**СОДЕРЖАНИЕ**

[1 Введение 3](#_Toc101115935)

[2 RAD model – быстрая разработка приложений 5](#_Toc101115936)

[2.1 Что такое rad модель и история ее появления 5](#_Toc101115937)

[2.2 Назначение rad модели 7](#_Toc101115938)

[2.3 Применение rad модели 8](#_Toc101115939)

[2.4 Основные принципы rad модели 9](#_Toc101115940)

[2.5 Преимущества и недостатки rad модели. Сравнение с другими видами подходов к разработке программных продуктов 10](#_Toc101115941)

[3 Agile model – гибкая методология разработки 12](#_Toc101115942)

[3.1 Что такое гибкая методология разработки и история ее появления 12](#_Toc101115943)

[3.2 Основные идеи agile-методов 13](#_Toc101115944)

[3.3 Основополагающие принципы agile manifesto 14](#_Toc101115945)

[3.4 Методологии в agile manifesto 15](#_Toc101115946)

[3.5 Преимущества и недостатки agile 17](#_Toc101115947)

[4 Заключение 19](#_Toc101115948)

[5 Список информационных источников 21](#_Toc101115949)

# 1 ВВЕДЕНИЕ

Исследуемая тема – подходы разработки программных продуктов.

Все проекты разные. Не существует идеальной системы управления проектами, подходящей для каждого из видов проектов. Также не существуют системы, которая бы подходила каждому руководителю и была удобна для всех членов команды. Однако за время существования проектного управления было создано не мало эффективных подходов, методик и стандартов, которые можно взять на вооружение.

Есть много подходов, таких как:

– Общие подходы

– «Waterfall Model» (каскадная модель или «водопад»)

– «V-Model»

– «Incremental Model» (инкрементная модель)

– «RAD Model» (rapid application development model или быстрая разработка приложений)

– «Agile Model» (гибкая методология разработки)

– «Iterative Model» (итеративная или итерационная модель)

– «Spiral Model» (спиральная модель)

Целью данной работы является изучение быстрой разработки приложений «RAD Model» и гибкой методологии разработки «Agile Model».

Задачи:

1. Изучить историю создания каждого из двух рассматриваемых в работе методов
2. Определить назначение каждого из двух рассматриваемых в работе методов
3. Выявить применение каждого из двух рассматриваемых в работе методов
4. Рассмотреть основные принципы каждого из двух рассматриваемых в работе методов
5. Определить преимущества и недостатки каждого из двух рассматриваемых в работе методов
6. Высказать свое мнение и подвести итоги по изученной теме

# 2 RAD MODEL – БЫСТРАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ

## **2.1 Что такое RAD модель и история ее появления**

**RAD** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) rapid application development — быстрая разработка приложений) — концепция организации технологического процесса разработки [программных продуктов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82), ориентированная на максимально быстрое получение результата в условиях сильных ограничений по срокам и бюджету и нечётко определённых требований к продукту. Эффект ускорения разработки достигается путём использования соответствующих технических средств и непрерывного, параллельного с ходом разработки, уточнения требований и оценки текущих результатов с привлечением заказчика.

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.

RAD создана в конце 1980-х как альтернатива более ранним [каскадной](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%81%D0%BA%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) и [итеративной](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0) моделям. С конца [XX века](https://ru.wikipedia.org/wiki/XX_%D0%B2%D0%B5%D0%BA) RAD получила широкое распространение.

Тот же термин используется в отношении программных инструментов быстрого [прототипирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и разработки ПО. Типичными качествами таких инструментов являются максимальная автоматизация рутинных операций и широкое использование [визуального программирования](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Создание концепции RAD явилось следствием сочетания ряда факторов, таких как:

• Неудовлетворённость методами разработки программ 1970-х и начала 1980-х годов, таких как «модель водопада» (англ. Waterfall model). Разработанные по аналогии с методами проектирования «железных» технических систем, они предусматривают поэтапное формирование требований, документирование, проектирование и реализацию. Их последовательное применение приводит к настолько медленному процессу создания программы, что зачастую даже базовые требования к программе успевают измениться до окончания разработки.

