
Сергей Толкачев
АКТИВНЫЕ ДАННЫЕ
философское программирование

Сергей Федорович Толкачев

Активные данные. Философское программирование

http://www.litres.ru/pages/biblio_book/?art=67934949
ISBN 9785005680914

Аннотация

Информационные технологии в будущем будут определяться «умными» устройствами и мультимодальными формами общения. В результате появится новая среда, в которой искусственный интеллект соединит вместе «Умный дом» и «Умное здоровье». В этой книге мы покажем, как модели биологических нейронов можно применить для построения активных структур данных, в задачах, связанных с обменом знаниями, пониманием смысла, контекстным поиском, диалогом и, в частности, при создании персональных помощников и чат-ботов.

Содержание

Активные данные Философское программирование

Сергей Федорович Толкачев

ISBN 978-5-0056-8091-4
Создано в интеллектуальной издательской системе Ridero

НЕ ДОГМЫ

«Наша теория не догма,
а руководство к действию,
говорили Маркс и Энгельс…»

Проект BeWiki
https://zoom.it/4VaM

Увертюра

– Все люди немножко программисты.
– Люди разные бывают…
– Программисты тоже.

Сетевой фольклор

Среди мудрых философских вопросов о смысле жизни, таких же древних, как и сама наша способность задавать вопросы, есть один, на который казалось бы легко найти ответ: *«Почему мы задаем вопросы?»* Однако и философ, и психолог, и Google – все ответят на него по-разному, что и не удивительно, поскольку никто на Земле не имеет полного знания о том, как устроен человеческий мозг. Вопрос относится к категории «Познай самого себя», а это по всей видимости невозможно, так как познающая система должна иметь информационные ресурсы, существенно превышающие ресурсы познаваемой системы. И среди обычных людей, как сам вопрос, так и ответы на него, воспринимаются скорее как любопытные, но не имеющее никакой практической пользы.

Пабло Пикассо сказал однажды: *«Компьютеры бесполезны. Они могут только давать ответы.»* Если начало этого утверждения эпатажно и спорно, то его вторая половина, применительно к первому поколению вычислительных ма-

шин – корректна и конструктивна. Но всё течет и развивается. И вот когда ЭВМ трансформировались в распределенные системы обработки знаний, у программистов появилась возможность ответить на вызов Пикассо: научить компьютер задавать *умные* вопросы. Ведь современный компьютер уже умеет задавать простые вопросы. А для поддержания «разумного разговора», нужны принципиально новые подходы не только к технической и программной архитектуре, но и к самой сути информационных процессов, с которыми придётся иметь дело следующему поколению программистов и пользователей.

«А что нужно, чтобы компьютер смог поддержать разумный разговор?» Для ответа на этот вопрос мало понимать, как он работает и что происходит внутри. К этому пониманию нужно будет добавить знания из самых разных областей науки, бизнеса, техники и искусства, и заодно ответить на сопутствующие вопросы: *«Как обучается компьютер? Как передать знания наиболее эффективным способом? В чем принципиальная разница между человеком и компьютером?»* Если мы сумеем соединить разные частички мозаики знаний вместе, эта увлекательная тема развернётся перед нами во всей своей многогранной гармонии. А с практической точки зрения – раскроются секреты, лежащие в основе построения чат-ботов, виртуальных помощников и множества других интеллектуальных инструментов,

для которых диалог является естественным интерфейсом.

Физики познают мир, наблюдая за внешними природными явлениями, а программистам, чтобы лучше понять компьютеры и научить их новым полезным навыкам, можно заглянуть внутрь самих себя. И иногда для этого достаточно поразмышлять над простыми последовательностями слов или букв.

«Мы живем, и мы учимся» – тривиальная фраза для тех, кто понимает русский язык. Но если читатель любопытен, то прежде чем продолжить чтение, он хотя бы на несколько секунд задумается над взаимосвязью между этими простыми словами. *Любопытство* – удивительно интересное свойство человеческого мозга, присущее от рождения всем людям. Иногда оно просыпается от одного простого слова, а иногда глубокие философские рассуждения проходят мимо и никакого интереса к ним человек не испытывает. Но каждый раз, когда любопытство просыпается, мы узнаём что-то новое.

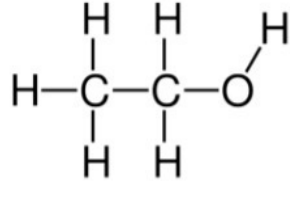
«We live to learn and we learn to live» – один из вариантов предыдущей фразы, записанный на английском языке. Использование другого языка сразу же разделило читателей на два класса – тех кто его понимает, и кто нет. При этом любопытный читатель, не зависимо от того, знает он английский или нет, постарается сделать обратный перевод, сопо-

ставит результат и возразит: *«Хотя слова в этих предложениях и одинаковы, но смысл в них совершенно разный!»*.

Эта фраза похожа на код Морзе, только вместо двух привычных символов, она передана пятью точками разного диаметра. Необычность этой записи заключается в том, что в ней используется алфавит *действий* – это фраза, которую можно отбарабанить пятью пальцами на столе или на «говорящем африканском барабане». Ведь самым первым техническим устройством, которое человек применил для передачи сообщений на расстояние, согласно Дж. Глейку, был именно барабан, и язык передаваемых сообщений был ритмически-тоновый. В ритмических языках временные интервалы между звуковыми символами позволяют конструировать достаточно сложные сообщения при помощи минимального алфавита и простейшего тонового генератора.

«GCA AGA GAT TGT…» – фрагмент ДНК, записан-

ный при помощи кодонов алфавита из четырёх оснований (A,C,G,T). Объяснить его до конца на сегодняшний день не сможет никто. Наши знания заканчиваются пониманием соответствий и микро действий, которые выполнит рибосома, интерпретируя молекулярные буквы. У нас есть некоторые статистические данные, мы знаем алфавит, знаем как происходит интерпретация, но не понимаем самого главного – смысла и логики программы, написанной неизвестным автором.



«ССНННННОН» – хорошо известное описание молекулы этанола, принадлежащей к огромному множеству молекул, которые построены из алфавита, состоящего из 118 атомов. Химические формулы молекул похожи на лингвистические конструкции – слова. Но в отличие от символов в алфавитах естественных или формальных языков, атомы *активны*, и связи (конкатенации) между ними многомерны!

Все разнообразные языки и алфавиты (молекулярные, лингвистические или технические), так или иначе, связаны с передачей, восприятием и преобразованием данных, информации и знаний в биологических, социальных или инженерных системах. При всём разнообразии этих систем, отличия между *биологическими языками* деления клеток, *естественными языками* общения людей и *языками программирования*, совсем не настолько велики, как это может показаться. Особенно для программистов, которые выполняют роль переводчиков, соединяя между собой мир людей и мир компьютеров, в каждом из которых говорят на своих языках. И этот уникальный опыт, накопленный программистами за многие годы, когда они по сути выполняют роль коммуникационного интерфейса между человеком и инженерными системами, может пригодиться самым неожиданным образом.

В этой книге речь пойдет о программировании *активных* данных. *«Чего только не программируют в наше время»* – заметит тут ворчливый программист и будет абсолютно прав. Запрограммировать можно практически ***всё***, и перефразируя известное высказывание Архимеда, настоящий программист может сказать: *«Дайте мне объект, и я его имплементирую.»* Правда, в отличие от Архимеда, которому не нужно было объяснять, что такое «*точка опоры»,* программист дол-

жен быть готов определить грамматику, семантику и прагматику этой фразы, специфицировать протокол «*дайте*», перечислить свойства «*мне*» и «*объект*», уточнить чем является «*и*» – конкатенацией или логической функцией, и т. д. Ведь он живет в мире, где все его слова, все его тексты, проверяет неумолимый компилятор, который только и делает, что снова и снова ищет ошибки. Стоило упомянуть слово «компилятор», как старый программист (старый – это тот, кто читал книги Д. Кнута и К. Джермейн, и знает разницу между MIX и PSW) немедленно возразил: *«Делать компилятору больше нечего – он занимается исключительно преобразованием одних строк в другие, и, если не получается, кричит, что не могу, спасайте. Ну да, делают это не все компиляторы. Например, JavaScript весьма толерантен и игнорирует любые ошибки. (Ю.Ш.)»*. Таков удел всех книг для программистов. С одной стороны, программирование – это формальная дисциплина, а с другой – сколько программистов, столько и формализаций.

Приступая к работе над этой книгой, автор, который всю свою жизнь пишет программы, не может просто так взять и выключить профессиональный способ формулирования мыслей, ведь для него книга – это та же программа, только написанная на другом языке. Но в отличие от программы, книгу мало написать корректно и оптимально, её важно написать к тому же и интересно! Наряду с этим, образованный

программист, имеющий некоторое представление о «принципе минимальной длины описания» или «Колмогоровской сложности», понимает, что гипотетически, существует способ передать то, что он хочет сказать, в кратчайшей форме. Но читательское восприятие – процесс сугубо индивидуальный, контекстно-зависимый и к тому же мультимодальный. Поэты и математики лучше других знают, как при помощи слов выразить человеческие чувства и рациональную логику, и для этого они используют одномерные языки. Музыканты и художники могут передавать гармонию и видение мира, в пространствах с бóльшим количеством измерений. В идеале, сочетание слов, ритмов, логики и рисунков создаёт в голове у человека самые глубокие и прочные ассоциации, основанные на фундаментальных формах сознания. А если помимо форм, прислушаться как в диалогах Пушкина о роли автора, Реньи о математике или в философских спорах Валентинова с Лениным о материальном и идеальном, возникает процесс обмена сомнениями, в котором каждый может поучаствовать со своими идеями и возражениями, то конечно нам нужно добавить в книгу виртуальный диалог с читателем и взгляд со стороны, например, в форме замечаний собеседника-оппонента…

Дойдя до этого параграфа, читатель-непрограммист может представить, что творится в голове у автора-программиста, а читатель-программист вправе ожидать хоть какое-ни-

будь формальное определение. И вот, чтобы помочь как можно быстрее разобраться и понять, насколько интересно и полезно будет продолжить чтение этой книги, попробуем не совсем формально и по возможности кратко, сформулировать её цель:

Активные данные – это виртуальные программируемые элементы, обладающие динамическими свойствами, аналогом которых являются биологические нейроны и нервы. В математике *данные* связаны с абстрактным понятием «переменная», а в программировании, *данные* – это значение или состояние какого-либо регистра или области памяти в конкретный момент времени. И в математике, и в программировании, операции над данными выполняет внешний процесс, который, собственно, и изменяет их значение, но в биологических системах, клетки обладают достаточной автономией и могут изменять свое состояние самостоятельно. В этой книге мы покажем, как модели биологических клеток можно применить для построения активных структур данных, при решении задач искусственного интеллекта, связанных с обменом знаниями, пониманием смысла, контекстно-зависимым поиском, диалогом и, в частности, при создании персональных помощников, виртуальных агентов и функциональных чат-ботов. Поскольку многое из того о чем здесь пойдет речь, еще только предстоит сделать, книгу можно рассматривать как бизнес-идею для

программистов и пользователей, а также для предпринимателей.

