|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

***\_\_\_\_*** **Анализ достоинств и недостатков итерационной модели Agile *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_ИУ5-34М\_\_\_\_\_\_ **\_\_\_\_\_ \_\_М.С. Шевчук\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_** **\_\_\_\_\_****Ю.Е. Гапанюк\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2023 г.***Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_ИУ5\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_В.И. Терехов\_\_

(И.О.Фамилия)

«\_04\_» \_\_\_\_сентября\_\_\_\_\_\_\_ 2023 г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме \_\_\_\_\_\_ Анализ достоинств и недостатков итерационной модели Agile \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_ИУ5-34М\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шевчук Михаил Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИССЛЕДОВАТЕЛЬСКАЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_КАФЕДРА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_\_ нед., 50% к \_\_\_\_ нед., 75% к \_\_\_ нед., 100% к \_\_\_\_ нед.

***Техническое задание*** \_\_\_\_\_\_\_\_\_провести анализ и сравненить опытным путем методы апскейлинга изображения в играх\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на 25

листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_04\_» \_\_\_\_сентября\_\_\_\_\_\_\_ 2023 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_ \_\_\_\_\_\_Ю.Е. Гапанюк\_\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_**М.С. Шевчук**\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

[Введение 3](#_Toc153201770)

[Понятие итерационной модели 4](#_Toc153201771)

[Разработка в итерационном конструировании 9](#_Toc153201772)

[Примеры итерационного проектирования моделей 12](#_Toc153201773)

[Преимущества 16](#_Toc153201774)

[Недостатки 19](#_Toc153201775)

[Вывод 23](#_Toc153201776)

[Список источников 24](#_Toc153201777)

## Введение

SDLC (жизненный цикл разработки программного обеспечения) является особенно огромным и изобилующим многочисленными действиями по тестированию и разработке, методами, методологиями, инструментами и другими. Он включает в себя интенсивное изложение и администрирование, расчет и расположение. Это просто следит за всеми определенными усилиями разработчиков программного обеспечения, что приложение или программное обеспечение благоприятно создано. Итерационная модель также является компонентом SDLC.

Это конкретное выполнение жизненного цикла разработки программного обеспечения, которое концентрируется на первичном, несложном выполнении, которое затем приводит к увеличению сложности и расширению характеристик для конечной системы. Вкратце, разработка в итерационной модели — это способ разбить разработку программного обеспечения огромного приложения на более короткие разделы.

## Понятие итерационной модели

Понятие жизненного цикла программного обеспечения появилось, когда программистское сообщество осознало необходимость перехода от кустарных ремесленнических методов разработки программ к технологичному промышленному их производству. Как обычно происходит в подобных ситуациях, программисты попытались перенести опыт других индустриальных производств в свою сферу. В частности, было заимствовано понятие жизненного цикла.

Аналогия жизненного цикла программного обеспечения с техническими системами имеет более глубокие корни, чем это может показаться на первый взгляд. Программы не подвержены физическому износу, но в ходе их эксплуатации обнаруживаются ошибки (неисправности), требующие исправления. Ошибки возникают также от изменения условий использования программы. Последнее же является принципиальным свойством программного обеспечения, иначе оно теряет свой смысл. Поэтому правомерно говорить о старении программ , хотя не о физическом старении, а о моральном.

Необходимость внесения изменений в действующие программы как из-за обнаруживаемых ошибок, так и по причине развития требований приводит по сути дела к тому, что разработка программного обеспечения продолжается после передачи его пользователю и в течение всего времени жизни программ. Деятельность, связанная с решением довольно многочисленных задач такой продолжающейся разработки, получила название сопровождения программного обеспечения (рисунок 1).

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Рисунок 1 – Разработка, использование и сопровождение ПО

Исторически развитие концепций жизненного цикла связано с поиском для него адекватных моделей. Как и всякая другая, модель жизненного цикла является абстракцией реального процесса, в которой опущены детали, несущественные с точки зрения назначения модели. Различие назначений применения моделей определяет их разнообразие.

Основные причины, из-за которых нужно изучать вопросы моделирования жизненного цикла программного обеспечения, можно сформулировать следующим образом.

Во-первых, это знание даже для непрофессионального программиста помогает понять, на что можно рассчитывать при заказе или приобретении программного обеспечения и что нереально требовать от него. В частности, неудобные моменты работы с программой, ее ошибки и недоработки обычно устраняются в ходе продолжающейся разработки, и есть основания ожидать, что последующие версии будут лучше. Однако кардинальные изменения концепций программы — задача другого проекта, который совсем необязательно будет во всех отношениях лучше данной системы.