• Осознание факта отличия разработки программного продукта от традиционных видов инженерной деятельности, состоящее в гораздо большей доступности предмета разработки для сложных неоднократных модификаций.

• Появление аппаратных мощностей и программных средств разработки, позволяющих быстро создавать работоспособные прототипы программных систем и их частей и автоматизировать их документирование. При использовании таких средств альтернативой подробному проектированию и тщательному предварительному документированию системы становится быстрое создание или модификация прототипа и его непосредственная оценка заказчиком.

• Успешный опыт разработки систем, основанной на работе небольшой по численности сплочённой команды разработчиков, непосредственно взаимодействующих между собой и с представителем заказчика.



Рисунок 1 – Джеймс Мартин (1933-2013)

Основателем RAD считается сотрудник [IBM](https://ru.wikipedia.org/wiki/IBM) Джеймс Мартин (рис.1), который в [1980-х](https://ru.wikipedia.org/wiki/1980-%D0%B5) годах сформулировал основные принципы RAD, основываясь на идеях Барри Бойма и Скотта Шульца. А в [1991 году](https://ru.wikipedia.org/wiki/1991_%D0%B3%D0%BE%D0%B4) Мартин опубликовал известную книгу, в которой детально изложил концепцию RAD и возможности её применения. В настоящее время RAD становится общепринятой схемой для создания средств разработки [программных продуктов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82).

## **2.2 Назначение RAD модели**

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков (рис.2) за срок порядка трёх-четырёх месяцев путём использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки. Технология RAD предусматривает активное привлечение заказчика уже на ранних стадиях — обследование организации, выработка требований к системе. Последнее из указанных свойств подразумевает полное выполнение требований заказчика как функциональных, так и нефункциональных, с учётом их возможных изменений в период разработки системы, а также получение качественной документации, обеспечивающей удобство эксплуатации и сопровождения системы. Это означает, что дополнительные затраты на сопровождение сразу после поставки будут значительно меньше. Таким образом, полное время от начала разработки до получения приемлемого продукта при использовании этого метода значительно сокращается.



Рисунок 2 – Небольшая команда разработчиков

## **2.3 Применение RAD модели**

RAD-технология не является универсальной, её целесообразно применять лишь если проект отвечает всем или некоторым из условий:

1. Сжатые сроки.

Требуется максимально быстро создать систему, отвечающую требованиям сегодняшнего дня. Увеличение сроков создаёт высокую вероятность настолько значительного изменения фундаментальных положений, регламентирующих автоматизируемую деятельность, что система морально устареет ещё до завершения проектирования.

1. Нечётко определённые и/или изменяющиеся по ходу разработки требования.

Заказчик весьма приблизительно представляет себе работу будущего программного продукта и не может четко сформулировать все требования к ПО. Требования могут быть вообще не определены к началу проекта либо могут изменяться по ходу его выполнения.

1. Ограниченный бюджет при готовности участия заказчика в разработке.

У заказчика нет средств на оплату работы большой команды проектировщиков и разработчиков в течение длительного времени, но имеется готовность выделить специалистов для постоянного непосредственного участия в разработке и оценки её текущего состояния.

1. Небольшие объёмы либо возможность разбиения проекта на функциональные компоненты.

Если предполагаемая система велика, необходимо, чтобы её можно было разбить на небольшие части, каждая из которых обладает четкой функциональностью и минимально зависит от других. Они могут выпускаться последовательно или параллельно (в последнем случае привлекается несколько RAD-групп).

1. Графический интерфейс пользователя.

Важнейший или один из важнейших компонентов системы. Именно в создании интерфейса RAD-технология даёт наибольшие преимущества, так как интерфейс демонстрируется непосредственно на прототипе, причём достаточно скоро после начала проекта. Возможно даже прямо привлечь представителя заказчика к проектированию интерфейса в визуальном редакторе. Этот подход позволяет избежать типичной ситуации, когда интерфейс, описанный пользователем в требованиях (как правило, без учёта технологических ограничений) ведёт себя на практике совсем не так, как рассчитывал пользователь, хотя формально система полностью соответствует документированным требованиям.