Теперь дотошный читатель может проверить близость определения термина *переменная* с соответствующей статьей в Википедии, математик—алгебраист (*А.В.К.)* добавит, что *«Согласно Н. Бурбаки, переменная в формальной математике – это вообще неопределенный термин»,* а старый программист снова возразит: *«В математике нет процессов. Непонятно, о какой конкретно математике идет речь. И хотелось бы иметь определение „знания“ и „понимания“ (Ю.Ш.)»*. И поскольку «*все они тоже правы*», перед автором возникает образ Сциллы и Харибды, между которыми нужно проложить маршрут повествования так, чтобы и увлечь любопытного читателя, и в то же время сделать книгу полезной для практического программиста.

Обмен знаниями – это такой же естественный процесс для человека, как и умение говорить прозой. Безусловно, что далеко не всякий разговор приводит к приобретению знаний, а протоколы и формы, в которых мы передаем и получаем знания, могут быть самыми разнообразными. Но в каждом случае, если речь идет об обмене знаниями, мы наблюдаем три составляющие этого процесса: *источник*, *сообщение* и *получатель*. *Источник* – автор, учитель, композитор и т. п., преобразует свои знания в *сообщение*, которое может

быть записано в виде книги, нотной партитуры, математической теории, и передано в форме диалога, лекции или беседы. А *получатель* – это тот, кто в результате приобретает новую информацию или навыки, позволяющие что-то сделать. Можно продолжить ряд вариантов этой тройки:

композитор → партитура → музыкант
учитель → урок → ученик
автор → книга → читатель

С появлением компьютера оказалось, что он также, наряду с человеком, может участвовать в процессе обмена знаниями, как в качестве источника, так и получателя. Вряд ли кого-нибудь сегодня удивит, что программист передает знания компьютеру в форме программы (*программист → программа → компьютер*) или компьютер, выступающий в роли учителя (*компьютер → урок → ученик*). А с помощью средств виртуализации, сам компьютер можно превратить в сообщение и тогда возможна схема передачи знаний в форме: *автор → компьютер → пользователь*.

В результате долгой эволюции человечество научилось учить и учиться. Обучение детей, строительная архитектура, управление производственными процессами и рецепты, которыми обмениваются поколения кулинаров и фармацевтов – это примеры передачи знаний, но с точки зрения про-

граммиста, их также можно отнести и к неформальному программированию. Но что значит неформальное? Почему научить ребенка, построить дом или приготовить обед менее формально, чем написать операционную систему или программу статистического анализа?

Этот риторический вопрос относится к давнему диалогу между той частью человечества, которая представляет формальные науки, и остальными людьми, которые занимается креативной деятельностью в повседневной жизни, не задумываясь о формализации и доказательстве корректности, иногда тривиальных, а иногда очень сложных, как бытовых,

так и производственных решений и процессов. С этим парадоксом постоянно сталкиваются программисты, которые, с одной стороны, используют инструменты и знания, основанные на формальных теориях, а с другой – применяют их при написании программ, в которых ошибки всегда были, есть и будут. Попытки построить теорию верификации программ, наверняка ещё где-то продолжаются, но иллюзии шестидесятых годов о том, что можно создать универсальный формальный доказатель корректности, у многих теоретиков программирования уже давно рассеялись. Этот спор не может выиграть ни одна из сторон, но можно надеяться, что, когда компьютер «поумнеет» до уровня понимания обычных вещей и способности задавать вопросы, чтобы перейти от простого к сложному, он сам сумеет добавить аргументы, как одной, так и другой стороне этого спора.

В основе классического программирования лежат алгоритмы, которые хорошо подходят для вычислительных и логических методов, и компьютер решает такие задачи значительно эффективней чем человек. Но при этом, алгоритмические методы плохо применимы для задач, связанных с самообучением, ассоциативным восприятием, глубоким пониманием и принятием сложных решений, которые человек решает сравнительно легко и естественно. Мозг человека или приблизительно девяносто миллиардов взаимосвязанных нейронов – это неалгоритмическая система, в которой

нет центрального процессора и нет управляющей программы. Способность человека создавать *новое* – сочинять музыку, поэтические произведения или программировать, пока не в состоянии повторить ни один компьютер, работающий под управлением алгоритма. Возможно, что это связано с тем, что обучаясь, мозг человека сам изменяет не только свое состояние, но также и свою структуру. Добавление же к вычислительной машине программ способных решать новые задачи – это безусловно *новое*, но это новое, созданное извне.

Компьютер и мозг хорошо взаимодополняют друг друга, хотя при этом они концептуально различны. В основе компьютера лежит централизованный процессор, исполняющий внешнюю программу, а мозг – децентрализованная система из множества динамических элементов, у которой нет конечной цели, а поведение является реакцией на внешние возбуждения. Различия между компьютером и человеком наглядны, но совсем не ясно, как может помочь программисту знание основ симфонической музыки, или наоборот, какая польза от теории информации, автору при работе над книгой? На первый взгляд, между этими понятиями нет прямых связей. Впрочем, если допустить, что знание, которое автор или программист создают и публикуют, является объектом с определенными свойствами, которые можно измерить, и этот объект воспринимает и интерпретирует некая

система (компьютер или человек), то такую модель уже можно анализировать.

С точки зрения программиста, способного написать и загрузить программу в компьютер, знание – это динамическое понятие, которое обладает важным свойством *аддитивности*. Если к тому, что находилось в компьютере (аппаратное и программное обеспечение), добавилась новая программа, то можно сказать, что *знание компьютера* увеличилось на некоторую величину соразмерную этой программе. Программист может также предположить, что по аналогии с компьютером, *знание человека* – это нервная система (нейроны и нервы) плюс приобретенное, тем или иным образом, нечто, которое изменило её состояние. Знания человека могут увеличиваться вследствие полученной информации, которая, как и в случае с компьютером, также может быть измерена. Эта аналогия, в дальнейшем будет служить отправной точкой для более глубокого уточнения этого понятия.

Программисты из будущего, будут также отличаться от современных, как прикладной математик из 70-х, отличается от веб-разработчика из 20-х. Пятьдесят лет, не такой уж и большой срок для художественной литературы, науки или музыки. Однако как у самих программ, так и у многих книг по программированию, время жизни очень быстротечно и напрямую зависит от популярности того или иного фор-

мального языка, поколения компьютеров или операционных систем. О программировании написано немало книг, и хотя лексиконы в них могут быть совсем непохожи, неизменным остаётся одно – они адресованы программистам. Такое узкое и конкретное определение целевого читателя, по сравнению с широкой аудиторией, приводит к заключению – чем шире аудитория, тем дольше живет произведение. Написанных же пятьдесят лет назад и до сих пор работающих программ или книг по программированию, которые продолжают читать, можно пересчитать по пальцам, в отличие от музыкальных произведений, живущих столетиями, и которые без проблем поймет и исполнит любой грамотный музыкант, не говоря уже о работах греческих философов.

Все программисты – писатели уже в силу самой своей профессии. Но всё то, что они напишут, будет читать компьютер и только компьютер может «понять», что они хотели сказать своими программами. Но бывает, что у программиста возникает желание написать так, чтобы его текст прочитал

другой человек. Написать без компилятора, пусть и с ошибками, нарушая логику, не очень оптимально и минимально, но зато от души. Постараться рассказать о своей профессии, о проблемах, а иногда и пофантазировать, живым человеческим языком. Вот такую цель и поставил перед собой автор, прекрасно понимая какой риск несёт в себе смешение аудиторий, языков и стилей. Но так же, как в компилируемую программу на языке Swift, можно вставить динамически интерпретируемые фрагменты на языке JavaScript, и это может быть одновременно и красиво и эффективно, так и здесь, мы будем смешивать естественный язык с формальными текстами вовсе не для того, чтобы добавить комментарии к программе и объяснить, как она работает, а скорее наоборот.

Традиционная специализация, как в науке, так и в инженерном деле, разделяет профессионалов на узкие, проблемно-ориентированные группы. Один из первых на это обратил внимание основоположник кибернетики Н. Винер: *«В настоящее же время лишь немногие ученые могут назвать себя или математиками, или физиками, или биологами, не прибавляя к этому дальнейшего ограничения… Именно такие пограничные области науки открывают перед надлежаще подготовленным исследователем богатейшие возможности. Но изучение таких областей представляет и наибольшие трудности… Очевидно также, что если физиолог, не знающий математики, работает вместе с ма-*

тематиком, не знающим физиологии, то физиолог не в состоянии изложить проблему в выражениях, понятных математику; математик, в свою очередь, не сможет дать совет в понятной для физиолога форме.»