Во-вторых, модели жизненного цикла — основа знания технологий программирования и инструментария, поддерживающего их. Программист всегда применяет в своей работе инструменты, но квалифицированный программист знает, где, когда и как их применять. Именно в этом помогают понятия моделирования жизненного цикла: любая технология базируется на определенных представлениях о жизненном цикле, выстраивает свои методы и инструменты вокруг фаз и этапов жизненного цикла.

В-третьих, общие знания того, как развивается программный проект, дают наиболее надежные ориентиры для его планирования, позволяют экономнее расходовать ресурсы, добиваться более высокого качества управления. Все это относится к сфере профессиональных обязанностей руководителя программного проекта.

В настоящей работе модели жизненного цикла представлены в таком виде, позволяющем рассматривать их, абстрагируясь от специфики разработки конкретных программных систем. Описываются традиционные модели и их развитие, приспособленное к потребностям объектно-ориентированного проектирования.

Модель жизненного цикла итерационной модели, которая не начиналась с целых условий. В частности, в модели разработка начинается с определения и выполнения единственного компонента программного обеспечения, который анализируется в соответствии с последующими спецификациями. Кроме того, в модели итеративного процесса итеративный процесс начинается с упрощенного выполнения небольшого набора программного реквизита, который итеративно улучшает развивающиеся варианты, пока вся система не будет выполнена и подготовлена для перераспределения. Каждый выпуск Итеративной модели разрабатывается в определенный и установленный период времени, известный как итерация.

Кроме того, итерация концентрируется на конкретной совокупности спецификаций. Каждый цикл заканчивается системой, которая используется, например, для конкретного результата итерации в реализуемом выпуске. Итерационная модель позволяет получить предыдущие этапы, на которых вносятся соответствующие изменения. Конечный результат продукта восстанавливается по окончании SDLC жизненного цикла разработки программного обеспечения.

Именно поэтому полученную модель (рисунок 2) называют итерационной. Впервые применённая в США ещё в 1957 году, многопроходная итерационная модель основана на концепции IID (Iterative and Incremental Development), включающей принципы итеративности (уточнение и детализация разрабатываемого программного обеспечения шаг за шагом), инкрементальности (увеличение функциональности ПО для каждой итерации) и эволюционности (максимальное использование наработок предыдущих итераций для последующих).

Изображение выглядит как диаграмма, Красочность, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 2 – Переход от каскадной к итерационной модели

Общепринятая модель жизненного цикла является идеальной, так как только очень простые задачи проходят все этапы без каких-либо итераций  — возвратов на предыдущие шаги технологического процесса. При программировании, например, может обнаружиться, что реализация некоторой функции очень громоздка, неэффективна и вступает в противоречие с требуемой от системы производительностью. В этом случае требуется перепроектирование, а может быть, и переделка спецификаций. При разработке больших нетрадиционных систем необходимость в итерациях возникает регулярно на любом этапе жизненного цикла как из-за допущенных на предыдущих шагах ошибок и неточностей, так и из-за изменений внешних требований к условиям эксплуатации системы.

Таковы мотивы классической итерационной модели жизненного цикла (рисунок 3).

Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание

Рисунок 3 – Классическая итерационная модель

Стрелки, ведущие вверх, обозначают возвраты к предыдущим этапам, квалифицируемые как требование повторить этап для исправления обнаруженной ошибки. В этой связи может показаться странным переход от этапа « Эксплуатация и сопровождение» к этапу «Тестирование и отладка». Дело в том, что рекламации, предъявляемые в ходе эксплуатации системы, часто даются в такой форме, которая нуждается в их перепроверке. Чтобы понять, о каких ошибках идет речь в рекламации, разработчикам полезно предварительно воспроизвести пользовательскую ситуацию у себя, т. е. выполнить действия, которые обычно относят к тестированию.

Классическая итерационная модель абсолютизирует возможность возвратов на предыдущие этапы. Однако это обстоятельство отражает существенный непреодолимый аспект программных разработок, не опирающихся на объектно-ориенти­ро­ван­ное проектирование: стремление заранее предвидеть все ситуации использования системы и невозможность в подавляющем большинстве случаев достичь этого. Все традиционные технологии программирования направлены лишь на то, чтобы минимизировать возвраты. Но суть от этого не меняется: при возврате всегда приходится повторять построение того, что уже считалось готовым.

