в поисковых системах? (Трубач Г.Г.)
- Как нейронные сети распознают картинки/звуки? (Трубач Г.Г.)
- На каком языке программирования в основном разрабатываются НС? (Трубач Г.Г.)
- Каковы перспективы развития НС? (Трубач Г.Г.)
- Какая наиболее используемая технология (язык программирования, например, или может какой-нибудь фреймворк) при создании нейронных сетей? (Щавровский С.А.)
- Пример компаний (команд) занимающихся нейронными сетями. Какие известные продукты есть у этих компаний (команд)? (Щавровский С.А.)
- Как происходит рисование картин нейронными сетями. (Ярошевич Я.О.)
- Какие есть известные типы сетей? Особенности сети Джордана. (Ярошевич Я.О.)
- Какие есть варианты будущего для нейронных сетей? (Ярошевич Я.О.)
- Какие есть продвижения в исследовании вопроса о возможности развития психологической интуиции у нейросетевых экспертных сетей? (Ярошевич Я.О.)

### [Работы и роботизация человека](#)

- Какие наиболее интересные работы используются в медицине? (Борисевич П.И.)
- Какие операционные системы используются роботами, какие у них есть особенности? (Борисевич П.И.)

- Каким образом роботизированный протез реагирует на нервные окончания человека? Для этого используется программное обеспечение? Оно как-то настраивается? (Борисевич П.И.)
- Вотты привёл определение робота, как замены человека, человекоподобной механической и проч. конструкции, манипулятора. Но ведь есть и роботы-пылесосы, роботы-животные и т.д. Так где грань? Плюс, опять же отталкиваясь от тобой данных определений, чем манипулируют роботы-пылесосы и роботы-животные? (Гетьман С.И.)
- Робот может заниматься искусством? (Гетьман С.И.)
- Деревянная птица не являлась копией человека и его функцией. Тогда как мы можем причислять эту конструкцию к роботам? Самолёт ведь тоже не робот. (Гетьман С.И.)
- Может ли робот выполнять НЕ РУТИННЫЕ действия? Как его для этого программировать (какие средства, алгоритмы)? Приведите примеры. (Гетьман С.И.)
- Какой вклад в робототехнику сделал Леонардо да Винчи кроме чертежа механического всадника? (Гетьман С.И.)
- Как вы относитесь ко взглядам А. Азимова на проблемы робототехники, выраженные с помощью 3 законов? А считаете ли вы эти законы достаточным условием безопасного существования человек и разумных машин = роботов? (Гетьман С.И.)
- Цитата из твоего доклада: "в 60-е годы роботы имели память как ЭВМ". Поясни это, потому что я представляю себе голову робота в виде целого коридора машин на лампах. (Гетьман С.И.)
- Как роботы связаны с миниатюризацией? (Гетьман С.И.)
- Люди с программируемыми протезами - киборги? (Гетьман С.И.)
- Мозг в железке - киборг. Железка в теле - робот. Правильно? (Гетьман С.И.)
- Могут ли роботы чувствовать? А андроиды видеть сны об электроовцах? (Гетьман С.И.)
- На каком языке проще всего запрограммировать робота? А какой самый популярный язык программирования для этого дела? (Гетьман С.И.)
- Ждут ли нас в будущем такие роботы, как Бендер из Футурамы? (Гетьман С.И.)
- Если всё-таки робот может чувствовать, что как он будет себя чувствовать, когда узнает, что он всего лишь плод серийного производства? (Гетьман С.И.)
- Возможна ли такая ситуация: один робот начинает выполнять действия другого робота без перепрограммирования (робо-мимикрия = подражание)? (Гетьман С.И.)
- ИГИЛ пользуется роботами? (Гетьман С.И.)
- Что ждёт человечество, когда оно перестанет быть рабочей силой номер 1? (Гетьман С.И.)
- Как человечество будет справляться с проблемой свободного места, если количество роботов будет расти экспоненциально? Надо заметить, что будут переполнены и свалки, и склады, и дома. Как решать проблему? (Гетьман С.И.)
- Роботы способны на самоубийство? (Гетьман С.И.)
- Наномашины - хорошо или плохо? (Гетьман С.И.)
- Можно ли занести вирус роботу? (Гетьман С.И.)
- Роботы могут родить детей? Каким способом? (Гетьман С.И.)
- Есть ли говорящие зубные протезы? Сколько памяти они вмещают? (Гетьман С.И.)
- Всегда ли у робота должно быть тело? А душа? (Гетьман С.И.)