В этой книге мы будем использовать знания из самых различных областей: биологии, математики, физики, музыки… Интерпретировать эти знания будет некий виртуальный любопытный программист, иногда поверхностно и разносторонне наивно, иногда профессионально глубоко, но всегда с целью понять суть, методы и практику программирования, и всегда готовый поделиться своими открытиями с другими. Программист, который немножко дилетант во всем, кроме своей профессии, решая задачи из самых разнообразных предметных областей, научился разбираться в каждой из них в достаточной степени, для того чтобы понимать основы, специальную лексику и оценивать результаты, по крайней мере, на уровне здравого смысла.

Есть одна область, в которой программы, не только помогают передавать знания от одной системы к другой, но и служат основой для эволюции огромного биологического мира. Поразительно, как похожи машинные коды, которые исполняются в компьютерах, и биологические программы или молекулы ДНК, которые интерпретируются внутри клетки, копируются, модифицируются и передаются другим клеткам!

И вполне вероятно, что это сходство не случайно. В середине прошлого века, когда компьютеров ещё не было и их нужно было придумать, один из основателей, заложивший архитектуру вычислительных машин на многие годы вперед, которая так и называется «Архитектура фон Неймана», в своей работе по структуре ЭВМ прямо ссылается на биологические модели нейронов, как прототип вычислительных элементов.

Математические и инженерные принципы компьютерного программирования общеизвестны и понятны, а вот биологическое программирование, которое лежит в основе репродукции клеток, соединения нейронов, функционирования головного мозга и других биологических систем, всё ещё остается областью начальных знаний. Но даже эти первичные знания о том, как функционируют клетки, можно применить для построения прикладных решений, реализуемых в вычислительных системах и при этом, такие решения могут оказаться хорошей основой для новых архитектур компьютеров и программного обеспечения.

Чем больше мы узнаём о природе биологических систем, о том, как происходит накопление и обмен знаниями, тем шире раздвигаются границы программирования. И хотя мы пока очень далеки от полного понимания того как работает код ДНК, и скорее всего никогда его не достигнем, можно предположить, что самая первая программа появилась

на Земле приблизительно три миллиарда лет назад. И если немного пофантазировать вместе с древними греками, произойти это могло вот так:

Давным-давно возникла в мире Любовь, и соединились Мать-Земля и Создатель-Уран. Изверглись вулканы, вздыбилась магма, образовались моря и континенты. И оплодотворил Уран Землю – Жизнью. Или, используя современную терминологию, запустил Создатель на репродуктивном интерпретаторе Рибосоме, программу первоначальной загрузки ДНК, способную модифицировать свой код, посредством РНК. С тех пор и развивается на нашей Земле Жизнь, в самых разнообразных формах. Будь то животное или микроб в природе, дерево или трава в лесу, брат или сестра в семье – все они равны перед своими прародителями, но также все они и разные между собой. Так задумал Создатель и так воплотила его желание Мать Земля…

Можно даже попробовать вообразить этого Создателя – Великого Программиста, который сотворил ДНК и Рибосому, загрузил их внутрь первичной самовоспроизводящейся клетки, запустил механизм размножения, и в результате последующих итераций появились и мы, и всё живое, что нас окружает. Хотя вполне вероятно, что всё было решительно иначе, никакого Великого Программиста изначально не было, и вся эта красота получилась из случайного со-

единения молекул. Но что бы ни произошло тогда на Земле, есть некоторые основания утверждать, что жизнь есть следствие глобального биологического программирования. Система биологических программ, которая имеется в каждой клетке и в каждом вирусе, в процессе фантастически огромного количества компиляций и интерпретаций, под воздействием различных мутирующих факторов, породила всё многообразие живых организмов, которые продолжают эволюционировать, с каждой очередной интерпретацией копии программы ДНК на Рибосоме. А когда пришло время, то и Человек, в свою очередь, сотворил и компьютер, и программирование, которые являются естественным продолжением своих биологических прототипов. И чем закончится эволюция компьютеров – кто знает?

Грезят на подобные темы не только программисты. Странные картины мира привычны также и физикам, например у Р. Фейнмана, вселенная – это *«атомы с сознанием, материя с любопытством.»*. А если уж продолжить фантазировать вместе с физиками совсем по-настоящему, то можно представить себе модель *рекуррентного Создателя*. Ведь когда один Создатель породил итеративный процесс, который привел к появлению другого Создателя, и тот, в свою очередь, породил следующий итеративный процесс, который… А что, если был Создатель, который придумал язык, в котором алфавит состоит из атомов (см. таблицу Менделеева),

а слова-молекулы образуются путем многомерной конкатенации букв этого алфавита. Затем, с помощью первоначальной загрузки (Большой Взрыв), он запустил универсальный процесс, и в результате из физического алфавита и слов, образуются молекулярные предложения, из которых и получается ***ВСЁ***, что с точки зрения конструктивного программиста, вполне логичная система рекуррентных отношений. И вернувшись к своей программе, конструктивный программист добавил к ней ещё одну строчку, ещё одну маленькую итерацию к процессу Великой Эволюции, и кто знает, может быть, когда-то нечто подобное делал другой Программист? – *«Но довольно…»*

Последние два слова – пример того, как глубоко различны все мы в восприятии и понимании! Этими словами заканчивается знаменитый монолог Гамлета в переводе Б. Пастернака и у того, кто читал и помнит эту версию, они определённо вызовут какую-нибудь, возможно, что и весьма глубокую, *ассоциацию*. Но для того, кто не читал или не помнит, реакция после прочтения этих двух слов будет совсем другой. Слово ***всё*** в начале этой главы, вместе со словом ***ВСЁ*** в конце, объединяют не логика, а принципиально иные по своей природе, глубокие ассоциативные связи, о которых и пойдет речь в этой книге.

Книга о программировании без примеров программ,

так же неполна, как и книга о математике без математических формул. Можно много фантазировать и интересно рассуждать о знаниях, программах и компьютерах, но практическая цель этой книги будет достигнута только тогда, когда программисты смогут применить результаты этих рассуждений в своих работах, и чтобы помочь в этом, все демонстрационные примеры, приведенные в этой книге, включая библиотеки подпрограмм и объектов, доступны в GitHub:

https://github.com/stolkachev

От слов к делу

В самом начале была Буква,
затем появилось Слово,
и уже потом Дело.

Формальные грамматики

Мы живем в мире машин, которые постоянно что-то делают: перевозят, разогревают, фрезеруют – или, если сформулировать более точно, выполняют определенные действия, которые можно измерить и описать в физических терминах: работа, мощность, КПД, энергия и пр. Здесь разносторонний программист обязательно должен добавить: *«Класс машин, помимо энергетических, должен включать подкласс* ***информационных машин****, которые тоже умеют кое-что делать».* Эх, как было бы удобно всё формализовать в виде классов, методов и отношений, если бы объектно-ориентированный подход приняли в качестве основы для рассуждений все остальные непрограммисты! Мечты…

Из повседневного опыта известно, что слова могут быть декларативными или императивными, и они могут *определять* или *инициировать* действия. Например, набор слов: *«Чтобы разогреть воду, включи плиту»* будет понятен любому современному человеку, и даже компьютеру, хотя

несколько столетий назад, эта фраза вызвала бы недоумение – ведь в то время для того, чтобы разогреть воду, нужно было *разжечь* огонь. Естественно, что один и тот же результат может быть получен в итоге различных последовательностей действий. А слова, как виртуальный мостик, связывают исполнительные механизмы, способные к конкретным действиям, с одной стороны, с процедурами или знаниями, задающими эти последовательности, с другой. Но и суть программирования заключается в формулировании чего бы то ни было – словами! Будь то математическая формула или экономическая модель, программист обязан передать их описание компьютеру при помощи комбинации слов. Желательно только не забывать при этом, о чем на основании своего опыта предупреждают известные мастера слов – математики: *«Слова – орудия опасные»* (Герман Вейль).

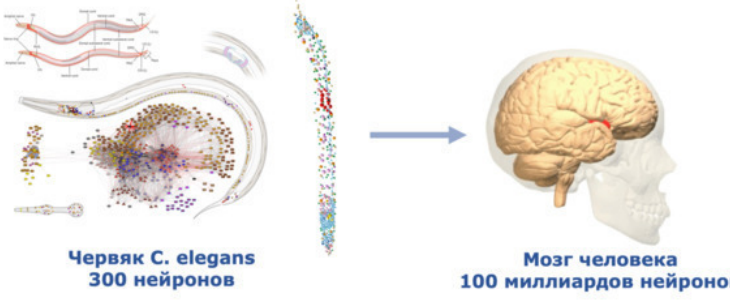
Мы употребляем подобные слова повсеместно и никаких сомнений не возникает, пока мы не зададимся вопросом, а что, собственно, есть *действие*, и чем *знание* отличается от *данных* и *информации*? Рассуждая над смыслом слов, нужно быть готовым к тому, что, как бы глубоко и формально мы не старались уточнить их определения, всё равно, согласно К. Гёделю и В. Гейзенбергу, они будут недостаточно точными, неполными или противоречивыми. Общепринятый подход к получению более «точных определений» основан на использовании более строгих математических фор-

мализаций. Мы же, напротив, будем рассчитывать на неформальное и интуитивное понимание, основанное на «глубоком многослойном обучении». Сравнительно недавно появилась новая теория – «Intelligent Learning» и один из её авторов, Владимир Вапник, высказал предположение, что если фактические знания передаются в сочетании с ассоциативно связанными «посторонними» идеями, то обучение происходит более эффективно, модификация знаний о предметной области не разрушает модель, а адаптация к новому происходит быстрее и легче.