Иное положение с объектно-ориентированными технологиями. Отказ от завершенности фаз и этапов, вместо чего предлагается распределять наращивание функциональности и интерфейсных возможностей по итерациям, позволяет ослабить требование переделки старого при возвратах. По существу, классическая схема остается верной, но только в рамках одной итерации и с одной важной поправкой: все полезное, что было сделано ранее, сохраняется. Понятно, что для программной системы в целом новый подход требует и новых моделей жизненного цикла, отражающих его особенности, отмеченные ранее. Об этом будет идти речь после изучения основных вариантов традиционных моделей жизненного цикла.

## Разработка в итерационном конструировании

Разработка ПО согласно концепции IDD сводится к разбиению этапа реализации на серию быстрых, лёгких и адаптивных итераций, оперативно приносящих результаты. Каждая итерация основана на PDCA-цикле Деминга (Plan-Do-Check-Act) и завершается демонстрацией потребителю полученного промежуточного продукта с целью скорейшего выявления потенциальных ошибок. Более того, в ходе выполнения итераций представление о конечном продукте изменяется, поэтому добавляются новые функциональные возможности. Продолжительность каждой итерации варьируется в пределах 1–6 недель, а начальный список требований к ПО вообще может отсутствовать.

Обычно разработка в итерационной модели используется в связи с накопительной разработкой, в которой расширенный цикл разработки программного обеспечения делится на более короткие секции, которые создаются друг на друге. Отныне итерационная модель используется в следующих схемах:

* Характеристики всей системы могут быть определены и поняты.
* Существенные спецификации определены, в то время как некоторые процедуры и необходимые улучшения возникают с процедурой процесса разработки.
* Новейшая технология должна использоваться и постоянно приобретаться командой разработчиков, пока она служит проекту.
* Это может быть изменено в будущем при условии, что существует мало характеристик и целей огромного риска. Источники с необходимыми способностями, которые недоступны и могут быть использованы на основе контракта для конкретных итераций.

Процедура построения итерационной модели является циклической, не похожа на обычные модели, которые сосредоточены на тщательной и поэтапной процедуре разработки. Во всей этой процедуре завершается только первичное планирование, повторяется меньшее количество этапов, постепенно усложняя каждый цикл и увеличивая итерации программного обеспечения.

Рассмотрим проект имплементации КИС согласно предлагаемой модели:

* идентификация и анализ требований, предъявляемых к КИС, а также приоритезация найденных требований (опционально);
* определение числа и продолжительности итераций разработки КИС, в случае наличия приоритезированных требований их распределение по итерациям разработки;
* доработка и кастомизация КИС, функциональное и интеграционное тестирование с последующей демонстрацией полученного продукта заказчику для уточнения требований (для всех итераций);
* проведение приёмочного тестирования;
* документирование реализованного программного решения;
* переход к продуктивной эксплуатации и поддержка.

Выполнение итераций подразумевает демонстрацию продукта заказчику, в результате чего выявляются дефекты и идентифицируются новые требования к ПО. Первый минус многопроходной модели формулируется достаточно просто: вновь появляющиеся требования и пожелания клиента могут выйти за временные рамки изначально обсуждённых итераций. Таким образом, необходимо некое мерило требований для отсекания маловажных, что в принципе противоречит самой концепции IID.

Второй минус выглядит куда более серьёзно: раньше речь шла исключительно о доработке КИС, теперь же непонятно, как быть с кастомизацией и интеграцией. Сначала разберёмся с кастомизацией. Настройке в КИС подлежат уровни данных и бизнес-процессов: организационная структура, процессы, основные и переменные данные. Невозможно выполнить настройку, например, бизнес-процесса в КИС, следуя принципу итеративности. Разумнее всего применить инкрементальный подход: разбить процесс на части и настроить одну часть бизнес-процесса в заданной итерации, а остальные – в следующих. Данный подход применим также к оргструктуре и данным. В итоге получается некий аналог требований: процессы, организационная структура и данные должны быть декомпозированы на объекты и распределены по итерациям. Дальше необходимо понять, полученные объекты включать в начальные или завершающие итерации? Ответ связан с вопросом интеграции.

