**ЗАТВЕРДЖЕНО**Науково-методичною радою Державного університету «Житомирська політехніка»

протокол від \_\_ \_\_\_\_\_\_\_ 20\_\_ р. №\_\_

**Гнучке управління проектами –**

**Agile project management**

Частина 1

Конспект лекцій

з дисципліни «**Гнучке управління проектами**»

для студентів спеціальностей 051 «Економіка»

освітньо-професійна програма «Економіка»

факультет бізнесу та сфери обслуговування

(назва факультету)

кафедра цифрової економіки та міжнародних економічних відносин

 (назва кафедри)

 **Розробник
Богоявленська Юлія Вячеславівна -** к.е.н., доц.,

Рекомендовано на засіданні кафедри цифрової економіки та міжнародних економічних відносин

 (назва кафедри)

\_\_\_ \_\_\_\_\_\_\_\_\_\_ 2021 р., протокол № \_\_\_

The Digital Business Models syllabus is developed in the framework of ERASMUS+ CBHE project “Digitalization of economic as an element of sustainable development of Ukraine and Tajikistan” / DigEco 618270 EPP-1-2020-1-LT-EPPKA2-CBHE-JP

This project has been funded with support from the European Commission. This document reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained there in.

Цей проект фінансується за підтримки Європейської Комісії. Цей документ відображає лише погляди автора, і Комісія не несе відповідальності за будь-яке використання інформації, що міститься в документі.

Житомир

2021

Гнучке управління проектами
[Електронний ресурс] : Конспект лекцій з дисципліни «Гнучке управління проектами» для здобувачів вищої освіти «Магістр» за спеціальністю: 051 «Економіка». – Житомир : Державний університет «Житомирська політехніка», 2021. – Режим доступу:

Конспект лекцій розроблено в рамках проєкту Erasmus+ “Діджиталізація економіки як елемент сталого розвитку України та Таджикистану (DigEco) 618270-EPP-1-2020-1-LT-EPPKA2-CBHE-JP”/ The proramm is developed in the framework of ERASMUS+ CBHE project “Digitalization of economic as an element of sustainable development of Ukraine and Tajikistan” / DigEco 618270-EPP-1-2020-1-LT-EPPKA2-CBHE-JP

Цей проект фінансується за підтримки Європейської Комісії. Цей документ відображає лише погляди автора, і Комісія не несе відповідальності за будь-яке використання інформації, що міститься в документі/This project has been funded with support from the European Commission. This document reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained there in.

Розробник: к.е.н., Богоявленська Юлія Вячеславівна

 Державний університет «Житомирська політехніка», 2021

 © Ю. В. Богоявленська, 2021

**ЗМІСТ**

[ВСТУП 5](#_heading=h.30j0zll)

[Лекция 1: ВВЕДЕНИЕ В AGILE 6](#_heading=h.tyjcwt)

[1.1 Описание курса 6](#_heading=h.3dy6vkm)

[1.2 Актуальность Agile 6](#_heading=h.2s8eyo1)

[1.3 Что такое Agile? 8](#_heading=h.2jxsxqh)

[1.4 Немного из истории 12](#_heading=h.23ckvvd)

[1.5 Об авторах Agile 14](#_heading=h.4f1mdlm)

[1.6 Как развивалась Agile 19](#_heading=h.3mzq4wv)

[1.7 Agile Manifesto 25](#_heading=h.338fx5o)

[1.8 Применение в различных предметных областях 27](#_heading=h.1a346fx)

[1.9 Краткие выводы 28](#_heading=h.1nia2ey)

[Лекция 2: ПРОЦЕССНЫЙ ПОДХОД КАК КОНКУРЕНТНОЕ ПРЕИМУЩЕСТВО 30](#_heading=h.1f7o1he)

[2.1. Процессный подход 30](#_heading=h.3z7bk57)

[2.2. Значимость процессного офиса во внедрении процессного подхода. Возможные варианты 37](#_heading=h.thw4kt)

[2.3. Движение в направлении гибкости. Пилотные процессы и управление изменениями 42](#_heading=h.4cmhg48)

[2.4. Регламент как основа процессного подхода 44](#_heading=h.2rrrqc1)

[2.5. Краткие выводы 47](#_heading=h.16x20ju)

[Лекция 3: ПРЕДПОСЫЛКИ ВОЗНИКНОВЕНИЯ AGILE 48](#_heading=h.3qwpj7n)

[3.1. Состояние области процессов разработки программного обеспечения 48](#_heading=h.261ztfg)

[3.2. Сравнение каскадного/итерационного/Agile процессов 52](#_heading=h.356xmb2)

[3.3 Эффективная таблетка от болезней? 57](#_heading=h.1kc7wiv)

[3.4 Для кого подходит, а для кого нет? 58](#_heading=h.44bvf6o)

[3.5. Краткие выводы 61](#_heading=h.2jh5peh)

[Лекция 4: ФИЛОСОФИЯ РАБОЧЕГО ПРОЦЕССА 63](#_heading=h.ymfzma)

[4.1. Типы Agile-методологий и их распространенность 63](#_heading=h.3im3ia3)

[4.2. Scrum - гибкий управленческий процесс 67](#_heading=h.4hr1b5p)

[4.3. Как воспитывать сотрудников? 70](#_heading=h.2wwbldi)

[4.4. Как управлять сопротивлением? 73](#_heading=h.1c1lvlb)

[4.5. Чем нужно управлять в Scrum 76](#_heading=h.2b6jogx)

[4.6. Значимость соблюдения процесса 78](#_heading=h.1pgrrkc)

[4.7. Инженерные практики 79](#_heading=h.49gfa85)

[4.8. Краткие выводы 82](#_heading=h.13qzunr)

[Лекция 5: РОЛИ SCRUM 84](#_heading=h.3nqndbk)

[5.1 Команда 84](#_heading=h.22vxnjd)

[5.2 Этапы командообразования 86](#_heading=h.415t9al)

[5.3 Разработчик 90](#_heading=h.3fg1ce0)

[5.4 Scrum-мастер 91](#_heading=h.1ulbmlt)

[5.5 Владелец продукта 92](#_heading=h.4ekz59m)

[5.6 Самоорганизация членов команды 94](#_heading=h.2tq9fhf)

[5.7 Прочие члены команды 95](#_heading=h.18vjpp8)

[5.8 Самоорганизующийся коллектив 96](#_heading=h.3sv78d1)

[5.9 Краткие выводы 99](#_heading=h.n5rssn)

[Лекция 6: ПЛАНИРОВАНИЕ 101](#_heading=h.375fbgg)

[6.1 Принцип быстрого планирования 101](#_heading=h.1maplo9)

[6.2 Поэтапное уточнение планов 103](#_heading=h.46ad4c2)

[6.3 Техника Planning Poker 105](#_heading=h.2lfnejv)

[6.4 Диаграмма сгорания работ 108](#_heading=h.3kkl7fh)

[6.5. Краткие выводы 110](#_heading=h.4jpj0b3)

[Лекция 7: ЭТАПЫ И МЕРОПРИЯТИЯ SCRUM 112](#_heading=h.2yutaiw)

[7.1. Sprint 112](#_heading=h.1e03kqp)

[7.2 Ежедневные встречи (daily) 116](#_heading=h.sabnu4)

[7.3 Груминг бизнес-задач 118](#_heading=h.1rf9gpq)

[7.4. Груминг технических задач 121](#_heading=h.4bewzdj)

[7.5 Обзор Спринта 122](#_heading=h.2qk79lc)

[7.6 Ретроспектива 124](#_heading=h.15phjt5)

[7.7 Краткие выводы 128](#_heading=h.3pp52gy)

[Лекция 8: АТРИБУТЫ SCRUM 129](#_heading=h.24ufcor)

[8.1 Story mapping 129](#_heading=h.jzpmwk)

[8.2. Пользовательские истории (User story) 132](#_heading=h.1j4nfs6)

[8.3. Определение приоритетов пользователей 134](#_heading=h.434ayfz)

[8.4. Доска задач 137](#_heading=h.xevivl)

[8.5 Бэклог продукта 139](#_heading=h.1wjtbr7)

[8.6 Бэклог спринта 142](#_heading=h.4gjguf0)

[8.7 Инкремент продукта 143](#_heading=h.2vor4mt)

[8.8. Принцип прототипирования 144](#_heading=h.1au1eum)

[8.9. Краткие выводы 145](#_heading=h.3utoxif)

[Лекция 9: ОЦЕНКА 147](#_heading=h.29yz7q8)

[9.1 PERT - оценка сроков 147](#_heading=h.p49hy1)

[9.2 Переход от оценки к обязательствам 150](#_heading=h.1o97atn)

[9.3 Сбалансированная система показателей Scrum-команды 152](#_heading=h.488uthg)

[9.4 Наработанная статистика результатов - фундамент прогнозирования и побед 154](#_heading=h.2ne53p9)

[9.5 Краткие выводы 155](#_heading=h.12jfdx2)

[Лекция 10: ИТОГИ И ПЕРСПЕКТИВЫ 157](#_heading=h.3mj2wkv)

[10.1 Сосуществование с альтернативными процессами последовательной разработки программного обеспечения 157](#_heading=h.21od6so)

[10.2 Обеспечение соответствия лучшим практикам и стандартам 158](#_heading=h.gtnh0h)

[10.3 Использование Lean-методологии в Scrum-процессе 161](#_heading=h.30tazoa)

[10.4 Продуктивность Scrum для цифровой трансформации 164](#_heading=h.1fyl9w3)

[10.5 Современная критика Agile 166](#_heading=h.3zy8sjw)

[10.6 Что дальше 168](#_heading=h.2f3j2rp)

# ВСТУП

Agile – это не просто модное веяние, которое является последовательностью определенных шагов, а действенный способ управления человеческим ресурсом, в котором учтены недостатки его предшественников и заложены достоинства, позволяющие владельцам организаций рассчитывать на достижение их ожиданий от применения программных продуктов. Гибкие процессы - это ступень в развитии подходов к управлению проектами, от которых будет зависеть каждая современная успешная компания.

*Знание* и умение оперировать понятиями и атрибутами Agile позволит менеджерам и профильным специалистам быть эффективными в мире создания и реализации проектов любого объема и степени сложности.

Курс адресован широкому кругу специалистов, которые заинтересованы в применении гибких методологий, но акцент сделан на менеджеров и профессионалов, задействованных в сфере разработки проектов информационных систем (т.е. в качестве примеров приведена реализация проектов разработки программного обеспечения). В то же время курс не претендует на звание "единого" источника знаний по Agile.