Слово *делать* относится как раз к такой категории интуитивно понятных, и положительный ответ на вопрос *«Может ли машина делать?»*, особенно если речь идет о физических действиях, не должен вызвать каких-либо сомнений (разве что филологических). Но и у программистов не видно серьезных оснований для возражений по поводу способности информационных машин – ведь стоит загрузить программу в компьютер, как в нем начинаются действия. Как в физике, так и в информатике, *действие* всегда приводит к изменению состояния, и это изменение в энергетической машине может быть измерено в джоулях, а в информационной – в битах. Мы не будем здесь углубляться в сравнение свойств физических и информационных систем, например закона сохранения энергии с его информационным аналогом, или определения работы в физических и информацион-

ных системах. Однако если эта тема заинтересует любопытного читателя, то можно быть уверенным, что его ждут интересные и неожиданные открытия.

Но ведь и человек способен к *делу*, равно как и множество живых существ, которые тоже совершают самые разнообразные осмысленные, и не очень, действия. Почему бы не расширить класс *машин*, добавив к нему биологические системы? При этом у нас появляется возможность изучить связи между информационными действиями и соответствующими физиологическими реакциями, на целом спектре разнообразных организмов, поскольку биологическая эволюция предоставила нам последовательный ряд: от примитивного червяка-нематоды *C. elegans*, до человека разумного, в которых связи между информационными и физическими органами уже достаточно хорошо изучены и описаны.



Если сравнить рефлекторные реакции в ответ на возбуждения в простейших организмах, с тем, как анализирует вопросы и генерирует ответы человек в процессе диалога, то можно проследить, как видоизменяются биологические системы, созданные с применением одного и того же базового конструктивного элемента – нейрона. Правда, проводить измерения внутри живых организмов дело тонкое и вряд ли в скором времени мы сможем создать универсальную структурно-функциональную модель, в которой для каждого органа будут получены энерго-информационные характеристики в джоулях и битах. Но характерные особенности и отличия в структурах различных по сложности организмов, могут дать хорошую пищу для рассуждений.

А *«Может ли машина мыслить?»* – с этого вопроса начинается, одна из самых популярных в истории вычислительной техники, статья А. Тьюринга «Вычислительные машины и разум». Этот вопрос вызвал бесконечное количество дискуссий и безусловно до сих пор, он является одним из самых интересных теоретических вызовов для многих программистов, и не только. Но если сформулировать его несколько иначе – *«Как машина может помочь человеку мыслить?»*, то из области сложных формальных проблем, с которыми столкнулся Тьюринг и его последователи, мы переходим к инженерным системам и техническим определениям, где результат должен быть конкретен и может быть из-

мерен понятным способом, например, как КПД физической машины, заменяющей ручной труд человека.

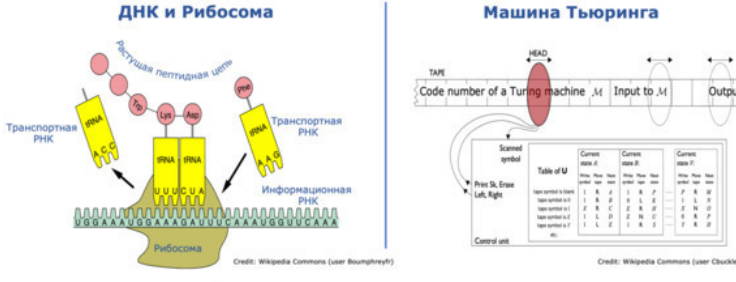
На первый взгляд, такая игра словами может показаться простой забавой для ума, однако, когда Тьюринг попытался найти формальный подход к ответу на свой вопрос, он тоже решил переформулировать его в других, менее двусмысленных терминах, и назвал эту проблему – «***игра*** *в имитацию*». В результате появился хрестоматийный «*тест Тьюринга*», который пытаются пройти поколения программистов. Если бы только Тьюринг смог предвидеть, что слово *имитация* в будущем, будет многими воспринято буквально и безотносительно к слову *игра*, то кто знает, возможно он попробовал бы подобрать ещё более аккуратные термины, и тогда, мы бы имели больше полезных функциональных чат-ботов, вместо множества «имитаторов», способных поддерживать лишь разговоры на общие темы.

Знаменитый вопрос Тьюринга можно рассматривать как продолжение его другой, не менее знаменитой, концепции универсальной вычислительной машины, которую он предложил в 1936 году. Для математиков машина Тьюринга, это несомненно прорыв, поскольку, согласно А. Черчу, Тьюринг – первый, кто ввел *физическое действие* в математическое понятие вычисления. В математике, с момента её возникновения, существует разрыв между «строго формаль-

ной» абстрактной теорией, и интерпретатором этой теории, которым всегда выступает другой математик. И каким бы строгим не было доказательство, в конце концов, проверяет его человек. У математиков нет компилятора, который является обыденным инструментом для программистов, а Тьюринг, если уж совсем упрощенно, предложил и обосновал возможность такого математического компилятора, по крайней мере, для вычислительных задач. Впрочем, при всей важности машины Тьюринга для фундаментальных исследований, программируют на ней лишь редкие математики и студенты курса «Теоретическое программирование», и вряд ли на всей Земле можно найти программиста, который использует её для решения своих практических задач.

Но есть в биологии чрезвычайно распространённый процесс, на который очень похожа универсальная модель вычислений Тьюринга! Если поместить рядом и сравнить две схемы – синтез протеина рибосомой и машину Тьюринга, то сходство основных функциональных блоков, логики работы и методов кодирования, становится наглядным. Рибосома считывает последовательности кодов, записанных на ленте информационной РНК, и на выходе конструирует структуру из соответствующих молекул. В машине Тьюринга, управляющее устройство считывает с ленты символы и следуя программе, выполняет ограниченный набор действий – перемещает головку, записывает новое значение на ленту

и т. п. И в рибосоме, и в машине Тьюринга, исполнение программ приводит к определенному результату. Обе программы – это линейные последовательности кодов, которые задаются простыми алфавитами. Обе машины получают программы извне и интерпретируют эти программы, превращая информацию команд в физические действия. Однако принципиальное отличие между двумя машинами всё-таки есть. Рибосома – это узко специализированная машина для производства органов и организмов, состоящих из разнообразных клеток, включая миоциты (мышечные клетки), нейроны и нервы, а машина Тьюринга – универсальная модель, попытка построить нечто, способное на всё. Если бы только Тьюринг мог предвидеть, что его машина окажется так поразительно похожа на рибосому и ДНК, может быть, вместо гипотетической модели универсального вычислителя, сегодня мы бы имели простой, но работающий генетический интерпретатор?



Синтез протеина и машина Тьюринга

Биологические машины представляют собой симбиоз энерго-информационных подсистем, и все действия в них инициируются внутренними механизмами управления. В отличие от биологических, физические и информационные машины создавались как инструменты, предназначенные для повышения производительности сначала физического, а затем умственного труда, и соответственно, в их основе лежит принцип руководства человеком, иными словами, программа управления всегда разрабатывается извне этих машин.

Энерго-информационные процессы в живых организмах, можно разделить на два типа: *репродуктивные* – когда при создании новой клетки, рибосома конструирует протеины под управлением программы ДНК/РНК, и *рефлекторные* – когда действия осуществляются моторными нейронами на основании состояния нервной системы. *Репродуктивный* и *рефлекторный* механизмы управления имеют четкое функциональное назначение: первый предназначен для создания исполнительных механизмов, а второй – для их эксплуатации. Если перед самым умным программистом на Земле поставить задачу спроектировать систему, в которой компоненты должны эволюционировать, приспосабливаться к внешней среде и самообучаться, то скорее всего, обладая всеми доступными знаниями, он пришел бы

к очень похожему решению. Для программирования *репродуктивных* систем подходят алгоритмические методы, а вот для *рефлекторных*, нужны принципиально новые инструменты и платформы, такие как интегрированные методы искусственного интеллекта и нейронное программирование.

Теория управления физическими машинами традиционно относилась к разделам математики и механики, вплоть до появления в 1948 году книги Н. Винера «Кибернетика, или управление и связь в животном и машине», которая внесла некоторый сумбур в эту область знаний. Новая наука предполагает, что существуют общие законы управления, которые можно описать в виде формальных математических уравнений, позволяющих рассчитать значения параметров воздействия на механические, биологические, социальные или экономические объекты, для достижения определённых целей. Так же, как и классическая механика Ньютона, которая решает множество задач известного типа, так и кибернетика с успехом применяется в широком спектре разнообразных систем. Но всякая «правильная» теория должна иметь область ограничения, и при достижении определенного уровня сложности, нужны новые теории и новые методы решения, и как никому другому, это хорошо известно всем прикладным программистам.

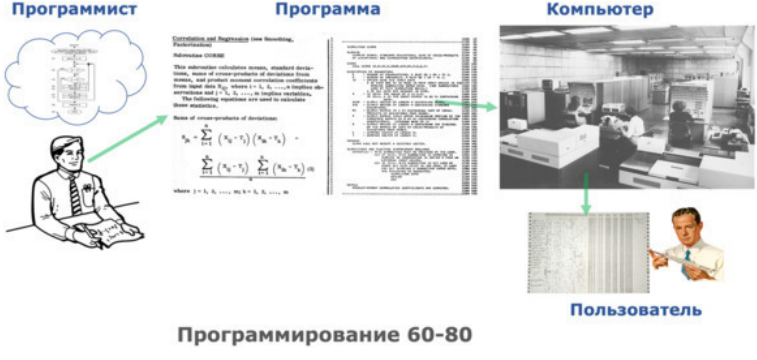
Современные системы управления используют искус-

ственный интеллект в сочетании с алгоритмическими методами и имеют развитый интерфейс к разнообразным аналого-цифровым преобразователям. В результате мы наблюдаем, как быстро прогрессируют, обучаются, становятся автономными и независимыми от непосредственного вмешательства человека самые разнообразные роботы и автоматизированные системы. «*Обучение машин*», «*передача знаний*», «*принятие решений компьютером*» – термины, которые из профессионального жаргона постепенно переходят в повседневный оборот. Если сравнительно недавно программирование было уделом избранных, то сегодня это массовая профессия, и в скором времени, это будет уже не только профессия, но и общедоступный элемент культуры, такой же, как умение писать и читать. И чтобы понять, как это произойдет, попробуем, используя математическую терминологию, экстраполировать процесс.