Интеграцию КИС можно разделить на внутреннюю и внешнюю. Под внутренней интеграцией подразумевается взаимодействие объектов и модулей КИС между собой, например, приход товара на склад должен порождать бухгалтерские проводки. В данном примере присутствуют две сущности различных модулей: приход и проводка, относящиеся к функциональности логистики закупки и финансам соответственно. Второй вид интеграции подразумевает взаимодействие КИС с внешними подсистемами, например, SRM, CRM, WMS и др. Если внутреннюю интеграцию в большей степени можно отнести к кастомизации КИС, то внешнюю – к настройке и доработке. Никакая доработка КИС не может вестись без наличия базовых компонентов системы, которые преимущественно заводятся путём настройки. Тем самым логично в начальные итерации включить кастомизацию КИС и интеграцию, и лишь затем доработку. Поэтому каждая итерация должна включать как функционально-модульное, так и интеграционное тестирование. Суммируя, видится следующая картина реализации итераций:

* начальные итерации должны включать настройку системы и внутреннюю интеграцию по инкрементальному принципу с целью подготовки основополагающего ядра КИС;
* последующие итерации содержат доработки КИС, использующие ранее подготовленные функции КИС путём кастомизации.

## Примеры итерационного проектирования моделей

Ниже приведены примеры дизайнов итерационных моделей:

#### 1. Пользовательские интерфейсы

Требование кинокомпании требует специальной системы для отслеживания оборудования, костюмов и реквизита. Включенный бизнес-раздел необычен для отображения спецификаций для технологических проектов. Чтобы избежать обычных обстоятельств, когда разработчики непрерывно работают в течение месяца и создают что-то, от чего заказчики будут отказываться, используется итеративная стратегия. Через неделю разработчики и пользователи сидят дважды, чтобы обсудить последние системы. Потребители знакомятся с продукцией и создают новые спецификации, которые устанавливают строгие приоритеты, сокращая количество рабочих дней. Таким образом, система оказывается полезной и простой в использовании.

#### 2. Графический дизайн

У креативного отдела рекламных агентств есть 2 недели, чтобы изготовить рекламную печать для клиента. У них есть предложение представить своим клиентам дни в течение. Со временем, охватывающим этот крайний срок, они уже прошли бы через повторяющуюся серию ответов от клиентов и были убеждены, что они будут работать и будут подтверждены клиентом.

#### 3. Архитектура

Фирма архитектуры оперативно производит модели и архитектурную анимацию для размещения в присутствии заказчика. Это удерживает клиента и делает его более вероятным, что фирма будет выбрана для конкретного проекта.

#### 4. Маркетинг

Сайт о моде для электронной коммерции больше никогда не начинает разрабатываться, чтобы заменить старый сайт. В качестве альтернативы пользовательский интерфейс (пользовательский интерфейс) постоянно изменяется и совершенствуется. Отслеживайте метрики и соотносите с изменениями. Разработки фирм являются экспертами, основанными на практиках, которые несопоставимы с отраслью.

#### 5. Городской дизайн

Город помогает общинам строить десятилетние программы, которые служат политикой и тактикой. Между тем, дело доходит до исполнения, городские дизайнеры пытаются что-то в меньших масштабах и собирают отзывы, прежде чем продвигаться во всем, как в программе. Например, конкретный дизайн схемы живой улицы может быть предпринят в течение года до того, как будет улучшено его дальнейшее использование.

Принципиальные моменты, в которых объектно-ориентированный подход к развитию проектов стоит сопоставить с традиционными последовательными методологиями, сводятся к следующему:

* Итеративность развития.

Начиная с фазы анализа и до завершения реализации, процесс объектно-ориенти­ро­­ванного проектирования в противоположность последовательному развитию строится как серия итераций, которой возможно предшествует определенный период последовательного изучения предметной области и задач проекта в целом (этапы определения требований и начального планирования ).

* Наращивание функциональности в соответствии со сценариями.

Наращивание функциональности проектируемого изделия представляется как развитие сценариев, которые соответствуют описаниям (диаграммам) взаимодействия объектов и отражают отдельные стороны функционирования. Эти описания предписывают развитие на этапе программирования операционной базы проекта: она вырабатывается исходя из сценариев уровня проектирования (конструирования). Полная функциональность состоит из функциональностей всех сценариев. Таким образом, данная стратегия довольно близка классическому методу пошаговой детализации, при использовании которого функциональность наращивается путем уточнения (доопределения) модулей нижнего уровня. Однако в отличие от этого метода итеративное наращивание требует, чтобы в результате каждой итерации изделие получало полностью готовую функциональность, планируемую реализуемым сценарием. Последующие итерации добавляют уже другую функциональность, которая планируется другим сценарием.