- Могут ли существовать роботы-потребители? Где и зачем они могут понадобиться? (Гетьман С.И.)
- Расскажи подробнее про нанороботов. В каком состоянии сейчас эта технология? Когда это будет возможно? (Грушевский А.А.)
- Расскажи про квантовых роботов. (Грушевский А.А.)
- Что такое Зал славы роботов? (Грушевский А.А.)
- Какой самый большой в мире робот? (Ипатов А.Е.)
- Какие из роботов являются уникальными в своей сфере? (Ипатов А.Е.)
- Какой из роботов наиболее популярен на сегодняшний день? (Ипатов А.Е.)
- Какой из роботов является наиболее маленьким на сегодняшний день и каковы его функции? (Ипатов А.Е.)
- Изначально роботов создавали для упрощения жизни человека. Можно ли утверждать, что сейчас их создают, чтобы полноценно заменить человека? (Михальцова А.Ю.)
- Как можно использовать роботов в сфере безопасности? (Михальцова А.Ю.)
- Может ли робот быть умнее человека? (Михальцова А.Ю.)
- Если робот может жить без органов, можно ли это как-то использовать для того, чтобы могли жить люди, у которых, например, отказалось сердце, проблемы с печенью и т.д.? (Михальцова А.Ю.)
- То есть ты называешь роботом обычные механизмы? Утка, часы, забиратель из печи :) (Ровдо Д.И.)
- А в чем разница между роботом и киборгом? (Ровдо Д.И.)
- Обязан ли робот обладать искусственным интеллектом? (Ровдо Д.И.)
- Почему ещё нету робота, который сильно похож на человека? (Трубач Г.Г.)
- На какие технологии повлияли роботы? (Трубач Г.Г.)
- Может ли произойти так, что роботы полностью заменят человека, а потом получится планета, населенная роботами? (Трубач Г.Г.)
- Как пишется софт для робота и на каком языке программирования? (Трубач Г.Г.)
- Существует ли компании, занимающиеся производством роботов для частного использования? (Щавровский С.А.)
- Приведите пример случаев интеграции роботов в тело человека, если такие интеграции имеют место быть. (Щавровский С.А.)
- Расскажите про применение роботов в космосе (мкс или марсе, например). (Щавровский С.А.)
- Может есть аналог R2D2 для починки труднодоступных поломок? (Щавровский С.А.)
- Расскажите про самые лучшие картины (фильмы или мультфильмы), в сюжете которых задействованы роботы. Далека ли наша реальность от сюжетов этих фильмов или мультфильмов? (Ярошевич Я.О.)
- Есть ли такие профессии, которые роботы могут полностью выполнять самостоятельно? То есть можно ли заменить некоторый вид человеческой деятельности машинной? (Ярошевич Я.О.)

- Какие полезные для медицинской сферы роботы были созданы в наши дни? (Ярошевич Я.О.)
- Есть ли роботы, которые умеют выражать человеческие эмоции? Например, плакать? (Ярошевич Я.О.)
- Почему количество людских жертв "от руки робота" растёт? Только ли из-за увеличения количества роботов? (Ярошевич Я.О.)
- Как появилась идея игры "Машинариум"? (Ярошевич Я.О.)
- Смотрели ли вы мультфильм "Мирс Пирс"? Если нет, то посмотрите и дайте анализ поведению роботов в этой картине. Может ли такое случиться в будущем? (Ярошевич Я.О.)

