Учреждение образования Федерации профсоюзов Беларуси
«Международный университет «МИТСО»

Факультет экономический
Кафедра информационных технологий

СОГЛАСОВАНО
Заведующий кафедрой
_____О.Б.Хорошко
29. 05 2026 г.

СОГЛАСОВАНО
Декан факультета
_____А.В.Ковтунов
07. 07 2026 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

ГИБКИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

для специальностей 6-05-0414-04 Управление информационными ресурсами

Составитель: Калинин М.А., старший преподаватель кафедры информационных технологий учреждения образования Федерации профсоюзов Беларуси «Международный университет МИТСО»

Рассмотрено и рекомендовано к утверждению на заседании кафедры информационных технологий
29. 05 2026 г., протокол № 11

Утверждено на заседании научно-методического совета учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО»
07. 07. 2026 г., протокол № 6

РЕЦЕНЗЕНТЫ:
Кафедра микропроцессорных систем и сетей УО «Белорусский государственный университет информатики и радиоэлектроники» (Кашникова И.В., заведующий кафедрой);
Гришко Н.И., доцент кафедры экономики и менеджмента УО ФПБ «Международный университет «МИТСО», к.т.н., доцент

Регистрационный № УД-072-26/э

Регистрационное свидетельство № 1202647968 от 20.07.2026

Актуализирован
заседание кафедры_____
_____ учреждения
образования Федерации профсоюзов Беларуси «Международный университет «МИТСО»
_____20____, протокол №

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 4
УЧЕБНАЯ ПРОГРАММА ПО ДИСЦИПЛИНЕ 5
I. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ 18
Тема 1. Существующие методологии разработки программного обеспечения. Их преимущества и недостатки 18
Тема 2. Введение в гибкие методы управления проектами и продуктами 25
Тема 3. Разработка программного обеспечения в цифровую эпоху 31
Тема 4. Новое лидерство и командная работа 39
Тема 5. Инженерные практики в гибкой разработке программного обеспечения 48
Тема 6. Переход организации к agile – выбор, пилотирование, масштабирование, трансформация 54
II. ПРАКТИЧЕСКИЙ РАЗДЕЛ 62
Планы и задания для семинарских занятий 62
III. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ 67
Тестовое задание 67
Примерный перечень вопросов к зачету 71
IV. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ 73
Перечень рекомендуемой литературы 73

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (далее – ЭУМК) подготовлен в соответствии с учебной программой учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО» по дисциплине «Гибкие методологии разработки программного обеспечения» для специальности 6-05-0414-04 «Управление информационными ресурсами».

Цель ЭУМК – обеспечение максимальной доступности и конкретности изучаемого материала по дисциплине. Практическая значимость состоит в том, что данное издание позволяет конкретизировать и спланировать учебную деятельность слушателей, разнообразить формы усвоения достаточно сложного материала.

Основными структурными элементами ЭУМК являются:

- учебная программа по дисциплине;
- теоретический раздел, включающий конспект лекций для теоретического изучения учебной дисциплины;
- практический раздел, содержащий учебно-методические материалы для поведения практических занятий, задачи для индивидуальной работы;
- раздел контроля знаний, в котором представлены контрольные вопросы по дисциплине, вопросы для самостоятельного изучения по всем предусмотренным учебной программой темам дисциплины;

Рекомендации по организации работы с ЭУМК. Для лучшего усвоения получаемых знаний слушателям рекомендуется сначала ознакомиться с учебной программой по дисциплине «Гибкие методологии разработки программного обеспечения». Усвоение материала по каждой из тем рекомендуется начинать с изучения соответствующей лекции («Теоретический раздел»), после чего следует проработать практические задания, решить индивидуальные задачи («Практический раздел»). Проверка знаний проводится на основе тестов, перечня вопросов к зачету («Раздел контроля знаний»).

Студентам рекомендуется использовать материалы ЭУМК для осуществления активной самостоятельной работы при изучении дисциплины, подготовки к аудиторным занятиям и экзамену.

Учреждение образования Федерации профсоюзов Беларуси
«Международный университет «МИТСО»

УТВЕРЖДАЮ
Ректор учреждения образования
Федерации профсоюзов Беларуси
«Международный университет «МИТСО»
_____В.М. Поздняков
16.12. 2025 г.
Регистрационный № УД -027/02-25/уч.

ГИБКИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебная программа учреждения высшего образования по учебной дисциплине для специальности

6-05-0414-04 Управление информационными ресурсами

2025 г.

Контрольный экземпляр

Учебная программа составлена на основе образовательного стандарта высшего образования ОСВО 6-05-0414-04-2023 (общее высшее образование), для специальности 6-05-0414-04 «Управление информационными ресурсами», утвержденного Постановлением Министерства образования Республики Беларусь 07.08.2023 г. № 240 и учебного плана учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО» для специальности 6-05-0414-04 «Управление информационными ресурсами», утвержденного 01.03.2023 регистрац. № 6-05-0414-04/уч.

СОСТАВИТЕЛЬ:
М.А.Калинин, старший преподаватель кафедры информационных технологий учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО»

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:
Кафедрой информационных технологий учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО»
(протокол № 4 от 28.11.2025);

Научно-методическим советом учреждения образования Федерации профсоюзов Беларуси «Международный университет «МИТСО»
(протокол № 2 от 16.12.2025)

СОГЛАСОВАНО
Проректор по учебной работе
_____М.А.Юрочкин

СОГЛАСОВАНО
Декан экономического факультета
_____А.В.Ковтунов

Нормоконтроль
ведущий специалист УМУ
_____Г.Д.Лагунович

Библиограф I категории
_____Н.А.Лёпа

I. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Дисциплина «Гибкие методологии разработки программного обеспечения» предназначен для формирования у студентов представления об организации процесса промышленной разработки программного обеспечения (ПО). По итогу студенты должны хорошо понимать, какие влияние оказывает принятая методология разработки на процессы проектирования, разработки и поставки программного обеспечения, а также на управление командой разработки. Рассматриваются различные методологии, их сильные и слабые стороны. Большое внимание уделяется вопросам применимости той или иной методологии в зависимости от целей и условий проекта, а также размера и структуры имеющейся команды разработки.

В рамках образовательного процесса по данной учебной дисциплине обучающийся должен приобрести не только теоретические и практические знания, умения и навыки по специальности, но и развить свой ценностно-личностный, духовный потенциал, сформировать качества патриота и гражданина, готового к активному участию в экономической, производственной, социально-культурной и общественной жизни страны.

Учебная дисциплина «Гибкие методологии разработки программного обеспечения» изучается во взаимосвязи с учебной дисциплиной «Технологии программирования» и «Алгоритмизация и программирование».

Освоение учебной дисциплины «Гибкие методологии разработки программного обеспечения» должно обеспечить у обучающихся формирование **специализированных компетенций**: использовать знания в области стандартов информационных систем, современных методов и подходов при разработке информационных систем

Цель преподавания учебной дисциплины «Гибкие методологии разработки программного обеспечения»: дать систему знаний по теоретическим и прикладным основам гибких методологий разработки программного обеспечения.

Задачи изучения учебной дисциплины:

формирование знаний о цели и основных задачах в области разработки программного обеспечения;

формирование знаний об основах методологии и базовых парадигмах разработки программного обеспечения;

изучение основ моделирования, проектирования и разработки программных систем;

изучение управления программными проектами (от инициации до реализации и внедрения);

освоение работы с различными методологиями, технологиями и платформами разработки программного обеспечения.

В результате изучения учебной дисциплины студент должен

знать:

принципы построения программного обеспечения различного назначения, основы проектирования, программирования, тестирования программных

продуктов, основные модели представления программных систем;

уметь:

проектировать архитектуру программного обеспечения, разрабатывать модели программных продуктов, используя современные нотации представления программных систем;

владеть:

базовыми навыками анализа предметных областей, навыками разработки программного обеспечения различного назначения с использованием автоматизированных систем проектирования.

Распределение аудиторных часов по видам занятий и семестрам.
Формы промежуточной аттестации

Семестр	Количество академических часов							Самостоят. работа	Форма промежуточной аттестации
	Всего	Аудит.	Из них						
			Лекции	Лабор. занятия	Практ. занятия	Семи-нары	УСР[1]		
Очная форма получения высшего образования									
6	90	40	18		16		6	50	зач.
Всего	90	40	18		16		6	50	

[1] Управляемая самостоятельная работа

II. СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

Тема 1. Существующие методологии разработки программного обеспечения, их преимущества и недостатки

Водопадная/каскадная методология. V-образная методология. Инкрементальная методология. RAD (быстрая) методология. Итерационная методология. Гибкая (Agile). Компонентно-ориентированная методология. Тяжеловесные и облегченные процессы. XP-процесс. Бережливая разработка программного обеспечения.

Тема 2. Введение в гибкие методы управления проектами и продуктами

Введение в гибкие методы управления проектами и продуктами. Источники и предпосылки появления Agile. VUCA-мир. Модель 4К. Методологии, практики и принципы Agile. Agile-манифест. Модели взаимодействия бизнеса и ИТ. Взаимосвязь Agile-подходов с другими областями знаний. Применение Agile-подходов вне ИТ-сферы. Business Agile, Agile-маркетинг, Agile-HR, Agile в госсекторе, образовании. Kanban, Lean, Кайдзен. Принципы бережливого производства, принципы непрерывного улучшения. Основные практики и ценности.

Тема 3. Разработка программного обеспечения в цифровую эпоху

Цифровые продукты и сервисы. Проектирование, ориентированное на пользователя. Концепция дизайн-мышления. UX, UI практики. Подход тестирования гипотез. Lean Canvas. Бережливый стартап. Видение, дорожная карта и бэклог продукта. Продуктовые метрики. Персоны и сценарии. Impact Map. User Story Map. Customer Journey Mapping. Customer Development. Unit-экономика.

Тема 4. Новое лидерство и командная работа

Новые модели лидерства. Не директивные способы управления. Servant Leadership. T-shape-специалисты. Менеджмент 3.0 Командная работа. Самодизайн, кроссфункциональность, самоорганизация. Роли в команде. Модель Белбина. Этапы формирования команды. Модель Такмана. Модель Шнейдера. Фичекоманды. Командные метрики.

Тема 5. Scrum фреймворк

Scrum фреймворк. Команды и роли, события, артефакты, правила. Scrum Guide. PMI Agile Practice Guide. Ценности, принципы, инкрементальность и итеративность.

Тема 6. Инженерные практики в гибкой разработке программного обеспечения

Инженерные практики в разработке: программирования, интеграции, управления, командные. Экстремальное программирование. Ценности,

принципы, практики. Дилемма проектирования. CI, CD, TDD, DevOps.

Тема 7. Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация

Выбор оптимальной методологии. Сравнение подходов. Модель бимодального ИТ. Agile-трансформация. Организационная культура организации. Модель Шнейдера. Модель Кеневин. 8 шагов Коттера.

III. УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА ДИСЦИПЛИНЫ

Очная (дневная) форма получения высшего образования

Номер раздела, темы	Название раздела, темы	Всего часов	Количество аудиторных часов					Самостоятельная работа	Литература	Форма контроля
			Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Количество часов УСР			
1	2	3	4	5	6	7	8	9	10	11
6 семестр										
1	Существующие методологии разработки программного обеспечения. Их преимущества и недостатки	3	4		4		2		[1]	УО, РПЗ, Р, УД
2	Введение в гибкие методы управления проектами и продуктами	4	4		2				[1]	УО, РПЗ, Р, УД
3	Разработка продуктов в цифровую эпоху	5	2		2				[1]	УО, РПЗ, Р, УД
4	Новое лидерство и командная работа	9	2		2				[1]	УО, РПЗ, Р, УД
5	Scrum фреймворк	9	2		2				[1]	УО, РПЗ, Р(ТА)[2]
6	Инженерные практики в гибкой разработке программного обеспечения	9	2		2		2		[1]	УО, РПЗ, Р, УД
7	Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация	5	2		2		2		[2]	УО, РПЗ, Р
	Всего по дисциплине	**90**	**18**		**16**		**6**			**зач.**

[2] Текущая аттестация

IV. ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

ОСНОВНАЯ ЛИТЕРАТУРА

1. Денисов, А. А. Современные проблемы системного анализа: Информационные основы : учеб. пособие / А. А. Денисов. – СПб. : Изд-во СПбГТУ, 2005. – 295 с.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Буч, Гради. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч. – М. : Бином; СПб. : Невский Диалект. 1998.

1. Макконнелл, Стив. Совершенный код. Мастер-класс / Стив Макконнелл. – М. : Русская редакция, 2010. – 896 с.

2. Таненбаум, М. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен. – СПб. : Питер, 2003. – 877 с.

3. Буч, Грейди. UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. – М. : – Изд-во ДМК, 2001 г., 430 с.

4. Ларман, Крэг. Применение UML 2.0 и шаблонов проектирования / Крэг Ларман. – М. : Вильямс, 2008. – 736 с.

5. Фаулер, М. Архитектура корпоративных программных приложений : пер. с англ. / М. Фаулер. – М. : Вильямс, 2006. – 544 с. : ил.

6. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб. : Питер, 2008. – 366 с.

СРЕДСТВА ОБУЧЕНИЯ

1. Мультимедийная аппаратура.

2. Электронные учебно-наглядные пособия (презентации, методические материалы).

СРЕДСТВА ДИАГНОСТИКИ РЕЗУЛЬТАТОВ УЧЕБНОЙ ДЕЯТЕЛЬНОСТИ

Диагностика результатов образовательной деятельности обучающихся осуществляется в ходе проведения всех видов занятий, самостоятельной работы и текущей аттестации по учебной дисциплине.

Основными **формами контроля** знаний по учебной дисциплине являются:

устный опрос	УО;
устный доклад	УД;
реферат	Р;
решение практических задач	РПЗ.

Текущая аттестация обучающихся проводится в течение семестра в целях периодического контроля и оценки результатов их учебной деятельности по учебной дисциплине с выставлением отметок.

Текущий контроль проводится строго в соответствии с учебно-методической картой дисциплины. Обучающийся должен в обязательном порядке участвовать во всех контрольных мероприятиях текущего контроля, предусмотренных учебной программой дисциплины.

Форма текущей аттестации по учебной дисциплине:

реферат Р.

К промежуточной аттестации по учебной дисциплине, модулю обучающиеся допускаются при условии успешного прохождения текущей аттестации, предусмотренной учебной программой в текущем семестре.

Формой промежуточной аттестации является:

зачет зач.

ПЕРЕЧЕНЬ ТЕМ СЕМИНАРСКИХ ЗАНЯТИЙ

Основная цель проведения семинарских занятий состоит в закреплении теоретического материала курса.

Тема 1. Существующие методологии разработки программного обеспечения. Их преимущества и недостатки.

Семинарское занятие 1. Существующие методологии разработки программного обеспечения. Их преимущества и недостатки.

Обеспеченность занятия: ПК.

Тема 2. Введение в гибкие методы управления проектами и продуктами.

Семинарское занятие 2. Введение в гибкие методы управления проектами и продуктами.

Обеспеченность занятия: ПК.

Тема 3. Разработка продуктов в цифровую эпоху.

Семинарское занятие 3. Разработка продуктов в цифровую эпоху.

Обеспеченность занятия: ПК.

Тема 4. Новое лидерство и командная работа.

Семинарское занятие 4. Новое лидерство и командная работа.

Обеспеченность занятия: ПК.

Тема 5. Scrum фреймворк.

Семинарское занятие 5. Scrum фреймворк.

Обеспеченность занятия: ПК.

Тема 6. Инженерные практики в гибкой разработке программного обеспечения

Семинарское занятие 6. Инженерные практики в гибкой разработке программного обеспечения

Обеспеченность занятия: ПК.

Тема 7. Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация.

Семинарское занятие 7. Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация.

Обеспеченность занятия: ПК.

ПЕРЕЧЕНЬ ЗАДАНИЙ И КОНТРОЛЬНЫХ МЕРОПРИЯТИЙ УСР

№ темы, (раздела)	Тема УСР	Кол-во часов	Метод. обеспечение	Форма контроля
3 курс, 6 семестр (6 часов)				
1	Существующие методологии разработки программного обеспечения. Их преимущества и недостатки.	2	Интернет-ресурсы	Реферат / Устный доклад
6	Инженерные практики в гибкой разработке программного обеспечения	2	Интернет-ресурсы	Реферат / Устный доклад
7	Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация.	2	Интернет-ресурсы	Реферат / Устный доклад

ТЕМАТИКА РЕФЕРАТОВ

1. Программные продукты (изделия).
2. Жизненный цикл программного обеспечения (ПО).
3. Метрология и качество ПО.
4. Критерии качества: сложность, корректность, надёжность, трудоёмкость.
5. Измерения и оценка качества ПО.
6. Процесс производства ПО: методы, технология и инструментальные средства.
7. Тестирование и отладка.
8. Документирование.
9. Проектирование программного обеспечения.
10. Технологический цикл разработки программных систем.
11. Коллективная работа по созданию программ.
12. Сравнительный анализ моделей ЖЦПО.
13. Оценка языка UML как инструмента бизнес-моделирования.
14. Сравнительный анализ жестких и гибких методологий разработки программ – XP, Scrum, RAD, RUP.
15. Сравнительный анализ и особенности применения моделей качества ISO, TQM, CMM, SPICE.