# Лекция 1: ВВЕДЕНИЕ В AGILE

План

1.1 Описание курса

1.2 Актуальность Agile

1.3 Что такое Agile?

1.4 Немного из истории

1.5 Об авторах Agile

1.6 Как развивалась Agile

1.7 Agile Manifesto

1.8 Применение в различных предметных областях

1.9 Краткие выводы

## 1.1 Описание курса

В этом курсе лекций речь идет о том, что позволит достичь поставленных показателей тем организациям и сотрудникам, которые заинтересованы в конечном результате, а точнее, в его воспроизводимости в поставленных условиях. Ведь именно способность следовать согласованному плану в рамках имеющихся ресурсов отличает компании-однодневки от тех предприятий, которые заинтересованы в стабильном и длительном существовании в своем сегменте бизнеса. Мы поговорим о специализированной процессной методологии Agile.

На сегодня издано достаточное количество литературы по Agile, но большинство авторов своей целевой аудиторией избрали будущих адептов гибких процессов, которые способны с головой погрузиться в их детали. Большинство курсов изобилует техническими и технологическими деталями.

Данный курс лекций преследует цель объективно осветить место и суть Agile в современном мире. Акцент сделан на доступность и наглядность изложения материала не столько для будущих участников гибких процессов, но в значительной степени на тех, кому необходимо понимать, как быть эффективным, используя Agile в своих профессиональных целях.

## 1.2 Актуальность Agile

Актуальность Agile обусловлена множеством различных технологических, организационных и социальных факторов.

Постепенно происходит глобальное смещение парадигмы управленческого мышления со следования тяжеловесным и длительным стратегиям на ситуационно-тактическое поведение. Именно такой тип организации и управления работ позволит успешно лавировать в современных динамичных реалиях окружающего мира.

Разработка длительного и основательного плана может так и не закончиться. В процессе его создания произойдет такое количество изменений, учет и отражение которых займет длительный период времени.

Именно для преодоления кризисов, связанных с дефицитом мыслей, идей, времени, нужно использовать гибкие процессы, суть которых состоит в адаптации к текущим условиям, акценте на работу с людьми и восприятии изменений как неизбежного блага, а не страшной проблемы.

Менеджеры и специалисты, которые изучили Agile и используют ее в своей профессиональной деятельности, безусловно, являются более компетентными и ценными кадрами для любой организации.

Чем Agile может быть полезна для вашей компании?Не стоит думать, что Agile является узкоспециализированной методологией, цель которой сводится к тому, чтобы создавать нужные и эффективные программные продукты. Безусловно, именно так и задумывалось, но после того как результаты ее применения оправдали все возможные ожидания, многие ответвления Agile были адаптированы под другие виды деятельности, и на текущий момент можно констатировать, что гибкие процессные методологии представляют собой определенного вида управленческий framework.

В связи с этим мы надеемся, что каждый *менеджер*, который приступил к изучению данного курса, изучая его, не просто будет пытаться воспринять изложенную тут информацию, но сможет применить приемы и методы, которые посчитает оптимальными для условий своей компании.

Помимо теории вы найдете множество конкретных приемов и практических методов, доказавших свою эффективность в операционной и проектной деятельности.

В сфере информационных технологий вопросы, связанные с процессами разработки программных продуктов, - одни из самых спорных и сложных. Начиная с момента широкого распространения компьютерных технологий, в 60-е годы XX века, специалисты пытаются отыскать универсальное средство, именуемое "серебряная пуля", с помощью которого удастся решить задачу создания сложных программных продуктов *быстро, дешево и качественно*. Его ищут до сих пор и, видимо, будут искать еще очень долго. Скажите, а в других компаниях, где происходит создание и реализация проектной деятельности, не теже цели и задачи?

Процессные методологии разработки информационных систем сменяют друг друга, но при этом сохраняется общая *закономерность*, связанная с фокусированием на оптимизации самого процесса и максимизацией конечного результата.

На текущий момент сложилось два основных противодействующих лагеря.

Адепты первого – это консерваторы, предпочитающие более основательные, "тяжелые" методологии, в которых четко и однозначно определена структура и последовательность выполняемых этапов *работ*. Типичным примером таких методологий является водопадная модель разработки информационных систем.

Приверженцы второго – это новаторы, выделившиеся из популярного ранее направления итерационной разработки, которые структуре и последовательности *работ* предпочитают эффективное профессиональное взаимодействие и гибкость в ответ на изменчивость внешнего мира. Именно о таких методологиях, которые принято называть гибкими, и пойдет речь в нашем курсе.

Все семейство гибких методологий на сегодняшний день находятся в непрерывном развитии:

* модернизируются и дополняются существующие подходы;
* разрабатываются новые методологии;
* расширяется сфера их использования.

## 1.3 Что такое Agile?

*Процессный подход*, как способ организации деятельности, будоражит умы и вызывает множество различных вопросов уже достаточно давно.

Когда речь заходит о "правильном" и эффективном управлении деятельностью, множество практиков приводят различные доводы в пользу того или иного подхода, который, по их мнению, является оптимальным способом организации работы.

Проходит время, меняются внутренние и внешние условия, различные факторы, и рынку необходимы новые инструменты, которые смогли бы позволить приспособиться компаниям к изменяющейся действительности и при этом демонстрировать все более устойчивый результат по сравнению с их предшественниками. Неизменным остается только наличие "человеческого ресурса" со всеми его достоинствами и недостатками. Попытка относиться к сотруднику, как к самозарядному автомату, который должен в сжатые сроки выстреливать результатом и быть готовым к использованию в любых, даже самых жестких условиях, постепенно отживает свое.

*Значимость* смены подобной парадигмы ознаменовалась тем, что в литературе взамен варварского термина "человеческий *ресурс*" стали использовать более мягкий - "человеческий капитал".

Но это не должно вводить в заблуждение тех, кто ставит во главу угла творческое начало и отрицает всякую *значимость* дисциплины. Сотрудник должен быть не винтиком в отлаженном механизме (не так уж много таких отлаженных механизмов, как показывает практика), а играть роль важной шестеренки, которая приводит в движение смежные процессы, обеспечивающие жизнь организации, каждого из профессионалов или тех, кто стремится им стать.

На сегодняшний день процессные методологии, ориентированные не только на конечный результат, но и на грамотное управление сотрудником как творческой и самоопределяющейся боевой единицей, показывают наибольший успех.

К примеру, можно привести *Lean*, *Six Sigma*, итерационный подход к разработке и реализации проектов, которые являются достаточно сложными к внедрению, последующей адаптации и развитию, но их постоянное применение позволяет компаниям вырваться в лидеры своих сегментов рынка и занять на них устойчивое положение.

Процесс есть всегда. Даже в том случае, когда речь идет о стихийном выполнении разнородных этапов, мы все равно говорим о процессе с низким уровнем зрелости, результат которого в большинстве случаев будет непредсказуемым.

Если руководство организации не устраивает ситуация, в которой оно не может влиять на достижение результата, то процессами обычно стремятся заниматься. Не просто заниматься, а делать это эффективно.

Главным помощником в достижении оптимального управления деятельностью в последнее время все больше и больше выступают *информационные технологии*, привносящие определенную меру порядка в окружающий организации хаос. Отлаженная информационная система или набор систем помогают компаниям оптимизировать и усовершенствовать их процессы до такой степени эффективности, когда возможное отрицательное влияние человеческого фактора сведено к минимуму.

Главная задача, которую необходимо решать каждому предприятию, пришедшему к необходимости применения ИС, - это то, что процессами их внедрения, разработки и развития надо также управлять.

В этой области сложилось несколько процессных методологий, каждая из которых имеет определенный багаж побед и поражений.

Первый и основной на сегодняшний день - последовательный подход, состоящий в четком следовании запланированной структуре работ, суть которых определена заранее.

Второй, наиболее перспективный и амбициозный, - гибкий подход, являющийся последователем итерационного подхода, который не отрицает первого, но цель которого состоит в снижении сложности необходимой структуры этапов при реализации проекта, повышении прозрачности выполняемого набора работ и использовании творческой инициативы людей как средства повышения результативности. Такими методологиями являются процессы семейства Agile.

Agile - процессная методология, состоящая из серии подходов (Scrum, Kanban и пр.) к разработке программного обеспечения или информационного продукта, ориентированная на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации, в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.

Выше приведено основное и общепринятое *определение* Agile, но, как это часто бывает в мейнстримных направлениях, которым Agile стала на сегодня, единого определения не существует и есть множество альтернативных толкований этого термина.

Приведем еще несколько специфичных толкований, которые дополнят ваше *представление* о его сущности.

*Agile - это framework для разработки и поддержки функционально сложных продуктов.*

*Agile - это framework, в рамках которого возможно решать сложные комплексные задачи и в то же время продуктивно и креативно разрабатывать продукты удовлетворительного для заказчика качества.*

*Agile - это framework, используемый для комплексного управления процессами.*

*Agile - это активно развивающаяся (с начала 90-x) процессная методология, цель которой заключается в оптимальной организации рабочего наукоемкого процесса, создании эффективных инструментов мониторинга и контроля за его исполнителями и производимыми результатами.*

Как вы заметили, приводя определения, мы использовали понятие framework, которое служит для обозначения профессионального подхода к решению всевозможных задач, являющихся ступенью к достижению поставленного результата.

В современном мире это понятие стало распространяться не только на профессиональную деятельность, но и на все проблемы, которые стоят перед специалистом. В нашем курсе мы дополним и расширим это понятие, так как, говоря о Agile, постепенно становится понятно, что это не просто подход к организации процессов, а эффективная жизненно-профессиональная философия.

В завершение скажем о том, что хоть Agile и родилась как способ организации специализированных процессов в области информационных технологий, но при этом ее с успехом используют во многих областях, где:

* влияние человеческого фактора на достижение конечного результата очень высоко;
* продукт, который получается в результате деятельности, представляет собой сложный и комплексный артефакт;
* в работе над продуктом задействована команда профессионалов, объединенная общими целями и задачами.

В утверждении, что Agile - это гибкий управленческий framework, нет лукавства. Это действительно так, что подтверждено на практике. Многие компании, основными процессами которых являются маркетинг, консалтинг и пр., уже успешно "живут" *по* Agile.

В нашем курсе мы уделим основное внимание самому популярному и авторитетному ответвлению Agile - Scrum. Основополагающие принципы и артефакты, о которых мы будем вести речь, используются во всех ответвлениях семейства Agile.

