УДК 004.048

Сотников Олег Олегович,
магистр делового администрирования, инженер по специальности информационные системы и технологии.
Основатель
ТОО «G1 Software Kazakhstan»
Республика Казахстан, г. Астана

СМЕНА ПАРАДИГМЫ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ С ПЕРЕХОДОМ ОТ ЧЕЛОВЕЧЕСКОГО ТРУДА НА ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Аннотация: *в настоящей работе рассмотрена эволюционная модель методов разработки программных продуктов в различных ИТ компаниях, новые тренды и изменения на рынке разработки программного обеспечения. Автор статьи на примере собственной компании в течении многих лет изучал существующие и перспективные подходы к разработке программного обеспечения, подходы к управлению командами разработки, механизмы управления эффективностью производственной деятельностью для ИТ компаний. Полученный в результате исследования концепт разработки программных продуктов дает многократный прирост эффективности труда разработчиков и продуктовых менеджеров, многократное снижение затрат на оплату труда персонала, снижает операционные риски и приносит огромную выгоду клиентам компаний, использующих инновационный концепт в производственном процессе. Предложенная концепция является революционной и окажет значительное влияние на рынок разработки программного обеспечения в будущем. Работа даст ответ на самый главный вопрос – как изменится подход к разработке программного обеспечения и почему человеческий труд в этой области очень быстро теряет актуальность.*

Ключевые слова: *искусственный интеллект в разработке программного обеспечения, информационные технологии, управление командой разработчиков, scrum, управление проектами, оптимизация бизнеса.*

Abstract: *this paper considers an evolutionary model of software development methods in various IT companies, new trends and changes in the software development market. For many years, the author of the article using his own company as a prototype studied existing and promising approaches to software development, approaches to managing development teams, and mechanisms for managing the efficiency of production activities for IT companies. The software product development concept obtained as a result of the study provides a multiple times increase in the efficiency of software engineers and product managers, reduces labor costs, significantly reduces operational risks, enhance the quality of the software development process. The proposed concept is revolutionary and will have a significant impact on the software development market in the future. The work will answer the most critical question - how will the approach to software development change and why human labor in this area very quickly becomes irrelevant.*

Keywords: *artificial intelligence for software development, information technologies, dev team management, scrum, project management, business optimization.*

Вот уже более восьми десятилетий человечество использует компьютеры в самых разных аспектах жизни – от первых научных проектов в 1930 годы до квантовых вычислений[1] и Интернета вещей в 2020м году. На протяжении всей истории существования компьютера были и инженеры со специальным образованием, которые программировали компьютерные блоки (да, только так и можно было назвать процесс задания логики в первые устройства), а с развитием технологий писали программы и целые платформы.

[1] Выставка CES 2020, стенд IBM, первое настоящее коммерческое применение нового квантового компьютера

Первые подходы к написанию программного кода были неэффективны и громоздки – ведь писались они, преимущественно используя машинные коды, бинарные значения и задавались в виде огромного количества переключателей. Практически до 1960 года инженерам приходилось использовать перфокарты, телетайпные ленты и табуляторы-перфораторы. С одной стороны, аппаратные возможности компьютеров тех времен не позволяли создавать хоть сколь-нибудь большие программы, но даже небольшие тексты исходных кодов приходилось выбивать, используя в качестве инструмента ввода перфокарты. Любая ошибка в этом процессе приводила к старту процесса практически с самого начала. Ни о какой эффективности не могло быть и речи. Лишь к 1980-му году появился первый относительно современный вид клавиатуры и вместе с этим гораздо более высокая скорость ввода данных.

Если говорить о языках программирования, то до 1950 года использовались бинарные значения и операторы, что приводило к медленному темпу разработки программ и сложной отладке работы. С 1950 года появился первый язык программирования Assembler[2], использующий низкоуровневые операнды для работы с данными и регистрами. Начиная с 1954 года стали появляться первые высокоуровневые языки программирования практически один за другим – FORTRAN, ALGOL, LISP, COBOL. Переход на высокоуровневые языки значительно ускорил разработку программных продуктов. К концу 1960-х годов в связи с ростом сложности программ и дальнейшим развитием программных средств возникла необходимость увеличить производительность труда разработчиков и это привело к разработке первых принципов структурного программирования. Методология была разработана Эдсгер Дейкстра[3] в 1968 году, когда он описал основные принципы структурного программирования. С развитием структурного программирования

[2] Язык ассемблера, машинно-ориентированный язык программирования низкого уровня. Его команды прямо соответствуют отдельным командам машины или их последовательностям.