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ВОПРОСОВ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ (ЗАЧЕТ)

1. Основные понятия программной инженерии. Официальная классификация процессов программной инженерии.
2. Базис процессов разработки ПО. Модель «классический жизненный цикл». Инкрементная модель.
3. Базис процессов разработки ПО. Модель «классический жизненный цикл». Спиральная модель.
4. Базис процессов разработки ПО. Модель «классический жизненный цикл». Компонентно-ориентированная модель.
5. Тяжеловесные и облегченные процессы. XP-процесс.
6. Бережливая разработка программного обеспечения. Ликвидация потерь. Встраивание качества. Формирование новых знаний.
7. Бережливая разработка программного обеспечения. Откладывание необратимых решений. Быстрая доставка заказчику. Уважение к людям. Оптимизация целого.
8. Введение в гибкие методы управления проектами и продуктами. Источники и предпосылки появления Agile. VUCA-мир. Модель 4К.
9. Введение в гибкие методы управления проектами и продуктами. Методологии, практики и принципы Agile. Agile-манифест.
10. Введение в гибкие методы управления проектами и продуктами. Модели взаимодействия бизнеса и ИТ. Взаимосвязь Agile-подходов с другими областями знаний.
11. Введение в гибкие методы управления проектами и продуктами. Применение Agile-подходов вне ИТ-сферы. Business Agile, Agile-маркетинг, Agile-HR, Agile в госсекторе, образовании.
12. Введение в гибкие методы управления проектами и продуктами. Kanban, Lean, Кайдзен.
13. Введение в гибкие методы управления проектами и продуктами. Принципы бережливого производства, принципы непрерывного улучшения. Основные практики и ценности.
14. Цифровые продукты и сервисы. Проектирование, ориентированное на пользователя. Концепция дизайн-мышления. UX, UI практики.
15. Подход тестирования гипотез. Lean Canvas. Бережливый стартап.
16. Видение, дорожная карта и бэклог продукта. Продуктовые метрики. Персоны и сценарии. Impact Map. User Story Map.
17. Видение, дорожная карта и бэклог продукта. Customer Journey Mapping. Customer Development. Unit-экономика.
18. Новые модели лидерства. Не директивные способы управления. Servant Leadership. T-shape-специалисты. Менеджмент 3.0 Командная работа.
19. Самодизайн, кроссфункциональность, самоорганизация. Роли в команде. Модель Белбина.
20. Этапы формирования команды. Модель Такмана. Модель Шнейдера. Фичекоманды. Командные метрики.

21. Scrum фреймворк. Команды и роли, события, артефакты, правила. Scrum Guide. PMI Agile Practice Guide. Ценности, принципы, инкрементальность и итеративность.

22. Инженерные практики в разработке: программирования, интеграции, управления, командные.

23. Экстремальное программирование. Ценности, принципы, практики. Дилемма проектирования. CI, CD, TDD, DevOps.

24. Выбор оптимальной методологии. Сравнение подходов. Модель бимодального ИТ.

25. Agile-трансформация. Организационная культура организации. Модель Шнейдера. Модель Кеневин. 8 шагов Коттера.

V. ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ

Название учебной дисциплины, с которой требуется согласование	Название кафедры	Предложения об изменениях в содержании учебной программы учреждения высшего образования по учебной дисциплине	Решение, принятое кафедрой, разработавшей учебную программу (с указанием даты и номера протокола)
Технологии программирования	информационных технологий	Предложений нет	протокол № 4 от 28.11.2025
Алгоритмизация и программирование	информационных технологий	Предложений нет	протокол № 4 от 28.11.2025

I. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

ТЕМА 1. СУЩЕСТВУЮЩИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ИХ ПРЕИМУЩЕСТВА И НЕДОСТАТКИ

Методология разработки программного обеспечения представляет собой набор принципов, правил, практик и процессов, которые определяют порядок создания, сопровождения и развития программных продуктов. Выбор методологии существенно влияет на сроки, бюджет, качество и успех проекта в целом. Исторически сложилось так, что первые методологии были жёсткими, предсказуемыми и ориентированными на полное документирование всех этапов, а затем появились гибкие подходы, которые делают ставку на скорость реакции на изменения и близкое взаимодействие с заказчиком. В рамках данной темы рассматриваются основные существующие методологии, их сильные и слабые стороны.

Водопадная или каскадная методология

Водопадная, или каскадная, методология является одной из самых первых и наиболее линейных моделей разработки. В её основе лежит принцип последовательного прохождения фаз проекта, где каждая следующая фаза начинается только после полного завершения предыдущей. Типичные фазы включают анализ требований, проектирование, реализацию, тестирование, внедрение и сопровождение. Переход на следующую фазу возможен только после того, как результат предыдущей фазы полностью задокументирован и утверждён.

Достоинством водопадной методологии является её простота и предсказуемость. Чёткое разделение этапов позволяет легко оценивать сроки и бюджет на начальном этапе, когда известны все требования. Дисциплина и строгий контроль на каждом этапе обеспечивают высокую документальную прозрачность, что важно для государственных проектов и систем, где требуется сертификация. Кроме того, каскадная модель хорошо подходит для проектов с фиксированными и неизменными требованиями, например для расчётных систем или критически важного программного обеспечения, где ошибки на ранних стадиях недопустимы.

Недостатки водопадной модели связаны с её негибкостью. Реальный заказчик часто не может сформулировать все требования на старте проекта, потому что он меняет своё представление о продукте по мере его создания. В водопадной модели изменения требований после завершения этапа анализа и проектирования практически невозможны или стоят очень дорого. Кроме того,

заказчик видит работающий продукт только на финальной стадии, когда уже поздно вносить серьёзные правки. Если в процессе тестирования обнаруживается ошибка в архитектуре или требовании, приходится возвращаться на несколько этапов назад, что приводит к значительному перерасходу ресурсов. Поэтому водопадная модель не подходит для проектов с высокой степенью неопределённости и для стартапов.

V-образная методология

V-образная методология является развитием водопадной модели и делает акцент на верификации и валидации продукта на каждом этапе разработки. Название V-образной методологии отражает её графическое представление, где левая ветвь буквы V соответствует нисходящим этапам разработки, от анализа требований до написания кода, а правая ветвь – восходящим этапам тестирования и интеграции, которые соответствуют каждому уровню требований.

На левой стороне выполняются этапы: анализ требований заказчика, системное проектирование, архитектурное проектирование, модульное проектирование и написание кода. На правой стороне каждому из этих этапов соответствует свой уровень тестирования. Модульному проектированию соответствует модульное тестирование. Архитектурному проектированию соответствует интеграционное тестирование. Системному проектированию соответствует системное тестирование. Анализу требований заказчика соответствует приёмочное тестирование.

Достоинством V-образной методологии является то, что тестирование планируется и разрабатывается параллельно с разработкой, а не откладывается на конец проекта. Это позволяет обнаруживать дефекты на ранних стадиях, когда их исправление обходится дёшево. Методология обеспечивает высокое качество продукта и подходит для проектов, где критически важны надёжность и безопасность, например в медицинском оборудовании, авиации и автомобилестроении.

Недостатки V-образной методологии во многом схожи с недостатками водопадной. Она предполагает, что требования полностью известны и стабильны на старте. Изменения требований на поздних этапах требуют пересмотра не только кода, но и тестовых сценариев, что очень трудоёмко. Кроме того, V-образная модель не очень хорошо подходит для проектов с длительным циклом разработки, где требования могут устареть до момента сдачи продукта.

Инкрементальная методология

Инкрементальная методология основана на идее создания продукта не целиком, а отдельными независимыми частями, которые называются инкрементами. Каждый инкремент представляет собой законченную, работоспособную часть функциональности, которая может быть передана

заказчику и использована им. Разработка каждого инкремента проходит последовательно этапы анализа, проектирования, реализации и тестирования, но инкременты могут разрабатываться один за другим или параллельно.

Достоинством инкрементальной методологии является то, что заказчик начинает получать реальную пользу от продукта уже после завершения первого инкремента. Это снижает риск несоответствия конечного продукта ожиданиям, так как заказчик видит работающие части и может корректировать требования для следующих инкрементов. Кроме того, ресурсы можно распределять не равномерно, а концентрировать на наиболее важных или наиболее рискованных инкрементах. Если проект сталкивается с бюджетными ограничениями, наименее критичные инкременты можно отложить на потом или вовсе отменить.

Недостатки инкрементальной методологии связаны с необходимостью тщательного проектирования интерфейсов между инкрементами. Если архитектура изначально спроектирована плохо, то добавление новых инкрементов может потребовать переработки уже созданных частей. Кроме того, инкрементальная модель требует хорошего управления конфигурацией и версионностью, чтобы избежать конфликтов при интеграции.

RAD или быстрая методология разработки

RAD, что расшифровывается как Rapid Application Development, методология быстрой разработки приложений, была предложена как ответ на длительные циклы водопадной модели. RAD делает ставку на прототипирование, итеративную разработку и активное использование инструментов автоматического генерирования кода. Жизненный цикл в RAD обычно сжат и ориентирован на максимально быстрое создание работающего прототипа, который затем уточняется и дорабатывается вплоть до получения финальной версии.

Достоинства RAD заключаются в высокой скорости разработки и близком взаимодействии с заказчиком, который постоянно видит прототипы и даёт обратную связь. Методология хорошо подходит для проектов с относительно небольшим бюджетом и сжатыми сроками, а также для проектов, где требования могут уточняться по ходу дела. RAD активно использует повторное использование компонентов, что экономит время и ресурсы.

Недостатки RAD связаны с тем, что такая методология требует высокой квалификации команды и хорошей дисциплины, иначе скорость приводит к хаосу и накоплению технического долга. RAD не подходит для критически важных систем, где требуется строгое документирование и формальное тестирование. Кроме того, методология плохо масштабируется на большие команды и проекты, где необходимо координировать работу множества разработчиков. RAD требует тесного участия заказчика на всех этапах, но не каждый заказчик готов тратить столько времени на проект.

Итерационная методология

Итерационная методология предполагает, что разработка ведётся циклами, называемыми итерациями, и на каждой итерации создаётся рабочая версия продукта, которая добавляет новую функциональность или улучшает существующую. В отличие от инкрементальной модели, где каждый инкремент добавляет законченную независимую функцию, в итерационной модели на каждой итерации продукт становится всё более полным, но ранние итерации могут содержать ещё не все функции.

Достоинством итерационной методологии является то, что она позволяет управлять рисками на ранних этапах. Самые рискованные и неясные функции разрабатываются в первых итерациях, чтобы как можно раньше понять, выполнимы ли они в принципе. Кроме того, заказчик регулярно видит прогресс и может корректировать приоритеты. Ошибки и недочёты обнаруживаются и исправляются на ранних итерациях, что снижает стоимость их исправления.

Недостатки итерационной модели связаны с тем, что она требует чёткого планирования и управления составом итераций. Без дисциплины итерации могут затягиваться, а продукт может никогда не достичь завершённого состояния. Кроме того, итерационная модель может требовать больше времени на рефакторинг и перепроектирование по сравнению с водопадной, потому что архитектура не может быть спроектирована полностью на старте и уточняется по ходу дела.

Гибкая методология или Agile

Гибкая методология, или Agile, представляет собой не одну конкретную методологию, а семейство подходов, объединённых общими ценностями и принципами, которые зафиксированы в Манифесте гибкой разработки программного обеспечения, опубликованном в две тысячи первом году. Основные ценности Agile включают людей и взаимодействие выше процессов и инструментов, работающий продукт выше исчерпывающей документации, сотрудничество с заказчиком выше согласования контрактов и готовность к изменениям выше следования первоначальному плану.

Достоинства Agile являются следствием этих ценностей. Agile-команды быстро адаптируются к изменениям требований, потому что изменения приветствуются на любом этапе. Заказчик участвует в процессе разработки на регулярной основе, например на демонстрациях по окончании каждой итерации, что снижает риск создания не того продукта. Короткие итерации, обычно от одной до четырёх недель, позволяют регулярно получать обратную связь и быстро исправлять ошибки. Agile способствует высокой мотивации команды, потому что разработчики сами организуют свою работу и несут коллективную ответственность за результат.

Недостатки Agile связаны с тем, что методология предъявляет высокие требования к квалификации и самоорганизации команды. Неопытные разработчики или менеджеры могут неправильно интерпретировать гибкость как отсутствие планирования и дисциплины, что приводит к хаосу. Отсутствие полной документации на старте может быть проблемой для крупных корпоративных проектов и для систем с длительным жизненным циклом, где нужна поддержка новыми разработчиками через несколько лет после написания кода. Кроме того, Agile требует очень близкого вовлечения заказчика, но не все заказчики могут или хотят уделять проекту столько времени. Наконец, Agile плохо масштабируется на очень большие команды из сотен человек без применения специальных фреймворков масштабирования.

Компонентно-ориентированная методология

Компонентно-ориентированная методология, или Component-Based Development, основана на принципе многократного использования готовых программных компонентов. Компонент в этом подходе представляет собой независимую, заменяемую часть системы, которая имеет чётко определённые интерфейсы и может быть использована в разных системах без изменений. Разработка системы сводится к подбору подходящих компонентов, их настройке и интеграции, а также к созданию недостающих уникальных компонентов.

Достоинством компонентно-ориентированной методологии является значительное сокращение времени и стоимости разработки за счёт повторного использования уже отлаженных и протестированных компонентов. Повышается надёжность системы, поскольку готовые компоненты уже были проверены в других проектах. Кроме того, система становится более модульной и легко модифицируемой: для изменения функциональности достаточно заменить один компонент на другой с тем же интерфейсом.

Недостатки связаны с тем, что поиск, оценка и адаптация готовых компонентов также требует времени и ресурсов. Кроме того, компоненты могут иметь скрытые дефекты или ограничения, которые не устраивают конкретный проект. Зависимость от поставщиков компонентов создаёт риски, если компонент перестаёт поддерживаться или меняется его интерфейс. Наконец, не для всех предметных областей существуют качественные готовые компоненты, и иногда разработчикам приходится создавать всё с нуля.

Тяжеловесные и облегчённые процессы

Всё многообразие методологий разработки можно разделить на две большие категории: тяжеловесные и облегчённые процессы. Тяжеловесные процессы, к которым относятся водопадная модель, V-образная модель, а также методологии типа RUP, характеризуются большим количеством формальных артефактов, детальным документированием, чёткими ролями и процедурами,

длительными циклами планирования и строгим контролем изменений. Такие процессы требуют значительных начальных вложений в планирование и проектирование, но обеспечивают предсказуемость и хорошо подходят для крупных команд и критически важных систем.

Облегчённые процессы, к которым относятся Agile и его реализации, такие как Scrum и Extreme Programming, характеризуются минимально необходимым документированием, короткими циклами, высокой степенью самоорганизации команды и приоритетом работающего кода над формальными артефактами. Облегчённые процессы быстрее адаптируются к изменениям, но требуют более высокой дисциплины и квалификации от команды. Выбор между тяжеловесным и облегчённым процессом зависит от размера проекта, критичности системы, требований к документации и культуры организации.

XP-процесс или экстремальное программирование

XP, или Extreme Programming, является одной из наиболее известных реализаций гибкой методологии. Экстремальное программирование делает акцент на инженерных практиках, которые доведены до экстремального уровня. Основными практиками XP являются парное программирование, когда два разработчика работают за одним компьютером, один пишет код, а второй его проверяет в реальном времени. Разработка через тестирование, когда сначала пишется автоматический тест, который падает, а затем пишется минимальный код, чтобы этот тест прошёл. Непрерывная интеграция, когда код всех разработчиков объединяется и тестируется несколько раз в день. Коллективное владение кодом, когда любой разработчик может править любой участок кода. Рефакторинг, то есть постоянное улучшение структуры кода без изменения его поведения. Короткие итерации, обычно одна-три недели. Простой дизайн, то есть создание самого простого возможного решения, которое работает на текущий момент.