## 1.4 Немного из истории

В феврале 2001 года на горнолыжном курорте в горном хребте Васатч, штат Юта (США), 17 человек, уважаемых специалистов в области разработки программного обеспечения, встретились пообщаться на общие темы (боль, идеи, надежды), покататься на лыжах и, конечно же, поесть…

Именно так, если верить авторитетным источникам, началась официальная история самого популярного отпрыска семейства Agile - Scrum.

К тому моменту возникла необходимость создания действенной и общеприменимой альтернативы тяжеловесным методам разработки программного обеспечения. Первые ростки, именуемые экстремальным программированием, адаптивная *разработка программного обеспечения*, кристалл, *FDD*, прагматичное *программирование* уже начали появляться на рынке, но для обширного распространения были слишком экзотичными (посадить за один *компьютер* двух разработчиков в одно и то же время было просто умопомрачительным решением для начала 2000-х годов).

Начиная с конца 80-х рынок информационных технологий, как в старушке Европе, так и в Соединенных Штатах Америки, начал готовиться (а вернее, его начали готовить именно те, кто в последующем станут авторами Agile) к прорыву.

Мирная революция назревала. Процессы разработки программного обеспечения должны были перестать быть уделом небольших псевдо- или околонаучных коллективов или одиночек, которые были гиками своего времени. Нужен был толчок, который позволит вывести развитие программного обеспечения на новый виток, инициировать следующую итерацию в развитии рынка программных продуктов.

Развитие электронно-вычислительной техники должно было стать дополнительным катализатором, подтолкнувшим к буму развития компьютерно-вычислительной техники и процессов разработки программных продуктов.

Поворот "лицом" к бизнес-заказчику, который ранее был далек от происходящего, и его активное знакомство с программными продуктами как средством оптимизации и повышения эффективности его бизнеса продемонстрировало всю мощь информационных технологий. Пиджак с галстуком, заинтересованный только в конечном результате, который сидит несколькими десятками этажей выше от отдела информационных технологий, должен стать вовлеченным в текущие процессы, сделаться их активным участником. Звучит очень хорошо и перспективно, но так просто, с наскока, открыть все двери одним движением ноги никто не позволил бы. Аудиторию заинтересованных сторон необходимо было ментально подготовить.

И эта подготовка началась издали, сначала с наиболее определенных участков человеческой деятельности, в которой не нужен широкий и очень быстрый полет творчества, - бухгалтерский и смежный с ним типы учета.

Постепенно *информационные технологии* стали отвоевывать все больше и больше пространства, проникать в емкие и сложные виды деятельности, занимая свое *место* среди прочих видов деятельности.

Экспоненциальный рост темпов развития информационных технологий привел к тому, что сроки на разработку информационных систем стали постоянно сокращаться.

Все меньше и меньше времени стали отводить на долгое и основательное планирование, последующее проектирование, написание полного пакета необходимой документации и последующее всеобъемлющее тестирование. Существующие процессы разработки программного обеспечения, а вернее - их методология, должны были адаптироваться к наступающей действительности.

Если сдвига не произошло бы, то все разрабатываемые программные продукты устаревали бы еще до того, как были полностью готовы к коммерческому использованию. Двигаться в направлении информатизации было необходимо, и это движение должно быть продуманным, но не слишком долгим.

Именно это и произошло с гибкими методологиями. Эволюционно, преобразовывая итерационный принцип разработки информационных систем, сначала полностью вовлекая в процессы только специалистов, разработчиков, тестировщиков и пр., а потом и смежные специальности, рынок готовился к появлению методологии, в которой одним из главных постулатов будет *универсальность*, но с сохранением профессиональной индивидуализации ее участников.

Agile появилась и начала набирать обороты на фоне быстрорастущего усложнения технологий и всеобщей информатизации различных бизнес-направлений деятельности.

Как говорят все те же авторитетные западные источники, на той судьбоносной встрече в "штате мормонов" появился *Agile Manifesto* - библия гибких процессов разработки программного обеспечения, о котором мы поговорим немного позже.

## 1.5 Об авторах Agile

К совершенствованию Agile приложили руку и умы многие столпы направления разработки программных продуктов области информационных технологий 90-х годов. Многие профессионалы не просто участвовали в развитии этого направления деятельности, а посвятили этому свои профессиональные карьеры.

Англоязычные источники среди прочих выделяют следующих авторов:

 Вэрд Кэнингхэм

Основатель одноименной компании, является директором в области исследований и разработок компании "Вэйт *Программное обеспечение*". Перед этим занимал пост главного инженера в компании "Научно-исследовательская компьютерная лаборатория Тектроникс".

Вэрд хорошо известен своим вкладом в разработку одной из практик объектно-ориентированного программирования, называемой экстремальным программированием, и созданной им технологией *Wiki*.

 Джим Хэнгсмит

Главный разработчик метода "Адаптивной разработки", входящей в семейство Agile, и *автор* книги с одноименным названием. Он одним из первых начал публично пропагандировать использование Agile на международных конференциях *по* разработке *ПО*. Джим в соавторстве с Мартином Фаулером создал статью-исследование "The Agile" в популярном журнале "*Разработка программного обеспечения*". Джим и Алистер Кокберн работали над созданием многих ответвлений Agile-семейства, которые были опубликованы в издательстве Addison-Wesley в виде серии книг *по* Agile.

 Эндрю Хант

Один из авторов книги "The *Pragmatic* Programmers" и соавтор бестселлера "Прагматичный программист: *путь* от новичка до Мастера". Между написанием книг и статей, произнесением вдохновляющих речей и игрой на пианино Эндрю находит время для своего консультационного бизнеса, специализирующегося на Agile. Энди создает *программное обеспечение* профессионально с 80-х годов и успел погрузиться в такие отрасли, как *телекоммуникации*, банковская и финансовая сферы, утилизация, медицина, графическое искусство и, конечно же, создание сервисов в Интернете. Энди задействован в компании Raleigh NC и со своим соавтором Дэйвом Томасом занимается распространением наиболее эффективных и хорошо себя зарекомендовавших методов разработки программного обеспечения в США. Он также является президентом *RTP*, отделения независимой ассоциации компьютерных консультантов и членом *ACM* и *IEEE*.

 Рон Джефрис

Основатель портала XProgramming.*com*, консультант компании *Object* Mentor и *автор* (совместно с Анной Эндерсон и Четом Хендриксоном) Extreme *Programming*, Рон был первым тренером *по* методологии Extreme *Programming*.

 Джон Керн

Джон питает страсть к оказанию успешной помощи клиентам в достижении ими ценности от использования разработанного им программного обеспечения. Его разнообразная карьера растянулась с исследовательской деятельности *по* созданию двигателя для реактивных самолетов до разработки летательного симулятора, он стал евангелистом (приверженцем-распространителем) объектно-ориентированного подхода c 90-х годов, начав с использования C++ и затем приступив к *Java*. Он первым опубликовал описание использования итерационной разработки при создании Lotus Notes 4.5 и 5.0. Он был мотивирован мантрой, которую произносил его друг Питер Код: "Часто, осязаемо, дающее результат". Он получил степень доктора наук, основал собственную компанию (Lightship, Inc.). В 1999 году он присоединился к Питеру Коду в его стартапе "TogetherSoft", где он создал профессиональную группу консультантов и работал над разработкой программных продуктов. Джон - соавтор реализации *Java*, работал вместе с Питером и Джефом ДеЛука над *FDD*. Джон постоянно занят поиском наилучших методов достижения поставленных целей с точки зрения используемой методологии работы и используемых технологий. Вы можете найти его *блог* *по* адресу <http://blogs.compuware.com/cs/blogs/jkern/>.

 Брайан Марик

Консультант *по* разработке и тестированию программного обеспечения. При разработке Agile выступал как представитель сообщества тестирования, который подчеркивал необходимость тестирования и документирования, особенно учитывая повышающуюся сложность и изменчивость реализации компьютерных систем.

 Роберт С. Мартин

Специалист в области разработки программного обеспечения с 1970 года. Президент и основатель *Object* Mentor Inc. - компании, которая занимается глубокой экспертизой с помощью гибких методологий, разработке программного обеспечения, обучении и разработки систем и сервисов для крупных корпораций. В 1995 году написал бестселлер "Проектирование приложений на *Object* Oriented C++ с применением метода Буча". В 1997 году был главным редактором книги "Шаблоны проектирования языков программирования". В 1999 году был редактором книги "More C++ Gems". Соавтор книги "XP in *Practice*", изданной в 2001 году. Непосредственно работал над книгой "Принципы, шаблоны и практика разработки программного обеспечения *по* Agile", изданной в 2002 году. С 1996 *по* 1999 год был главным редактором отчетов *по* С++. Опубликовал множество статей в различных журналах и выступает в качестве постоянного докладчика на международных конференциях.

 Майк Бидл

Основатель и исполнительный директор компании e-Architects Inc. Один из самых ранних евангелистов философии Agile. Майк специализируется на создании масштабных информационных систем, в работу над которыми, как правило, вовлекается множество распределенных команд.

 Ари Ван Беннекум

Разработчик, консультант, практикующий тренер. Один из самых активных пропагандистов Agile-процессов. Внутренний двигатель Scrum-cообщества.

 Алистер Кокберн

Основатель компании Humans *and* *Technology*, *автор* многих современных ответвлений в области ИТ, один из отцов-основателей ядра методологии Agile. Спонсировал первоначальное развитие и продвижение Agile. До сих пор является практикующим профессионалом, *деятельность* которого направлена на создание техноэкосистем.

 Мартин Фаулер

Исполнительный директор *по* исследованиям крупной компании в области разработки программного обеспечения и консалтинга. *Автор* книг *по* аналитическим шаблонам, использованию Uml, рефакторингу, планированию и другим видам активностей в направлении Agile. Несмотря на то, что Мартин Фаулер напрямую не участвовал в разработке и продвижении Agile, профессионалы относят его к числу тех, кто внес в ее развитие наибольший вклад.

 Кен Швайбер

Президент компании *ADM*. Разработчик, проектный *менеджер*, консультант, инициировавший массовый переход от тяжеловесных методологий в сторону Agile. Один из авторов ответвления Scrum (вместе с Д. Сазерлендом), которое на сегодняшний день имеет наибольшую популярность среди Agile-направлений.

 Джеф Сазерленд

Директор *по* технологиям крупной компании. Соавтор Scrum. *Архитектор* крупных информационных наукоемких систем.