Двуликий Янус: инженер - ученый?

«Ученые изучают то, что уже есть, инженеры создают то, чего никогда не было». В этой цитате, авторство которой Интернет приписывает А. Эйнштейну, очень доходчиво показано различие между теоретическими исследованиями ученых и практической деятельностью инженеров. Ученым требуется время для наблюдений и размышлений. Научная деятельность, со времен Аристотеля – это процесс получения знаний, состоящий из определенной последовательности шагов. С момента появления гипотезы, её проверки и публикации, до подтверждения теории научным сообществом, может пройти много времени, однако от этого зависит и качество науки и, соответственно, образования. В классических науках, времени всегда хватало – история механики насчитывает более двух тысяч лет, пятьсот лет развива-

ется теория электричества, а вот в программировании глубокие изменения происходят в такие короткие промежутки, что наука просто не успевает за темпами развития компьютерных технологий. Наверное, поэтому и возник разрыв между теоретическими исследованиями в области программирования и полноценным профессиональным обучением, позволяющим инженерам рационально применять научные результаты в практике.



Программирование 60-80

Современное программирование появилось в середине 50-х годов, как система из трех взаимодополняющих элементов – *программист, программа и компьютер*. Компьютеры тогда были, пусть и внушительных размеров, но всё же понятные вычислительные машины с известной логикой и состояниями. Практически все программисты имели либо математическое, либо специальное инженерное образо-

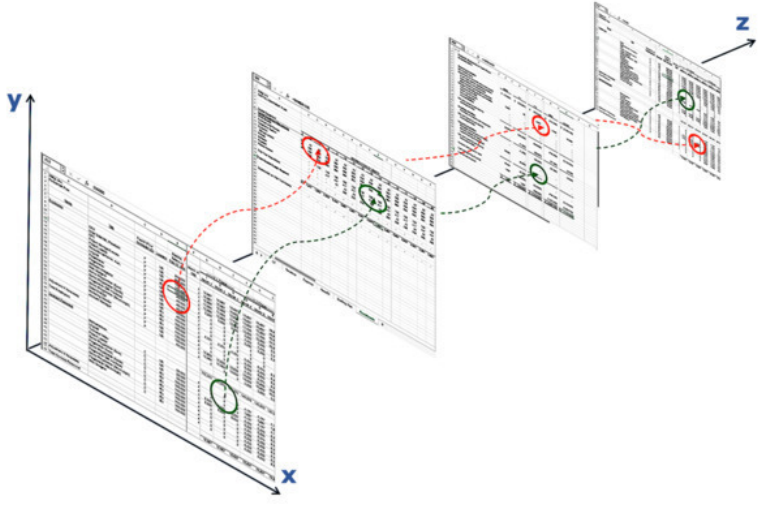
вание. Системные программисты разрабатывали операционные системы, компиляторы и специальные утилиты для ЭВМ, а программисты-математики создавали прикладные программы, которые представляли собой реализации различных алгоритмов решения численных задач. Такая модель программирования напоминала геоцентрическую систему Птолемея и многие надеялись, что рано или поздно, будет создана универсальная теория языков и верификации программ, что сделает их похожими на «вечные» математические конструкции.

Но компьютеры развиваются стремительно, и за сравнительно небольшой период времени, прошедший с момента появления первых ЭВМ, мы наблюдаем уже несколько эволюций, которые принципиально изменили мир информационных технологий. Если в самом начале программирование было ориентированно на численные методы, где основным объектом вычислений является *число*, то на следующем этапе, наравне с числами, в программах появляются *данные* и, как сформулировал в 1976 году Н. Вирт:

Программа = Алгоритм + Структуры Данных

В эти же годы появляются серверы и персональные компьютеры, объединенные в простейшие сети, а наряду с прикладными математиками, в информатику приходит всё больше и больше инженеров, специализирующихся на решении

задач обмена и организации данных. Тогда же появилась принципиально новая форма программирования – электронные таблицы, которые позволяют «непрограммистам» создавать сложные модели из данных, связанных друг с другом функциональными отношениями в многомерном виртуальном пространстве.



Многомерные модели данных в Excel

Это принципиально новое решение соответствует естественному для человека способу пространственного мышления и позволяет, используя простую систему координат, не только систематизировать структуры и отношения между

различными данными, но и передавать изменения непосредственно от точки, в которых они происходят, ко всем взаимосвязанным объектам. В электронных таблицах *действия инициируются данными*, и это принципиально отличает их от алгоритмических систем, в которых данные изменяются в результате интерпретации последовательности команд в программе.

Программист-зануда здесь справедливо возразит, что *«за всем этим стоит простая система ссылок и таблиц значений, которые пересчитываются в ответ на событие, возникающее при изменении координат указателя мышки»*. Но любопытный программист сможет представить стоящие за этим рисунком реальные физические процессы, в связанных между собой исполнительных элементах, очень похожих на сеть биологических нейронов или на вычислительную модель в аналогово-цифровых комплексах, которую можно получить, соединив кибернетику Н. Винера с машиной фон Неймана. Ещё раз заметим, что в своих моделях и Винер, в аналоговой, и фон Нейман, в цифровой, ссылаются на нейрон в качестве прототипа вычислительного элемента.

Сегодня аналогово-цифровые преобразователи используются повсеместно: от сенсоров, переключателей и микрофонов в «умном доме», до промышленных роботов и медицинских имплантатов, соединенных различными видами

как проводной, так и беспроводной связи. На смену серверам пришли *сервисы*, и компьютер, как целостная вычислительная машина, трансформировался в набор фактически неограниченных ресурсов в облаках. Числа, с которыми имеют дело программисты при решении алгоритмических задач, составляют очень небольшой процент от постоянно увеличивающихся по объёму, как структурированных, так и неструктурированных данных, распределённых в глобальных и локальных сетях Интернет. А одно из новых ключевых направлений в программировании 2020, это интерактивность и поддержка прямого взаимодействия между бизнесом и потребителями.

В 1968 году вышел первый том монографии Д. Кнута «Искусство программирования». В то время, количество программистов в мире исчислялось десятками тысяч, а профессионалом мог считаться лишь тот, кто сумел бы решить большинство из приведенных в этой книге упражнений. На обучение этому у среднего студента уходило несколько лет, и трудно было представить тогда программиста без специального инженерного или математического образования. В 2020 году программированием занималось уже более двадцати миллионов человек, и эта профессия превратилась в одну из самых массовых и доступных. Многие из современных программистов ничего не слышали о книге Д. Кнута, что, впрочем, не мешает им создавать полезные прикладные

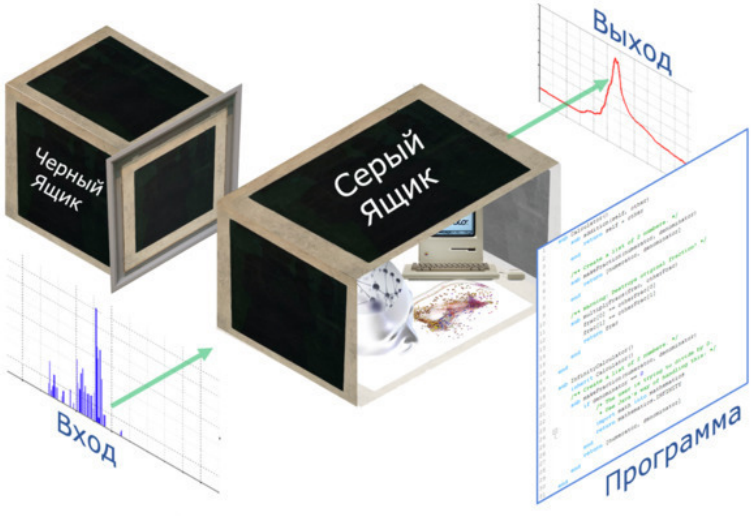
решения. Изменилась природа приложений, и соответственно появились новые технологии и инструменты. Если первоначально программирование было предназначено для автоматизации работы вычислительных машин и реализации алгоритмов специально подготовленными профессионалами, то сегодня, это в первую очередь инструмент для накопления и передачи знаний, а основы программирования и информатики включены школьную программу для начальных классов.

Программирование 2020

За эти годы в мире информатики было решено множество проблем. Но чем шире горизонт – тем больше открывается неизвестного, и вместе с этим появляются новые задачи, среди которых, в первую очередь нужно выделить обработку знаний. До сих пор, основным объектом программи-

рования являлись данные. Знания – это значительно более сложная система, в которой данные соединяются с процедурами, а обмен знаниями, это контекстно-зависимый процесс. В процессе обмена могут участвовать группы, где каждый участник имеет своё уникальное состояние.

Данные, информация и знания – три основополагающие категории, как в биологических, так и в компьютерных системах. Термин *данные* относится к неопределимым аксиоматическим понятиям, которые, чаще всего объясняют, используя косвенную рекурсию. Например, согласно Википедии, *данные* – это *факты* или *события*, а статистик и специалист по машинному обучению определит их как *числа* или *вектор чисел*. Но если продолжить уточнение: а что есть *факты* или *числа*, то круг очень быстро замкнется. Однако если мы зададим разумные области ограничений, то оказывается, что этот термин поддается конструктивному определению.