1. Низкая вычислительная сложность.

Обработка данных в проекте сводится к комбинированию типовых операций, все или большинство из которых уже реализованы в виде доступных библиотек. Оригинальных алгоритмов обработки данных либо вообще не требуется, либо они достаточно просты и могут быть реализованы быстро и без особых затруднений.

## **2.4 Основные принципы RAD модели**

Принципы RAD технологии направлены на обеспечение трёх основных её преимуществ:

1. Высокой скорости разработки
2. Низкой стоимости
3. Высокого качества

Достигнуть высокого качества программного продукта весьма непросто и одна из главных причин возникающих трудностей заключается в том, что разработчик и заказчик видят предмет разработки (ПО) по-разному:

• Инструментарий должен быть нацелен на минимизацию времени разработки.

• Создание прототипа для уточнения требований заказчика.

• Цикличность разработки: каждая новая версия продукта основывается на оценке результата работы предыдущей версии заказчиком.

• Минимизация времени разработки версии, за счёт переноса уже готовых модулей и добавления функциональности в новую версию.

• Команда разработчиков должна тесно сотрудничать, каждый участник должен быть готов выполнять несколько обязанностей.

• Управление проектом должно минимизировать длительность цикла разработки.

Принципы RAD применяются не только при реализации, но и распространяются на все этапы жизненного цикла, в частности на этап обследования организации, построения требований, анализ и дизайн.

Модель быстрой разработки приложений включает следующие фазы:

– Бизнес-моделирование: определение списка информационных потоков между различными подразделениями. На этом этапе пользователи, менеджеры и IT-специалисты обсуждают задачи проекта, его объём, системные требования, а также сложности, которые могут возникнуть при разработке. Фаза завершается согласованием ключевых моментов с RAD-группой и получением от руководителей проекта разрешения на продолжение.

– Моделирование данных: информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.

– Моделирование процесса: информационные потоки связывают объекты для достижения целей разработки.

– Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.

– Тестирование: тестируются новые компоненты и интерфейсы.

## **2.5 Преимущества и недостатки RAD модели. Сравнение с другими видами подходов к разработке программных продуктов**

Технология быстрой разработки приложений (RAD) позволяет обеспечить:

* быстроту продвижения программного продукта на рынок;
* интерфейс, устраивающий пользователя;
* лёгкую адаптируемость проекта к изменяющимся требованиям;
* простоту развития функциональности системы.

Является противоположностью жесткого водопадного метода, она не берет в расчет требования и документацию проекта. Для названия такого подхода иногда используют термин «совместная разработка прикладных систем» (свидетельствующий об участии потребителя в процессе разработки), или быстрая разработка прикладных систем (RAD –Rapid Application Development).

Если требования к системе чётко определены и не могут меняться, вовлечение заказчика в процесс разработки не потребуется и более эффективной может быть традиционная иерархическая разработка (каскадный метод). Также RAD не даёт практически никаких преимуществ в проектах, основная сложность которых определяется необходимостью реализации сложных, нестандартных алгоритмов обработки данных, а интерфейс пользователя либо отсутствует как таковой, либо очень прост и совершенно стандартен.

# 3 AGILE MODEL – ГИБКАЯ МЕТОДОЛОГИЯ РАЗРАБОТКИ

## **3.1 Что такое гибкая методология разработки и история ее появления**

Гибкая методология разработки (англ. agile software development, agile-разработка) — обобщающий термин для целого ряда подходов и практик, основанных на ценностях Манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе. Применяется как эффективная практика организации труда небольших групп (которые делают однородную творческую работу) в объединении с управлением ими комбинированным (либеральным и демократическим) методом.

Большинство гибких методологий нацелены на минимизацию рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели. Каждая итерация сама по себе выглядит как программный проект в миниатюре и включает все задачи, необходимые для выдачи мини-прироста по функциональности: планирование, анализ требований, проектирование, программирование, тестирование и документирование. Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации команда выполняет переоценку приоритетов разработки.