Стоит отметить, что Agile является полностью открытой методологией, и поучаствовать в ее дальнейшем развитии и популяризации может любой из нас.

## 1.6 Как развивалась Agile

Описание процесса зарождения и развития Agile стоит начать с ситуации, происходившей в отрасли в середине прошлого века.

Начался так называемый "первый кризис программирования", который заключался в том, что *стоимость* разработки программного обеспечения стала приближаться к стоимости аппаратуры, а при анализе дальнейших трендов развития ситуации были прогнозы, что к концу прошлого века все человечество будет задействовано в процессах разработки информационных систем.

Затем произошел прорыв развития электроники и, как следствие, увеличилась *производительность* компьютеров, а *стоимость* их стала резко снижаться. Ограничения на разработку, связанные с аппаратными средствами, стали исчезать или трансформироваться в ограничения на *ПО*. Таким образом, умение разрабатывать новые программы диссонировало с теми требованиями, которые заказчики предъявляли к этим самым программам. Это привело к тому, что *разработка программного обеспечения* перестала быть просто "кодированием", а стала формироваться в виде конкретных концепций, поддерживающих полный *жизненный цикл* разрабатываемых программ (формирование идеи, *анализ* требований, проектирование функциональности, разработка *ПО*, тестирование *ПО*, внедрение, *поддержка* с последующей утилизацией).

Смещение фокуса со специализированного процесса разработки в сторону полноценной методологии и инструментария, поддерживающего его выполнение, привело к появлению целого мира программной инженерии и, как следствие, к промышленной разработке программного обеспечения, цель которой заключалась в сокращении стоимости создания программ.

Все необходимые направления, выполняемые в процессе промышленной разработки и необходимые для получения результата, стали формироваться в виде конкретных процессных направлений разработки *ПО*, каждый из которых в дальнейшем переосмысливался, обобщался и соответствующим образом формализовывался.

Это привело к появлению конкретных практических методик, каждая из которых подтвердила свою успешность на массиве практических *работ*.

Если мы рассмотрим любую современную методологию управления проектом или моделями жизненного *цикла* проекта *по* разработке информационных систем, согласно SWEBOK, то выглядеть она будет следующим образом:

* традиционная (каскадная, водопадная) модель;
* спиральная модель;
* итеративная и инкрементная модель.

*SWEBOK - документ, в котором описаны эталонные методики по всем стадиям разработки программного обеспечения.*

*SWEBOK является одним из самых авторитетных стандартов в области разработки программного обеспечения, постоянно переиздается и расширяется на предмет поэтапной детализации наиболее сложных и важных направлений работ.*

*Каскадная модель*(рис. 1.1) разработки программного обеспечения является первой и, как следствие, самой критикуемой. При работе с данной методологией предполагается последовательное выполнение этапов разработки. Такая структура не дает изменить требования к программному продукту до самого релиза (до выхода образца готовой продукции).



Рисунок 1.1 **–** Каскадная модель разработки

Чем более масштабный проект, тем больше изменений накапливается. Реализация изменений в следующей версии продукта иногда становится нецелесообразной. Продукт необходимо писать с нуля. Таким образом, *стоимость* работоспособной версии неоправданно сильно растет. Это приводит к тому, что *процент* успешно завершенных проектов ничтожно мал *по* сравнению с другими методологиями. Но при этом для части проектов, которые затрагивают, к примеру, *безопасность* жизнедеятельности, строго поставленные требования и высокая степень формализации являются основополагающим и необходимым фактором.

Кроме того, традиционная модель играет важную роль. Она налагает на процесс разработки требование крайне необходимой для него дисциплинированности, с помощью которой удается благополучно обходить неструктурированные процессы типа "пишем и правим написанное".

Традиционная модель внесла фундаментальный вклад в понимание процессов разработки следующими утверждениями:

* процесс должен подчиняться дисциплине, разумному планированию и управлению;
* полная реализация продукта должна быть отложена до полного понимания целей этой реализации.

Следующей *по* времени возникновения и эволюционному развитию стала *спиральная модель* ( рис. 1.2). Как это часто бывает с последователем, в ней попытались исправить все основные проблемы ее предшественника.



Рисунок 1.2 –Спиральная модель разработки

Набор фаз и их структура в спиральной модели соответствуют водопадной модели, но в спиральной методологии каждая фаза завершается этапом прототипирования и управления рисками. Этап прототипирования после каждой фазы проекта позволяет определить, насколько текущее состояние проекта соответствует первоначальному плану. *По* итогам прототипирования выполняется либо переход к следующей фазе, либо возвращение на одну из предыдущих фаз для выполнения необходимых корректировок.

Следующим витком в развитии процессов разработки программного обеспечения стала итеративная модель. Она предполагает *разбиение* жизненного *цикла* на последовательные итерации, каждая из которых, *по* сути, является самодостаточным циклом *по* созданию функциональности программного продукта.

*Истоки концепции итеративной разработки прослеживаются в относящихся к 1930-м годам работах эксперта по проблемам качества продукции Уолтера Шеварта из Bell Labs. Важной вехой в истории является осуществленный в 50-е годы проект по разработке сверхзвукового реактивного самолета X-15. По мнению участников этих работ, применение спиральной методологии в значительной степени определило дальнейший успех проекта.*

Плавно переходя от методологий, которые являлись историческими вехами на пути возникновения Agile (их подробное сравнение будет проведено в следующих главах), каждая из которых явила собой ступень в эволюционном развитии понимания того, как эффективно на текущий момент нужно разрабатывать *программное обеспечение*, мы плавно подошли к теме Agile.

Стоит отметить, что Agile-семейство, *по* сути, является ответвлением итеративной модели разработки программного обеспечения. При описании любой из гибких методологий упоминается принцип разделения на итерации, который нашел отражение во многих артефактах, применяемых в Agile (task *board* и пр.).

Следующие вехи в развитии Agile заслуживают упоминания:

* 1992 год:
	+ *Crystal Methods.* Семейство методологий Crystal послужило основой для развития современных методологий разработки программного обеспечения. Agile была создана людьми, которые "вышли" из движения Crystal. Автором Crystal является Алистер Коберн. Методология может быть применима к командам, состоящим из шести-восьми участников, расположенных в одном месте, работающих над созданием программных систем, не являющихся критичными для жизни пользователей.

*Crystal* фокусируется на:

* частой поставке работающего кода конечным пользователям;
* разумных улучшениях;
* всепроникающей коммуникации (osmotic communication) между членами команды, расположенными в одном месте.

Под всепроникающей коммуникацией подразумевается непосредственная передача информации, в том числе путем подслушивания или наблюдения за вещами, происходящими вокруг.

* 1993 год:
	+ *Refactoring.* Термин "рефакторинг" был введен Билом Опдайком в статье под названием "Creating Abstract Superclasses by Refactoring". В Википедии приведено следующее описание рефакторинга: "Процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы".
* 1994 год:
	+ *Dynamic Systems Development Method (DSDM).* DSDM был разработан консорциумом, являющимся объединением поставщиков и производителей программного обеспечения. Цель их работы - совместными усилиями разработать и распространить независимый framework для быстрой разработки приложений с использованием накопленного опыта.

DSDM фокусируется на восьми основных принципах:

* учитывать потребности бизнеса;
* поставлять вовремя;
* взаимодействовать;
* никогда не снижать качество;
* создавать постепенно с самых основ;
* разрабатывать итерационно;
* непрерывно и ясно общаться;
* демонстрировать управляемость.
* 1995 год:
	+ *Scrum and Pair Development.* Scrum был разработан совместно Джефом Сазерлендом и Кеном Швабером, которые представили доклад на конференции OOPSLA'95 в Остине, штат Техас.
* 1997 год:
	+ *Feature Driven Development.* Методология Feature Driven Development (FDD) изначально была разработана Джефом ДеЛукой.

Основные практики, описанные в *FDD*, следующие:

* моделирование объектной модели домена предметной области;
* разработка на основе улучшений;
* индивидуальное владение кодом;
* команды, организованные по направлениям улучшений;
* инспекции;
* управление конфигурацией;
* регулярные сборки продукта;
* видимость прогресса и результатов.
* 1999 год:
	+ *Adaptive Software Development.* Джим Хэнгсмит сформулировал концепцию Adaptive System Development и опубликовал книгу с таким же названием. Идея выросла из его работы по методологиям быстрого создания приложений (RAD).
* 2002 год:
	+ *Новые идеи в Agile.* Test Driven Development (TDD). Концепция разработки через тестирование изначально появилась в XP в виде подхода "сначала тест". Чуть позже эта техника была описана более подробно Кентом Беком в его книге "Test Driven Development: By Example".
* 2003 год:
	+ *Lean Software Development.* Последователи Agile считали, что следующим витком ее развития будет являться трансформация с процессной методологией Lean. Время подтвердило обоснованность этой идеи. Термин такого ответвления Agile был введен Мэри и Томом Поппендик в 2003 году.

Это *адаптация* бережливых принципов производства к разработке программного обеспечения. Рассматриваются семь основных принципов:

* устранять расходы;
* усиливать обучение;
* принимать решение как можно позже;
* поставлять как можно раньше;
* оказывать поддержку команде;
* встраивать целостность;
* видеть картину в целом.

Agile на текущий момент своего развития - в большей степени свод знаний *по* организации работы с психологической точки зрения. Agile помогает проявлять творческую составляющую, умение работать в команде, навыки коммуникации и прочее. Техническая сторона организации *работ* все больше уходит на второй план, но при этом Agile содержит достаточное количество инженерных практик, используя которые, можно достичь качественного конечного результата. Эти практики с успехом применяются в сочетании с другими методологиями.

## 1.7 Agile Manifesto

После того как у нас сложилось целостное *представление* об историческом пути Agile, логично перейти к рассмотрению документа, который является основополагающим для многих процессов, принадлежащих этому семейству.

Этим документом является *Agile Manifesto*. *По* сути, его и документом назвать сложно. Это, скорее, небольшая памятка, которая содержит 4 ценности и 12 принципов работы. Далее мы приведем полное содержание *Agile Manifesto*, как оно представлено на официальном сайте (<http://agilemanifesto.org/>) разработавшего его консорциума.

**Ценности:**

1. **Люди и взаимодействие** важнее процессов и инструментов.
2. **Работающий продукт** важнее исчерпывающей документации.
3. **Сотрудничество с заказчиком** важнее согласования условий контракта.
4. **Готовность к изменениям** важнее следования первоначальному плану.