[3] Эдсгер Дейкстра - голландский учёный, труды которого оказали влияние на развитие информатики и информационных технологий; один из разработчиков концепции структурного программирования, исследователь формальной верификации и распределённых вычислений. Тьюринговский лауреат (1972)

следующим достижением было внедрение в языки программирования процедур и функций. То есть, если есть алгоритм, который выполняется несколько раз, то его можно описать в виде функции или процедуры, а при необходимости выполнения ее можно вызывать из разных участков общего кода приложения. Программный код в данном случае становился меньше и это способствовало созданию модульных программных продуктов.

Развивая высокоуровневые языки программирования человечество подошло к рубежу 2020 года имея лишь слегка усовершенствованные инструменты разработки программных продуктов. Стоит отметить, что каждый новый виток прогресса развития как языков программирования, так и подхода к архитектуры программных продуктов приводил к эволюционному улучшению этих процессов, однако мы не увидели ни одной по настоящему революционной смены парадигмы за практически столетие развития.

Какими инструментами оперируют современные инженеры и разработчики программного обеспечения. В 2020 году, как и десятилетия назад человек все еще самостоятельно пишет программный код, ищет и исправляет ошибки в нем, работает с конечными заказчиком программного продукта и итеративно достигает результата. Практически каждый год появляются новые языки программирования, при этом старые отмирают. Например, компании Google и Microsoft в мае 2020 года объявили о намерении отказаться от использования языков программирования C и C++[1] из-за проблем с безопасностью кода и высокой стоимостью поддержи написанных на них программных продуктов, заменяя новыми более прогрессивными языками Go, Rust, SWIFT, Kotlin.

Для ускорения старта разработки программных продуктов в обиход разработчика прочно вошли такие понятия как шаблонизаторы приложений, бойлерплейты[4] и командные утилиты для создания преднастроенных прототипов приложений. Это позволило значительно сократить необходимое

[4] Бойлерплейт – в программировании «заготовка для проекта», содержащая в себе стандартный набор кода для запуска тестового приложения.

время на создание проекта с нуля с нужной архитектурой, сейчас в большинстве фреймворков это делается автоматически.

С появлением процедур и функций в языках программирования 1960х годов появился и термин «переиспользование программного кода». В самом начале в это понятие вкладывалась лишь возможность повторного вызова кусочка кода из разных мест программы. Однако этот подход значительно эволюционировал в огромные библиотеки подключаемых модулей, где термин «переиспользование программного кода» стал означать не только использование собственных разработок инженеров, трудящихся над проектом, но и полученные сторонние модули и библиотеки кода из специализированных хранилищ. Так появились репозитории готовых модулей NPM, GitHub Package Registry и многие другие.

С развитием языков программирования и методологий разработки, появлялись новые паттерны проектирования MVC, Observer, Singleton и другие. Совершенствовался подход к управлению командами разработки – от Waterfall[5] до Kanban[6] и scrum[7]. Не изменился лишь один ключевой фактор – программные продукты все еще разрабатываются человеком.

Основные проблемы разработки программных продуктов человеком. Если внимательно присмотреться с тому, что среднестатистический человек умеет делать на стабильно высоком уровне – мы скорее всего найдем лишь один пример – создавать новые инструменты и развивать их. В применении к разработке программного обеспечения человек крайне несовершенное существо, и вот почему.

[5] Waterfall (другие названия модель «Водопада», каскадная модель) - модель процесса разработки ПО, в которой процесс разработки выглядит как поток, последовательно проходящий фазы. Предложена У. У. Ройсом в 1970 году.

[6] Kanban - метод управления разработкой, реализующий принцип «точно в срок» и способствующий равномерному распределению нагрузки между работниками. При данном подходе весь процесс разработки прозрачен для всех членов команды. Задачи по мере поступления заносятся в отдельный список, откуда каждый разработчик может извлечь требуемую задачу.

[7] Scrum – итеративный подход к разработке программного обеспечения, впервые описанный Хиротака Такэути (Hirotaka Takeuchi) и Икудзиро Нонака(Ikujiro Nonaka) в статье The New Product Development Game (Harvard Business Review, январь-февраль 1986).