### [Интернет вещей](#)

- Как интернет вещей может помочь или уже помогает в бытовом плане? (Белый А.А.)
- Какой необходимый критерий для вступления в интернет вещей? (Белый А.А.)
- Можно ли отнести концепт автомобиля без водителя к интернету вещей? (Белый А.А.)
- Каким образом будет происходить управление такими вещами? Или они не будут требовать нашего участия? (Белый А.А.)
- Интернетом вещей можно назвать возможность самостоятельного взаимодействия между устройствами через интернет, без участия человека? (Борисевич П.И.)
- Какие есть интересные примеры сетей устройств, которые можно отнести к интернету вещей?
- Какой прогноз на развитие интернета вещей в ближайшем будущем? (Борисевич П.И.)
- Какими функциями обладал тот тостер из рассказа, который дал начало интернету вещей? (Гетьман С.И.)
- Почему в период с 90-х до конца 2000-х интернет вещей были в затишье? (Гетьман С.И.)
- Есть ли в интернет вещей потенциал, на котором можно заработать деньги? (Гетьман С.И.)
- интернет вещей - это исключительно бытовая техника, или это может быть использоваться где-то ещё? (Гетьман С.И.)
- Чего стоит опасаться больше всего при внедрении интернет вещей? Говорилось, что система должна САМА принимать решения, а это, несложно догадаться, может повлечь необратимые последствия. (Гетьман С.И.)
- Может ли так произойти, что интернет вещей не найдёт своё применение в реальном мире из-за того, что невозможно всё предугадать чтобы выстроить любой ход событий? (Гетьман С.И.)
- Какими транспортными и сетевыми протоколами пользуются интернет вещей? (Гетьман С.И.)
- Верно ли, что в интернете вещей обязательно будет применён искусственный интеллект? (Григорьев А.В.)
- Будет ли интернет вещей как-то пересекаться с человеческим? Или это совершенно отдельная сеть? (Григорьев А.В.)
- Что такое цикл зрелости новых технологий? (Грушевский А.А.)
- Какие неожиданные решения в области умного дома существуют? (Грушевский А.А.)

- Как можно стандартизировать интернет вещей? (Грушевский А.А.)
- Какие из интернет вещей сейчас наиболее популярны? (топ 5, которые применяются в той или иной сфере) (Ипатов А.Е.)
- Какие фишки уже есть в модели умного дома? (Ипатов А.Е.)
- Как ты считаешь каким образом это все может повлиять на людей в частности? (Ипатов А.Е.)
- Как можно коротко описать сформулированные тобой концепции и их цели? (Лебедев Н.А.)
- Можно ли IoT считать новой самостоятельной веткой в индустрии? Почему? (Лебедев Н.А.)
- Для Brillo открыт исходный код. А где можно посмотреть на него и что мы там увидим? (Лебедев Н.А.)
- Можно ли smart-house запрограммировать под себя? (Михальцова А.Ю.)
- Как происходит взаимодействие между вещами? (Михальцова А.Ю.)
- Можно ли интернет вещей рассматривать как часть искусственного интеллекта? (Михальцова А.Ю.)
- Почему Apple так слабо двигается в этом направлении? Неужели они согласны отдать рынок Google? (Ровдо Д.И.)
- Не опасно ли открывать исходный код? Google о нашей безопасности позаботится, а люди, которые захотят поиграться и насоздают своих умных вещей? (Ровдо Д.И.)
- Как приходят в голову вообще идеи типа умных вещей? Просто расширяют сферу применения уже созданного? Или, может, прислушиваются к предположениям учёных столетней давности? (Ровдо Д.И.)
- Что тостер мог делать в интернете? (Трубач Г.Г.)
- В чем отличие связи машина к машине от связи человек к человеку? (Трубач Геннадий)
- Получается, что интернет-вещей связано с искусственным интеллектом и нейронными сетями? (Трубач Геннадий)
- Как интернет вещей может помочь с уборкой квартиры? Можно ли оснастить свой дом так, чтобы производилась самоуборка без вмешательства человека? Сколько это будет стоить, если можно? (Ярошевич Я.О.)
- На что способна самая умная кофемашина? (Ярошевич Я.О.)
- Машина без водителя близка к реальности? (Ярошевич Я.О.)

### [3D печать](#)

- Можно ли на 3д-принтере напечатать 3д-принтер? (Белый А.А.)
- Можно ли делать детям игрушки самим дома? (Белый А.А.)
- В будущем реально ли иметь 3д-принтер или его аналог на крупном производстве, например, изготовление домов, статуй и других крупных конструкций? (Белый А.А.)
- Когда появились первые 3D принтеры? (Борисевич П.И.)
- Какие основные отличия лазерной технологии создания слоёв от струйной? (Борисевич П.И.)
- Какие распространённые операционные системы используются для управления 3D принтером? (Борисевич П.И.)
- Какой язык программирования используется при 3Д-печати? (Гетьман С.И.)