* Ничто не делается однократно.

Последовательный подход предполагает, что анализ завершен перед конструированием, завершение которого предшествует программированию. Перекрытие этапов (см. п. 1.4) ослабляет это предположение, но принципиально ситуацию не меняет. В большинстве объектно-ориенти­ро­­ванных проектов анализ никогда не завершается в течение всего развития проекта, а процесс конструирования сопровождает разработку в ходе всего ее жизненного цикла.

* Оперирование на размножающихся фазах подобно.

Как в начале проектирования, на последующих итерациях анализ предшествует конструированию, за которым следует программирование, тестирование и другие виды работ.

При объектно-ориентированном проектировании в ходе итеративного наращивания обыкновенно выполняются вполне традиционные этапы:

* Определение требований, или планирование итерации, — фиксируется, что должно быть выполнено на данной итерации в виде описания области, для которой планируется разработать функциональность на данной итерации, и что для этого нужно. Обычно этот этап включает отбор сценариев, которые должны быть реализованы на данной итерации.
* Анализ — исследуются условия выполнения планируемых требований, проверяется полнота отобранных сценариев с точки зрения реализации требуемой функциональности.
* Моделирование пользовательского интерфейса — коль скоро итерация должна обеспечивать функционально законченную реализацию, требуется определить правила взаимодействий, необходимые для активизации требуемых функций. Модель интерфейса представляет пользовательское представление поведения объектов данной итерации.
* Конструирование — обычная декомпозиция проекта, проводимая в объектно-ориентированном стиле. Конструирование включает построение или наращивание иерархии системы классов, описание событий и определение реакции на них и т. д. В ходе конструирования определяются объекты, реализуемые и/или доопределяемые на данной итерации, и набор функций (методов объектов), которые обеспечивают решение задачи данной итерации.
* Реализация (программирование) — программное воплощение решений, принятых для данной итерации. Необходимым компонентом реализации здесь считается автономная проверка соответствия составляемых модулей их спецификациям (в частности, должно быть обеспечено требуемое поведение объектов).
* Тестирование — этап комплексной проверки результатов, полученных на данной итерации.
* Оценка результатов итерации — этап включает работу, связанную с рассмотрением полученных результатов в контексте проекта в целом. В частности, должно быть выяснено, какие задачи проекта можно решать с учетом результатов итерации, на какие ранее поставленные вопросы получены ответы, какие новые вопросы возникают в новых условиях.

## Преимущества

Снижение рисков — раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта; большая фокусировка на основных задачах; динамическое формирование требований и управление ими.

Организация эффективной обратной связи проектной команды с потребителем, создание продукта, реально отвечающего его потребностям.

Быстрый выпуск минимально ценного продукта (MVP) и возможность вывести продукт на рынок и начать эксплуатацию гораздо раньше.

Следует обязательно признать преимущества итеративной модели перед ее выполнением в SDLC (жизненный цикл разработки программного обеспечения). Существенным преимуществом этой модели является то, что она выполняется на предыдущих этапах процесса разработки программного обеспечения, что позволяет разработчикам и тестировщикам максимально быстро определять дефекты проектирования или функциональности, что, кроме того, позволяет им выполнять восстановительные действия при недостаточном бюджете. Другие преимущества или улучшения этих моделей:

* В жизненном цикле разработки программного обеспечения можно заранее создать несколько возможностей.
* Он эффективно универсален для постоянно меняющихся требований проекта, а также клиента.
* Это лучшее, что подходит для проворных компаний.
* Кроме того, по разумной цене можно изменить диапазон спецификаций в Итерационной модели.
* Совместное развитие может быть организовано.
* Изучение и устранение неполадок, в то время как меньше итераций просто.
* Опасности распознаются и исправляются путем итерации, и каждая итерация может быть просто обработана.
* В модели итерации сжатое время расходуется на запись, а расширенное время предоставляется для обрисовки.
* снижение воздействия серьёзных рисков на ранних стадиях проекта, что ведет к минимизации затрат на их устранение;
* организация эффективной обратной связи проектной команды с потребителем (а также заказчиками, стейкхолдерами) и создание продукта, реально отвечающего его потребностям;
* акцент усилий на наиболее важные и критичные направления проекта;
* непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом;
* раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта;
* более равномерная загрузка участников проекта;
* эффективное использование накопленного опыта;
* реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении.
* затраты распределяются по всему проекту, а не группируются в его конце.