Agile-методы делают упор на непосредственном общении лицом к лицу. Большинство agile-команд расположены в одном офисе. Основной метрикой agile-методов является рабочий продукт. Отдавая предпочтение непосредственному общению, agile-методы уменьшают объём письменной документации по сравнению с другими методами.

В течение 1990-х годов ряд легких методов разработки программного обеспечения развивался в ответ на преобладающие тяжелые методы, которые критики называли чрезмерно регулируемыми, планируемыми и микроуправляемыми. К ним относятся: быстрая разработка приложений (RAD) с 1991 года; унифицированный процесс и метод разработки динамических систем с 1994 года; Scrum, с 1995 года; Crystal Clear и экстремальное программирование (XP), как с 1996 года; и функционально-ориентированная разработка, начиная с 1997 года. Хотя все они возникли до публикации Манифеста гибкой методологии разработки программного обеспечения, теперь они все вместе называются гибкими методами разработки программного обеспечения.

В феврале 2001 года в штате Юта США был выпущен «Манифест гибкой разработки программного обеспечения». Он являлся альтернативой управляемым документацией «тяжеловесным» практикам разработки программного обеспечения, таким как «метод водопада», являвшимся золотым стандартом разработки в то время. Данный манифест был одобрен и подписан представителями методологий: экстремального программирования, Crystal Clear[en], DSDM, Feature driven development, Scrum, Adaptive software development, Pragmatic Programming. Гибкая методология разработки использовалась многими компаниями и до принятия манифеста, однако вхождение Agile-разработки в массы произошло именно после этого события.

Среди этой группы (называвшей себя «Agile Alliance») авторов, подписавших манифест, были Майк Биддл, Кент Бек, Алистер Кокбёрн, Уорд Каннингем, Мартин Фаулер, Джим Хайсмит, Энди Хант, Рон Джеффрис, Роберт Мартин, Стивен Дж. Меллор, Кен Швабер, Джефф Сазерленд, Дейв Томас.

## **3.2 Основные идеи Agile-методов**

Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto. Agile не включает практики, а определяет ценности и принципы, которыми руководствуются команды.

Agile Manifesto содержит 4 основные идеи и 12 принципов. Примечательно, что Agile Manifesto не содержит практических советов.

Основные идеи:

• люди и взаимодействие важнее процессов и инструментов;

• работающий продукт важнее исчерпывающей документации;

• сотрудничество с заказчиком важнее согласования условий контракта;

• готовность к изменениям важнее следования первоначальному плану.

## **3.3 Основополагающие принципы Agile Manifesto**

• наивысшим приоритетом признается удовлетворение заказчика за счёт ранней и бесперебойной поставки ценного программного обеспечения;

• изменение требований приветствуется даже в конце разработки (это может повысить конкурентоспособность полученного продукта);

• частая поставка работающего программного обеспечения (каждые пару недель или пару месяцев с предпочтением меньшего периода);

• общение представителей бизнеса с разработчиками должно быть ежедневным на протяжении всего проекта;

• проекты следует строить вокруг заинтересованных людей, которых следует обеспечить нужными условиями работы, поддержкой и доверием;

• самый эффективный метод обмена информацией в команде — личная встреча;

• работающее программное обеспечение — лучший измеритель прогресса;

• спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;

• постоянное внимание к техническому совершенству и хорошему проектированию увеличивают гибкость;

• простота как искусство не делать лишней работы очень важна;

• лучшие требования, архитектура и проектные решения получаются у самоорганизующихся команд;

• команда регулярно обдумывает способы повышения своей эффективности и соответственно корректирует рабочий процесс.

## **3.4 Методологии в Agile Manifesto**

Существуют методологии, которые придерживаются ценностей и принципов, заявленных в Agile Manifesto. Ниже представлены некоторые из них:

• Agile Modeling — набор понятий, принципов и приёмов (практик), позволяющих быстро и просто выполнять моделирование и документирование в проектах разработки программного обеспечения. Не включает в себя детальную инструкцию по проектированию, не содержит описаний, как строить диаграммы на UML. Основная цель: эффективное моделирование и документирование; но не охватывает программирование и тестирование, не включает вопросы управления проектом, развёртывания и сопровождения системы. Однако включает в себя проверку модели кодом.