Достоинства XP включают очень высокое качество кода благодаря постоянному тестированию и рефакторингу, низкое количество дефектов и высокую скорость разработки за счёт минимизации лишней работы. Парное программирование способствует передаче знаний внутри команды и снижает риск ухода ключевого сотрудника. Недостатки XP связаны с тем, что не все разработчики любят и умеют работать в паре, а парное программирование может казаться неэффективным менеджерам, привыкшим к индивидуальной оценке производительности. XP требует от заказчика постоянного присутствия и готовности отвечать на вопросы, что не всегда выполнимо. Кроме того, практики XP, особенно тестирование и рефакторинг, требуют высокой технической культуры, которая достигается не во всех командах.

Бережливая разработка программного обеспечения

Бережливая разработка программного обеспечения, или Lean Software Development, возникла как перенос принципов бережливого производства, разработанных в компании Toyota, в область создания программного обеспечения. Lean-подход фокусируется на устранении любых потерь, то есть всего, что не добавляет ценности для конечного пользователя. К потерям в разработке программного обеспечения относятся частично выполненная работа, лишние функции, переделки, ожидания, передача задач между людьми и излишняя документация.

Принципы бережливой разработки включают устранение потерь, усиление обучения, принятие решений как можно позже, когда информации больше, но не настолько поздно, чтобы задержать проект. Кроме того, это быстрая доставка ценности заказчику, уважение к людям и оптимизация всей системы, а не отдельных её частей. Бережливая разработка близка к Agile по духу, но делает больший акцент на потоке создания ценности и экономическом обосновании решений.

Достоинства Lean включают повышение производительности и сокращение времени выхода на рынок за счёт устранения потерь. Команда фокусируется только на том, что действительно нужно заказчику, а не на гипотетических функциях, которые могут никогда не пригодиться. Принятие решений в последний ответственный момент позволяет избежать перепроектирования.

Недостатки бережливой разработки связаны с тем, что её принципы требуют зрелой и дисциплинированной команды, способной правильно идентифицировать потери и приоритизировать работу. Не для всех организаций подходит философия принятия решений на поздних этапах, так как традиционные бюджеты требуют детального планирования на годы вперёд. Кроме того, Lean-подход может вступать в противоречие с традиционной корпоративной культурой, где поощряется загрузка сотрудников на сто процентов, даже если эта загрузка не создаёт ценности.

ТЕМА 2. ВВЕДЕНИЕ В ГИБКИЕ МЕТОДЫ УПРАВЛЕНИЯ ПРОЕКТАМИ И ПРОДУКТАМИ

Введение в гибкие методы управления проектами и продуктами

Гибкие методы управления проектами и продуктами, объединяемые под общим названием Agile, представляют собой подход, который делает ставку на адаптивность, итеративность, тесное взаимодействие с заказчиком и готовность к изменениям. В отличие от традиционных методологий, где процесс жёстко планируется на старте и редко меняется, Agile предполагает работу короткими циклами, по завершении каждого из которых заказчик видит работающий результат и может скорректировать дальнейшее направление разработки. Сегодня Agile применяется не только в разработке программного обеспечения, где он возник, но и в управлении продуктами, маркетинге, HR, образовании и даже в государственном секторе .

Источники и предпосылки появления Agile

Agile появился не на пустом месте. Его возникновению предшествовало осознание того, что классические методологии, прежде всего водопадная модель, плохо работают в условиях быстро меняющихся требований и высокой неопределённости. В конце девяностых годов двадцатого века многие специалисты в области разработки программного искали альтернативы тяжеловесным процессам, которые требовали детального документирования и длительного согласования, но при этом часто приводили к созданию продуктов, не соответствующих реальным потребностям заказчика. Толчком к формализации нового подхода стала встреча семнадцати независимых практиков на горнолыжном курорте в штате Юта в феврале две тысячи первого года. Среди них были Кент Бек, Алистер Кокбёрн, Мартин Фаулер, Кен Швабер, Джефф Сазерленд и другие известные фигуры. Результатом этой встречи стал Agile-манифест, который зафиксировал общие ценности и принципы гибкой разработки .

VUCA-мир

VUCA-мир – это термин, который описывает современную среду, в которой действуют компании и организации. VUCA – это аббревиатура, составленная из первых букв четырёх английских слов. Volatility означает нестабильность, то есть частые и непредсказуемые изменения рыночной ситуации. Uncertainty означает неопределённость, когда трудно предсказать развитие событий из-за множества факторов. Complexity означает сложность, когда события и процессы имеют множество взаимосвязанных элементов, и трудно понять причинно-следственные связи между ними. Ambiguity означает неоднозначность, когда одна и та же информация может интерпретироваться по-

разному, и нет однозначно правильного решения. Именно в условиях VUCA-мира традиционные методы управления, ориентированные на стабильность и предсказуемость, перестают работать. Agile возник как ответ на вызовы VUCA-мира, предлагая инструменты для быстрой адаптации к изменениям и работы в условиях неопределённости .

Модель 4К

Модель 4К не является строгим термином из классического Agile-манифеста, но часто используется в контексте обучения и управления для описания ключевых компетенций, необходимых в XXI веке. Четыре К расшифровываются как критическое мышление, креативность, коммуникация и кооперация. В контексте гибких методологий эти качества особенно важны, потому что Agile-команды строятся на самоорганизации, коллективном принятии решений и постоянном обмене информацией. Критическое мышление помогает анализировать проблемы и не принимать решения на основе предположений. Креативность позволяет находить нестандартные решения. Коммуникация обеспечивает эффективное взаимодействие внутри команды и с заказчиком. Кооперация означает умение работать вместе для достижения общей цели, а не просто выполнять свою узкую функцию.

Методологии, практики и принципы Agile

Agile – это не одна конкретная методология, а скорее зонтичный термин, объединяющий семейство подходов, которые разделяют общие ценности и принципы. К Agile-методологиям относятся Scrum, Kanban, Extreme Programming, Lean и другие. Каждая из этих методологий предлагает свой набор практик и инструментов, но все они следуют принципам, заложенным в Agile-манифесте. Если говорить о структуре, то Agile включает три уровня. Первый уровень – это менталитет, или mindset, то есть определённое мировоззрение и установки, нацеленные на гибкость и быструю реакцию. Второй уровень – это философская основа, зафиксированная в четырёх ценностях и двенадцати принципах Agile-манифеста. Третий уровень – это конкретные практики и фреймворки, которые реализуют эти принципы на практике: спринты в Scrum, доски в Kanban, парное программирование в XP и так далее .

Agile-манифест

Agile-манифест является основополагающим документом для всех гибких методологий. Он был опубликован тринадцатого февраля две тысячи первого года и содержит четыре ключевые ценности. Первая ценность: люди и взаимодействие важнее процессов и инструментов. Это означает, что даже самые совершенные процедуры и программные средства не заменят живого общения и доверия между членами команды. Вторая ценность: работающий продукт важнее

исчерпывающей документации. Документация важна, но главная цель разработки – это создание ценности для пользователя, а не написание толстых томов технических спецификаций. Третья ценность: сотрудничество с заказчиком важнее согласования условий контракта. Вместо того чтобы прятаться за формальными договорённостями, Agile-команда стремится к открытому диалогу с заказчиком на всём протяжении проекта. Четвёртая ценность: готовность к изменениям важнее следования первоначальному плану. План необходим, но он не должен быть догмой; изменения требований приветствуются даже на поздних стадиях разработки, если они делают продукт более ценным для пользователя .

Помимо четырёх ценностей, манифест включает двенадцать принципов. Среди них наивысшим приоритетом является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения. Изменение требований приветствуется даже на поздних стадиях разработки, и Agile-процессы позволяют использовать эти изменения для обеспечения заказчику конкурентного преимущества. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе. Над проектом должны работать мотивированные профессионалы, и для них нужно создать условия, обеспечить поддержку и полностью довериться им. Непосредственное общение признаётся наиболее практичным и эффективным способом обмена информацией. Работающий продукт является основным показателем прогресса. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать текущий темп бесконечно. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта. Простота, то есть искусство минимизации лишней работы, крайне необходима. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд. И наконец, команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы .

Модели взаимодействия бизнеса и ИТ

В традиционных моделях взаимодействия бизнеса и ИТ, характерных для водопадной методологии, бизнес-заказчик формулирует требования на старте проекта, после чего передаёт их разработчикам и практически не участвует в процессе до момента приёмки готового продукта. Это часто приводило к разрыву между тем, что ожидал бизнес, и тем, что в итоге было создано. Agile предлагает принципиально иную модель. Вместо единовременной передачи требований организуется постоянно действующее сотрудничество. Бизнес-заказчик становится полноправным членом команды или, как минимум, регулярно

участвует в планировании итераций, демонстрациях результатов и ретроспективах. Продуктовый владелец, или Product Owner, является представителем интересов бизнеса внутри команды. Он отвечает за формирование и приоритизацию бэклога, то есть списка задач, и за то, чтобы команда разрабатывала наиболее ценные функции в первую очередь. Такая модель позволяет бизнесу быстро реагировать на изменения рынка, корректировать приоритеты и получать реальную отдачу от инвестиций уже на ранних этапах проекта .

Взаимосвязь Agile-подходов с другими областями знаний

Agile не существует в вакууме. Он впитал в себя идеи из разных областей знаний. Из теории управления качеством, в частности из работ Эдвардса Деминга, пришли принципы непрерывного улучшения процессов. Из бережливого производства, или Lean manufacturing, разработанного в компании Toyota, были заимствованы идеи устранения потерь, визуализации потока создания ценности и уважения к людям. Из теории сложных систем пришло понимание того, что в сложной среде невозможно всё спланировать заранее, и процесс должен быть адаптивным. Из психологии и социологии пришли знания о мотивации, командной динамике и самоорганизации. Таким образом, Agile является междисциплинарным подходом, объединяющим лучшие практики из разных областей.

Применение Agile-подходов вне ИТ-сферы

Хотя Agile зародился в разработке программного обеспечения, сегодня он успешно применяется во многих непрофильных сферах. Business Agile означает применение гибких принципов к управлению бизнесом в целом, включая стратегическое планирование, бюджетирование и работу целых подразделений. Agile-маркетинг использует спринты, ежедневные стендапы и ретроспективы для управления маркетинговыми кампаниями, тестирования гипотез и быстрой адаптации к изменениям рынка. Agile-HR внедряет гибкие подходы в управление персоналом: регулярную обратную связь вместо ежегодных оценок, формирование кросс-функциональных команд под задачи, гибкую настройку процессов найма и адаптации. Agile в госсекторе применяется при разработке цифровых государственных услуг, где также важна быстрая обратная связь от граждан и возможность улучшать сервис итеративно, а не ждать завершения многолетних контрактов. Agile в образовании используется для построения учебных программ, где студенты работают над проектами короткими итерациями, получают обратную связь и улучшают свои работы. Известны примеры успешного применения Agile в банковском секторе, как, например, трансформация IT-департаментов Райффайзен Банка, где команды перешли от

жёсткой иерархии к кросс-функциональным трайбам и теперь ежемесячно выпускают новые релизы мобильного приложения .

Kanban, Lean, Кайдзен

Методологии Kanban, Lean и Кайдзен тесно связаны с Agile и часто используются вместе с ним. Kanban – это метод визуализации потока работ, при котором задачи отображаются на доске с колонками, например «К выполнению», «В работе», «На проверке», «Готово». Kanban ограничивает количество задач, находящихся одновременно в работе, что позволяет избежать перегрузки команды и выявлять узкие места в процессе. В отличие от Scrum, Kanban не требует фиксированных итераций и позволяет выпускать изменения непрерывно по мере готовности задач.

Lean, или бережливое производство, это философия, фокусирующаяся на устранении всех видов потерь, то есть всего, что не добавляет ценности для конечного пользователя. В разработке программного обеспечения к потерям относятся частично сделанная работа, лишние функции, ожидания, переделки, передача задач между людьми, излишняя документация и неиспользуемый код. Lean-подход стремится создать непрерывный поток ценности, при котором каждая задача максимально быстро проходит от идеи до доставки пользователю.

Кайдзен – это японский термин, означающий непрерывное улучшение. Кайдзен предполагает, что каждый сотрудник, независимо от своей должности, постоянно ищет способы сделать свою работу лучше, проще и эффективнее. В Agile-командах принцип Кайдзен реализуется через ретроспективы, которые проводятся по завершении каждой итерации. На ретроспективе команда обсуждает, что прошло хорошо, что можно улучшить, и принимает конкретные действия для улучшения процесса в следующей итерации .

Принципы бережливого производства и принципы непрерывного улучшения

Принципы бережливого производства, сформулированные в Toyota, включают определение ценности с точки зрения клиента, выявление потока создания ценности, организацию непрерывного движения потока, создание системы вытягивания, когда следующая задача берётся только тогда, когда есть свободная мощность для её выполнения, и стремление к совершенству через постоянное улучшение. Эти принципы легли в основу Lean-подхода в разработке программного обеспечения.

Принципы непрерывного улучшения включают в себя цикл PDCA, то есть Plan-Do-Check-Act, что означает планируй-делай-проверяй-воздействуй. Этот цикл, известный как цикл Деминга, лежит в основе многих систем управления качеством. В Agile цикл PDCA реализуется на каждом уровне: в ежедневных стендапах, где команда проверяет прогресс и корректирует действия, в

планировании итераций, в демонстрациях и в ретроспективах. Непрерывное улучшение означает, что процесс никогда не считается идеальным и законченным; команда постоянно ищет возможности для оптимизации и устранения потерь .

Основные практики и ценности

Основные практики Agile варьируются в зависимости от выбранного фреймворка, но существует ряд общих практик, характерных для большинства гибких подходов. Это работа короткими итерациями, обычно от одной до четырёх недель. Ежедневные короткие встречи команды, или стендапы, на которых каждый отвечает на три вопроса: что сделал вчера, что планирует сделать сегодня и какие есть препятствия. Планирование итерации, где команда отбирает задачи из бэклога для выполнения в следующем цикле. Демонстрация результата заказчику в конце каждой итерации. Ретроспектива, на которой команда анализирует свою работу и договаривается об улучшениях. Визуализация задач с помощью досок. Приоритизация бэклога, где самые ценные и важные задачи находятся наверху списка.

Что касается ценностей, помимо четырёх ценностей Agile-манифеста, многие фреймворки добавляют свои дополнительные ценности. Например, в Scrum часто выделяют пять ценностей: смелость, чтобы принимать сложные решения и говорить правду; фокус, чтобы в рамках итерации сосредоточиться на задачах, взятых в работу; открытость, чтобы прозрачно показывать как успехи, так и проблемы; уважение, чтобы ценить мнение каждого члена команды; и преданность, чтобы брать на себя обязательства и выполнять их . Эти ценности и практики в совокупности создают ту уникальную культуру, которая позволяет Agile-командам работать эффективно даже в самых нестабильных и неопределённых условиях.

ТЕМА 3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ЦИФРОВУЮ ЭПОХУ

Цифровые продукты и сервисы

В цифровую эпоху основным результатом разработки программного обеспечения становятся не просто программы, а цифровые продукты и сервисы. Цифровой продукт – это достаточно широкое понятие, которое включает в себя чисто программные продукты, то есть программы как товар, готовые работоспособные приложения разной функциональности и назначения, поставляемые потребителю на носителях или через интернет с необходимой документацией и гарантиями от разработчика . Сюда же относятся продукты и онлайновые услуги, основанные на сборе, анализе и статистической обработке больших объемов данных, а также цифровые решения определённых реальных бизнес-проблем, разрабатываемые преимущественно методами программной инженерии . Некоторые цифровые продукты могут иметь как программные, так и физические составляющие, но их ключевая особенность заключается в том, что основная ценность для пользователя создаётся именно программным обеспечением. Цифровые продукты и сервисы сегодня широко применяются практически во всех сферах жизни и деятельности – в бизнесе, промышленности, медицине, образовании – и обеспечивают повышение эффективности, функциональности, производительности и улучшение качества жизни . Важно различать цифровые товары и цифровые услуги. Цифровой товар представляет собой файл, который можно скачать и сохранить на устройство пользователя, после чего использовать без подключения к интернету . Цифровая услуга, напротив, требует интерактивного режима и потребляется одновременно с передачей, то есть для её использования необходимо постоянное подключение к сети, как, например, в случае с онлайн-кинотеатром или онлайн-переводчиком.