Таким образом, не отрицая важности того, что справа, все-таки больше ценится то, что слева.

**Основные принципы:**

1. Наивысшим приоритетом является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
2. Изменение требований приветствуется даже на поздних стадиях разработки.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
6. Непосредственное общение является наиболее практичным и эффективным способом обмена информацией, как с самой командой, так и внутри команды.
7. Работающий продукт - основной показатель прогресса.
8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
10. Простота - искусство минимизации лишней работы - крайне необходима.
11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Изложенные ценности и принципы говорят сами за себя и в каких-то длительных и пространных рассуждениях не особо нуждаются.

Акцент сделан на создании ощутимой выгоды для заказчика. В *Agile Manifesto* учтены все те сложности, которые являются критичными для водопадной и итерационной моделей разработки программного обеспечения, с целью их возможной минимизации.

Каждая из ценностей и принципов будет рассмотрена нами более подробно в последующих лекциях. Главное, на чем хочется сосредоточить внимание сейчас, - с помощью *Agile Manifesto* консорциумом профессионалов была заложена успешная попытка выработки процессной методологии, в которой стало реальным взаимопроникновение информационных технологий и сферы деятельности, заинтересованный не просто в применении современных высокотехнологических наработок, а в их постоянном использовании, развитии и внедрении в операционную *деятельность*.

Барьеры, которые возникали с середины прошлого века между интеллектуалом-разработчиком и представителем бизнеса, связанные с непониманием друг друга, превращались в барьеры при создании информационных систем, которые не демонстрировали ожидаемой от них полезности и стали постепенно рушиться.

## 1.8 Применение в различных предметных областях

Несмотря на кажущуюся специализированность Agile, данная методология расширяет сферу своего влияния и постепенно становится новой культурой управления компаниями (Toyota) и проникает в различные сферы деятельности.

Одним из основных постулатов использования Agile является разработка эффективного конечного продукта.

Результатом практически каждого направления деятельности является именно продукт, выраженный в виде результата, облеченного в определенную специфику конкретного вида деятельности.

Даже если речь идет о таком консервативном и "бесконечном" процессе, как *бухгалтерский учет*, то можно провести параллель не с разработкой программного обеспечения, а с таким этапом его жизненного *цикла*, как *поддержка* и развитие.

Развитие продукта итерациями, определяемыми временным интервалом, вряд ли подойдет, но если говорить об итерациях, которые будут определяться способом работы над конкретной задачей, то все становится вполне логичным.

*В последнее время получает широкое распространение способ управления, называемый холакратией. Это подход к управлению и распределению ответственности, который позволяет избавиться от иерархии в компании. Каждый сотрудник (холакратия предлагает вместо слова "сотрудник" использовать слово "партнер") попадает в условия, которые на постсоветском пространстве называются "работать, как для себя", непрерывно что-то изменяет и улучшает в той части бизнеса, за которую он ответственен. И это не просто слова, это логическое продолжение Agile, которое имеет примеры успешного применения (проект "Кнопка").*

Основное в применении Agile - это желание учиться и развиваться. Профессионалы, которые стремятся к использованию новых идей и передовых практик, как правило, с удовольствием становятся на рельсы Agile.

Аgile - подход инновационного переосмысления создания нового продукта или услуги. В его основе очень простая идея: каждый участник процесса, каждый сотрудник этой "конвейерной сборки" должен вовлекаться в процесс переосмысления своих задач и общего дела. Каждый может остановить конвейер и внести свои рациональные предложения.

## 1.9 Краткие выводы

Первая лекция, в которой приведен краткий обзор исторической ситуации, связанной с возникновением такого направления деятельности современного мира, как *разработка программного обеспечения*, подошла к концу. Мы заронили основные повествовательные зерна, которые будем проращивать в дальнейших главах курса. Как вы могли понять, тема гибких процессных методологий является сложным, но при этом интересным и доступным пониманию направлением, который приковывает к себе внимание. Ожидания от его применения очень высоки. Но на успех могут рассчитывать только те, у кого уже сложилось цельное *представление* о том, что *процессный подход* к организации деятельности - это сложная управленческая *парадигма*, которая нуждается в постоянном контроле и управлении. В противном случае ожидаемый эффект от Agile вряд ли будет достигнут.

В лекции "Процессный подход как конкурентное преимущество" будет изучен процессный подход, его значимость для современных компаний, рассмотрены наиболее эффективные варианты его внедрения и последующей адаптации и основные артефакты, без которых его использование не будет *по*-настоящему оптимальным.

**Питання до теми:**

1. Чем Agile может быть полезна для вашей компании?
2. Что такое Agile?
3. Приведите условия применимости Agile.
4. Приведите ценности Agile.
5. Приведите основные принципы Agile.
6. Что такое «холакратия»?

# Лекция 2: ПРОЦЕССНЫЙ ПОДХОД КАК КОНКУРЕНТНОЕ ПРЕИМУЩЕСТВО

План

2.1. Процессный подход

2.2. Значимость процессного офиса во внедрении процессного подхода. Возможные варианты

2.3. Движение в направлении гибкости. Пилотные процессы и управление изменениями

2.4. Регламент как основа процессного подхода

2.5. Краткие выводы

## 2.1. Процессный подход

**Смысл процессного управления**

Компании для завоевания лидерства на рынке многие годы тестируют различные подходы к управлению, такие как: управление по целям, управление поручениями, управление проектами и управление бизнес-процессами. Каждый из управленческих подходов хорош для той или иной ситуации, которая складывается из отраслевой принадлежности организации, ее зрелости, конкурентности рынков, на которых функционирует компания, личности руководителя компании и множества других факторов.

Для предприятий, которые работают на неконкурентных рынках, система управления может быть любой. Например, в российских монополиях чаще можно увидеть систему управления поручениями с соответствующей системой контроля их исполнения.

Управление бизнес-процессами больше всего необходимо производственным компаниям и компаниям из сферы услуг, работающих на конкурентных рынках. Им приходится бороться за свои позиции, улучшая качество, оптимизируя стоимость или сокращая сроки. Поэтому за долгие годы управление бизнес-процессами стало стандартом де-факто для большинства международных компаний.

В украинских условиях процессное управление востребовано коммерческими компаниями, работающими в конкурентной среде, - банками, страховыми компаниями, компаниями розничной торговли, сферой оказания услуг и многими другими отраслями.

Смысл процессного управления предельно прост. Потоки работ, проходящие через компанию, требуют участия представителей различных подразделений, часто имеющих различные цели, задачи и показатели. При этом часто никто не заинтересован в эффективности всего сквозного бизнес-процесса, проходящего через всю компанию. Главное - как-то сделать свою часть внутри бизнес-процесса, после чего передать задачу далее. В результате такого функционально-ориентированного подхода в компании нет ответственного за весь бизнес-процесс целиком, что приводит к серьезной неэффективности организации работ и постоянным конфликтам между представителями различных функциональных подразделений, которые еще больше снижают результативность процесса. Фактически мощность бизнес-процесса определяется мощностью самого слабого звена в бизнес-процессе, что часто сводит на нет усилия остальных подразделений по оптимизации своей части бизнес-процесса.

Смысл управления бизнес-процессами заключается в определении кросс-функциональных или сквозных бизнес-процессов, их формализации в том или ином формате и последующем их анализе, а главное - их оптимизации. Чем больше в бизнес-процессе участников, тем больше в нем несостыковок, и поэтому особое внимание нужно уделять именно кросс-функциональным процессам. Процессы, которые называются сквозными, еще важнее, они начинаются вне организации и заканчиваются вне организации, именно по организации сквозных бизнес-процессов клиенты и другие заинтересованные стороны создают свое мнение о компании, и поэтому сквозные процессы - первые кандидаты на управление и оптимизацию.

Если говорить о цикле управления бизнес-процессом, то в первую очередь необходимо сделать описание, или, как еще говорят, моделирование бизнес-процесса, которое показывает всю неэффективность существующего бизнес-процесса. А анализ фактических значений показателей бизнес-процесса позволяет не только проанализировать существующую ситуацию, но и определить те показатели, которые должны быть усовершенствованы.

На практике существует множество способов оптимизации, но самое главное, что после их применения должна стать понятна целевая картина бизнес-процесса, то есть то состояние бизнес-процесса "как должно быть", которое и необходимо внедрить в организации.

Далее на основании целевой картины необходимо начать изменение процесса и его внедрение в деятельность новых правил. Чаще всего это делается с помощью нормативных документов (регламентов и стандартов), информационных систем, а также обучения участников бизнес-процесса. Именно внедрение бизнес-процессов "как должно быть" является наиболее трудоемкой задачей, ведь нужно изменить компанию, ее сотрудников вместе с их привычками и представлениями, а возможно даже и уволить часть не согласных с новыми правилами.

**Моделирование, анализ и оптимизация бизнес-процессов**

Ключевым инструментарием внедрения процессного подхода к управлению на первых этапах являются графические инструменты моделирования бизнес-процессов, а также текстовые и табличные редакторы, в которых моделируются или описываются бизнес-процессы, а также создаются регламенты, закрепляющие целевое состояние бизнес-процесса.

На настоящий момент времени для графического моделирования бизнес-процессов MS Visio является наиболее распространенным инструментом, при этом моделировать бизнес-процессы можно в различных нотациях. Однако рынок инструментария не стоит на месте, и для моделирования простых схем бизнес-процессов сейчас возможно использовать совершенно бесплатные облачные приложения, которые позволяют нарисовать бизнес-процесс фактически в любой нотации.

Некоторые компании вообще обходятся без инструментов графического моделирования бизнес-процессов, используя презентации для графической иллюстрации и текст или таблицу для описания необходимых деталей бизнес-процессов, что позволяет им ограничиться классическими офисными редакторами.

На украинском рынке есть более серьезный инструментарий, позволяющий строить целостную модель бизнес-процессов с взаимосвязями как между отдельными моделями процессов, так и с элементами организационной структуры, классификатором информационных систем, стратегическими целями и ключевыми показателями результативности, а также документами в различных форматах, данными в информационных системах и другими артефактами. Наибольшее распространение среди серьезных промышленных систем для моделирования и анализа бизнес-процессов, используемых в крупнейших украинских компаниях, сейчас имеет система ARIS. В то же время есть пример импортозамещения на украинском рынке - система Business Studio.

Преимуществом таких систем является обеспечение связности модели описания деятельности, а также возможность получения в автоматическом режиме регламентов бизнес-процессов, положений о подразделениях, а также любой другой информации на основании построенных моделей бизнес-процессов.