«Черный ящик» Малевича и Программа

Один из концептуальных способов описания поведения и построения моделей самых разнообразных сложных систем, носит название «Метод чёрного ящика». В классическом чёрном ящике важно только то, что *входит* и *выходит*. Именно это мы и будем рассматривать как *данные*. Входные и выходные *данные* – это поток сигналов, который из всего огромного разнообразия сигналов внешнего мира, с одной стороны выделяет и воспринимает, а с другой, генерирует и возвращает обратно, конкретная система. Если же мы приоткроем *чёрный ящик* и заглянем внутрь, то у нас появляются некоторые знания, с помощью которых можно

определить, какой компонент воспринимает сигналы на входе, а какой отвечает за их генерацию на выходе, в теперь уже *сером ящике*. И наконец у программистов, вход и выход связаны *программой*, где каждый шаг понятен и определен. При таком подходе у нас появляются количественные характеристики входных и выходных компонент, такие как пропускная способность, формат сигнала, объемы памяти и др. Только не нужно забывать, что декомпозируя *ящик*, можно этот же подход применить на любом уровне и для любого функционального блока внутри. Так, например, минимальным компонентом в компьютере можно считать ячейку памяти, а в биологической системе – отдельную клетку, хотя всегда можно продолжить декомпозицию, как клетки, так и ячейки памяти.

В классической модели компьютера данные поступают и передаются вовне через устройства ввода/вывода. В биологических системах, *данные* – это разнообразные физические взаимодействия или молекулярные объекты, которые могут восприниматься сенсорными клетками, и на которые реагируют рефлекторные механизмы. *Данные* могут генерироваться *источником* целенаправленно, например радиостанцией, которая создаёт и передает сообщения в сеть слушателей, или человеком, порождающим поток слов во время разговора в интернете. Это могут быть любые физические сигналы, возникающие в окружающем нас мире: космиче-

ское излучение астрономических объектов или молекулы запаха цветка в воздухе. И все эти сигналы превращаются в *информацию* в тот момент, когда *получатель* воспринимает их и интерпретирует в соответствии со своим состоянием. И уже из *информации* могут быть получены *знания*, которые образуют индивидуальную ассоциативно-связанную систему фактов и процедур.

Динамические связи нейронов

В биологических системах, физиология *знаний* – это изменения *состояний* нейронов, *свойств* нервов и появление новых ассоциативных или моторных *связей*. Мы еще слабо представляем, как всё это происходит в комплексе, но некоторые детали этого процесса уже более или менее понятны. Например, для того, чтобы установить новую связь, целевой нейрон выделяет *нетрины* – специальные макромолекулы,

которые привлекают аксоны, а исходный нейрон инициирует рост нерва, который «по запаху» должен найти свою цель. Всё выглядит элементарно для пары нейронов. Но если попробовать представить себе сложнейшую композицию асинхронных процессов, где один исполнитель начинает, другой подхватывает и продолжает, и всё это происходит в миллиардах нейронов, между которыми появляются триллионы связей, и в этой гармонии возникают новые индивидуальные знания, то становятся понятным, почему Д. Кнут и Э. Дейкстра сравнивают программирования с музыкой, и утверждение биохимика, что *«мы живем до тех пор, пока внутри нас звучит симфония запахов»*.

Биохимики – люди в меру циничные, поскольку знают, что человеческие *чувства* – это производные химических процессов, которые в норме инициируются нервной системой. Для того, чтобы то или иное чувство возникло, нужно заставить эту систему работать, и хотя человек может и обмануть самого себя, испытав иллюзию чувств при помощи искусственных наркотических субстанций, но ничего нового при этом у него не возникнет. *«Получается, что когда мы узнаём нечто, у нас возникает биохимическое* ***Чувство Нового****? И пока наш мозг способен выделять нетрины, мы способны удивляться и учиться?»* – спросит романтичный программист. Вполне возможно, что *чувство нового*, это одно из основополагающих и мотивирующих в индивидуаль-

ном развитии думающего существа. И живое существо активно живет и развивается лишь до тех пор, пока его мозг способен воспринимать и устанавливать новые связи между нейронами. А в понимании того, как и почему нетрины притягивают аксоны, и лежит ключ к созданию систем по обработке знаний.

Эволюция биологических систем происходит от простейшей, *но далеко не простой*! клетки, к простейшим, а в последующем, ко всё более усложняющимся организмам. В какой-то момент у организмов появляется нервная система, у которой уже на следующем этапе развития, формируется головной мозг.

клетка → системы клеток → нейроны → головной мозг

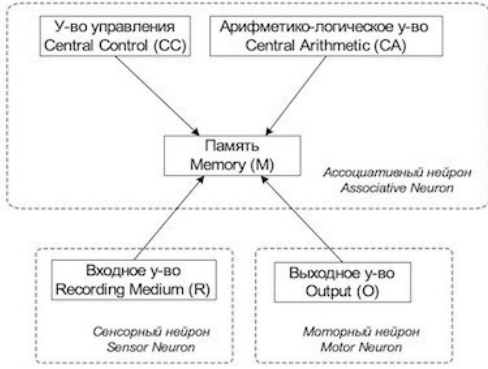
Развитие информационных машин происходит похожим образом: от простых компьютеров – к сетям и облакам. Затем, на очередном этапе эволюции, появляется искусственный интеллект, у которого в дальнейшем должен будет сформироваться аналог головного мозга – *система искусственных нейронов*?

компьютер → интернет → искусственный интеллект →???

Клетки являются исполнительными механизмами – машинами, которые выполняют заданные действия, под управлением внешней программы РНК. Эта программа задает правила и определяет поведение любой биологической системы. Программа может допускать варианты, вероятности, изменять реакцию в соответствии с состоянием и значением тех или иных параметров, но интерпретация программы РНК рибосомой, происходит в соответствии с алгоритмом, который заложил в нее «автор». Клетка может породить новую, модифицированную клетку, но она не может учиться и приобретать знания.

Точно также и компьютеры были созданы для того, чтобы строго исполнять последовательности инструкций. Идеи и цели, которые находились в голове у программиста при создании программы, не доступны центральному процессору, который должен получить команду и выполнить ее, в соответствии с правилами, которые заложил в него конструктор, и которые ожидает от него программист. Многие поколения классических компьютеров были построены на основе принципов, заложенных в архитектуре фон Неймана. В процессе работы над ней, фон Нейман соединил доступные для него знания физиологии нервной деятельности, с математическими формализациями и инженерными решениями. Структура его вычислительной машины состоит из трех блоков: вход,

обработка и выход, по аналогии с тремя типами нейронов: сенсорные, ассоциативные и моторные. Однако если сравнить современные модели нейрона и рибосомы, то окажется, что его архитектура значительно ближе к рибосоме чем к нейрону!



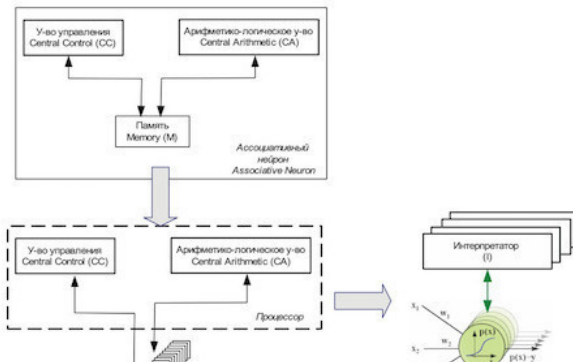
Архитектура фон Неймана

В современных компьютерах оперативная память содержит множество однотипных элементов, количество которых сравнимо с количеством клеток в биологических системах, однако все эти элементы являются пассивными, и в отличие от биологических нейронов не являются функционально полными устройствами.

Первоначально фон Нейман объединил *устройство*

управления, *арифметико-логическое устройство* и *память* в один блок, который он назвал ассоциативным нейроном или центральным процессором. На первый взгляд может показаться, что его процессор напоминает модель нейрона, однако биологический ассоциативный нейрон в отличие от пассивного запоминающего элемента обладает самостоятельной активностью. Компьютер на ассоциативных нейронах так никогда и не был реализован – вместо множества активных элементов, из которых он должен был бы состоять следуя первоначальной схеме фон Неймана, классический компьютер построен из пассивных ячеек памяти, связанных с единственным центральным процессором.

Первое поколение искусственных нейронных моделей, основанное на работе МакКалока и Питтса «*A logical calculus of the ideas immanent in nervous activity*», было построено исходя из предположения, что нервная активность биологических систем подчиняется закону – «*всё или ничего*», и как следствие этого, «*…нервную деятельность, нейронные события и отношения между ними можно рассматривать с помощью логики высказываний*». Искусственный нейрон, построенный по этой схеме, содержит два основных блока – *сумматор* и *функциональный преобразователь*, а сама сеть, состоящая из таких нейронов, предполагается неизменной во времени.



От архитектуры фон Неймана к нейронной архитектуре

Ограничения архитектуры фон Неймана стали заметны, практически сразу же после появления построенных на её основе промышленных версий компьютеров и языков программирования. Джон Бэкус, был один из первых, кто проанализировав узкие места машин Тьюринга и фон Неймана, предложил принципиально новую модель – функциональное программирование, или, если применить более современную терминологию – распределенную систему функциональных объектов с динамическими связями. И хотя в его работе используется сложный математико-логический способ анализа проблемы и описания решения, если настойчивый программист с базовым математическим образовани-

ем уделит немного времени чтобы разобраться в сути его небольшой статьи: *«Can programming be liberated from the von Neumann style?»*, то для него станет понятной связь между моделью λ-исчислений и биологическими нейронами.