Проектирование, ориентированное на пользователя

Проектирование, ориентированное на пользователя, или User-Centered Design, является одним из наиболее технологичных подходов к созданию удобного программного обеспечения. Основная идея этого метода заключается в использовании специальных моделей, которые помогают адекватно определить набор задач, которые необходимо решать пользователям, и способы организации информации, позволяющие упростить их решение . В рамках этого подхода строится модель ролей, которая представляет собой список ролей пользователей системы, где каждая роль – это группа связанных задач и потребностей некоторого множества пользователей . Модель ролей определяет связи между ролями, которые могут уточнять друг друга, включать друг друга или просто быть похожими, а также выделяет набор из одной-трёх центральных ролей, на которые в основном и будет нацелено проектирование . Для каждой роли могут

быть определены профили, включающие обязанности, умения, типичные варианты взаимодействия с системой, критерии удобства, необходимые функции и возможные убытки от ошибок . Кроме того, строится модель задач, которая базируется на сущностных вариантах использования, где в рамках сценариев выделяются только цели и задачи пользователя, а не конкретные его действия, что помогает освободиться от неявных предположений о наличии определённых элементов интерфейсов .

Концепция дизайн-мышления

Дизайн-мышление, или Design Thinking, – это методология, ориентированная на человека и используемая для решения сложных проблем и создания инновационных продуктов. Дизайн-мышление включает пять основных этапов. Первый этап – эмпатия, то есть погружение в контекст пользователя, сбор информации о его потребностях, болях и мотивах через наблюдения и интервью . Второй этап – фокусировка, на котором необходимо проанализировать собранную информацию, выявить ключевую проблему и сформулировать её с точки зрения пользователя . Третий этап – генерация идей, когда команда использует методы мозгового штурма для поиска максимально широкого спектра возможных решений, а затем отбирает наиболее конструктивные из них, применяя, например, метод шести шляп . Четвёртый этап – прототипирование, то есть создание упрощённых и недорогих версий решения, будь то бумажные макеты, цифровые прототипы или другие варианты, позволяющие быстро проверить идею . Пятый этап – тестирование, когда прототипы показываются реальным пользователям для сбора обратной связи, которая затем используется для доработки решения. Важной особенностью дизайн-мышления является его итеративность: команда может возвращаться на предыдущие этапы на основе полученных результатов тестирования .

UX и UI практики

UX/UI-дизайн является неотъемлемой частью разработки цифровых продуктов в современном мире. UX-дизайн, или User Experience Design, отвечает за то, как интерфейс работает, и фокусируется на общем опыте пользователя, предполагая изучение его потребностей для последующего проектирования структуры и логики сценариев . В ходе UX-дизайна продумывается навигация: UX-дизайнер разрабатывает структуру приложения так, чтобы пользователь мог легко добраться с главного экрана до оформления и оплаты заказа, например, в интернет-магазине. Также продумываются сценарии использования: как удобнее и быстрее найти нужный товар, добавить его в корзину и совершить покупку . UI-дизайн, или User Interface Design, отвечает за то, как интерфейс выглядит, и касается визуальной части, включая выбор цветовой палитры, шрифтов, иконок и других элементов . UI-дизайнер работает над визуальным стилем продукта и

добивается согласованности элементов, баланса между информативностью, структурированностью и понятностью. Простыми словами, UX отвечает за функциональность продукта, а UI – за его визуальную составляющую . Современные тренды в UX/UI-дизайне включают минимализм, то есть упрощение интерфейсов и использование только необходимых элементов, чтобы помочь пользователям сосредоточиться на содержании и избежать информационной перегрузки. Популярны тёмные темы, обеспечивающие комфортное чтение в условиях низкой освещённости и экономящие заряд батареи на мобильных устройствах. Микроанимации делают страницы и экраны более живыми и интерактивными, помогая пользователям понять, что произошло после их действий. Голосовые интерфейсы интегрируются в приложения с развитием голосовых помощников. Универсальность означает, что приложения и сайты становятся более адаптивными и отзывчивыми, обеспечивая отличный пользовательский опыт на любых устройствах .

Подход тестирования гипотез

Тестирование гипотез является ключевым элементом современной разработки в цифровую эпоху. Вместо того чтобы строить долгосрочные планы на основе непроверенных предположений, команды формулируют гипотезы – конкретные утверждения о том, какое изменение в продукте приведёт к какому измеримому улучшению – и затем проверяют их эмпирически с минимальными затратами. Для этого используется методология HADI-циклов, которая включает четыре этапа: Гипотеза, или Hypothesis, то есть предположение о том, как улучшить продукт; Действие, или Actions, то есть реализация изменения для небольшой выборки пользователей; Данные, или Data, то есть сбор и анализ метрик, показывающих эффект от изменения; Выводы, или Insights, то есть решение о том, внедрять изменение для всех пользователей, дорабатывать его или отказаться от идеи . Тестирование гипотез позволяет проверять гипотезы с минимальными затратами времени и ресурсов, получать обратную связь от реальных пользователей и быстро вносить улучшения .

Lean Canvas

Lean Canvas – это модель для быстрого анализа бизнеса или его отдельного направления, которая представляет собой одностраничный шаблон бизнес-модели, адаптированный специально для стартапов и новых продуктов . Lean Canvas включает девять блоков. Целевая аудитория – это группы пользователей или клиентов, для которых создаётся решение, а также их характеристики и контекст. Проблемы – это три-пять ключевых проблем или болей целевой аудитории, которые продукт призван решить. Уникальное торговое предложение, или УТП, представляет собой краткое обещание ценности: какую пользу получает клиент и чем предложение отличается от альтернатив. Решения

– это основные функции, сервисы или подходы продукта, которые закрывают заявленные проблемы. Каналы коммуникации – это способы взаимодействия с клиентами, через которые происходит привлечение, коммуникация и продажи. Источники дохода – это источники монетизации и финансовые потоки, то есть за что платит клиент и как формируется выручка. Затраты – это основные статьи расходов, связанные с разработкой, продвижением и операционной работой продукта. Ключевые метрики – это показатели, по которым будет оцениваться успешность продукта, например конверсии, удержание, выручка, активность пользователей. Несправедливые преимущества – это то, что сложно повторить конкурентам: уникальные ресурсы, знания, технологии, доступы или репутационный капитал . Lean Canvas позволяет получить целостное представление о продукте и используется для проверки гипотез: блоки регулярно пересматриваются и корректируются по мере получения новой информации.

Бережливый стартап

Методология Lean Startup, или бережливый стартап, была предложена Эриком Рисом и используется для быстрого тестирования идей, выбора бизнес-моделей и создания только востребованных продуктов . В основе методологии лежит цикл Build-Measure-Learn, то есть Создать-Измерить-Обучиться. Сначала команда создаёт минимальную версию решения, называемую MVP, то есть Minimum Viable Product, или минимально жизнеспособный продукт, который содержит только базовую ценность и позволяет проверить ключевые гипотезы с минимальными затратами. Затем команда измеряет реакцию пользователей на этот MVP, собирая данные об использовании, поведении и обратную связь. На основе полученных данных команда делает выводы и решает, продолжать ли развитие в выбранном направлении, вносить коррективы или менять курс, то есть совершать поворот, или pivot . Ключевые принципы бережливого стартапа включают подтверждение фактами, то есть использование научного подхода и проверку гипотез опытным путём, а также учёт инноваций с помощью критериев и показателей, которые помогают оценить успех развития или провала стартапа . Методология Lean Startup помогает стартапам сокращать расходы и избегать ситуации, когда продукт разрабатывается долго и тщательно, но в итоге оказывается никому не нужным.

Видение, дорожная карта и бэклог продукта

Видение продукта, или Product Vision, представляет собой долгосрочное стратегическое представление о том, каким продукт станет в будущем и какую ценность он будет приносить пользователям и бизнесу. Видение отвечает на вопрос, зачем мы делаем этот продукт и какую проблему в мире решаем. Дорожная карта продукта, или Product Roadmap, является высокоуровневым планом развития продукта на горизонте от нескольких месяцев до нескольких

лет. Дорожная карта показывает основные вехи, крупные функциональные блоки и примерные сроки их реализации, но не содержит детальных задач. Бэклог продукта, или Product Backlog, представляет собой упорядоченный по приоритету список всего, что может потребоваться в продукте, включая новые функции, исправления ошибок, технические улучшения и исследовательские задачи. Владелец продукта, или Product Owner, отвечает за формирование и поддержание бэклога, а также за его приоритизацию таким образом, чтобы команда всегда работала над наиболее ценными для пользователя и бизнеса задачами.

Продуктовые метрики

Продуктовые метрики являются количественными показателями, которые помогают оценить успешность продукта, прогресс в достижении целей и эффективность принимаемых решений. Одной из ключевых концепций в области продуктовых метрик является метрика Полярной звезды, или North Star Metric. Это единственный показатель, который отражает ценность продукта для клиентов, то есть насколько хорошо компания решает главную проблему своей аудитории . Правильная метрика Полярной звезды соответствует трём условиям. Когда метрика растёт, растёт и бизнес компании. Показатель напрямую связан с тем, зачем клиенты используют продукт. И команда может улучшить метрику своими действиями: новой функциональностью, изменением интерфейса, исправлением ошибок . Например, для платформы управления проектами это может быть количество активных проектов, для музыкального стримингового сервиса – общее время прослушивания, для CRM-системы – количество закрытых сделок через систему за месяц . Метрика Полярной звезды фокусирует всю компанию на одной цели, помогает расставлять приоритеты и ускоряет принятие решений, поскольку вместо двадцати графиков достаточно показать динамику одного показателя . Она помогает избежать ловушки показателей тщеславия, таких как количество зарегистрированных пользователей, которые могут создавать иллюзию успеха, но не отражают реальную ценность продукта .

Персоны и сценарии

Персоны, или Personas, представляют собой вымышленных, но реалистичных персонажей, которые олицетворяют различные сегменты целевой аудитории продукта. Каждая персона имеет имя, фотографию, демографические характеристики, цели, потребности, боли, поведенческие паттерны и контекст использования продукта. Создание персон помогает команде лучше понимать своих пользователей, принимать решения, ориентируясь на конкретные типажи, а не на абстрактную массу, и избегать ошибки, когда разработчики проектируют продукт под себя. Персоны строятся на основе реальных данных, собранных в ходе исследований: интервью, опросов, анализа поведения. Сценарии, или

Scenarios, описывают типичные ситуации использования продукта от лица конкретной персоны. Сценарии отвечают на вопросы: в каком контексте пользователь обращается к продукту, какую цель он хочет достичь, какие шаги для этого предпринимает, какие возникают препятствия. Сценарии помогают команде проектировать пользовательский путь и проверять, насколько продукт соответствует реальным потребностям в реальных ситуациях.

Impact Map

Impact Mapping является методом стратегического планирования, который помогает связать цели продукта с конкретными действиями команды. Impact Map визуально представляет собой карту, в центре которой находится цель, или Why. Вокруг цели располагаются акторы, или Who, то есть роли пользователей или стейкхолдеров, чьё поведение необходимо изменить для достижения цели. От каждого актора расходятся влияния, или Impacts, которые описывают, как именно должно измениться поведение этого актора. И наконец, от каждого влияния расходятся deliverables, или What, то есть конкретные функциональные возможности или изменения в продукте, которые должны вызвать желаемое изменение поведения. Impact Mapping позволяет избежать ситуации, когда команда сосредоточена на создании функций, не понимая, как эти функции влияют на поведение пользователей и достижение бизнес-целей.

User Story Map

User Story Mapping – это метод, который помогает команде и заказчику увидеть картину целиком и понять, как отдельные пользовательские истории, или user stories, складываются в полноценный пользовательский опыт. Карта историй строится следующим образом. По горизонтали откладывается последовательность действий пользователя, то есть шаги, которые он выполняет для достижения своей цели. Это называется пользовательским путешествием. По вертикали откладываются альтернативные способы выполнения этих шагов или дополнительные детали. Карта историй помогает команде приоритизировать разработку: сначала создаётся «скелет» карты, то есть минимальная версия, которая позволяет пользователю пройти весь путь от начала до конца с базовым уровнем качества, а затем добавляются дополнительные возможности, то есть «мышца». User Story Mapping особенно полезна на этапе планирования релизов, так как позволяет увидеть, какие функции необходимы для минимально жизнеспособного продукта, а какие могут быть отложены на потом.

Customer Journey Mapping

Customer Journey Map, или карта пути клиента, представляет собой визуализацию опыта взаимодействия пользователя с продуктом или компанией на протяжении всего времени контакта, от первого знакомства до завершения

использования. Карта включает этапы пути, действия пользователя на каждом этапе, его мысли, эмоции, точки контакта с продуктом и болевые точки, то есть места, где возникают трудности или негативные эмоции. Customer Journey Map помогает команде увидеть продукт глазами пользователя, выявить разрывы и неудобства в пользовательском опыте и определить приоритеты для улучшения. В отличие от User Story Map, которая фокусируется на выполнении конкретной задачи и часто используется непосредственно в разработке, Customer Journey Map охватывает более широкий контекст, включая этапы до использования продукта и после него, и часто используется на уровне бизнес-анализа и маркетинга.

Customer Development

Customer Development, или CustDev, представляет собой подход, при котором развитие продукта строится на основе регулярного общения с клиентами. Customer Development проводится в рамках Discovery, на этапе аналитики, до начала активной разработки продукта, чтобы получить данные об ожиданиях, болях и потребностях пользователей до того, как команда начнёт создавать решение . Существует несколько инструментов и методов Customer Development. Проблемное интервью направлено на выявление истинных болей и потребностей клиентов на ранних стадиях разработки продукта. Особенность этого метода заключается в том, что необходимо задавать вопросы, раскрывающие проблемы, а не их решения, например: как часто возникает эта проблема, когда в последний раз вы сталкивались с ней . Решенческое интервью помогает проверить, готово ли предлагаемое решение решить проблему пользователя и готов ли клиент платить за него. Клиентам предлагают оценить прототип или концепцию продукта и спрашивают, насколько решение отвечает ожиданиям, какие функции следует улучшить и готовы ли они использовать продукт и платить за него . Глубинное интервью позволяет получить детализированную и развёрнутую информацию о мотивациях, установках и поведении целевой аудитории в форме неструктурированного или полуструктурированного разговора, где интервьюер активно слушает и задаёт уточняющие вопросы, чтобы дойти до истинной причины, которая движет поведением пользователя . UX-тестирование проверяет, насколько аудитории удобно пользоваться продуктом: пользователю даётся задание выполнить определённые действия, и наблюдатели фиксируют, где возникают трудности, какие кнопки не были замечены, какие действия вызывают путаницу . Количественные интервью представляют собой опросы с закрытыми вопросами на больших выборках для выявления общих трендов среди целевой аудитории .

Unit-экономика

Unit-экономика, или юнит-экономика, представляет собой метод анализа бизнес-модели, при котором все финансовые показатели рассчитываются на

одного юнита. Юнитом может быть один пользователь, один заказ, один клиент, одна транзакция или любая другая единица, приносящая доход. Основная цель unit-экономики – понять, прибыльна ли каждая единица бизнеса, и определить, масштабируема ли бизнес-модель. Две ключевые метрики unit-экономики – это Customer Acquisition Cost, или CAC, то есть стоимость привлечения одного платящего клиента, включающая все затраты на маркетинг и рекламу, и Lifetime Value, или LTV, то есть чистая прибыль, которую приносит один клиент за всё время использования продукта. Здоровым считается соотношение LTV к CAC не менее трёх к одному, то есть клиент должен приносить в три раза больше, чем стоил его привлечение . В unit-экономике также используются такие метрики, как средний чек, или Average Order Value – AOV, маржинальность, частота покупок или использования в месяц и время окупаемости затрат на привлечение клиента. Unit-экономика позволяет принимать обоснованные решения о ценообразовании, маркетинговых бюджетах и приоритетах развития продукта.

ТЕМА 4. НОВОЕ ЛИДЕРСТВО И КОМАНДНАЯ РАБОТА

Новые модели лидерства

В цифровую эпоху представления о лидерстве претерпели значительные изменения. Если раньше лидер воспринимался как главный эксперт, который знает всё лучше других и отдаёт приказы, то сегодня такой авторитарный стиль управления становится неэффективным и даже вредным. Современные исследования показывают, что семьдесят три процента сотрудников уходят из компаний не из-за задач, а именно из-за стиля руководства, а команды с высокой степенью автономии демонстрируют рост эффективности почти на триста процентов по сравнению с теми, где практикуется микроменеджмент .

Авторитарный стиль управления, основанный на жестком контроле и страхе наказания, создаёт несколько серьёзных проблем. Во-первых, резко падает скорость принятия решений, потому что любое действие должно быть согласовано с руководителем. Во-вторых, инициатива сотрудников полностью исчезает: при микроконтроле люди делают только то, что им сказали, и перестают предлагать идеи. В-третьих, происходит массовое выгорание – как сотрудников, которые живут в постоянном стрессе, так и самого руководителя, который пытается контролировать всё и вся . Поэтому сегодня на смену «командиру» приходит «архитектор среды» – лидер, который не давит, а создаёт условия для раскрытия потенциала команды. Такой лидер строит свою работу на трёх опорах: доверие как инструмент ускорения, культура как стратегический актив и энергия команды как источник роста .