Недостатком промышленных систем моделирования и анализа бизнес-процессов является сложность их освоения пользователями из бизнес-подразделений, что часто приводит к локализации работ по моделированию бизнес-процессов в процессном офисе - подразделении, сопровождающим внедрение процессного подхода в компании.

Применение простейших инструментов из комплекта MS Office или даже бесплатных облачных инструментов, таких как draw.io и bpmn.io, позволяет расширить круг сотрудников, описывающих бизнес-процессы внутри компании, за счет представителей бизнес-подразделений.

Сейчас можно выбирать между различными формами моделирования бизнес-процессов, но так или иначе лидерство пока за текстовой и табличной формой описания бизнес-процессов, в этом формате процесс может описать любой руководитель или сотрудник. В графическом формате чаще рисуют бизнес-аналитики, используя стандартные нотации моделирования бизнес-процессов, такие как BPMN и EPC.

На первых этапах описания и анализа бизнес-процесса очень важно правильно выделить бизнес-процесс, описав все его составляющие, собрав проблематику и фактические значения ключевых показателей результативности по процессу.

И уже после того, как первичные данные собраны, ключевой становится необходимость подвергнуть существующий бизнес-процесс оптимизации либо через сокращение времени, либо через сокращение стоимости, либо через повышение качества.

Можно заниматься оптимизацией процесса по двум параметрам из трех, но вот оптимизация по трем параметрам редка, поскольку если будет сокращена длительность процесса, то увеличится стоимость процесса или упадет его качество - проектный треугольник пока еще никто не отменял.

В рамках анализа и оптимизации бизнес-процессов очень полезно использовать различные референтные модели, а также анализировать примеры реализации аналогичного бизнес-процесса в компаниях той же отрасли.

Однако самое сложное - это не поиск целевого состояния бизнес-процесса, а его внедрение в деятельность компании. Ведь нужно не просто описать новую схему работы, но и убедить участников процесса следовать новым правилам. В большинстве российских компаний чаще всего составляется регламент, контроль над исполнением которого возлагается на того или иного руководителя, однако в век информационных технологий уже появились информационные системы, которые помогают закрепить целевой бизнес-процесс в организации и контролировать соблюдение закрепленных правил. Эти системы появились вместе с технологией процессного управления и изначально назывались workflow.

**Автоматизация бизнес-процессов**

Как и множество информационных систем, workflow-системы стали реакцией ИТ-рынка на появление процессного подхода к управлению, именно в workflow-системах появился такой объект, как бизнес-процесс, который настраивался разработчиком или системным аналитиком при внедрении workflow-системы, и в соответствии с которым маршрутизировались задачи в рамках исполнения бизнес-процесса. Впоследствии вместе с расширением функционала workflow-системы приобрели название BPMS (Business Process Management Suite), сейчас данные системы называются iBPM (intelligent Business Process Management).

По мнению многих руководителей, автоматизация бизнес-процессов часто является единственным результативным способом внедрить бизнес-процесс "как должно быть" в деятельность в точном соответствии с целевой моделью. И именно поэтому в некоторых компаниях стали использоваться системы iBPM для автоматизации некоторых кросс-функциональных бизнес-процессов, а некоторые компании даже смогли замкнуть цикл управления для ключевых бизнес-процессов, внедрив помимо систем iBPM ИТ-решения класса BI (Business Intelligence) для анализа процессных показателей.

Серьезной вехой в применении систем автоматизации бизнес-процессов стала возможность не программировать бизнес-процесс, а настраивать его через формирование графической нотации в формате BPMN, что обеспечило быстроту внесения изменений в автоматизированные бизнес-процессы.

При этом нужно понимать, что стандартизация бизнес-процессов с применением информационных технологий может потребовать решений различного класса, тут и CRM-системы, и ERP, и SCM, при этом разнообразие существующих ИТ-решений на рынке требует выбора информационной системы под особенности того или иного бизнес-процесса. При этом хорошо известно, что большинство типовых ИТ-решений имеют внутри жестко зашитый бизнес-процесс, и поэтому автоматизация с помощью данных ИТ-решений часто типизирует бизнес-процессы в соответствии с запрограммированными алгоритмами, что жестко закрепляет бизнес-процесс в информационных системах.

Однако не все бизнес-процессы можно автоматизировать типовыми ИТ-решениями, для многих процессов можно не найти ИТ-решений, и тут может использоваться или собственная разработка, или различные системы-конструкторы, которые позволяют создать специальное решение для автоматизации именно этого бизнес-процесса.

Но существует еще более сложный случай, когда бизнес-процесс настолько специфичен, что под него не найти типового решения, и в дополнении бизнес-процесс необходимо регулярно изменять. Типовые системы в данном случае не подходят, а используя собственную разработку, придется постоянно переписывать код созданной информационной системы, что отрицательно скажется на скорости внесения изменений в бизнесе. Именно для данных бизнес-процессов, которые, как правило, и являются сквозными или кросс-функциональными, и применяются системы класса iBPM, которые изначально и были предназначены для автоматизации взаимодействия сотрудников в соответствии с правилами, зашитыми в бизнес-процессе. Фактически iBPM-системы позволяют закрепить логику взаимодействия и время исполнения отдельных операций, а также обеспечивают высокую скорость внесения изменений в автоматизированный бизнес-процесс, что позволяет создать регламент в электронном формате с встроенными функциями мониторинга и контроля ключевых параметров бизнес-процесса.

Примеры проектов показывают, что iBPM-система вполне может заменить руководителя при маршрутизации задач внутри одного бизнес-процесса, учитывая при назначении задачи на исполнителя, множество параметров, начиная от загрузки сотрудников и заканчивая их статистикой по операционной эффективности, рассчитанной на базе исторических данных.

Можно сказать, что с помощью iBPM-систем автоматизируется управленческая рутинная задача маршрутизации задач между исполнителями с контролем сроков их исполнения и возможностью перестраивать бизнес-процесс без изнурительного программирования. iBPM-системы в автоматическом режиме эскалируют нарушения по срокам, что позволяет строить достаточно жесткие системы мотивирования, зависящие от исполнения отдельных задач всеми участниками бизнес-процесса.

**Контроллинг бизнес-процессов**

Не всегда удается использовать iBPM-системы для автоматизации бизнес-процесса, поскольку часто процесс исполняется в нескольких типовых информационных системах, которые в том или ином виде интегрированы между собой. Тогда для контроля бизнес-процесса возможно использовать решение класса Process Intelligence (PI), в котором можно собрать и проанализировать данные из различных информационных систем, поддерживающих исполнение бизнес-процесса.

Помимо систем класса PI для анализа бизнес-процессов могут быть использованы специализированные ИТ-решения. Такие как Process Mining - в решениях этого класса на основании данных о фактическом исполнении бизнес-процесса выстраивается графическая модель бизнес-процесса, а также системы класса Business Activity Monitor (BAM), которые собирают в оперативном режиме статистику по исполняемому процессу.

С помощью систем класса PI и BAM можно проводить анализ как временных, так и частотных показателей бизнес-процесса, при этом для анализа фактической логики взаимодействия участников бизнес-процесса можно использовать инструментарий Process Mining, с помощью которого из статистики исполнения отдельных действий можно собрать фактическую модель исполненного процесса для последующего анализа.

При организации контроллинга бизнес-процесса чаще всего анализируются показатели времени исполнения той или иной операции, а также среднего времени исполнения экземпляра бизнес-процесса за тот или иной период.

Сокращение времени исполнения отдельной операции достигается нормированием и контролем работы участников бизнес-процесса, тогда как секрет сокращения среднего времени исполнения бизнес-процесса заключается в поиске "бутылочных горлышек", в которых задача по процессу находится в очереди, ожидая, когда будет доступен ресурс для ее исполнения.

Отдельной группой показателей для анализа бизнес-процесса являются показатели конверсии, которые показывают эффективность отдельных операций процесса, например, сколько коммерческих предложений перешло в договор или сколько кандидатов на работу прошло собеседование с руководителем подразделения. Эти частотные показатели чаще всего анализируются в массовых бизнес-процессах, где происходит обработка большого числа заявок и заказов.

Отдельным направлением контроллинга бизнес-процесса является его стоимость. На основании среднего времени исполнения операции с учетом стоимости единицы рабочего времени исполнителя, а также затрат на организацию рабочего места можно подсчитать стоимость одной операции. И уже на этом основании рассчитать стоимость случая единичного исполнения бизнес-процесса и затем получить возможность расчета стоимости бизнес-процесса за период.

Анализ стоимости бизнес-процесса позволяет правильно перераспределить затраты по услугам, оценить эффективность мероприятий по автоматизации бизнес-процесса и даже проанализировать результативность автоматизации бизнес-процесса. При этом практика проектов по расчету стоимости бизнес-процессов показывает, что руководители часто не видят фактическую стоимость бизнес-процессов, что приводит к принятию неэффективных с экономической точки зрения решений.

Так или иначе, организация контроллинга бизнес-процесса позволяет организовать обратную связь из бизнес-процесса, что позволяет контролировать в регулярном режиме его результативность, а главное - оценивать эффективность мероприятий по его оптимизации.

## 2.2. Значимость процессного офиса во внедрении процессного подхода. Возможные варианты

**От центра компетенции к процессному офису**

Процессный подход к управлению требует кросс-функциональной ответственности за бизнес-процессы. В некоторых компаниях функциональные руководители договариваются между собой, синхронизируя сквозные или кросс-функциональные бизнес-процессы на границах перехода между подразделениями. Но такая ситуация складывается не везде.

Во многих компаниях существуют бизнес-процессы, в которых участвует множество подразделений с пересекающимися интересами, при этом при попытке оптимизировать такой бизнес-процесс менеджерам компании не всегда хватает времени или управленческих полномочий для его оптимизации.

Именно несогласованность действий нескольких функциональных подразделений приводит к задержкам в выполнении работ, ошибкам, бюрократическим процедурам, не добавляющим качества в бизнес-процесс, а лишь съедающим дополнительные ресурсы.

Таким образом, внедрение процессного подхода в организации необходимо поддержать организационными мероприятиями и созданием специализированного центра компетенции или даже отдельного подразделения для определения бизнес-процессов и последующего их усовершенствования.

На первом этапе центр компетенции может заключаться в папке на сервере, где сложены обучающие материалы по управлению бизнес-процессами и регулярной информационной рассылки по заинтересованным лицам. В дальнейшем такая рассылка может перерасти в отдельное подразделение, занимающееся внедрением процессного подхода в организации.