В нашей компании мы долгие 10 лет занимались разработкой программных продуктов для самых разных заказчиков из государственного и коммерческого сектора, за нашими плечами множество сложных и необычных проектов от высоконагруженных систем до биометрических систем с продвинутыми механизмами искусственного интеллекта. Накопив большое количество опыта и проведя ряд исследований, мы выделили наиболее острые проблемы процесса разработки любого программного продукта.

1. **Архитектурные ошибки**. Насколько не была бы опытная команда разработчиков и архитекторов, ошибки при разработке архитектуры программных продуктов были и остаются одной из самых «дорогих» ошибок для исправления. Неверно выбранная архитектура модулей, компонентов и механизмов взаимодействия неизбежно приводит к остановке разработки продукта и дорогостоящим изменениям.

2. **Человеческий фактор или человеческие ошибки**. Несмотря на уровень развития современных интегрированных сред разработки программного обеспечения (IDE), где среда пытается подсказать и исправить ошибки разработчика остается все еще огромное поле для совершения ошибок программистов – от использования неверных функций и процедур, до банальной невнимательности и нежелания уделить больше внимания продумыванию кода. Каждый кто хоть раз занимался написанием программного кода самостоятельно или управлял командами разработки знает, как часто встречаются такого рода ошибки и как много времени уходит на их исправление.

3. **Непрерывный рост технического долга**. В программировании термин «технический долг» используется, когда необходимо обозначить несовершенство написанного программного кода, требующего оптимизации и улучшения. Часто с ростом размера проекта технический долг начинает расти как снежный ком и команде разработки через каждые несколько месяцев требуется время чтобы переписать старый код, значительно улучшив его. Такая процедура называется рефакторингом и приводит к остановке разработки нового функционала иногда на месяцы.

4. **Стоимость изменений.** В процессе разработки программного продукта мир не стоит на месте и вместе с изменениями появляются новые бизнес требования к продукту. Для минимизации рисков, связанных с изменением бизнес требований были придуманы достаточно эффективные методологии управления проектами – Agile и scrum, организующие итеративный подход к разработке программного продукта, во многом спасающие при изменение бизнес требований на основе первых версий продукта, но не снижающие рисков внешних факторов (изменения рынка, действия конкурентов и др).

5. **Время вывода на рынок (Time to market).** С ускорением скорости жизни и инноваций, для большинства бизнесов запуск программных продуктов в кратчайшие сроки становится жизненно важной задачей. Даже применяя все современные методы управления проектами, с сильной и опытной командой разработки уходят месяцы на запуск первых версий продукта, что может быть непозволительно долго для некоторых типов заказчиков.

6. Стоимость разработки. Согласно аналитическому отчету Glassdoor[8], средняя годовая заработная плата инженера по разработке программного обеспечения в районе залива Сан-Франциско составляет $135 000, не считая бонусов и социального пакета. Зарплаты инженеров по разработке программных продуктов в странах СНГ составляют от $18 000 до $120 000 в год. С ростом спроса на разработку программных продуктов растет и спрос на лучших специалистов в востребованных областях, что в свою очередь вызывает повсеместный рост заработных плат и перегретость рынка человеческого капитала.

Каждая из приведенных выше проблем появилась не сегодня и существует десятилетиями. Помимо вышеприведенных факторов в специализированных областях существуют и другие проблемы, порой более значительные, но не

[8] Glassdoor.com – крупнейший веб сервис, где бывшие и текущие сотрудники компаний анонимно делятся размерами своих заработных плат и бонусных пакетов. https://www.glassdoor.com

вписывающиеся в основную массу проектов по разработке программного обеспечения.

Единственно верное решение – отказ от человеческого труда. Проанализировав все частые проблемы разработки программных продуктов, наша команда пришла к выводу, что основным узким местом является человек. Ведь именно ему требуется большая конкурентная заработная плата, у него сегодня может быть неудачный день или он просто не выспался, у него могут быть недостаточные знания и недостаточное количество опыта, ведь это он не смог продумать написанный им программный код хотя бы на 10 шагов вперед и теперь нужно будет переписать огромное количество кода, ведь это он неправильно понял требования к функционалу продукта и создал немного не то.

Все вышеперечисленно отлично иллюстрирует что единственным решением проблемы может быть только исключения человека из процесса разработки программного продукта и замена его на платформы, которые бы самостоятельно создавали программные продукты с помощью нейросетей и искусственного интеллекта и лучших практик.