Интересно, что современные исследователи выделяют несколько новых интегративных моделей лидерства. Среди них трансформационное лидерство, когда лидер вдохновляет команду на достижение целей, выходящих за рамки личных интересов; харизматическое лидерство, основанное на личном обаянии и способности увлекать за собой; визионерское лидерство, когда лидер формирует привлекательный образ будущего; и цифровое лидерство, которое учитывает специфику управления виртуальными командами и работы в условиях цифровой трансформации . При этом ключевое отличие новых моделей от классических заключается в их интегративности – они учитывают не одну характеристику лидера, а целый комплекс: личностные качества, поведение, ситуационный контекст и процессы влияния .

Недирективные способы управления

Недирективные способы управления – это подходы, при которых руководитель не даёт готовых решений и прямых указаний, а создаёт условия, в которых сотрудники сами находят оптимальные пути достижения целей. Одним из ключевых инструментов здесь становится коучинговый подход. Вместо того чтобы сказать «делай так», лидер задаёт наводящие вопросы: «Как ты сам видишь

решение этой проблемы?», «Что мешает тебе достичь результата?», «Какие варианты ты рассматривал?». Такой подход развивает ответственность и уверенность у сотрудников, поскольку они сами находят ответы и сами принимают решения .

Важным аспектом недирективного управления является предоставление права на ошибку. Как отмечают современные управленцы, без ошибок нет развития. Конечно, это не означает вседозволенность: на начальном этапе ошибки допустимы и даже приветствуются как часть обучения, но по мере роста компетенций сотрудник должен быть готов нести ответственность за свои действия . Кроме того, недирективное управление предполагает индивидуальный подход к каждому сотруднику: кого-то важно похвалить, кого-то – мягко скорректировать, а иногда – применить финансовые стимулы. Главное – опираться на факты и конкретные данные, а не на эмоции .

Современные лидеры также активно используют принцип «управление через прозрачность и контекст». Это означает, что руководитель не контролирует каждый шаг, а задаёт ясные цели, понятные правила и общий контекст, в котором команда действует автономно. Доверие при таком подходе не является слепым – оно подкреплено прозрачностью процессов и регулярной обратной связью .

Servant Leadership

Servant Leadership, или лидерство-служение, является одной из наиболее влиятельных концепций в современном менеджменте. Термин был введён Робертом Гринлифом и означает такую модель лидерства, при которой главная цель руководителя – служить своей команде, создавая условия для её роста и успеха. В отличие от традиционного лидерства, где власть исходит от вершины иерархии, лидер-слуга ставит потребности команды на первое место.

Лидер-слуга выполняет несколько ключевых функций. Во-первых, он устраняет препятствия, мешающие команде работать эффективно. Во-вторых, он обеспечивает команду всеми необходимыми ресурсами. В-третьих, он заботится о профессиональном развитии каждого члена команды. В-четвёртых, он создаёт психологически безопасную среду, в которой люди не боятся ошибаться и предлагать новые идеи . При этом лидер-слуга не занимается микроменеджментом и не решает за команду её задачи – он доверяет профессионализму сотрудников и даёт им свободу действий в рамках agreed-upon целей.

Особенно важна модель Servant Leadership в контексте работы с высококвалифицированными специалистами, которые сами являются экспертами в своей области. Такие сотрудники не нуждаются в постоянном контроле и указаниях – им нужна поддержка, доверие и возможность влиять на свою работу . Лидер-слуга как раз и создаёт такую среду, где талантливые люди могут раскрыть свой потенциал.

T-shape-специалисты

T-shape-специалист – это человек, чей профиль компетенций напоминает по форме букву «Т». Вертикальная палочка буквы означает глубокие экспертные знания в одной или нескольких предметных областях. Горизонтальная перекладина означает способность взаимодействовать со специалистами из других областей, понимать их задачи и язык, а также готовность выходить за рамки своей узкой специализации.

Концепция Т-образных специалистов стала особенно актуальной с переходом на гибкие методологии. В кросс-функциональных командах, где рядом работают разработчики, тестировщики, аналитики и дизайнеры, умение понимать коллег и при необходимости подменять их становится критически важным. T-shape-специалист не только глубоко знает свою область, но и обладает широким кругозором, эмпатией к другим ролям и навыками коммуникации. Команда, состоящая из Т-образных специалистов, способна быстрее адаптироваться к изменениям и решать сложные задачи, поскольку знания и опыт в ней распределены более равномерно.

Формирование Т-образных специалистов требует от компании создания условий для горизонтального развития. Сотрудникам дают возможность пробовать себя в смежных областях, обучаться новым навыкам, участвовать в кросс-функциональных проектах. Некоторые компании практикуют ротацию ролей, чтобы сотрудники могли лучше понять работу своих коллег и расширить кругозор . Такой подход не только повышает гибкость команды, но и снижает риск выгорания, поскольку работа становится более разнообразной.

Менеджмент 3.0

Менеджмент 3.0 – это подход к управлению, предложенный Юргеном Аппело, который объединяет принципы гибкой разработки, теорию сложных систем и современные представления о мотивации и лидерстве. Название «Менеджмент 3.0» подразумевает эволюцию управленческих парадигм: версия 1.0 – это классический командный менеджмент, версия 2.0 – управление качеством и процессный подход, а версия 3.0 – управление, ориентированное на людей и самоорганизацию.

Менеджмент 3.0 рассматривает организацию не как машину, которой можно управлять с помощью рычагов и кнопок, а как живой организм, который самоорганизуется и адаптируется к окружающей среде. Роль менеджера при этом меняется: он не контролирует и не приказывает, а создаёт условия для того, чтобы команды могли эффективно работать. Ключевые практики Менеджмента 3.0 включают делегирование полномочий командам, развитие компетенций сотрудников, улучшение коммуникации, внедрение системы поощрений,

основанной на внутренней мотивации, и постоянное совершенствование процессов.

Важным элементом Менеджмента 3.0 является набор игр и упражнений, которые помогают командам обсуждать сложные вопросы управления в безопасной и игровой форме. Например, игра «Delegation Poker» помогает определить, какой уровень делегирования полномочий подходит для той или иной ситуации, а игра «Moving Motivators» позволяет выявить, что именно мотивирует каждого члена команды.

Командная работа. Самодизайн, кроссфункциональность, самоорганизация

Современные гибкие методологии делают ставку на командную работу, основанную на трёх ключевых принципах: самодизайне, кроссфункциональности и самоорганизации. Эти три понятия тесно связаны, но обозначают разные аспекты зрелой команды.

Самодизайн команд – это практика, при которой сотрудники самостоятельно формируют команды под конкретные задачи, исходя из своих интересов и компетенций . В отличие от традиционного подхода, где команды назначаются руководителем, самодизайн предполагает, что инициатива переходит к самим участникам. Процесс обычно проходит в форме специальной сессии, где фасилитатор помогает участникам соблюдать правила, но не вмешивается в принятие решений. Сотрудники изучают доступные проекты, советуются друг с другом и размещают свои карточки рядом с выбранной командой. Если в какой-то команде не хватает нужных компетенций, участники обсуждают, кто готов пересмотреть свой выбор. Процесс может повторяться несколько раз, пока не будет достигнуто оптимальное распределение .

Самодизайн повышает вовлеченность и мотивацию, поскольку люди получают возможность влиять на свою работу и развиваться в желаемом направлении. Кроме того, эта практика помогает избежать выгорания, так как сотрудники могут менять проекты и команды внутри компании, не думая об увольнении ради лучших условий . Важно отметить, что самодизайн не следует путать с самоорганизацией работы. Самоорганизующаяся команда действительно сама решает, как выполнять задачи в рамках спринта, но обычно не выбирает, кто входит в состав участников. Самодизайн же делегирует командам часть полномочий, которые раньше считались зоной менеджмента .

Кроссфункциональность означает, что в команде собраны все необходимые компетенции для выполнения работы от начала до конца без привлечения внешних специалистов. В контексте разработки программного обеспечения кросс-функциональная команда может включать аналитика, разработчиков, тестировщика, дизайнера и специалиста по эксплуатации. Такая команда способна самостоятельно реализовать функциональность от идеи до поставки

пользователю, не дожидаясь, пока свою часть работы сделает другая команда. Кроссфункциональность ускоряет доставку ценности и снижает потери, связанные с передачей задач между командами .

Самоорганизация – это способность команды самостоятельно определять, как лучше достичь поставленной цели. Самоорганизующаяся команда сама распределяет задачи, сама выбирает способы их решения, сама контролирует свой прогресс и сама корректирует свои действия на основе обратной связи . Принципы самоорганизации тесно связаны с идеями Дэниела Пинка о внутренней мотивации: автономия (желание самому управлять своей работой), мастерство (желание становиться лучше в том, что важно) и цель (желание делать что-то значимое) .

Роли в команде. Модель Белбина

Для эффективной работы команды важно не просто собрать группу профессионалов, но и обеспечить баланс ролей, которые эти профессионалы выполняют в процессе совместной деятельности. Одной из самых известных моделей командных ролей является модель Мередита Белбина. В результате многолетних исследований Белбин выделил девять ролей, которые должны быть представлены в сбалансированной команде.

Первая группа ролей – мыслительные роли. Генератор идей генерирует новые идеи и нестандартные решения. Он креативен, изобретателен, свободно мыслит, но может быть непрактичным и терять интерес, когда идея требует доработки. Наблюдатель-оценщик анализирует идеи и предложения, объективно оценивает их сильные и слабые стороны, стратегичен и проницателен, но может быть излишне критичным и медленным в принятии решений. Специалист – это человек с глубокими знаниями в узкой области, который обеспечивает команду экспертизой, но его вклад ограничен его специализацией.

Вторая группа – исполнительские роли. Председатель-координатор направляет команду к целям, чётко формулирует задачи, принимает решения, хорошо чувствует сильные стороны членов команды. Формирователь – это движущая сила команды, он бросает вызов, преодолевает препятствия, динамичен и амбициозен, но может быть нетерпелив и провоцировать конфликты. Исполнитель превращает идеи в практические действия, дисциплинирован, надёжен, эффективен, но может быть негибким и медленно адаптироваться к изменениям.

Третья группа – командные роли. Душа команды поддерживает гармонию и командный дух, заботится о потребностях коллег, дипломатичен и чуток, но может быть нерешителен в кризисных ситуациях. Исследователь ресурсов ищет внешние возможности, устанавливает контакты, общителен и воодушевляет команду, но быстро теряет интерес после первоначального энтузиазма. Доводчик дорабатывает детали, находит ошибки, контролирует сроки, добросовестен и

педантичен, но может тревожиться по пустякам и испытывать трудности с делегированием.

Модель Белбина не означает, что в команде должны быть все девять ролей в лице разных людей – один человек может выполнять несколько ролей. Важно, чтобы все роли были в той или иной степени покрыты, потому что отсутствие какой-либо роли создаёт дисбаланс. Например, отсутствие генератора идей приводит к стагнации, отсутствие доводчика – к ошибкам и срыву сроков, а отсутствие души команды – к конфликтам и падению морального духа.

Этапы формирования команды. Модель Такмана

Процесс формирования эффективной команды не происходит мгновенно. Любая группа людей, начинающая работать вместе, проходит через несколько закономерных этапов, описанных Брюсом Такманом в его модели развития команды. Модель включает пять стадий, хотя изначально их было четыре, а пятая была добавлена позже.

Первая стадия – формирование, или Forming. На этом этапе участники только знакомятся друг с другом. Они вежливы, избегают конфликтов, пытаются понять, каковы правила и нормы поведения в этой группе. Люди испытывают некоторую неопределённость и тревогу: что от них ожидают, кто здесь главный, как они впишутся в коллектив. На этой стадии команда демонстрирует высокую зависимость от лидера, который должен дать ясные указания и задать направление.

Вторая стадия – бурление, или Storming. Это самый сложный и конфликтный этап. Участники начинают проявлять свои истинные характеры, возникают разногласия по поводу целей, процессов и распределения ролей. Люди спорят, соперничают, формируют подгруппы. На этой стадии многие команды распадаются, потому что не выдерживают напряжения. Задача лидера – помочь команде конструктивно разрешать конфликты, не подавляя их, но и не давая им разрушить команду.

Третья стадия – нормирование, или Norming. Команда преодолевает конфликты и начинает вырабатывать общие нормы и правила работы. Участники договариваются о том, как принимать решения, как распределять задачи, как давать обратную связь. Появляется чувство сплочённости и командного духа. Люди начинают доверять друг другу и помогать коллегам.

Четвёртая стадия – работа, или Performing. Это стадия зрелой, высокоэффективной команды. Команда способна самостоятельно решать сложные задачи, быстро адаптироваться к изменениям и достигать поставленных целей. Взаимодействие происходит легко и естественно, конфликты решаются конструктивно, люди понимают сильные и слабые стороны друг друга. На этой стадии команда нуждается в минимальном внешнем управлении.

Пятая, добавленная позже стадия – расформирование, или Adjourning. Команда завершила свою задачу и готовится к расставанию. Участники могут испытывать чувство потери и тревогу по поводу будущего. Задача лидера – помочь команде завершить работу достойно, зафиксировать достижения, провести ретроспективу и поддержать членов команды в переходный период.

Знание модели Такмана помогает лидерам понимать, на каком этапе находится их команда, и применять соответствующие стили управления. На стадии формирования нужно больше директивности, на стадии бурления – фасилитации конфликтов, на стадии нормирования – поддержки, а на стадии работы – делегирования.

Модель Шнейдера

Модель Шнейдера, также известная как модель культурных аттракторов, описывает четыре типа организационной культуры, которые притягивают разные типы людей. Эта модель полезна для понимания того, почему одни люди хорошо работают в определённых командах, а другие – нет, и как формировать команды, учитывая культурные предпочтения сотрудников.

Первый тип культуры – культура контроля. В такой культуре ценятся предсказуемость, стабильность, порядок и дисциплина. Люди здесь работают с чёткими правилами, процедурами и иерархией. Такой тип культуры характерен для банков, государственных учреждений, производственных предприятий. Сотрудники, предпочитающие этот тип культуры, ценят ясность и определённость.

Второй тип – культура компетенции. Здесь ценятся знания, экспертиза, инновации и совершенство. Люди стремятся быть лучшими в своей области, постоянно учатся и соревнуются в профессионализме. Такая культура характерна для исследовательских центров, высокотехнологичных компаний, университетов. Сотрудники в такой культуре ценят признание своей экспертности и возможность заниматься сложными задачами.

Третий тип – культура сотрудничества. Здесь ценятся командная работа, гармония, взаимопомощь и хорошие отношения. Люди работают вместе, поддерживают друг друга, заботятся о командном духе. Такая культура характерна для некоммерческих организаций, семейных компаний, творческих коллективов. Сотрудники в такой культуре ценят принадлежность к команде и позитивную атмосферу.

Четвёртый тип – культура культивации. Здесь ценятся рост, развитие, самореализация и творчество. Люди стремятся раскрыть свой потенциал, экспериментировать, создавать новое. Такая культура характерна для стартапов, архитектурных бюро, рекламных агентств. Сотрудники в такой культуре ценят свободу и возможность самовыражения.

Модель Шнейдера показывает, что люди склонны выбирать ту культуру, которая соответствует их ценностям, и что разные типы команд требуют разных культурных установок. При формировании команды важно учитывать не только профессиональные компетенции, но и культурные предпочтения участников.

Фичекоманды

Фичекоманды, или feature teams, – это один из способов организации команд в крупных Agile-проектах. В отличие от компонентных команд, которые специализируются на отдельных частях системы (например, команда базы данных, команда пользовательского интерфейса, команда серверной логики), фичекоманды являются кросс-функциональными и берут на себя реализацию конкретной функциональности от начала до конца.

Переход от компонентных команд к фичекомандам является важным этапом Agile-трансформации. В компонентной структуре каждая функция требует координации между несколькими командами, что порождает длительные ожидания, потери при передаче и размытую ответственность. В структуре фичекоманд одна команда обладает всеми необходимыми компетенциями для реализации функции, что позволяет доставлять ценность пользователям намного быстрее.

Практика показала, что переход на фичекоманды может быть непростым, особенно в крупных организациях, где годами выстраивалась компонентная специализация. Однако использование методов, таких как самодизайн команд, может существенно облегчить этот переход. В описанном кейсе группа из сорока человек в ходе воркшопа самостоятельно сформировала пять фичекоманд, что позволило значительно ускорить поставку и повысить вовлечённость сотрудников.

Командные метрики

Измерение эффективности команды является важной, но непростой задачей. Традиционные метрики, такие как количество написанного кода или отработанных часов, плохо отражают реальную ценность, создаваемую командой. В гибких подходах используются иные метрики, которые фокусируются на результате, а не на процессе.