Такие подразделения в теории называются процессными офисами, хотя на практике могут встречаться и другие названия, например, департамент по оптимизации бизнес-процессов или департамент стандартизации бизнес-процессов, или даже департамент описания бизнеса и регламентации отношений, что в сокращении будет смотреться очень здорово - ДОБРО.

Вне зависимости от названия процессный офис является подразделением, которое содействует в описании, регламентации и оптимизации бизнес-процессов в компании. Для обеспечения поддержки внедрения процессного подхода на уровне всей компании помимо процессного офиса должен быть организован специализированный комитет, в задачи которого входит постановка задач на описание и оптимизацию бизнес-процессов, а также разбор произошедших кейсов и анализ лучших практик с целью запустить процесс непрерывного совершенствования деятельности организации.

Частота заседаний данного комитета может отличаться в зависимости от организации, где-то это ежегодное заседание с докладом об изменениях, сделанных в бизнесе, а где-то это может быть еженедельное заседание, на котором рассматриваются кейсы, утверждаются регламенты и даются задачи на стандартизацию или оптимизацию тех или иных процедур.

Цель данного комитета - сделать совершенствование существующих бизнес-процессов регулярным, дать руководителям подразделений и владельцам бизнес-процессов задачи на оптимизацию тех или иных процессов, а также в оперативном режиме рассматривать конфликтные ситуации, которые возникают в бизнес-процессах из-за рассогласованности действий участников бизнес-процесса, и стараться на системном уровне исключить повторение конфликтных ситуаций в будущем.

При этом задачи оптимизации бизнес-процессов необходимо ставить функциональным руководителям на уровне заместителей генерального директора или директора департамента, и именно они должны рассматривать свои бизнес-процессы и предлагать мероприятия по их усовершенствованию. Процессный офис в данном случае выступает как внутренний ресурс, который может обучить моделированию бизнес-процессов, формированию регламентов, а также методам анализа и оптимизации бизнес-процессов. Если задачи описания, анализа и оптимизации бизнес-процессов полностью передать в процессный офис, то с большой вероятностью сама технология процессного управления не будет внедрена в бизнес-подразделения, которые с радостью отдадут все задачи по оптимизации и регламентации бизнес-процессов процессному офису, сами наблюдая со стороны, как их деятельность будут пытаться усовершенствовать.

В российских компаниях существует множество процессных офисов, которые не имеют организационных рычагов для оптимизации бизнес-процессов, и тогда они сосредоточиваются на описании существующих бизнес-процессов, закрепляя их текущее состояние в виде графических моделей или регламентов.

Наиболее эффективна схема, в которой бизнес подразделения самостоятельно формируют видение существующего бизнес-процесса, совместно с процессным офисом и ИТ-подразделением создают видение процесса "как должно быть", после чего активно участвуют во внедрении целевого бизнес-процесса. Именно такая структура опробована в большинстве международных компаний. Ведь при активном участии бизнес-подразделений в описании, анализе и оптимизации бизнес-процессов процессный офис можно сделать достаточно компактным по штатной численности или вообще вывести его на аутсорсинг.

Ведь задача процессного офиса заключается не в описании, анализе и оптимизации существующих бизнес-процессов вместо менеджеров, а в продвижении культуры и инструментария процессного управления среди менеджмента компании при активной поддержке верхнего руководителя.

**Ответственность за бизнес-процесс**

Одним из ключевых вопросов в процессном управлении является определение ответственности за бизнес-процессы. Каждое функциональное подразделение отвечает за свою часть бизнес-процесса, но как только ставится задача найти ответственного за весь бизнес-процесс, среди функциональных руководителей возникает конфликт, что требует вмешательства руководителя, находящегося по управленческой иерархии выше их. Некоторые компании решают вопросы конфликтов, возникающих в рамках кросс-функциональных процессов, совещаниями, на которых разбираются детали ведения операционной деятельности, а также рассматриваются те или иные кейсы. Если в рамках совещания вырабатывать общие правила и фиксировать их в формате моделей или текстового описания, то со временем необходимость таких совещаний отпадет, а между подразделениями возникнет понимание, закрепленное в формализованном виде, как необходимо взаимодействовать по бизнес-процессу.

Если самостоятельного взаимодействия на уровне руководителей и исполнителей подразделений в рамках бизнес-процесса не складывается, то количество конфликтов растет, а на совещаниях конструктива все меньше. В таком случае сотрудники процессного офиса могут выполнять функцию рефери по разграничению позиций сторон с одновременной формализацией достигнутых договоренностей.

Есть вариант, когда в рамках кросс-функционального бизнес-процесса может быть определен старший руководитель, который становится "главнее", чем другие участники бизнес-процесса, - это владелец процесса. Его задача - не только добиться синхронизации действий участников процесса, но и обеспечить контроль результатов, а также инициацию мероприятий по оптимизации бизнес-процессов.

Фактически владелец процесса отвечает за результат процесса, несмотря на то что в процессе участвуют сотрудники других подразделений. В рамках управления процессом появляется рамочная ответственность - за качество исполнения отдельной операции отвечает участник бизнес-процесса - сотрудник функционального подразделения, тогда как за форматы и своевременность передачи и приема результатов работ по всему бизнес-процессу между подразделениями отвечает владелец процесса. Именно владелец процесса анализирует организационные и информационные разрывы, возникающие на границах подразделений, а также определяет мероприятия по их устранению.

Расчет фактических значений показателей по процессу тоже входит в зону ответственности владельца, ведь именно он заинтересован в организации контроллинга бизнес-процесса для понимания точек неэффективности, которые негативно влияют на целевые показатели процесса. На практике владельцем процесса может быть руководитель уровня директора департамента или даже выше. При этом над владельцем процесса есть еще одна роль - это куратор процесса, который определяет целевые показатели, формирует стратегические установки, а также распоряжается бюджетом по бизнес-процессу.

Назначение владельцев процесса очень схоже с назначением руководителя проекта, разница лишь в том, что руководитель проекта назначается на разовую активность, в которой участвуют несколько исполнителей от функциональных подразделений, а владелец процесса отвечает за регулярную деятельность. При этом руководитель проекта ориентирован на планирование будущего плана работ, тогда как владелец процесса сосредоточен на анализе прошлого с целью в дальнейшем сделать его эффективнее.

**Варианты организации процессного офиса**

В крупных компаниях часто возникает вопрос, как должен быть построен процессный офис и каковы должны быть его функции. Существует три способа организации процессного офиса: первая подразумевает централизацию на уровне штаб-квартиры, вторая - полную децентрализацию по подразделениям, и третья, подразумевающая гибридную организацию, часть процессного офиса находится в штаб-квартире, а часть рассредоточена по подразделениям. В случае полной централизации процессный офис часто настолько отдаляется от реальных бизнес-процессов, что теряет компетенцию в бизнесе и превращается в бесполезную структуру, занимающуюся лишь стандартизацией бизнес-процессов. И хотя стандартизация тоже полезна, проникновение процессных технологий в бизнес-подразделения затруднено.

В случае полной децентрализации процессного офиса не существует единого координационного центра, что приводит к несовместимости получаемых результатов и дублированию работ. Примерами неудобства децентрализованной формы организации процессного офиса могут быть использование разных нотаций для моделирования бизнес-процессов, закупка различных инструментов дли описания бизнес-процессов, различные шаблоны регламентирующей документации и многое-многое другое.

Гибридная форма построения процессного офиса признана наиболее эффективной, поскольку наверху находится минимальная структура, отвечающая за методологию и общие правила, а на уровне подразделений сосредоточены основные ресурсы процессного офиса, помогающие бизнесу заниматься описанием, анализом и оптимизацией бизнес-процессов.

Численность процессного офиса может варьироваться от одного человека до нескольких сотен, при этом чем больше процессных задач расположено внутри основных бизнес-подразделений, тем процессный офис компактнее, а распространенность процессных технологий шире.

## 2.3. Движение в направлении гибкости. Пилотные процессы и управление изменениями

Один из непростых вопросов в применении процессного подхода заключается в придании построенным бизнес-процессам гибкости. В компаниях очень часто можно увидеть, что построенный *бизнес-процесс* "костенеет" со временем и перестает отвечать изменившимся требованиям потребителей. Если процесс автоматизирован в информационной системе, в которую сложно внести изменение, то процесс становится не просто "закостеневшим", а "окаменевшим", ведь для его изменения нужно переписать существующую информационную систему, на что у ИТ-*подразделения*, как правило, нет ни времени, ни ресурсов.

Именно поэтому в случаях внедрения процессного подхода необходимо особое внимание уделять не только поддержанию гибкости существующих бизнес-процессов на организационном уровне, но и обеспечению гибкости бизнес-процесса на уровне ИТ-систем.

Помимо информационных систем гибкости процессов мешает нежелание его участников меняться самим и изменять существующий *бизнес-процесс*. Человеческой психике свойственно использовать проверенные варианты действий, и поэтому убедить участников бизнес-процесса в том, что им теперь надо работать по-другому - быстрее и качественнее, да еще и меньшим составом, - бывает достаточно непросто. И здесь необходимо искать сторонников - тех, кто хорошо воспринимает изменения, делать их агентами изменений и, пользуясь поддержкой владельца процесса, обучать всех остальных участников новым алгоритмам работы.

Наверное, 70% ресурсов в проекте оптимизации бизнес-процесса приходится именно на внедрение новой версии бизнес-процесса в *деятельность* организации. При этом если процесс критичный или затрагивает множество участников, то необходимо перед внедрением бизнес-процесса его тщательно протестировать - например, в формате деловой игры или используя средства имитационного моделирования, а уже затем опробовать процесс на пилотном участке или регионе. Как правило, в рамках пилотного внедрения бизнес-процесса обнаруживаются нюансы, о которых забыли подумать при разработке целевого состояния бизнес-процесса "как должно быть", а при внедрении они становятся видны.

Российская специфика, заключающаяся в управлении бизнес-процессами через рассылку бумажных регламентов, описывающих *бизнес-процесс* "как должно быть", часто нивелирует проблему внедрения новых бизнес-процессов, ведь регламент выслан, и руководители надеются, что люди стали работать по-другому. В некоторых компаниях это срабатывает, а в некоторых сотрудники сразу удаляют *письмо* с регламентом, даже не читая его.