Несмотря на то, что идея об исключении человека из процесса разработки программных продуктов не нова и попытки предпринимались множество раз. В целом подход получил название RAD или Rapid Application Development, где вместо написания всего кода использовалось блочное программирование и визуальный подход. К сожалению, расцвета эта концепция не получила и в конце 2000 годов появился новый подход, который был назван low-code. В основе подхода лежит максимальное снижения объема написанного кода за счет использования готовых модулей, интерфейсов, визуальных бизнес-процессов. Обычно для таких платформ выбирается один из языков высокого уровня – JavaScript или Java для написания кода и для приложений требуется какая-либо среда выполнения.

На протяжении последних двух лет наша команда занимается исследованием и развитием совершенно нового подхода в создании

программных продуктов, получившего название no-code. В основе этого подхода лежит принцип отсутствия необходимости написания программного кода, крайне низкие требования к уровню пользователей и максимальная автоматизация процесса с помощью ИИ и лучших практик. Программный продукт создается только на основе визуального интерфейса, визуального построения моделей данных и визуального редактора бизнес-процессов.

Формально, все что делает пользователь в подходе no-code это задает требования к будущему программному продукту, все остальные операции выполняются автоматически платформой и ИИ. Отдельно стоит отметить, что в результате тысяч тестов и множества прототипов мы пришли к подходу, когда после изменения пользователем любых требований (даже незначительных) самый простой и разумный способ – генерировать код приложения с нуля. Несмотря на противоестественность такого подхода с точки зрения человека, для компьютера создания приложения очередной раз с нуля обходится быстрее и проще, кроме того, архитектура продукта всегда остается оптимальной несмотря на постоянное внесение изменений.

Эффективность и применимость подхода

В рамках тестирования нового подхода, нашей команде удалось создать огромное количество программных продуктов, не написав не строчки кода. При сравнении эффективности написания приложения с помощью труда программистов и использования no-code платформы мы пришли к выводу, что применение нового подхода дает более чем 20-кратное сокращение времени на создание аналогичных программных продуктов. Скорость написания готового кода на тестовых испытаниях составляла от 4 до 12 тысяч строк кода в секунду и при среднем размере приложения занимала до 10 секунд.

Избавившись от необходимости использования человеческого труда мы получили огромное преимущество во многих областях и решение практически всех поставленных проблем.

1. **Качество программного продукта.** При написании кода с помощью ИИ, нам удалось добиться максимального соответствия

результирующего программного кода практикам используемого языка программирования и подходу к построению архитектуры. Архитектура больше не зависит от личного мнения и опыта конкретного разработчика или архитектора, она создается в момент получения технических требований от автора приложения на основании лучших подходов и натренированных нейронных сетей. Качество создаваемого программного продукта больше не зависит от уровня знаний и опыта разработчиков, участвующих в проекте.

2. **Мультипликативные эффект при улучшении алгоритмов генераторов.** При обнаружении какой-либо ошибки в генерируемом коде, исправления вносятся в алгоритм генерации приложений, а не в само приложение, в котором была обнаружена ошибка. Таким образом все сгенерированные в прошлом приложения могут быть автоматически пересобраны с учетом изменений в алгоритмах генерации. С помощью такого подхода достигается высокий уровень эволюционного развития алгоритмов создания исходного кода приложений и мультипликативный эффект при внедрении улучшений.

3. **Полное отсутствие технического долга и его проявлений.** Учитывая подход к генерации приложения каждый раз, когда происходят какие-либо изменения в требованиях к создаваемому приложению, весь программный код создается заново с учетом всех текущих требований к приложению. Таким образом, в любой момент времени кодовая база приложения оптимальна, не имеет лишнего или устаревшего кода, свободна от ошибок бизнес-логики.

4. **Сокращения сроков разработки программного продукта.** Использование no-code платформы позволяет значительно сократить общий срок разработки программного продукта более чем в 20 раз по сравнению традиционным подходом. Основное сокращение времени разработки приложения происходит за счет более высокой скорости написания программного кода, использования готовых элементов интерфейса создаваемого приложения, отсутствия затрат времени на рефакторинг (при отсутствии технического долга рефакторинг кода больше не требуется), автоматического

создания всех применимых (стандартных для текущих требований) бизнес-процессов обработки данных. Согласно статистике, полученной после создания 1877 приложений в рамках тестирования платформы, 98,3% затраченного времени приходится на создание и уточнение бизнес требований к приложению и построению моделей данных будущего приложения. Учитывая особенность подхода к разработке приложения, получений бизнес требований от пользователя не может считаться в полной мере затратами на разработку программного продукта и нашли отражений в полученной статистике.