• Agile Unified Process (AUP) упрощенная версия IBM Rational Unified Process (RUP), разработанная Скоттом Амблером, которая описывает простое и понятное приближение (модель) для создания программного обеспечения для бизнес-приложений.

• Agile Data Method — группа итеративных методов разработки программного обеспечения, в которых требования и решения достигаются в рамках сотрудничества разных кросс-функциональных команд.

• DSDM основан на концепции быстрой разработки приложений (RAD). Представляет собой итеративный и инкрементный подход, который придаёт особое значение продолжительному участию в процессе пользователя/потребителя.

• Экстремальное программирование (Extreme programming, XP).

• Feature driven development (FDD) — функционально-ориентированная разработка. Используемое в FDD понятие функции или свойства системы достаточно близко к понятию прецедента использования, используемому в RUP, существенное отличие — это дополнительное ограничение: «каждая функция должна допускать реализацию не более, чем за две недели». То есть если сценарий использования достаточно мал, его можно считать функцией. Если же велик, то его надо разбить на несколько относительно независимых функций.

• Getting Real — итеративный подход без функциональных спецификаций, использующийся для веб-приложений. В данном методе сперва разрабатывается интерфейс программы, а потом её функциональная часть.

• OpenUP — это итеративно-инкрементальный метод разработки программного обеспечения. Позиционируется как лёгкий и гибкий вариант RUP. OpenUP делит жизненный цикл проекта на четыре фазы: начальная фаза, фазы уточнения, конструирования и передачи. Жизненный цикл проекта обеспечивает предоставление заинтересованным лицам и членам коллектива точек ознакомления и принятия решений на протяжении всего проекта. Это позволяет эффективно контролировать ситуацию и вовремя принимать решения о приемлемости результатов. План проекта определяет жизненный цикл, а конечным результатом является окончательное приложение.

• Scrum устанавливает правила управления процессом разработки и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения. Использование этой методологии дает возможность выявлять и устранять отклонения от желаемого результата на более ранних этапах разработки программного продукта.

• Бережливая разработка программного обеспечения (lean software development) использует подходы из концепции бережливого производства.

Важно помнить, что Agile — это общая концепция. Где-то под циклом понимается путь от анализа и планирования до выпуска, где-то это просто этап разработки, связанный с определённым уровнем сложности. Но везде есть 4 стадии:

1. Планирование. Ограниченный по времени этап из-за большого числа итераций и постоянно меняющихся требований.

2. Обратная связь. Она может поступать от клиентов, программистов, тестировщиков и самих разработчиков — любых заинтересованных лиц.

3. Реализация. Собственно сама разработка кода и графики.

4. Тестирование. Автоматическое или ручное выявление ошибок и несоответствий требованиям заказчика.

Описанная концепция универсальна для всех гибких методологий, которые также имеют общие плюсы и минусы.

## **3.5 Преимущества и недостатки Agile**

Из плюсов можно отметить:

1. Максимальная удовлетворённость клиента. У него есть постоянный контакт с разработчиками, его пожелания быстро находят отклик в проекте.

2. Благоприятная атмосфера в коллективе. Даже в случае разногласий эффективнее все решать словами.

3. Высокие цели. Постоянный фидбек предполагает внимание к мелочам, эргономике, соответствию трендам.

4. Безопасность. Множество итераций исключает существование ошибок, допущенных в самом начале.

Отрицательные моменты:

1. Слабое внимание к документации. В случае возникновения претензий и конфликтных ситуаций аргументов в свою защиту у исполнителя просто не будет.

2. Проблемы с реализацией комплексных продуктов. Если в самом начале разработки были заложены ограничения — на поздних этапах исправлять их будет сложно.

3. Вероятность отказа клиентов в процессе разработки. Если у заказчика нет четкого понимания, что он хочет видеть на выходе — выполнять его требования очень сложно.