Одной из наиболее известных метрик является скорость команды, или velocity, в Scrum. Скорость показывает, сколько единиц функциональности команда завершает за одну итерацию. Эта метрика полезна для планирования, но её нельзя использовать для сравнения разных команд, поскольку единицы измерения субъективны.

Важными метриками являются время выполнения задачи, или lead time, и время цикла, или cycle time. Lead time – это время от момента поступления запроса до момента его доставки пользователю. Cycle time – это время от

момента, когда команда начала работу над задачей, до момента её завершения. Сокращение этих метрик указывает на повышение эффективности.

Также используются метрики качества, такие как количество дефектов, обнаруженных после релиза, и время восстановления после сбоя. Метрики удовлетворённости команды, измеряемые с помощью регулярных опросов, показывают моральное состояние и риск выгорания. Метрики бизнес-ценности, которые сложнее всего измерить, пытаются оценить, насколько созданная функциональность повлияла на бизнес-показатели.

Важно помнить, что любая метрика может стать вредной, если она используется для наказания команды или сравнения команд между собой. Метрики должны служить инструментом для улучшения, а не средством контроля . Команды должны сами участвовать в выборе метрик и анализировать их динамику на ретроспективах, чтобы принимать решения о том, что и как улучшить.

ТЕМА 5. ИНЖЕНЕРНЫЕ ПРАКТИКИ В ГИБКОЙ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Инженерные практики в разработке: программирования, интеграции, управления, командные

Инженерные практики играют ключевую роль в гибкой разработке программного обеспечения. Если Agile-манифест задаёт ценности и принципы, а фреймворки, такие как Scrum, определяют роли и процессы, то именно инженерные практики обеспечивают техническое качество продукта и позволяют команде двигаться быстро, не накапливая технический долг. Эти практики можно разделить на несколько категорий в зависимости от того, на какой аспект разработки они направлены.

Практики программирования касаются того, как разработчики пишут код. Сюда относятся разработка через тестирование, или TDD, парное программирование, рефакторинг, простота дизайна, работа с кодовой базой через небольшие частые коммиты и следование стандартам кодирования. Эти практики направлены на то, чтобы код был чистым, понятным, тестируемым и легко изменяемым.

Практики интеграции фокусируются на том, как изменения, сделанные разными разработчиками, объединяются в общую кодовую базу. Главной практикой здесь является непрерывная интеграция, или CI, которая требует, чтобы разработчики интегрировали свой код несколько раз в день, а каждое изменение автоматически проверялось сборкой и тестами. Сюда также относится управление версиями с помощью систем, таких как Git, и стратегии ветвления.

Практики управления связаны с тем, как команда организует свою работу и взаимодействует с заказчиком. Сюда относятся короткие итерации, планирование и демонстрации результатов, ретроспективы для непрерывного улучшения, визуализация задач с помощью досок Kanban или Scrum и метрики для отслеживания прогресса.

Командные практики касаются взаимодействия внутри команды. Это парное программирование, когда два разработчика работают за одним компьютером, коллективное владение кодом, когда любой член команды может править любой участок кода, работа в общем пространстве, или open space, ежедневные стендапы и культура конструктивной обратной связи. Все эти практики в совокупности создают среду, в которой высокое качество продукта и быстрая скорость разработки не противоречат друг другу, а наоборот, поддерживают друг друга.

Экстремальное программирование

Экстремальное программирование, или Extreme Programming, сокращённо XP, является одной из наиболее технически ориентированных гибких

методологий. XP была разработана Кентом Беком в конце девяностых годов и представляет собой набор инженерных практик, доведённых до экстремального уровня. Название отражает суть методологии: если какая-то практика полезна, её нужно делать постоянно и в максимально возможной степени.

XP возникла как реакция на проблемы традиционной разработки, где изменения требований стоили очень дорого, качество кода неуклонно падало со временем, а разработчики тратили большую часть времени на отладку, а не на создание новой ценности. XP предлагает альтернативу: делать короткие итерации, писать тесты до кода, программировать в парах, постоянно улучшать дизайн и интегрировать код несколько раз в день. Несмотря на свою «экстремальность», многие практики XP стали стандартом де-факто в современной разработке, особенно практика разработки через тестирование и непрерывная интеграция.

Ценности, принципы и практики XP

Экстремальное программирование базируется на пяти основных ценностях. Первая ценность – коммуникация. В XP коммуникации придаётся исключительное значение, поскольку большая часть проблем в разработке возникает именно из-за недопонимания. Парное программирование, ежедневные стендапы, коллективное владение кодом – все эти практики направлены на то, чтобы информация свободно циркулировала внутри команды и между командой и заказчиком. Вторая ценность – простота. XP призывает делать самое простое решение, которое работает сейчас, и не добавлять функциональность, которая может понадобиться в будущем. Простота позволяет двигаться быстрее и легче адаптироваться к изменениям. Третья ценность – обратная связь. В XP обратная связь поступает по множеству каналов: тесты дают мгновенную обратную связь о корректности кода, заказчик демонстрирует готовую функциональность в конце каждой итерации, парное программирование даёт непрерывную обратную связь о качестве кода. Четвёртая ценность – смелость. Смелость в XP означает готовность вносить изменения, даже если они затрагивают уже работающий код, готовность отказаться от неправильного решения, даже если в него уже вложены усилия, и готовность говорить правду о статусе проекта. Пятая, добавленная позже ценность – уважение. Уважение к коллегам означает, что каждый член команды заслуживает доверия и поддержки, и что никто не должен совершать действий, которые ухудшат работу других, например ломать сборку или оставлять нечитаемый код.

На основе этих ценностей XP формулирует двенадцать принципов, которые уже упоминались в контексте Agile-манифеста, но в XP они приобретают конкретное техническое наполнение. Ключевой принцип – быстрая обратная связь: чем быстрее команда узнаёт о проблеме, тем дешевле её исправить.

Практики XP являются воплощением этих ценностей и принципов. К основным практикам XP относятся следующие. Парное программирование, когда два разработчика работают за одним компьютером: один пишет код, а второй в реальном времени его проверяет, предлагает улучшения и думает о более высокоуровневых аспектах. Разработка через тестирование, или TDD, когда сначала пишется автоматический тест, который не проходит, а затем пишется минимальный код, достаточный для прохождения теста. Рефакторинг, то есть постоянное улучшение структуры кода без изменения его поведения. Простой дизайн, означающий создание самого простого решения, которое работает на текущий момент. Непрерывная интеграция, когда код всех разработчиков объединяется и тестируется несколько раз в день. Коллективное владение кодом, когда любой член команды может изменить любой участок кодовой базы. Стандарты кодирования, чтобы код выглядел так, как будто его писал один человек. Метафора системы, то есть общая для всей команды понятная история, описывающая, как устроена система. Планирование игры, когда заказчик и разработчики совместно планируют содержание следующей итерации. Короткие итерации, обычно от одной до трёх недель. Постоянная доступность заказчика, или on-site customer, который всегда готов ответить на вопросы команды. И наконец, работа с устойчивой скоростью, чтобы команда не перерабатывала и могла поддерживать темп бесконечно долго.

Важно понимать, что практики XP не являются независимыми – они поддерживают друг друга. Например, без разработки через тестирование рефакторинг становится слишком рискованным. Без непрерывной интеграции парное программирование теряет половину своей эффективности. Без коллективного владения кодом команда не может быстро реагировать на изменения. Поэтому XP рекомендуется внедрять целиком, а не выбирать отдельные практики.

Дилемма проектирования

Дилемма проектирования, также известная как дилемма Бекса, является классической проблемой в разработке программного обеспечения, которую XP решает своим подходом. Суть дилеммы заключается в следующем. С одной стороны, предварительное проектирование, когда архитектура тщательно продумывается до написания кода, позволяет избежать дорогостоящих переделок в будущем. С другой стороны, в условиях изменяющихся требований значительная часть предварительных проектных решений может оказаться ненужной или даже вредной, потому что требования изменятся, и спроектированная архитектура перестанет соответствовать реальным потребностям.

Традиционный подход, характерный для водопадной модели, делает ставку на детальное проектирование на старте. XP предлагает иной путь: эволюционное

проектирование, при котором архитектура системы развивается постепенно по мере того, как команда узнаёт больше о предметной области и потребностях пользователей. Ключевой практикой, позволяющей реализовать эволюционное проектирование, является рефакторинг. Рефакторинг – это процесс изменения внутренней структуры кода без изменения его внешнего поведения. Благодаря рефакторингу команда может начать с очень простого дизайна, который работает, а затем, по мере добавления новой функциональности, постепенно улучшать архитектуру, сохраняя её чистой и гибкой.

Дилемма проектирования разрешается в XP с помощью принципа «сделай самое простое решение, которое работает, и будь готов изменить его, когда появятся новые требования». При этом XP не отрицает проектирование полностью – перед началом работы над сложной функциональностью разработчики могут провести быстрый дизайн-сешн с доской и маркерами. Но это проектирование является «достаточно хорошим», а не исчерпывающим. Оно даёт направление, но не заковывает команду в жёсткие рамки.

CI, CD, TDD, DevOps

Четыре понятия – CI, CD, TDD и DevOps – являются краеугольными камнями современной гибкой разработки и часто упоминаются вместе, хотя обозначают разные, но взаимосвязанные практики.

TDD, или Test-Driven Development, разработка через тестирование, как уже упоминалось, является практикой написания кода, при которой тест пишется до реализации. TDD работает по короткому циклу, который называется «красный-зелёный-рефакторинг». Сначала разработчик пишет тест, который заведомо не проходит, потому что тестируемая функциональность ещё не реализована. Цвет «красный» означает, что тест упал. Затем разработчик пишет минимальный код, достаточный для того, чтобы тест прошёл. Цвет «зелёный» означает, что все тесты проходят. Затем, с уверенностью, что тесты защищают от регрессий, разработчик выполняет рефакторинг, то есть улучшает структуру кода, не меняя его поведения. Цикл занимает от нескольких секунд до нескольких минут и повторяется постоянно. TDD приводит к нескольким важным результатам. Во-первых, создаётся полный набор автоматических тестов, который позволяет смело вносить изменения, не боясь что-то сломать. Во-вторых, код, написанный через TDD, как правило, является модульным и слабо связанным, потому что писать тесты для сильно связанного кода очень трудно. В-третьих, TDD служит в качестве дизайна: процесс написания теста заставляет разработчика думать об интерфейсе класса или функции до того, как он будет реализован, что часто приводит к лучшему дизайну.

CI, или Continuous Integration, непрерывная интеграция, это практика, при которой все разработчики регулярно, несколько раз в день, интегрируют свои изменения в общую кодовую базу. Каждая интеграция автоматически

проверяется: код компилируется или собирается, прогоняются все автоматические тесты, выполняются статические анализаторы кода и другие проверки. Если что-то идёт не так, интеграция считается «сломанной», и команда обязана исправить проблему в первую очередь. Ключевые принципы CI включают поддержание единой кодовой базы в системе контроля версий, автоматизацию сборки, автоматизацию тестирования и фиксацию изменений в основной ветке не реже одного раза в день. CI решает проблему «ада интеграции», когда разработчики работали независимо неделями или месяцами, а затем мучительно объединяли свои изменения, сталкиваясь с сотнями конфликтов и ошибок. При CI интеграция происходит постоянно и в небольших масштабах, поэтому проблемы обнаруживаются и исправляются почти сразу после их возникновения.

CD имеет два значения, которые дополняют друг друга. Первое значение – это Continuous Delivery, непрерывная поставка. Continuous Delivery означает, что код готов к развёртыванию в любой момент. Другими словами, автоматизированный конвейер сборки и тестирования построен таким образом, что прохождение всех проверок гарантирует, что данная версия кода может быть безопасно развёрнута в продуктивной среде. Само развёртывание при этом может оставаться ручным действием – например, бизнес-заказчик нажимает кнопку, когда хочет выпустить новую версию. Второе значение – это Continuous Deployment, непрерывное развёртывание. Оно идёт на шаг дальше: каждая версия кода, успешно прошедшая все автоматические проверки, автоматически развёртывается в продуктивной среде без участия человека. Continuous Deployment возможен только при очень высоком уровне автоматизации тестирования и уверенности в качестве кода. И Continuous Delivery, и Continuous Deployment являются продолжением идеи CI и требуют хорошо настроенного CI-конвейера. В современной индустрии CD также часто ассоциируется с GitOps – подходом к управлению конфигурациями, использующим Git и автоматические операторы, такие как ArgoCD, которые синхронизируют состояние продуктивной среды с конфигурацией, хранящейся в Git-репозитории.

DevOps – это более широкая концепция, чем CI, CD или TDD. DevOps, образованное от слияния слов Development и Operations, представляет собой и набор практик, и культурную философию, направленную на объединение команд разработки и эксплуатации. Традиционно эти две группы были разделены: разработчики писали код и стремились к быстрым изменениям, а эксплуатационники, или админы, заботились о стабильности систем и стремились к минимальным изменениям. Это порождало конфликт целей и длительные циклы передачи. DevOps устраняет этот разрыв, объединяя обе функции в кросс-функциональные команды и автоматизируя всё, что связано со сборкой, тестированием, развёртыванием и мониторингом.

Ключевые практики DevOps включают автоматизацию инфраструктуры с помощью кода, то есть Infrastructure as Code, или IaC, когда сервера, сети и другие компоненты описываются в файлах конфигурации, которые хранятся в системе контроля версий наравне с кодом приложения. Сюда же входят автоматизированное тестирование среды, мониторинг и оповещение об инцидентах, сбор и агрегация логов, а также культура «blameless post-mortems», то есть разбора инцидентов без поиска виноватых, с целью улучшения системы, а не наказания людей. Отдельно стоит упомянуть практику feature flags, или фича-флагов, которая позволяет включать и выключать функциональность на лету, не выполняя развёртывание новой версии приложения. Feature flags стали одним из стандартов индустрии и органично дополняют практики CD, позволяя доставлять неактивные фичи в продуктивную среду без риска для пользователей, а затем включать их в нужный момент.

Все четыре рассмотренные практики – TDD, CI, CD и DevOps – тесно переплетены. TDD даёт уверенность в качестве кода на микроуровне. CI обеспечивает частую и безопасную интеграцию изменений. CD доставляет эти изменения в продуктивную среду быстро и надёжно. DevOps создаёт культуру и инфраструктуру, в которой всё это возможно. Вместе они формируют основу для тех мощных темпов разработки, которые демонстрируют современные высокоэффективные IT-команды.

ТЕМА 6. ПЕРЕХОД ОРГАНИЗАЦИИ К AGILE – ВЫБОР, ПИЛОТИРОВАНИЕ, МАСШТАБИРОВАНИЕ, ТРАНСФОРМАЦИЯ

Выбор оптимальной методологии

Выбор методологии разработки программного обеспечения не может быть универсальным для всех проектов и организаций. Оптимальный подход зависит от множества факторов, включая характер проекта, степень неопределённости требований, размер команды, организационную культуру и требования к надёжности и безопасности. Не существует единственной «лучшей» методологии – есть методологии, которые лучше или хуже подходят для конкретного контекста. Поэтому перед началом Agile-трансформации или выбора методологии для конкретного проекта необходимо проанализировать ситуацию и принять осознанное решение.

Ключевыми факторами, влияющими на выбор методологии, являются следующие. Степень определённости требований: если требования полностью известны и стабильны на старте, можно рассмотреть традиционные подходы. Если требования будут меняться, гибкие методологии подходят лучше. Размер и распределённость команды: для небольших команд, работающих в одном помещении, хорошо подходят Scrum и XP. Для крупных распределённых команд могут потребоваться фреймворки масштабирования, такие как SAFe или LeSS. Критичность системы: для систем, где ошибки могут привести к человеческим жертвам, например в авиации или медицине, может потребоваться более формальный подход с обширной документацией и верификацией. Культура организации: если организация привыкла к жёсткой иерархии и формальным процедурам, внедрение Agile может встретить сопротивление, и выбирать более постепенные стратегии.

Важно понимать, что одна организация может одновременно использовать разные методологии для разных типов проектов. Например, для поддержки унаследованной критической системы может использоваться водопадная модель с строгим контролем изменений, а для разработки нового инновационного сервиса – Scrum или Kanban. Способность сочетать разные подходы и выбирать подходящий инструмент под задачу является признаком зрелой организации.