5. **Минимизаций стоимости изменений.** Если проводить аналогию с традиционным подходом к разработке программного обеспечения человеком – практически любое изменений в требованиях к продукту приводят к необходимости проводить изменения в множестве алгоритмов, модулей и иногда структурах данных, которые в свою очередь тянут за собой еще большее количество изменений. Все это приводит к постоянно возрастающим затратам времени и ресурсов разработчиков, увеличивая технический долг за счет внесения изменений в продукт которые могут не соответствовать заданной архитектуре продукта. В другой стороны, использование no-code платформы позволяет исключить возникновение технического долга и неоптимальную архитектуру приложения за счет постоянно создания приложения «с нуля» каждый раз при внесении изменений. Внесение изменений в разрабатываемый продукт становится рутинной и быстрой операцией, позволяющей эволюционно улучшать конечный продукт.

6. **Снижение общей стоимости разработки программного продукта.** Учитывая тот факт, что при использовании no-code платформы исключается необходимость найма команды разработчиков и архитекторов, стоимость конечного продукта в основном состоит из стоимости пользования no-code платформой, которая является намного более низкой и хорошо прогнозируемой, в отличии человеческих ресурсов.

Как мы видим, описанные результаты перехода с использования человеческого труда на использование автоматизированных no-code платформы

позволяет значительно повысить эффективность процесса разработки программных продуктов, избавиться от накопившихся проблем и значительно повысить качество конечного продукта.

Заключение

Однозначно можно сказать, что будущее разработки программного обеспечения за автоматизированными инструментами создания кода, интеллектуальными системами создания бизнес-логики и визуальными интерфейсами. Согласно статистике многих аналитических компаний[2], в 2020 году начался бурный рост рынка low-code и no-code платформы. Переход на онлайн работу множества компаний по всему миру из-за продолжающейся пандемии COVID-19 заставляет акционеров и руководителей компаний все большие ресурсы вкладывать в разработку корпоративных сервисов и систем для организации удаленной работы сотрудников, решения производственных задачи и оптимизации численности персонала через автоматизацию различных процессов. С ростом спроса на разработку программного обеспечения растут и требования к скорости доставки программного обеспечения с момента разработки до внедрения. В конкуренции за покупателей, компании по разработке программного обеспечения будут все сильнее смотреть в сторону платформ для снижения себестоимости разработки и повышения привлекательности за счет более коротких сроков разработки с более высоким качеством конечного продукта.

В настоящий момент на рынке уже имеется пример, как один отдельный тип программного обеспечения теперь создается автоматизированными платформами. Эта область – создание веб сайтов и лэндинг-страниц, где для упрощения и удешевления процесса разработки огромное количество пользователей начало пользоваться двумя крупнейшими конструкторами WIX и Tilda[9] для создания веб страниц. Эти решения буквально переформатировали рынок разработки веб-сайтов и лэндинг-страниц за счет отсутствия

[9] WIX и Tilda - международные облачные платформы для создания и развития интернет-проектов, которая позволяет конструировать сайты и их мобильные версии на HTML5 с помощью инструментов drag-and-drop.

необходимости найма разработчиков программного обеспечений или самостоятельного написания программного кода.

По мере взросления no-code платформ для разработки программного обеспечения, все большее количество пользователей будут отказываться от классического прохода к разработке программного обеспечения с наймом команд разработчиков и архитекторов в пользу самостоятельного создания продуктов без необходимости написания кода. Массовый переход на использование таких платформ приведет к революционному изменению рынка программного обеспечения во всем мире и значительно ускорит дальнейшие процессы развития.

Библиографический список:

1. Forbes.com, Google Just Gave Millions Of Users A Reason To Quit Chrome, https://www.forbes.com/sites/gordonkelly/2020/05/29/google-chrome-critical-security-vulnerability-warning-firefox-update-chrome-browser/#73a845f81418
2. Gartner, Magic Quadrant for Enterprise Low Code Application Platforms, 8 August 2019, Paul Vincent, Kimihiko Iijima, Jason Wong, Mark Driver Yefim Natis