Существует глубокое заблуждение, что *«нейрон прост»* и представляет собой *элемент* с известными функциональными свойствами, которые можно формально специфицировать. При этом упускаются из вида хорошо известные факты:

– как и всякая клетка, нейрон может размножаться;

– нейрон способен инициировать соединения с другими нейронами;

– нейрон может принимать или отказываться (attraction, repulsion) от приглашений на соединение, поступающие от других клеток;

– внутри нейрона есть механизм, который реагирует на изменение электро-магнитного и биохимического состояния внешней для него среды и, соответственно, у него есть генератор ответных электрических или химических сигналов;

– нейрон имеет внутреннюю долговременную память – при помощи рибосомы он способен копировать и интерпретировать молекулы РНК, в которых могут храниться разнообразные данные.

Эти свойства нейрона, а также ещё многое, чего мы пока не знаем, позволяют утверждать, что *нейрон – не прост*! Это

не элемент, а сложная система, и ближайшим аналогом ему является вовсе не процессор или запоминающий элемент памяти, а целостный компьютер. Если исходить из этой аналогии, то нервная система похожа на сеть компьютеров, или – ***интернет нейронов***.

Способность нейрона связываться с другими нейронами и с другими функциональными клетками, изменять свое состояние в соответствии с внутренними процессами и внешними сигналами, запоминать значения и генерировать выходные импульсы – является важным свойством, которое предполагает новый подход к анализу нервной деятельности, но одновременно открывает программистам пути для использования опыта биологических систем при решения принципиально нового класса задач.

Когда-то Аристотель в своих рассуждениях о природе знаний, выделил два типа поведения: подчиненное и рассудительное. Подчиненное поведение – это исполнение инструкций без необходимости или возможности *понимать* цель. А рассудительное – способность определить пользу действия и разработать инструкцию для исполнителя. Человек, способный понять – способен сформулировать цель и определить способ достижения. Так же и программист создает последовательности инструкций для исполнения компьютером аналогично тому, как менеджер или технолог создают проце-

дуры для исполнителей в офисе или на производстве. В биологических системах, для того чтобы *понять*, используются динамические многослойные нейронные сети, способные генерировать новые ассоциации. Создание новых ассоциативных связей – это то, что отличает высшую нервную деятельность от любых, сколь угодно сложных, исполнительных механизмов или систем интерпретации программ. Нервная система червяка-нематоды принципиально отличаются от человека не только количеством нейронов, но и тем, что наш мозг постоянно создаёт новые связи, а у червяка, после этапа формирования, они остаются одними и теми же на протяжении всего периода его существования. И наверное поэтому люди все разные, а червяки – одинаковые.

Какой бы сложной не была программа, до тех пор, пока исполнительный механизм будет следовать ее логике, этот механизм не создает новых знаний. И в этом отличие машин Тьюринга или фон Неймана от динамических нейронных моделей. Машины такого типа не могут работать без загруженных в них программ, а наш головной мозг работает без центрального процессора и без внешнего программиста. Теперь на вопрос: *«Может ли машина мыслить?»* рассудительный программист мог бы ответить так: *«Одна машина мыслить не может точно, а вот множество взаимосвязанных машин с определенными новыми свойствами, похоже, что да. И кто знает, возможно, что Интернет – это про-*

тотип нового поколения умных машин?».

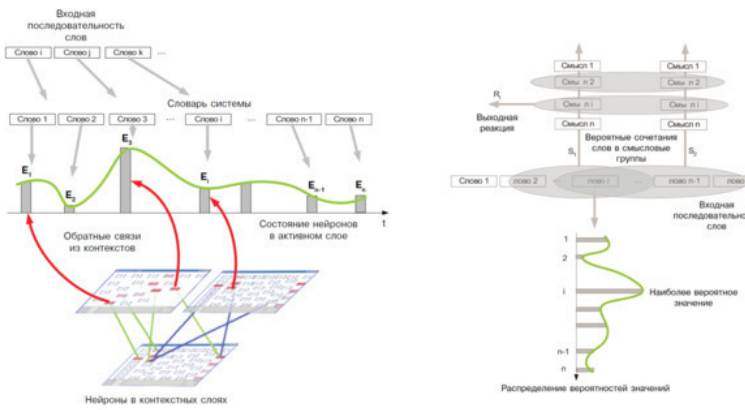
Возвращаясь к ДНК и базовым моделям вычислительных машин, в которых данные интерпретируются однозначно, можно сказать, что в основе этого лежит *буква*. В формальных грамматика такой объект называется *Терминалом*. Действительно, для заданного алфавита, любая система должна распознавать букву из этого алфавита, однозначно. И любая буква, поступающая на вход такой системы, также должна иметь единственное значение. После того, как входное устройство передало сообщение процессору, буквосочетания будут интерпретироваться в соответствии с логикой внутренней программы, которая может быть весьма сложной и иметь свое состояние. Но если заглянуть в память машины, мы увидим статические последовательности «букв, которые могут быть изменены только центральным процессором по командам программы.

Элементарные буквы из которых построены РНК и компьютерная программа

Хотя алфавиты, с которыми работают компьютеры, меняются часто, неизменной остается двоичная система, на базе которой строятся более мощные структуры – машинные слова, шестнадцатеричные символы, юникод, и т. п. А вот базовая спецификация языка, на котором записаны коды биологических программ не меняется на протяжении миллиарда лет! И более того, это язык, в основании которого лежит очень простой алфавит. Можно сказать, что геном человека – это программа, написанная всего четырьмя буквами.

Если бы нам удалось построить аналогичную основу систем программирования, то продолжительность жизни программ увеличилась бы существенно, и это то направление, в котором работают конструкторы виртуальных машин.

Совсем иная ситуация возникает, когда мы используем *слова*. *Слово* естественного языка не может быть определено однозначно, потому что его интерпретация происходит в системах с различными ассоциативными связями и состояниями. Одно из ключевых свойств нейронных сетей, это *неоднозначная* интерпретации входного сообщения, которая зависит от контекста или состояния этой сети. Каждый человек или каждая активная нейронная сеть, получив сообщение, поймет его по-разному и более того, одно и то же сообщение будет воспринято иначе одной и той же системой в разное время. Интересно, что такое свойство нейронных систем, хорошо согласуется с физическими свойствами неопределенности и относительности – два наблюдателя не в состоянии синхронизировать свои знания.



Последовательности слов и динамический контекст

Поток слов, поступающих в нейронную сеть, вызывает возмущение ее состояния, которое можно представить себе как волнение поверхности воды от падения капель дождя. Такой образ, совсем далекий от строгих определений, к котором привыкли алгоритмические программисты, вряд ли смутит человека, не знакомого с формальными грамматиками, но неоднократно наблюдавшего волнения воды. А поскольку наша задача при создании новых «умных» систем, будет очень похожа на моделирование волновых процессов, мы надеемся, что подобные аналогии помогут с формированием образного представления о том, как такое решение может работать.

Дело и деньги

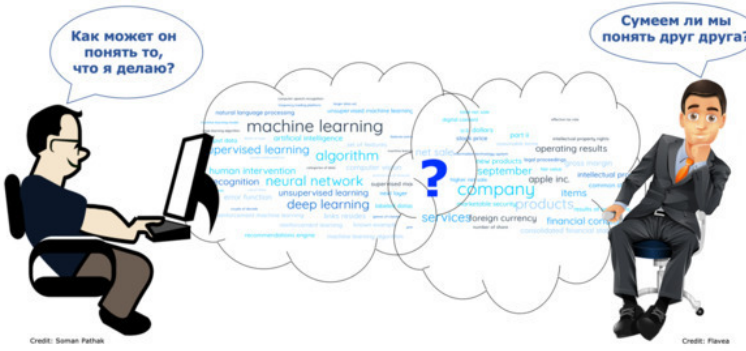
Богатый – это тот, у кого денег всегда достаточно.

Почти математическое определение (Р. Ф.)

Среди программистов не бывает бедных. Они по сути своей профессии всегда должны иметь достаточно средств или возможностей, чтобы получить доступ к компьютеру. И не обязательно его покупать – очень часто программисты работают на чужих компьютерах, но именно «власть» над компьютерами, физическими или виртуальными, делает программиста собственником в самом положительном смысле этого слова. Некоторые программисты зарабатывают много денег, а у других, денег достаточно, чтобы получать удовольствие от любимого дела. *Достаточно* – очень интересное слово! В одном случае оно может определять условие, а в другом, количество, как и категория *бесконечность*, которая в одной математике – число, а в другой – процесс. В древности, люди употребляли слово *много* для обозначения совокупности предметов, а правила счета в самой старой троичной системе (*1, 2, много*) были очень просты: *1+1=2; 2+1=много; много+много=много*. По аналогии с такой троичной арифметикой слово *достаточно* можно применить как критерий в нечёткой логике человеческих ценностей.

Для программистов различные логики и операции над лингвистическими значениями, являются естественной и рутинной нормой. Например, вместо двоичной системы счисления или булевой логики, в современном программировании для определения двоичного состояния бита памяти, используются *троичное* значение, которое может быть: **1, 0** или **null** (если значение неопределено или недоступно) и большинство современных компиляторов это учитывают. А такие лингвистические переменные, как «*лучше*» или «*хуже*», согласно Л. Заде, можно свести к функции или набору числовых параметров, и уже к ним применять стандартные алгебраические операции.

И вот здесь на нашей сцене появляется деловой человек (бизнесмен), для которого подобные логические и лингвистические проблемы не относятся к категории значимых, а понятие богатство по обыкновению, это количество, которого всегда не достаточно.



Программист и бизнесмен

Если посадить рядом бизнесмена и программиста, разговор между ними получается очень сложным. У делового человека и у программиста разное практически все: словарный запас, парадигмы мышления, критерии, стиль жизни, и ко всему этому, долгая история взаимного недоверия. Но есть нечто, объединяющее их вопреки всем различиям. *Дело* или действие, это то, ради чего, в конечном счете, программист пишет программы, и если во всех рассуждениях, наряду с физиком и математиком, он будет прислушиваться к мнению делового человека, а иногда и обращаться к нему с идеями или предложениями, то от этого безусловно выиграет не только наша книга.