- Какие науки необходимо знать и учить для становления как специалист 3Д-печати? (Гетьман С.И.)
- Какие материалы поддерживает данный вид печати? (Гетьман С.И.)
- Можно ли печатать еду? По каким схемам? (Гетьман С.И.)
- Возможна ли сейчас печать готовых объектов, состоящих из деталей? (Григорьев А.В.)
- Печатают ли сейчас какие-либо технологические объекты? (Микросхемы, чипы, носители данных и т.д.) (Григорьев А.В.)
- Насколько развита печать органов? (Григорьев А.В.)
- Особенности солнечного 3д принтера? (Грушевский А.А.)
- Есть ли какие-нибудь ограничения на модели для 3д принтеров, кроме размера? (Грушевский А.А.)
- Когда стоимость 3д принтеров станет доступной "всем"? (Грушевский А.А.)
- Каким образом происходит процесс 3D-печати? (Ипатов А.Е.)
- Какие материалы используются в печати? (Ипатов А.Е.)
- Какой самый масштабный принтер и что было напечатано? (Ипатов А.Е.)
- А чем еще необычным можно заправлять 3д принтер? (Ровдо Д.И.)
- Заменят ли когда-нибудь 3д принтеры заводы? (Ровдо Д.И.)
- Насколько долговечны сейчас детали, напечатанные на 3д принтере? (Ровдо Д.И.)
- Как 3D повлияли на Software Engineering? (Трубач Г.Г.)
- На 3D принтере можно делать съедобные вещи. То есть можно делать все и из всего на 3D принтере? (Трубач Г.Г.)
- Насколько развито изготовление протезов на 3D принтере? (Трубач Г.Г.)
- Какие интересные (удивительные) вещи уже сейчас печатаются на 3D принтере? (Щавровский С.А.)
- Печатали ли органы на 3д принтере и какие? (Щавровский С.А.)
- Как можно распознать убийство, совершенное с помощью оружия из 3д принтера? (Ярошевич Я.О.)
- Сколько людей в мире обладают 3д принтерами? (Ярошевич Я.О.)
- В каких сферах науки нужно иметь познания, чтобы составить модель для 3д принтера? (Ярошевич Я.О.)

## 8. Компьютерная и информационная безопасность

### Компьютерная и информационная безопасность

- Как работает вирус троян (каким образом он получает информацию о введённых ключах авторизации)? (Белый А.А.)
- Назовите самое примитивное устройство, с которого можно взломать компьютерную сеть. (Белый А.А.)
- Какое самое масштабное преступление, связанное с хакерством? (Белый А.А.)
- Как злоумышленник может получить исходный код мобильного приложения? (Борисевич П.И.)

- На что в первую очередь следует обратить внимание разработчику, чтобы защитить своё приложение? (Борисевич П.И.)
- Почему операционные системы, основанные на ядре Linux, считают наиболее защищёнными от вирусов? (Борисевич П.И.)
- Что можно рассказать о безопасности и анонимности систем луковичных серверов? (Тог, к примеру) (Гетьман С.И.)
- Какие существуют стандарты шифрования? (Гетьман С.И.)
- Что такое DDoS-атаки? (Гетьман С.И.)
- Какие есть различия между безопасностью персонального компьютера и безопасностью сервера / mainframe машины? (Гетьман С.И.)
- Как аппаратно обеспечивается экран? (Гетьман С.И.)
- Если о шифре / безопасности системы знают уже двое, то её можно взломать. Как системами безопасности обходится такой барьер? (Гетьман С.И.)
- На любом ли языке программирования можно написать вирус? Почему? (Григорьев А.В.)
- Какие наказания грозят за хакерскую деятельность? (Григорьев А.В.)
- Можно ли законно зарабатывать, будучи хакером? (Григорьев А.В.)
- Как правильно удалять аккаунты, какую-либо информацию? (Ипатов А.Е.)
- Как работает брандмауэр? (Ипатов А.Е.)
- Как используется цифровая подпись? (Ипатов А.Е.)
- По каким критериям выделяют те или иные типы вирусов? (Ипатов А.Е.)
- Топ 10 хакеров и их атак. (Ипатов А.Е.)
- Как хоть как-нибудь защитить себя от тех или иных атак (хотя бы несколько способов)? (Ипатов А.Е.)
- А почему бы не ввести на всех сайтах виртуальную клавиатуру? И тогда кейлогер не страшен. (Ровдо Д.И.)
- Какой вирус самый забавный? (Ровдо Д.И.)
- А есть ли на Linux антивирусы? (Ровдо Д.И.)
- Какими способами можно положить сайт? (Трубач Г.Г.)
- Какие существуют способы защиты от DDoS атак? (Трубач Г.Г.)
- Как защищаются операции с денежными средствами? (Трубач Г.Г.)
- Как можно получить доступ к удаленному компьютеру? (Трубач Г.Г.)
- Как хакеры подбирают пароли от различных аккаунтов? (Трубач Г.Г.)
- Популярен ли хакинг сейчас? (Щавровский С.А.)
- Каково отношение к хакерам? (Щавровский С.А.)
- Что такое DDoS-атака? (Щавровский С.А.)
- Какие самые необычные компьютерные вирусы имели место? (Топ 5). В чем заключалась уловка, как впоследствии происходило заражение? (Ярошевич Я.О.)
- Какой уровень наказания хакеров в нашей республике? А в какой стране самое жестокое наказание для хакеров? (Ярошевич Я.О.)
- Какие были самые известные истории, связанные с хакерством? (Ярошевич Я.О.)