Сравнение подходов

Для осознанного выбора методологии полезно сравнивать различные подходы по ряду критериев. Водопадная модель предлагает предсказуемость и дисциплину, но негибка и плохо подходит для проектов с меняющимися требованиями. Scrum обеспечивает регулярную обратную связь и адаптивность, но требует высокой вовлечённости команды и заказчика. Kanban даёт гибкость и визуализацию потока, но не предоставляет фиксированных итераций и ролей. XP

обеспечивает высокое техническое качество через инженерные практики, но требует высокой дисциплины и может быть ресурсоёмкой. Lean фокусируется на устранении потерь и потоке ценности, но требует зрелой команды, способной самостоятельно идентифицировать потери.

Практика показывает, что не существует единственного правильного выбора. Как отмечают эксперты, методология должна выбираться в зависимости от состояния системы, в рамках которой реализуется проект . Для простых систем с очевидными причинно-следственными связями оптимальны контрольные списки и стандартные процедуры. Для упорядоченных систем, где связи уже не очевидны и требуют экспертного анализа, подходят методы классического проектного управления, такие как Waterfall, PMI или PRINCE2. Для запутанных систем, где причинно-следственные связи не ясны, а результат непредсказуем, наиболее эффективным является Agile, поскольку он позволяет строить гипотезы и проверять их в коротких итерациях .

Модель бимодальных ИТ

Концепция бимодальных ИТ, или двухскоростных информационных технологий, была предложена аналитической компанией Gartner для описания ситуации, когда в организации сосуществуют два режима ИТ-деятельности, требующих разных подходов к управлению .

Первый режим, или mode 1, ориентирован на стабильность, предсказуемость и надёжность. В этом режиме работают системы, которые являются критически важными для функционирования бизнеса: процессинговые системы банков, системы управления запасами, бухгалтерские системы. Требования к таким системам обычно хорошо известны и редко меняются. Ошибки могут иметь серьёзные последствия. Поэтому здесь применяются традиционные методологии с длительными циклами планирования, тщательным тестированием и строгим контролем изменений. Это мир «водопада», ITIL и долгосрочного планирования.

Второй режим, или mode 2, ориентирован на инновации, скорость и адаптивность. В этом режиме разрабатываются новые цифровые сервисы, мобильные приложения, интернет-порталы, где требования постоянно меняются, а время выхода на рынок является критическим фактором. Здесь применяются гибкие методологии, такие как Scrum и Kanban, короткие итерации, постоянная обратная связь от пользователей. Этот режим характеризуется высокой степенью неопределённости, нелинейным характером причинно-следственных связей и слабой предсказуемостью результатов .

Основная сложность бимодального подхода заключается в том, как совместить эти два режима в рамках одной организации. В реальной жизни mode 1 нельзя рассматривать как нечто незыблемое на все времена. Как отмечают эксперты, от шестидесяти до семидесяти процентов изменений, внесённых в

процессы mode 2, требуют взаимной синхронизации с процессами mode 1 . Например, новое мобильное приложение для интернет-банка может потребовать изменений в основной банковской системе, которая традиционно развивается медленно и осторожно. Поэтому многие организации движутся от бимодальности к более интегрированным моделям, где гибкость проникает и в традиционные ИТ, но с учётом их специфики.

Agile-трансформация

Agile-трансформация представляет собой гораздо более глубокий процесс, чем простое внедрение Scrum или Kanban в нескольких командах. Это системное изменение того, как организация создаёт ценность, принимает решения, распределяет ресурсы и оценивает успех. Agile-трансформация затрагивает структуру организации, процессы, системы мотивации, культуру и менталитет сотрудников. Она может длиться годы и требует сильной поддержки высшего руководства.

Важным выводом из практики Agile-трансформаций является необходимость идти шаг за шагом, а не пытаться изменить всё и сразу. Например, опыт трансформации в Cisco показал, что сначала необходимо сформировать Agile-навыки в командах, и только затем пытаться изменять полный жизненный цикл разработки программного обеспечения . Также было отмечено, что критически важно получить и поддерживать поддержку лидерства, а также учитывать влияние инструментов и систем на внедрение Agile .

Agile-трансформация также тесно связана с изменениями в организационной культуре, что будет рассмотрено далее. Без изменения культуры даже самое правильное внедрение процессов может остаться поверхностным и не дать ожидаемых результатов.

Организационная культура организации

Организационная культура является одним из самых мощных, но часто недооцениваемых факторов, влияющих на успех Agile-трансформации. Культура – это набор разделяемых ценностей, убеждений, норм и артефактов, которые определяют, как люди ведут себя в организации и как принимаются решения. Если Agile-практики вступают в противоречие с доминирующей культурой, они либо отторгаются, либо формально выполняются, но не приносят пользы. Поэтому перед началом трансформации важно «прощупать» текущую культуру, понять её сильные стороны и возможные конфликты с Agile-ценностями. Как отмечается в практических руководствах, для понимания культуры, которую вы пытаетесь изменить, необходимо сначала измерить текущее состояние и затем сопоставить его с моделью культуры .

Модель Шнейдера

Модель Шнейдера, разработанная Уильямом Шнейдером, описывает четыре типа организационной культуры, которые определяются по двум осям. Первая ось показывает ориентацию организации: на компанию или на людей. Вторая ось показывает фокус внимания: на реальность, то есть на то, что есть сейчас, или на возможность, то есть на то, что может быть в будущем .

Первый тип культуры – культура сотрудничества, или collaboration, находится в квадранте «ориентация на людей» и «фокус на реальность». В такой культуре ценятся командная работа, гармония, взаимопомощь и хорошие отношения. Люди работают вместе, поддерживают друг друга, заботятся о командном духе. Решения часто принимаются консенсусом.

Второй тип – культура контроля, или control, находится в квадранте «ориентация на компанию» и «фокус на реальность». Здесь ценятся предсказуемость, стабильность, порядок и дисциплина. Люди работают по чётким правилам и процедурам, иерархия важна. Решения принимаются на верхних уровнях и спускаются вниз. Отчётность и контроль являются ключевыми механизмами.

Третий тип – культура компетенции, или competence, находится в квадранте «ориентация на компанию» и «фокус на возможность». Здесь ценятся знания, экспертиза, инновации и совершенство. Люди стремятся быть лучшими в своей области, постоянно учатся и соревнуются в профессионализме. Принимаются решения, основанные на экспертизе и лучших практиках.

Четвёртый тип – культура культивации, или cultivation, находится в квадранте «ориентация на людей» и «фокус на возможность». Здесь ценятся рост, развитие, самореализация и творчество. Люди стремятся раскрыть свой потенциал, экспериментировать, создавать новое. Решения часто принимаются на основе видения и интуиции.

Agile и Scrum, как отмечают эксперты, являются в первую очередь коллаборативными, то есть культурой сотрудничества, с вторичным фокусом на культивацию . Это означает, что Agile-команды процветают в среде, где ценятся сотрудничество, самоорганизация, доверие и непрерывное обучение. Если основная культура организации – это культура контроля с её иерархией, формальными отчётами и микроменеджментом, внедрение Agile столкнётся с серьёзными препятствиями. Культура контроля может подавлять инициативу, страховать команды от принятия решений и требовать предсказуемости там, где её не может быть . Поэтому Agile-трансформация часто требует не только изменения процессов, но и эволюции культуры в сторону большей коллаборации и культивации.

Модель Кеневин

Модель Кеневин, или Cynefin framework, была разработана Дэйвом Сноуденом в две тысячи третьем году для классификации систем и проблем по характеру причинно-следственных связей и помощи в выборе соответствующих стратегий принятия решений . Модель выделяет четыре домена, а также пятый домен беспорядка.

Первый домен – простые системы, или Simple. В простых системах причинно-следственные связи очевидны любому агенту системы. Они просты, предсказуемы, повторяемы и понятны. В такой системе, если вы сделаете А, вы всегда получите Б. Стратегия действий здесь: определить проблему, категоризировать её, применить наилучшую практику. Примером может служить стандартная процедура регистрации пользователя или инструкция по сборке мебели.

Второй домен – упорядоченные системы, или Complicated. В упорядоченных системах причинно-следственные связи существуют, но они не очевидны. Для их обнаружения требуется экспертный анализ. Разные эксперты могут предложить разные решения, и все они могут быть верными. Стратегия действий: определить проблему, проанализировать возможные причины, применить хорошую практику. Примером может служить диагностика поломки автомобиля или проектирование архитектуры здания.

Третий домен – запутанные системы, или Complex. В запутанных системах причинно-следственные связи не ясны и не определёны. Система состоит из множества взаимодействующих агентов, и результат их взаимодействия непредсказуем. Два одинаковых действия могут дать разные результаты. Стратегия действий: не пытаться предсказать, а проводить эксперименты, строить гипотезы, проверять их и адаптироваться. Это область «зондируй-чувствуй-реагируй», то есть probe-sense-respond. Примером может служить вывод нового продукта на рынок или управление командой творческих специалистов.

Четвёртый домен – хаотичные системы, или Chaotic. В хаотичных системах причинно-следственных связей нет вообще. Ситуация кризисная, факты отсутствуют или бессмысленны. Стратегия действий: действовать немедленно, чтобы стабилизировать ситуацию, вывести систему из хаоса, а затем уже анализировать. Это область «действуй-чувствуй-реагируй», то есть act-sense-respond. Примером может служить реагирование на кибератаку или на внезапное падение сервера в продакшене.

Пятый домен – беспорядок, или Disorder. Это состояние, когда непонятно, в каком из четырёх доменов вы находитесь.

Модель Кеневин имеет прямое отношение к выбору методологии разработки . Для простых систем оптимальны контрольные списки и стандартные операционные процедуры. Для упорядоченных систем хорошо подходят

традиционные методы проектного управления, такие как Waterfall и PRINCE2. Для запутанных систем, где неопределённость высока, а результат непредсказуем, Agile является наиболее подходящим подходом, поскольку позволяет строить гипотезы и проверять их в коротких итерациях. Хаотичные системы требуют немедленных действий по стабилизации, и здесь могут применяться любые методы, которые помогут остановить кризис.

Понимание модели Кеневин помогает организациям избегать ошибки применения Agile там, где он не нужен. Не все проекты требуют гибкости. Если вы разрабатываете систему, где все требования известны и не изменятся, а цена ошибки очень высока, использование Agile может быть неоправданным. И наоборот, если вы работаете в зоне высокой неопределённости, попытка применить жёсткое планирование обречена на провал.

8 шагов Коттера

Модель восьми шагов Джона Коттера является одной из наиболее известных и практичных моделей управления организационными изменениями. Профессор Гарвардской школы бизнеса Джон Коттер предложил эту модель в тысяча девятьсот девяносто пятом году, и с тех пор она широко используется для проведения трансформаций в компаниях по всему миру . Модель особенно эффективна в условиях кризиса, когда компания теряет позиции и должна срочно адаптироваться к новым условиям. Коттер подчёркивает, что любые масштабные изменения требуют чёткого плана и постепенного внедрения, и его модель помогает не просто внедрить реформу, а изменить мышление внутри компании, чтобы новое стало частью повседневной работы . Опыт Agile-трансформаций, например в Cisco, подтверждает полезность следования структурированному процессу изменений, такому как модель Коттера, для осознания текущего состояния и определения того, что нужно сделать в первую очередь .

Первый шаг – создать чувство срочности, или Create a sense of urgency. Изменения начинаются только тогда, когда компания понимает, что иначе нельзя. Необходимо показать команде реальные угрозы: данные, тренды, мнения потребителей. Без этого мотивация будет недостаточной, а реформы быстро заглохнут . В контексте Agile-трансформации это означает донесение до всей организации, что рынок меняется, конкуренты становятся быстрее, и традиционные методы разработки уже не позволяют успевать за требованиями клиентов.

Второй шаг – сформировать коалицию лидеров изменений, или Build a guiding coalition. Нельзя внедрять изменения в одиночку. Нужна группа людей, способных вести за собой других, – авторитетных, энергичных, готовых взять ответственность. Такая команда становится ядром изменений, принимает ключевые решения и вдохновляет остальных . В Agile-трансформации это могут

быть Agile-чемпионы, Scrum-мастеры и лидеры, которые первые внедряют новые практики и помогают другим.

Третий шаг – разработать стратегическое видение и инициативы, или Form a strategic vision and initiatives. Изменения невозможны без ясного видения: куда движется компания и зачем. Видение – это конкретное понимание будущего состояния бизнеса. На этом этапе формируются ключевые инициативы, которые помогут достичь цели . Видение Agile-трансформации может звучать так: «Мы хотим сократить время выхода новой функциональности на рынок с шести месяцев до двух недель, повысив при этом качество и удовлетворённость клиентов».

Четвёртый шаг – донести до сотрудников своё видение изменений, или Enlist a volunteer army. На этом этапе важно не только сформулировать видение, но и донести его до каждого сотрудника. Коммуникация должна быть простой, понятной и вдохновляющей. Цель – чтобы каждый понял, зачем нужны изменения и какую роль он в них играет . В Agile-трансформации это проведение презентаций, обучение, создание внутренних сообществ практиков.

Пятый шаг – устранить препятствия, или Enable action by removing barriers. Даже при хорошем плане могут возникнуть барьеры: устаревшие процессы, сопротивление сотрудников или нехватка ресурсов. На этом этапе важно выявить и устранить эти препятствия, чтобы ничего не мешало внедрению изменений . В Agile-трансформации это может включать изменение системы бюджетирования, которая требует годового планирования всех проектов, или пересмотр системы оценки эффективности, которая поощряет индивидуальные, а не командные результаты.

Шестой шаг – планировать краткосрочные победы, или Generate short-term wins. Люди верят в изменения, когда видят реальные результаты. Поэтому важно планировать и достигать краткосрочных успехов, которые демонстрируют правильность выбранного пути и мотивируют команду . В Agile-трансформации это может быть успешное завершение первого спринта одной пилотной командой, или досрочная поставка функциональности, или значительное улучшение качества.

Седьмой шаг – закрепить достигнутые изменения и продолжать движение, или Sustain acceleration. Изменения – это долгий процесс, требующий постоянного развития. После первых успехов важно не останавливаться: нужно ставить новые цели, внедрять свежие идеи и укреплять доверие к переменам. Коттер считал, что успех приходит через настойчивую работу, регулярную обратную связь и постоянные небольшие улучшения . В Agile-трансформации это означает расширение числа Agile-команд, углубление инженерных практик и постепенное изменение культуры во всей организации.

Восьмой шаг – интегрировать новые подходы в корпоративную культуру, или Institute change. Чтобы изменения были устойчивыми, они должны стать

частью корпоративной культуры. Это означает, что новые ценности, нормы и практики закрепляются в повседневной работе компании, в системах мотивации, в процессах найма и развития персонала . В Agile-трансформации это финальный этап, когда гибкость становится не «проектом», а образом мышления всей организации.

Модель Коттера подчёркивает, что изменения – это не одномоментное событие, а длительный процесс, и успех зависит от прохождения всех шагов в правильном порядке. Пропуск или недооценка какого-либо шага может привести к тому, что изменения не приживутся или даже провалятся.

II. ПРАКТИЧЕСКИЙ РАЗДЕЛ

ПЛАНЫ И ЗАДАНИЯ ДЛЯ СЕМИНАРСКИХ ЗАНЯТИЙ

Семинарское занятие 1. Существующие методологии разработки ПО

Вопросы для подготовки:

1. Каковы основные этапы водопадной (каскадной) методологии и в каких проектах её применение оправдано?
2. В чём заключается ключевое отличие V-образной методологии от классической водопадной модели?
3. Чем различаются инкрементальная и итерационная методологии разработки?
4. Каковы основные преимущества и ограничения RAD (быстрой) методологии разработки?
5. Дайте характеристику понятиям «тяжеловесные» и «облегчённые» процессы разработки ПО.

Контрольные вопросы по теме:

1. Перечислите и кратко охарактеризуйте известные вам методологии разработки ПО.
2. Назовите преимущества и недостатки водопадной методологии.
3. Для каких проектов подходит V-образная методология, а для каких нет?
4. В чём заключается суть инкрементальной модели разработки?
5. Опишите сферу применения и ограничения RAD-методологии.
6. Каковы основные ценности и принципы гибкой методологии (Agile)?
7. Что такое компонентно-ориентированная методология и в чём её сильные стороны?
8. Дайте сравнительную характеристику тяжёловесных и облегчённых процессов.
9. Каковы основные практики XP (экстремального программирования)?
10. Сформулируйте принципы бережливой разработки программного обеспечения.

Семинарское занятие 2. Введение в гибкие методы управления проектами и продуктами

Вопросы для подготовки:

1. Что такое VUCA-мир и как он повлиял на появление Agile?
2. Назовите четыре ключевые ценности Agile-манифеста.
3. Каковы три уровня в структуре Agile (менталитет, философия, практики)?
4. В чём разница между Kanban и классическим Lean-подходом?

5. Что такое цикл PDCA (Деминга) и как он связан с непрерывным улучшением?