Каждый получает достоверную оценку потребителя при отображении дизайна и схемы продукта для потребителей для обратной связи.

Итеративная разработка ПО — это процесс создания программного обеспечения, который осуществляется небольшими этапами, в ходе которых ведется анализ полученных промежуточных результатов, выдвигаются новые требования и корректируются предыдущие этапы работы.

Жизненный цикл проекта при итерационной разработке разбит на последовательность итераций, каждая из которых, по сути, является проектом в миниатюре, то есть включает в себя все процессы разработки ПО (сбор и анализ требований, составление спецификаций, непосредственную реализацию, тестирование и запуск), но в рамках одной итерации разрабатывается не весь проект, а только его версия или отдельная часть.

Как правило, цель каждой итерации — это получение версии ПО, включающей в себя как новые или преработанные возможности, реализованные в ходе текущей итерации, так и функциональность всех предыдущих итераций. Результат же финальной итерации содержит всю требуемую функциональность продукта.

Бюджет и сроки, необходимые для реализации финальной версии обычно изначально не устанавливаются, так как не определяется общий объём работ и требования формируются по ходу реализации.

Итеративная, итерационная, инкрементная и эволюционная разработка — фактически, это синонимы.

Итеративность (iteration, «повторение») в данном случае означает подход, основаный на выполнении задач в рамках «мини-проектов», инкрементность (increment «увеличение») означает последовательное добавление функционала к разрабатываемому продукту, а эволюционность (evolutio, «развёртывание») — процесс развития продукта, напоминающий естественное развитие биологических видов.

Водопадная и итеративная модели разработки

Метафорически сравнение водопадной и итеративной моделей разработки часто описывают на примере разработки транспортного средства.

В случае с «водопадом» сначала описываются требуемые характеристики автомобиля, затем по этим требованиям разрабатывается проектная документация. После составления проектной документации собираются отдельные узлы автомобиля и происходит их взаимная интеграция. Результат сборки тестируется на соответствие проектной документации и после этого созданный автомобиль передается заказчику. Все эти этапы занимают достаточно продолжительное время, а пригодный для использования продукт заказчик получает только в самом конце.

В случае с итеративным подходом всё несколько иначе. Изначально ставится задача разработки транспортного средства. И результатом первой итерации может быть вариант такого транспортного средства — например, самокат. Для него не нужен двигатель внутреннего сгорания и собрать его можно в десятки раз быстрее, чем автомобиль. Да, самокат проигрывает автомобилю по очень многим характеристикам, но он всё же более эффективен для передвижения, чем хождение пешком. Результатом второй итерации может быть уже самокат с электродвигателем. На третьей итерации — у самоката могут быть увеличены колеса и он превратится в электровелосипед. На четвертой — электровелосипед может быть оснащён ДВС и станет мотоциклом.

По сути, с каждой итерацией повышаются функциональные возможности. И пока сторонники водопада ждут готовность создаваемого автомобиля, любители итерационного подхода уже пользуются транспортным средством. И вполне может быть, что получившийся в итоге мотоцикл — более правильный бизнес‑результат.

## Недостатки

Тем не менее, хотя итеративная модель является удивительно выгодной, с ней связаны некоторые недостатки и недостатки, так как каждый этап итерации является жестким без наложений. Кроме того, могут возникнуть проблемы со структурой или схемой системы из-за того, что не все реквизиты собраны в начале полного жизненного цикла. Дополнительные недостатки итерационной модели упомянуты ниже:

* Могут потребоваться расширенные ресурсы.
* Несмотря на то, что цена изменения ниже, она не всегда соответствует спецификациям изменения.
* Требуется дополнительное признание администрации.
* Это не подходит для более коротких проектов.
* Для экспертизы способностей требуются чрезвычайно опытные ресурсы.
* Продвижение проекта в значительной степени зависит от этапов оценки рисков.
* Определение постепенно увеличивается, возможно, интерпретация всей системы.
* Целостное понимание возможностей и ограничений проекта очень долгое время отсутствует.
* При итерациях приходится отбрасывать часть сделанной ранее работы.
* Добросовестность специалистов при выполнении работ всё же снижается, что психологически объяснимо, ведь над ними постоянно довлеет ощущение, что «всё равно всё можно будет переделать и улучшить позже»

**Rational Unified Process** (RUP) — методология разработки программного обеспечения, созданная компанией Rational Software

**В основе RUP лежат следующие принципы:**

* Ранняя идентификация и непрерывное (до окончания проекта) устранение основных рисков.
* Концентрация на выполнении требований заказчиков к исполняемой программе, анализ и построение модели прецедентов.
* Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.
* Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.
* Постоянное обеспечение качества на всех этапах разработки проекта (продукта).
* Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.