## Антивирусы

- Антивирусы на мобильных устройствах, какие у них есть особенности? (Борисевич П.И.)
- Почему антивирус Касперского сильно тормозит всю систему, что он делает такого сложного? (Борисевич П.И.)
- Какие файлы обновляются в антивирусе ESET, что входит в несколько килобайт его обновления и достаточно ли их ему для надёжной защиты компьютера? (Борисевич П.И.)
- Когда стали появляться первые антивирусы? (Борисевич П.И.)
- Почему Avast такой медленный? (Гетьман С.И.)
- В чём преимущества антивируса Касперского? (Гетьман С.И.)
- Чем Касперский наделал в своё время столько шума, что стал русским антивирусом номер один? (Гетьман С.И.)
- Встречаются ли антивирусы со встроенным фаерволлом? (Гетьман С.И.)
- Стоит ли пользоваться антивирусом на Linux? (Гетьман С.И.)
- Как ведёт себя антивирус в случае заражения вирусом его исходных и рабочих файлов? Баз сигнатур? (Гетьман С.И.)
- Есть ли качественные оффлайн-антивирусы? (Гетьман С.И.)
- У Windows есть собственные средства борьбы с угрозами. Так что, востребованы ли сейчас антивирусы? (Григорьев А.В.)
- Возможны ли ошибки в базах сигнатур? (Например, занесение невредоносного кода кода в базу.) (Григорьев А.В.)
- Есть ли среди бесплатных антивирусов объективный лидер? (Григорьев А.В.)
- (Грушевский А.А.)
- Какие из антивирусов являются наиболее эффективными, а какие наоборот лучше вообще не использовать? (Ипатов А.Е.)
- Существуют ли какие-либо алгоритмы для выявления той или иной атаки? (Ипатов А.Е.)
- Как вообще происходит выявление той или иной атаки? (Ипатов А.Е.)
- В чём суть функции anti-work? Что она делает? (Ипатов А.Е.)
- Как и кем осуществляется поддержка словаря сигнатур? (Лебедев Н.А.)
- Какой из методов защиты наиболее успешен/популярен? (Лебедев Н.А.)
- Какие математические методы участвуют в анализе кода? Можно ли говорить о высокой эффективности этих методов? (Лебедев Н.А.)
- Как распознается новый вирус? Обратная связь? (Лебедев Н.А.)
- Правда, что качество антивируса прямо пропорционально количеству потребляемых им ресурсов? (Лебедев Н.А.)
- Какие есть антивирусы под Linux? (Михальцова А.Ю.)
- Можно ли утверждать, что вирусы пишутся специально, чтобы потом продавать для них антивирусы? (Михальцова А.Ю.)
- Может ли компьютер нормально функционировать без антивируса, если нет подключения к интернету? (Михальцова А.Ю.)
- Эффективны ли антивирусы для мобильных устройств? (Трубач Г.Г.)

- Рейтинг использования антивирусов? (Трубач Г.Г.)
- Как происходит карантин, лечение зараженного файла? (Трубач Г.Г.)
- Существуют ли open source антивирусы? (Трубач Г.Г.)
- Особенности Norton anti-virus? Почему он сейчас один из наиболее популярных? (Трубач Г.Г.)
- Каким образом разрабатывается вредоносное программное обеспечение? (вопрос слишком абстрактный, поэтому конкретизирую) Как разработать своего червя? (Щавровский С.А.)
- Пробовали ли вы создавать вредоносное ПО? (Щавровский С.А.)
- Как выстроена защита от вредоносного ПО на огромных серверах компаний-гигантов (Google, AWS)? (Щавровский С.А.)
- Как антивирусы распознают так называемые "zip-бомбы"? (Ярошевич Я.О.)
- Нужно ли ставить антивирус на телефон? (Для разных ОС) (Ярошевич Я.О.)
- Какие есть самые лучшие антивирусы для ОС Windows? (Ярошевич Я.О.)
- Платили ли вы когда-нибудь за антивирусы? (Ярошевич Я.О.)