Контрольные вопросы по теме:

1. Каковы исторические предпосылки возникновения Agile?
2. Охарактеризуйте VUCA-мир и его влияние на управление проектами.
3. Что такое модель 4К и как она связана с гибкими методологиями?
4. Перечислите и раскройте четыре ценности Agile-манифеста.
5. Назовите любые пять принципов из двенадцати принципов Agile.
6. Опишите модели взаимодействия бизнеса и ИТ в традиционном и гибком подходах.
7. Приведите примеры применения Agile вне ИТ-сферы.
8. Что такое Kanban и в чём его отличие от Scrum?
9. Охарактеризуйте принципы бережливого производства (Lean).
10. В чём суть принципа Кайдзен (непрерывного улучшения)?

Семинарское занятие 3. Разработка продуктов в цифровую эпоху

Вопросы для подготовки:

1. Что такое цифровой продукт и чем он отличается от цифрового сервиса?
2. Опишите пять этапов методологии дизайн-мышления (Design Thinking).
3. В чём разница между UX и UI-дизайном?
4. Что такое HADI-цикл и как он используется для тестирования гипотез?
5. Дайте определение понятию Customer Development (CustDev) и его основным методам.

Контрольные вопросы по теме:

1. Что такое цифровые продукты и сервисы? Приведите примеры.
2. В чём суть проектирования, ориентированного на пользователя (User-Centered Design)?
3. Опишите концепцию дизайн-мышления и её этапы.
4. Дайте характеристику UX и UI-практикам.
5. Как организуется тестирование гипотез в разработке продуктов?
6. Что такое Lean Canvas? Перечислите и раскройте его девять блоков.
7. В чём суть методологии «Бережливый стартап» и цикла Build-Measure-Learn?
8. Что такое продуктовое видение, дорожная карта и бэклог?
9. Какие продуктовые метрики вы знаете? Что такое метрика Полярной звезды?
10. Что такое персоны, сценарии, Impact Map и User Story Map?

11. Опишите Customer Journey Mapping и его назначение.

12. Что такое Customer Development и какие методы в него входят?

13. Что такое unit-экономика и какие ключевые метрики она использует (CAC, LTV)?

Семинарское занятие 4. Новое лидерство и командная работа

Вопросы для подготовки:

1. В чём суть модели Servant Leadership (лидерство-служение)?
2. Что такое T-shape-специалист и почему он важен для Agile-команд?
3. Какие девять ролей выделяет модель Белбина?
4. Опишите пять стадий формирования команды по модели Такмана.
5. Что такое фичекоманды и чем они отличаются от компонентных команд?

Контрольные вопросы по теме:

1. Каковы особенности новых моделей лидерства в цифровую эпоху?
2. Что такое недирективные способы управления и в чём их преимущества?
3. Раскройте концепцию Servant Leadership.
4. Дайте характеристику T-shape-специалиста.
5. Что такое Менеджмент 3.0 и какие практики он включает?
6. Раскройте понятия самодизайна, кроссфункциональности и самоорганизации команд.
7. Перечислите и охарактеризуйте роли в команде по модели Белбина.
8. Опишите этапы формирования команды по модели Такмана.
9. Что такое модель Шнейдера и как она помогает в формировании команд?
10. Что такое фичекоманды и какова их роль в Agile-трансформации?
11. Какие командные метрики используются в гибкой разработке?

Семинарское занятие 5. Scrum фреймворк

Вопросы для подготовки:

1. Каковы три основных артефакта Scrum (Product Backlog, Sprint Backlog, Increment)?
2. Каковы три основных роли в Scrum (Product Owner, Scrum Master, Development Team)?
3. Что такое спринт и какова его оптимальная продолжительность?
4. Какие события (церемонии) предусмотрены в Scrum?
5. Что такое Definition of Done (DoD) и для чего она нужна?

Контрольные вопросы по теме:

1. Что такое Scrum и какова его роль в семействе Agile?
2. Перечислите и раскройте содержание трёх артефактов Scrum.
3. Охарактеризуйте роли Product Owner, Scrum Master и команды разработки.
4. Что такое спринт и как определяется его длительность?
5. Перечислите и опишите все Scrum-события: планирование спринта, ежедневный стендап, обзор спринта (демо), ретроспектива спринта.
6. Что такое Definition of Done (DoD) и чем она отличается от Definition of Ready (DoR)?
7. Как оцениваются задачи в Scrum (story points, планирование покера)?
8. Как измеряется скорость команды (velocity) в Scrum?
9. Как происходит приоритизация бэклога продукта?
10. Как масштабируется Scrum на несколько команд (Nexus, LeSS, Scrum@Scale)?

Семинарское занятие 6. Инженерные практики в гибкой разработке ПО

Вопросы для подготовки:

1. Каковы четыре ценности экстремального программирования (XP)?
2. В чём суть практики разработки через тестирование (TDD) и её цикла «красный-зелёный-рефакторинг»?
3. Что такое непрерывная интеграция (CI) и какие проблемы она решает?
4. В чём разница между Continuous Delivery и Continuous Deployment (CD)?
5. Что такое DevOps и какие практики он включает (IaC, мониторинг, feature flags)?

Контрольные вопросы по теме:

1. На какие категории делятся инженерные практики в гибкой разработке?
2. Охарактеризуйте методологию экстремального программирования (XP).
3. Назовите ценности, принципы и основные практики XP.
4. В чём заключается дилемма проектирования и как она решается в XP?
5. Что такое TDD? Опишите цикл TDD.
6. Что такое рефакторинг и почему он важен для поддержания чистоты кода?
7. Что такое непрерывная интеграция (CI) и каковы её ключевые принципы?
8. Что такое Continuous Delivery и Continuous Deployment?
9. Что такое DevOps и какие проблемы он решает?

10. Какие инструменты используются для CI/CD (Jenkins, GitLab CI, GitHub Actions)?
11. Что такое Infrastructure as Code (IaC) и зачем он нужен?
12. Что такое feature flags (фича-флаги) и как они связаны с CD?

Семинарское занятие 7. Переход организации к Agile – выбор, пилотирование, масштабирование, трансформация

Вопросы для подготовки:

1. Какие факторы следует учитывать при выборе оптимальной методологии разработки?
2. Что такое бимодальные ИТ (mode 1 и mode 2) и в чём их особенности?
3. Охарактеризуйте четыре типа организационной культуры по модели Шнейдера.
4. Какие четыре домена выделяет модель Кеневин (Cynefin)?
5. Перечислите восемь шагов управления изменениями по Коттеру.

Контрольные вопросы по теме:

1. Как выбирать оптимальную методологию в зависимости от контекста проекта?
2. Проведите сравнительный анализ водопадной, итерационной и гибкой методологий.
3. Что такое бимодальные ИТ и когда они применяются?
4. Что такое Agile-трансформация и какие аспекты организации она затрагивает?
5. Как организационная культура влияет на успех Agile-трансформации?
6. Опишите четыре типа культуры по модели Шнейдера. Какие из них совместимы с Agile?
7. Опишите модель Кеневин. В каком домене применение Agile наиболее эффективно?
8. Перечислите и раскройте восемь шагов управления изменениями Джона Коттера.
9. Как осуществляется пилотирование Agile в организации?
10. Какие существуют фреймворки для масштабирования Agile (SAFe, LeSS, Nexus)?
11. Каковы типичные ошибки и антипаттерны при Agile-трансформации?
12. Как измерить успех Agile-трансформации?

III. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

ТЕСТОВОЕ ЗАДАНИЕ

Вопрос 1. Какая методология предполагает последовательное прохождение фаз анализа, проектирования, реализации и тестирования, где каждая следующая фаза начинается только после полного завершения предыдущей?

А) Scrum
Б) Водопадная (каскадная)
В) Экстремальное программирование
Г) Lean

Вопрос 2. Какая из перечисленных методологий делает акцент на устранении всех видов потерь и создании непрерывного потока ценности?

А) RAD
Б) V-образная
В) Lean
Г) Инкрементальная

Вопрос 3. Что из перечисленного является ценностью Agile-манифеста?

А) Следование первоначальному плану важнее готовности к изменениям
Б) Исчерпывающая документация важнее работающего продукта
В) Люди и взаимодействие важнее процессов и инструментов
Г) Согласование контрактов важнее сотрудничества с заказчиком

Вопрос 4. Как расшифровывается аббревиатура VUCA?

А) Volatility, Uncertainty, Complexity, Ambiguity
Б) Vision, Understanding, Clarity, Action
В) Value, Usability, Cost, Accessibility
Г) Verification, Validation, Certification, Audit

Вопрос 5. Какой этап в методологии дизайн-мышления предполагает создание упрощённых и недорогих версий решения для проверки гипотез?

А) Эмпатия
Б) Фокусировка
В) Генерация идей
Г) Прототипирование

Вопрос 6. Что такое Customer Development (CustDev)?

А) Методология управления разработкой ПО

Б) Подход, при котором развитие продукта строится на основе регулярного общения с клиентами

В) Технология непрерывной интеграции кода

Г) Фреймворк для масштабирования Agile

Вопрос 7. В чём основное отличие UX-дизайна от UI-дизайна?

А) UX отвечает за визуальное оформление, UI – за логику работы

Б) UX отвечает за общий опыт и удобство использования, UI – за визуальную составляющую интерфейса

В) UX используется только в мобильных приложениях, UI – в веб-разработке

Г) UX и UI – это одно и то же понятие

Вопрос 8. Какая роль в модели командных ролей Белбина отвечает за генерацию новых идей и нестандартных решений?

А) Исполнитель

Б) Доводчик

В) Генератор идей

Г) Душа команды

Вопрос 9. Какая стадия формирования команды по модели Такмана характеризуется возникновением конфликтов и разногласий между участниками?

А) Forming

Б) Storming

В) Norming

Г) Performing

Вопрос 10. Что такое T-shape-специалист?

А) Специалист, который работает только в одной узкой области

Б) Специалист, имеющий глубокие знания в одной области и широкий кругозор в смежных

В) Специалист по тестированию программного обеспечения

Г) Менеджер, управляющий несколькими Agile-командами

Вопрос 11. Какая из практик экстремального программирования (XP) предполагает, что два разработчика работают за одним компьютером?

А) Разработка через тестирование (TDD)

Б) Непрерывная интеграция (CI)

В) Парное программирование

Г) Коллективное владение кодом

Вопрос 12. Что означает аббревиатура TDD?

А) Test-Driven Development

Б) Task-Driven Design

В) Technical Debt Document

Г) Total Design Delivery

Вопрос 13. Какая практика предполагает, что все разработчики несколько раз в день интегрируют свои изменения в общую кодовую базу с автоматической проверкой?

А) Continuous Deployment

Б) Continuous Integration (CI)

В) Refactoring

Г) Code Review

Вопрос 14. Что такое DevOps?

А) Инструмент для автоматизации сборки

Б) Фреймворк для управления проектами

В) Культура и набор практик, объединяющих разработку и эксплуатацию

Г) Методология тестирования программного обеспечения

Вопрос 15. Какая модель организационной культуры выделяет такие типы, как культура контроля, культура сотрудничества, культура компетенции и культура культивации?

А) Модель Такмана

Б) Модель Кеневин

В) Модель Шнейдера

Г) Модель Коттера

Вопрос 16. В каком домене модели Кеневин (Cynefin) применение Agile наиболее эффективно?

А) Простые системы (Simple)

Б) Упорядоченные системы (Complicated)

В) Запутанные системы (Complex)

Г) Хаотичные системы (Chaotic)

Вопрос 17. Какой из восьми шагов управления изменениями Коттера предполагает создание у сотрудников понимания необходимости перемен и срочности?

А) Разработать стратегическое видение

Б) Устранить препятствия

В) Создать чувство срочности

Г) Планировать краткосрочные победы

Вопрос 18. Что такое бимодальные ИТ (Bimodal IT)?

А) Сочетание двух методологий: Scrum и Kanban

Б) Сосуществование двух режимов: стабильного (mode 1) и инновационного (mode 2)

В) Использование двух систем контроля версий

Г) Разработка на двух языках программирования одновременно

Вопрос 19. Что такое метрика Полярной звезды (North Star Metric)?

А) Единственный показатель, отражающий ценность продукта для клиента

Б) Метрика, показывающая скорость работы команды

В) Показатель количества дефектов в релизе

Г) Метрика удовлетворённости сотрудников

Вопрос 20. В чём суть методологии «Бережливый стартап» (Lean Startup)?

А) В детальном планировании всех этапов разработки на старте проекта

Б) В быстром тестировании гипотез через цикл Build-Measure-Learn и создание MVP

В) В использовании только бесплатного программного обеспечения

Г) В привлечении максимального количества инвесторов на ранней стадии

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ВОПРОСОВ К ЗАЧЕТУ

1. Основные понятия программной инженерии. Официальная классификация процессов программной инженерии.
2. Базис процессов разработки ПО. Модель «классический жизненный цикл». Инкрементная модель.
3. Базис процессов разработки ПО. Модель «классический жизненный цикл». Спиральная модель.
4. Базис процессов разработки ПО. Модель «классический жизненный цикл». Компонентно-ориентированная модель.
5. Тяжеловесные и облегченные процессы. XP-процесс.
6. Бережливая разработка программного обеспечения. Ликвидация потерь. Встраивание качества. Формирование новых знаний.
7. Бережливая разработка программного обеспечения. Откладывание необратимых решений. Быстрая доставка заказчику. Уважение к людям. Оптимизация целого.
8. Введение в гибкие методы управления проектами и продуктами. Источники и предпосылки появления Agile. VUCA-мир. Модель 4К.
9. Введение в гибкие методы управления проектами и продуктами. Методологии, практики и принципы Agile. Agile-манифест.
10. Введение в гибкие методы управления проектами и продуктами. Модели взаимодействия бизнеса и ИТ. Взаимосвязь Agile-подходов с другими областями знаний.
11. Введение в гибкие методы управления проектами и продуктами. Применение Agile-подходов вне ИТ-сферы. Business Agile, Agile-маркетинг, Agile-HR, Agile в госсекторе, образовании.
12. Введение в гибкие методы управления проектами и продуктами. Kanban, Lean, Кайдзен.
13. Введение в гибкие методы управления проектами и продуктами. Принципы бережливого производства, принципы непрерывного улучшения. Основные практики и ценности.
14. Цифровые продукты и сервисы. Проектирование, ориентированное на пользователя. Концепция дизайн-мышления. UX, UI практики.
15. Подход тестирования гипотез. Lean Canvas. Бережливый стартап.
16. Видение, дорожная карта и бэклог продукта. Продуктовые метрики. Персоны и сценарии. Impact Map. User Story Map.
17. Видение, дорожная карта и бэклог продукта. Customer Journey Mapping. Customer Development. Unit-экономика.
18. Новые модели лидерства. Не директивные способы управления. Servant Leadership. T-shape-специалисты. Менеджмент 3.0 Командная работа.
19. Самодизайн, кроссфункциональность, самоорганизация. Роли в

команде. Модель Белбина.

20. Этапы формирования команды. Модель Такмана. Модель Шнейдера. Фичекоманды. Командные метрики.

21. Scrum фреймворк. Команды и роли, события, артефакты, правила. Scrum Guide. PMI Agile Practice Guide. Ценности, принципы, инкрементальность и итеративность.

22. Инженерные практики в разработке: программирования, интеграции, управления, командные.

23. Экстремальное программирование. Ценности, принципы, практики. Дилемма проектирования. CI, CD, TDD, DevOps.

24. Выбор оптимальной методологии. Сравнение подходов. Модель бимодального ИТ.

25. Agile-трансформация. Организационная культура организации. Модель Шнейдера. Модель Кеневин. 8 шагов Коттера.

IV. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

ПЕРЕЧЕНЬ РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Денисов, А. А. Современные проблемы системного анализа: Информационные основы : учеб. пособие / А. А. Денисов. – СПб. : Изд-во СПбГТУ, 2005. – 295 с.

2. Буч, Гради. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч. – М. : Бином; СПб. : Невский Диалект. 1998.

3. Макконнелл, Стив. Совершенный код. Мастер-класс / Стив Макконнелл. – М. : Русская редакция, 2010. – 896 с.

4. Таненбаум, М. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен. – СПб. : Питер, 2003. – 877 с.

5. Буч, Грейди. UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. – М. : – Изд-во ДМК, 2001 г., 430 с.

6. Ларман, Крэг. Применение UML 2.0 и шаблонов проектирования / Крэг Ларман. – М. : Вильямс, 2008. – 736 с.

7. Фаулер, М. Архитектура корпоративных программных приложений : пер. с англ. / М. Фаулер. – М. : Вильямс, 2006. – 544 с. : ил.

8. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб. : Питер, 2008. – 366 с.