RUP использует итеративную модель разработки. В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

Microsoft Solutions Framework (MSF) — методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

MSF представляет собой согласованный набор концепций, моделей и правил.

Модель проектной группы MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта. Данная модель определяет ролевые кластеры, их области компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществить свою миссию по воплощению проекта в жизнь.

Модель проектной группы MSF разрабатывалась в течение нескольких лет и возникла в результате осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп.

В соответствии с моделью MSF проектные группы строятся как небольшие многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта. Проектную группу объединяет единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание самосовершенствоваться.

Проблемы внедрения при использовании итерационной модели. В некоторых областях спиральная модель не может применяться, поскольку невозможно использование/тестирование продукта, обладающего неполной функциональностью (например, военные разработки, атомная энергетика и т. д.). Поэтапное итерационное внедрение информационной системы для бизнеса возможно, но сопряжено с организационными сложностями (перенос данных, интеграция систем, изменение бизнес-процессов, учетной политики, обучение пользователей). Трудозатраты при поэтапном итерационном внедрении оказываются значительно выше, а управление проектом требует настоящего искусства. Предвидя указанные сложности, заказчики выбирают каскадную модель, чтобы "внедрять систему один раз".

Проблемы с архитектурой и накладные расходы — при работе с хаотичными требованиями и без проработанного глобального плана архитектура приложения может пострадать, а на её приведение к адекватному виду могут потребоваться дополнительные ресурсы. По сути, за возможность менять требования в ходе создания продукта, приходится так или иначе расплачиваться.

## Вывод

Разработка программного обеспечения идет с многочисленными процедурами и методами. Ему нужны модели, еще один внешний компонент и инструменты для достижения триумфальных достижений. Один из методов разработки итеративной модели позволяет эффективно использовать скудные масштабы разработки программного обеспечения. Это одна из главных практик в методах гибкой разработки, в которых действия повторяются по мере продвижения проекта со спецификациями.

Резюмируя вышесказанное, применение итерационной модели вполне логично для доработки КИС, настройка же потребует дополнительных манипуляций. Несмотря на статистику, гласящую, что порядка 70% функционала иностранных КИС требуют доработки, пока многопроходная модель применяется в России достаточно редко. Возможно, причина кроется в том, что предпочтение отдаётся максимальному использованию стандартного функционала КИС, то есть кастомизации, против его доработки.

## Список источников

1. https://ru.education-wiki.com/3617065-iterative-model
2. https://studopedia.ru/7\_103765\_iteratsionnaya-model-stadii-dostoinstva-nedostatki.html
3. Степанов Д.Ю. Обзор проектных документов при внедрении корпоративных информационных систем // Вопросы экономических наук. – 2014. – т.70, №6. – С. 54-62.
4. Гвоздева Т.В., Баллод Б.А. Проектирование информационных систем: учебное пособие. – Ростов н/Д.: Феникс, 2009. – 508 с.
5. Зараменских Е.П. Управление жизненным циклом информационных сисием.- М.: Юрайт, 2017. - 431 с.
6. Larman C., Basili V.R. Iterative and Incremental Development: A Brief History // Computer. - 2003. – Vol.36, №6. – P.47-56.
7. Левочкина Г., Калянов Г., Васильев Р. Управление развитием информационных систем. – М.: Горячая Линия – Телеком, 2014. – 376 c.
8. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Пятое издание. – М.: Олимп-Бизнес, 2014. – 586 с.
9. Стеллман Э., Грин Д. Постигая Agile. Ценности, принципы, методологии. – М.: Манн, Иванов и Фербер, 2018. – 448 c.
10. https://corpinfosys.ru/archive/2018/issue-1/48-2018-1-models
11. https://studfile.net/preview/16493243/page:4/
12. https://prog.bobrodobro.ru/18752