4. Влияние иерархии. Руководители принимают все решения, так как все процессы цикла реализуются параллельно. Важность рядовых программистов минимальна.

Agile-модель будет очень полезна начинающим компаниями и стартапам, когда необходимо привлекать клиентов и завоевывать авторитет. В крупных компаниях Agile в чистом виде применяется редко, лишь для отдельных проектов или отделов.

# 4 ЗАКЛЮЧЕНИЕ

У человечества за всю историю накопился внушительный список успешно реализованных сложных проектов. Самые смелые человеческие начинания требовали слаженной работы большого количества людей. А это подразумевает сложную систему управления проектами. Говоря простыми словами, управление проектами – это управление и организация всего, что нужно для достижения цели. Проектное управление позволяет добиться успеха.

Изучив RAD модель и Agile-методы, можно сделать выводы, что для каждого проекта нужен свой подход. Кому-то нужна скорость, кому-то нужна простота, кому-то нужен непосредственный контакт «заказчик-разработчик», из-за чего нет общего подхода к решению поставленной задачи. И именно поэтому каждый подход имеет право на существование, ведь в каждом варианте есть свои плюсы и минусы.

Так, при agile-подходе часто пренебрегают созданием плана («дорожной карты») развития продукта, равно как и управлением требованиями, в процессе которого и формируется такая «карта». Гибкий подход к управлению требованиями не подразумевает далеко идущих планов (по сути, управления требованиями просто не существует в данной методологии), а подразумевает возможность заказчика вдруг и неожиданно в конце каждой итерации выставлять новые требования, часто противоречащие архитектуре уже созданного и поставляемого продукта. Такое иногда приводит к катастрофическим «авралам» с массовым рефакторингом и переделками практически на каждой очередной итерации.

Кроме того, считается, что работа в agile мотивирует разработчиков решать все поступившие задачи простейшим и быстрейшим возможным способом, при этом зачастую не обращая внимания на правильность кода с точки зрения требований нижележащей платформы (подход «работает — и ладно»), при этом не учитывается, что код может перестать работать при дальнейшем изменении. Это приводит к снижению качества продукта и накоплению дефектов.

В свою очередь, RAD не даёт практически никаких преимуществ в проектах, основная сложность которых определяется необходимостью реализации сложных, нестандартных алгоритмов обработки данных, а интерфейс пользователя либо отсутствует как таковой, либо очень прост и совершенно стандартен.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Материал с сайта Serviceproekt.ru. Гибкая методология разработки (Agile). Как и когда применять этот метод. Главные составляющие agile организации. <https://serviceproekt.ru/finance/gibkaya-metodologiya-razrabotki-agile-agile-kak-i-kogda/> (Дата обращения: 20.02.2022)
2. Материал из Википедии – свободной энциклопедии. RAD (программирование). <https://ru.wikipedia.org/wiki/RAD_(программирование)> (Дата обращения: 20.02.2022)
3. Материал из Википедии – свободной энциклопедии. Быстрая разработка приложений. <https://en.wikipedia.org/wiki/Rapid_application_development> (Дата обращения: 20.02.2022)
4. Материал из Википедии – свободной энциклопедии. Гибкая методология разработки. <https://ru.wikipedia.org/wiki/Гибкая_методология_разработки> (Дата обращения: 20.02.2022)
5. Материал из Википедии – свободной энциклопедии. Agile software development. <https://en.wikipedia.org/wiki/Agile_software_development> (Дата обращения: 20.02.2022)
6. Материал с сайта koreajob.ru. Agile примеры. Гибкая методология разработки (Agile). Ключевые моменты в применении Agile. <https://koreajob.ru/agile-primery-gibkaya-metodologiya-razrabotki-agile-klyuchevye/> (Дата обращения: 20.02.2022)
7. Эндрю Стеллман, Дженнифер Грин. Постигая Agile. O’Reilly 2015. <https://www.mann-ivanov-ferber.ru/books/postigaya-agile/> (Дата обращения: 20.02.2022)