Все деньги в современном мире создаются или контролируются информационными технологиями, а программи-

сты, это те, кто создают и контролируют эти самые информационные технологии. *«Если бы ещё знать, что такое деньги?»* вдруг возникает дерзкая мысль у наивного программиста. Ведь не зря же появились биткойн и множество его вариантов. Правда, хитрые бизнесмены быстро привязали биткойны к доллару, и – пропала независимая мера! Но если деньги выступают в качестве меры стоимости, то, следуя логике измерения энерго-информационных действий в *джоулях* и *битах*, почему бы не предложить универсальную меру, в которой эти две единицы будут объединены? И тут расчетливый программист декларирует: *«Если построить энерго-информационную модель, в которой единицы измерения энергии и информации – **джоуль** и **бит**, будут связаны интуитивно понятными отношениями, то это позволит создать принципиально новую денежную единицу, назовем ее условно ДжоБит, которая может прийти на смену биткойну и даже доллару.»*. Тогда на энерго-информационном рынке возможны вот такие отношения подобия:

1 грамм молока ~ 1 байт знаний ~ 1 ДжоБит
1 грамм колбасы ~ 1 байт веселья ~ 1 ДжоБит
3 минуты программиста ~ 1 минута врача ~ 2 ДжоБит

В такой модели становится возможным обосновать соответствие между трудом информационным и трудом физиче-

ским. И вековая мечта философов о равенстве, о справедливом обмене на рынке производства и потребления, становится теоретически достижимой. Ведь если можно измерить любой труд, то и результатами труда можно обмениваться на основе понятных и соизмеряемых величин. Старый лозунг социалистов: «*От каждого – по способностям, каждому – по труду*» будет оставаться идеалистической догмой, до тех пор, пока не появится сначала теоретическая, а затем и практически осуществимая возможность измерить труд в двух его составляющих: физической и информационной.

Если это произойдет, будет устранена ключевая проблема в экономической модели наивных коммунистов, в которой только физический труд рабочих оценивается на основе измерений и нормативов, а знания и удовольствия относятся к непостижимой области идеализма. Одновременно и теория капитализма приобретёт радикально иной смысл, когда рыночные отношения, по аналогии с законом сохранения энергии в физике, будут служить основой для оптимизации новой экономики, в которой термины «*бесплатное*» и «*прибыль*» не имеют смысла. Равенство людей, должно быть основано на том, что они получают равные деньги за равный труд, независимо от того, где и как они работают. Один программист может написать программу за неделю, а другой – за две, но в результате оба совершат одина-

ковую работу и должны получить за неё одинаковую плату. Но если за одинаковое время один программист заработает сто единиц, а другой десять, то это вполне возможно и нормально, и подтверждается старыми статистическими данными измерений различных параметров технологии программирования, которые показали, что производительность труда среди программистов в среднем отличается приблизительно в десять раз.

«Получается, что когда в проект приглашают людей из других стран, и за ту же самую работу им платят меньше, то это форма дискриминации? Или эксплуатации? Становится понятной суть поведения некоторого подмножества множества посредников (финансовых, маркетинговых, страховых и др.), которые разнообразными способами отнимают у людей деньги, не создавая ничего полезного и прикрываются понятием прибыли, якобы обоснованной теорией капитализма. Мы должны бороться против ***дешёвой*** *рабочей силы!»* – возбуждённо воскликнет программист-демократ и с ним наверняка согласятся многие беспартийные участники производственных отношений.

Однако программисту не следует слишком далеко углубляться в самые различные области знаний, с которыми его сталкивает профессиональная деятельность, и особенно – в политические споры! Интерес программиста к предмету,

с которым его на короткое время сводит судьба, как правило, дилетантский. Но все-таки, в логике и умению быстро разобраться в сути проблемы, программистам не откажешь. И вот если провести экспресс-анализ архитектуры, да и самой возможности построения системы поддержки универсальной денежной единицы, основанной на энерго-информационной мере, то аргументы и предложения расчетливого программиста могут показаться вполне разумными.

Можно начать с *информационного барьера* В. М. Глушкова. Рассуждая о способности человека обрабатывать информацию, он пришел к выводу о необходимости создания новой, безбумажной экономики. В рассуждениях Глушкова, совершенно далеких от монетарных проблем, есть вроде бы тривиальный, но в то же время очень важный формальный момент: способности человека потреблять энергию и тратить ее на работу, так же, как и воспринимать и обрабатывать информацию – ограничены, и стало быть, существует барьерная функция, которая позволяет вычислить объективное значение суммарного производственно-потребительского порога человечества в джоулях и битах. И эта величина может послужить основой для определения не только оптимального объема данных на веб-странницах в Интернете, но и предельного количества универсальных *ДжоБитов*, по аналогии с золотым запасом или максимальным количеством биткойнов. А если известна предельная величина и мы

сможем построить баланс, то это и есть основа для универсальных расчетов. Таким образом, вместо сложного и закрытого механизма печати и перераспределения денег, которую мы имеем сегодня, можно построить прозрачную и основанную на понятных показателях систему, в которой деньги будут обладать всеми свойствами привычных мер, таких как *масса*, *длина*, *температура* и т. п. Все основные компоненты и технологии, которые потребуются для её создания, имеются в наличии, включая протоколы блокчейн, методы измерений энергии и информации, а также расчетливых программистов, которые могут всё это реализовать. Дело за малым.

Подобные крамольные идеи приходят в головы программистов довольно часто. Иногда они бывают утопические, как *ДжоБит*, но иногда и конструктивные. Современная история знает примеры, когда такие идеи помогали деловым людям заработать очень много денег! Особенно в наиболее благоприятное для инвестиций время, когда в вычислительной технике происходят кардинальные изменения и когда закладываются решения и платформы, которые будут определять развитие компьютеров на десятилетия вперед.

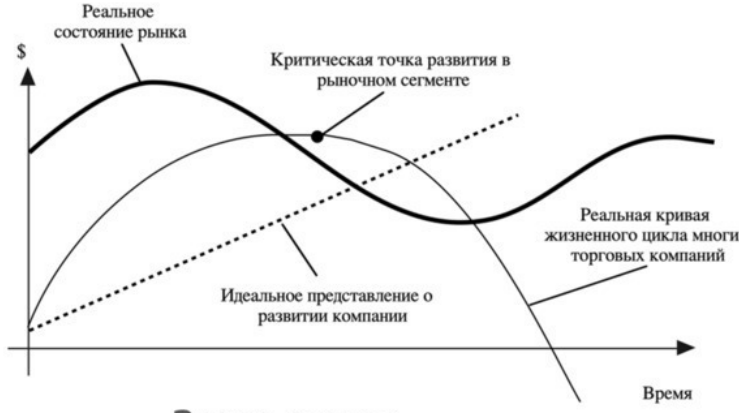
Этапы и бизнесы

Начало нового периода – это идеальное время для софтверных стартапов, и большинство гигантов компьютерной индустрии возникли как раз в такие моменты: Apple и Microsoft были созданы в конце 70-х, одновременно с первыми персональными компьютерами. Amazon и Google – в конце 90-х, как раз накануне появления первых вычислительных облаков. Если спросить старого программиста, который помнит эпоху ЭВМ, бум персональных компьютеров и расцвет облаков, о том, когда наступит следующий этап и чем он может отличаться, то его ответ может выглядеть вот так: *«Нужно просто к слову **облако** добавить слово **персональное**».*

Название «*Персональные облака*» – идеально подходит для обозначения нового этапа в эволюции вычислительной техники. Не сложно спрогнозировать, что в информационных технологиях в ближайшем будущем будут доминиро-

вать гипер-персональные умные устройства и мультимодальные формы общения: с одной стороны, человек будет окружен разнообразными датчиками, экранами, микрофонами, гаджетами, смартфонами, а с другой – ему будут доступны неограниченные информационные ресурсы, как в локальных, так и в удаленных облаках. И вследствие этого появится новая среда, в которой виртуальные помощники, обладающие искусственным интеллектом, соединят вместе «Умный дом» и «Умное здоровье», «Профессиональных помощников» с «Персональными», и эта новая среда будет способна совершать физические действия, общаться на естественном языке и самое главное – понимать конкретного человека.

Можно по-разному интерпретировать двадцатилетние периоды в эволюции вычислительной техники, например, наложить полусинусоиды на технологические фазы, и определить участки, на которых та или иная платформа достигает пика своего развития. Но вот однажды успешный бизнесмен решил объяснить программисту, в чем состоит секрет успеха в бизнесе, и согласно его модели, рынок подчиняется «*закону колеса*», а правило успеха он сформулировал следующим образом: *«Бизнес нужно начинать в нижней точке цикла, а продавать в наивысшей. О. Т.»*, что неплохо согласуется с аксиомой об инвестициях в моменты переходных периодов и совпадает с поведением успешных компаний, о которых шла речь выше.



Закон колеса

В рациональном мире потребления всё характеризуется экономическими показателями, и задача новой технологии сводится к их улучшению. Чтобы понять *почему* и *как* Персональные облака смогут изменить нашу жизнь к лучшему, и одновременно принести прибыль деловым людям, попробуем определить связь между возможностями новых информационных технологий и экономическими показателями. Для этого рассмотрим структурно-функциональную модель рыночных отношений с точки зрения программиста-потребителя, который, с одной стороны, владеет системным анализом, а с другой, как и все остальные потребители, заинтересован в оптимизации и повышении эффективности про-

цесса получения услуг и продуктов в этом самом мире потребления.

Следуя традициям концептуального проектирования, в этой модели будут представлены три группы интересов: *потребителей*

Конец ознакомительного фрагмента.