Технология внедрения через регламент становится нерезультативна, если новый *бизнес-процесс* и его *представление* на бумаге делаются без участия руководителей подразделений, или руководители подразделений сами отстраняются от участия в проектировании бизнес-процессов. Если они не участвовали в создании бизнес-процесса и соответствующего регламента, то и при его внедрении будут сложности. Скорее всего, возникнет мнение, что "мы тут работаем, а вы нам своими бизнес-процессами и правилами только мешаете". Такая корпоративная культура, скорее всего, не позволит изменить существующие *бизнес-процессы*.

Для обеспечения внедрения новых бизнес-процессов в корпоративную практику нужно вовлекать весь руководящий состав и операционных сотрудников в работу над бизнес-процессами "как должно быть", ведь иначе новый процесс не будет отражать всех нюансов и не будет принят сотрудниками в качестве норматива для работы, если они не будут участвовать в разработке нового бизнес-процесса.

На практике сложно что-то изменить в существующих бизнес-процессах, если сотрудникам сверху спускается регламент или стандарт, который выработан без их участия и обсуждения. Именно поэтому столь важно развивать корпоративную культуру, направленную на непрерывное совершенствование бизнес-процессов на уровне силами всех сотрудников компании, и здесь процессный *офис* должен быть поддержан руководством в части необходимости переноса *работ* по перепроектированию процессов внутрь операционных подразделений, а не создавать отдельное подразделение, которое в отрыве от бизнеса будет разрабатывать бизнес-правила и стандарты, часто без участия представителей бизнеса.

Одним из инструментов вовлечения сотрудников компании в непрерывное совершенствование бизнес-процессов является "банк идей" - это *бизнес-процесс*, в рамках которого происходит генерация идей по оптимизации бизнес-процессов, *ранжирование* предложений, их фильтрация и отбор на основании тех или иных параметров.

Фактически в оптимизацию бизнес-процессов вовлекаются все сотрудники компании, а специально созданная система мотивации сподвигает их не только на генерацию идей, но и на их последующее внедрение. Организация "банка идей" позволяет показать всю неэффективность существующих бизнес-процессов, подняв ее с уровня исполнителей на уровень руководства. И хотя при генерации идей большая их часть может оказаться "мусором", даже 10% идей, пригодных к внедрению, могут дать серьезный рывок оптимизации бизнес-процессов в компании.

## 2.4. Регламент как основа процессного подхода

Как уже говорилось, при внедрении процессного подхода необходимо сначала определить существующий *бизнес-процесс*, а после его изменения внедрить его через регламент или информационную систему в *деятельность*.

Фактически одним из инструментов внедрения процессного управления на практике является разработка, согласование, утверждение, рассылка и *контроль* исполнения регламентов бизнес-процессов, ведь именно регламент содержит описание бизнес-процесса "как должно быть" и является нормативным документом, в соответствии с которым участники бизнес-процесса должны взаимодействовать между собой.

И хотя описание бизнес-процесса в виде графической модели является наиболее качественным, если придерживаться определенных правил и структуры, то можно описывать *бизнес-процессы* и в простом текстовом редакторе, иногда прилагая графическую схему бизнес-процесса в виде приложения к регламенту.

Существует множество примеров различных регламентов, некоторые из которых достигают сотни листов. Однако чем больше по объему регламент бизнес-процесса, тем меньше шансов, что его прочтут, и еще меньше - что станут исполнять. Именно поэтому лучше сразу разрабатывать предельно короткий и "сухой" регламент объемом не более пяти листов текста без учета приложений.

На вычитку и согласование регламента уйдет немало времени управленцев, и поэтому чем "суше" будет текст документа, тем меньше времени уйдет на его вычитку и обсуждение.

В начале документа нужно указать цель регламента, например, установление порядка взаимодействия при расчете заработной платы или сокращение времени обработки заявки клиента. Далее необходимо сделать раздел с терминами и сокращениями, которые будут использоваться в регламенте бизнес-процесса: например, ДУП - Департамент управления персоналом или ЦФО - центр финансового обеспечения.

После терминов и сокращений необходимо расположить общую часть регламента, куда разместить все положения, касающиеся бизнес-процесса в целом, например, система мотивирования разрабатывается для руководителей уровня директора департамента. Отдельной строкой в конце общей части регламента желательно прописать, для кого предназначен данный регламент: например, регламент предназначен для сотрудников ДУП и руководителей ЦФО.

Ну и самая главная часть, в которой будет содержаться описание регламентируемого бизнес-процесса. Эту часть необходимо заполнять максимально короткими фразами по следующей структуре. Кто и когда (специалист по оплате труда, еженедельно, до 18.00 пятницы). Затем, переведя строку и сдвинув нумерацию вглубь, можно написать перечень действий, которые исполняются специалистом (согласовывает систему мотивации для новых сотрудников, передает систему мотивации директору ДУП).

Таким образом шаг за шагом описывается весь *бизнес-процесс* с указанием перечня документов, которые передаются по процессу, и информационных систем, которые используются для выполнения той или иной *операции*. Если необходимо указать комплект документов, возникающий в процессе, то это можно сделать либо в общих положениях, либо в тексте описания бизнес-процесса.

Как правило, даже очень масштабные *бизнес-процессы* могут быть описаны в такой структуре. При этом нужно иметь в виду, что краткое описание редко встречающихся сценариев исполнения бизнес-процессов можно вынести в *приложение* или в общие положения: например, в случае необходимости подготовить предложение в срочном порядке руководитель ЦФО формирует рабочую группу в составе… Такой подход позволит сосредоточиться на основном сценарии протекания бизнес-процесса.

После согласования и утверждения регламента он становится нормативным документом, по которому должны действовать участники бизнес-процесса. Фактически регламент содержит правила взаимодействия подразделений при работе в том или ином бизнес-процессе.

Есть у регламентов в российских условиях одно слабое *место* - это необязательность их исполнения, вызванная сложностью контроля исполнения всех правил регламента. Проверить, насколько сотрудники работают по регламенту, можно двумя способами. В первом случае это *аудит* с анализом деятельности и ее проверкой на соответствие заложенным в регламенте процедурам. Второй вариант - это разбор конфликтов, возникающих в бизнес-процессе, *информация* о которых передается владельцу процесса от участников, если один из них начинает нарушать регламент в *ущерб* другим. Второй вариант - это система самоконтроля исполнения процесса, когда *исполнитель*, получающий задачу в рамках бизнес-процесса, контролирует качество работы предыдущего исполнителя и соответствие его действий алгоритмам и временным ограничениям, закрепленным в регламенте.

Если в компании нет никакой культуры работы в соответствии с утвержденными регламентами, то приходится привлекать на помощь *информационные технологии*. И в этом случае приходится "заставлять" людей работать по правилам, прописанным в регламенте, автоматизируя процесс в информационной системе, чтобы она в автоматическом режиме контролировала соответствие всех действий заложенным правилам.

Этот способ требует больших затрат, поскольку для автоматизации необходимо закупить софт и силами разработчика создать ИТ-решение, которое будет автоматизировать процесс, не позволяя нарушить его логику и контролируя сроки исполнения отдельных операций и эффективность каждого из участников процессов. Но после автоматизации бизнес-процесса владелец получит мощный *инструментарий*, который позволит управлять бизнес-процессом с минимальными потерями времени.

## 2.5. Краткие выводы

Процессное управление решает проблемы, возникающие в организационных структурах, построенных по функциональному принципу, за счет определения и оптимизации сквозных и кросс-функциональных бизнес-процессов в организации.

Подразделением, которое сопровождает внедрение процессного подхода в организации, является процессный *офис*.

Сотрудники бизнес-подразделений должны при поддержке процессного офиса описывать, анализировать и совершенствовать свои *бизнес-процессы*.

Наиболее трудоемким является внедрение бизнес-процессов в практику деятельности компаний, что требует изменения не только самих процедур, но и корпоративной культуры в компании.

Инструментом внедрения новых бизнес-процессов в *деятельность* организации являются либо организационно-распорядительные документы, такие как регламенты и стандарты, либо *информационные системы*, в которых закрепляются все правила маршрутизации задач по бизнес-процессу, а главное - обеспечиваются средства контроля над параметрами исполнения процесса.

**Питання до теми:**

# Лекция 3: ПРЕДПОСЫЛКИ ВОЗНИКНОВЕНИЯ AGILE

План

3.1. Состояние области процессов разработки программного обеспечения

3.2. Сравнение каскадного/итерационного/Agile процессов

3.3 Эффективная таблетка от болезней?

3.4 Для кого подходит, а для кого нет?

3.5. Краткие выводы

## 3.1. Состояние области процессов разработки программного обеспечения

О том, насколько все неутешительно в области внедрения, разработки и сопровождения программных продуктов, не говорил только совсем ленивый консультант.

Наиболее актуальные обзоры и собранная *статистика* авторитетных отечественных и западных изданий демонстрируют, что признанные и наиболее известные методологии разработки программного обеспечения не оправдывают возложенные на них ожидания.

В качестве примера приведем исследование Standish *group*, проведенное в 2015 году. Оно было выполнено *по* результатам анализа данных о внедрении программных продуктов за период с 2001 *по* 2010 год. Фундаментом исследования послужила *информация* *по* более чем 10 000 проведенных проектов различной степени сложности ( рис. 3.1).



Рисунок 3.1 – Статистика проектов в разрезе процессных методологий за период с 2001 по 2010 г.

На приведенной диаграмме продемонстрировано, что удовлетворенность от использования классического водопадного подхода значительно ниже в сравнении с применением Agile.

Общий *тренд* очевиден. Современные компании нашли в Agile успешную альтернативу классическому подходу управления информационными технологиями.

Причины столь требовательного отношения к самой сфере, а главное - к ее результатам, достаточно полно раскрыты в лекции "Введение в Agile" . Дополнительного внимания заслуживает факт повышающейся интеграции между информационными системами и всеми остальными направлениями жизнедеятельности современного мира.

Затруднительно представить направление деятельности, в котором зависимость от информационных систем была бы незначительна. Технологичные продукты, автоматизирующие операционные процессы, выполняют все те функции, которые возлагали на них в самом фантастичном фильме, созданном не более 10 лет назад. Фантастика становится реальностью. *Информатизация* материализуется не только в виде социотехнических систем, но и в виде повсеместно используемых сервисов.

Не стоит сильно упиваться локальными победами. Предстоит еще поработать над многими глобальными вопросами, не решенными к сегодняшнему моменту и возникающими *по* ходу наступающей эры информатизации.

Основные проблемы и сложности, которые явно можно выделить, порождаются следующими причинами:

* постоянное усложнение предметных областей, автоматизируемых информационными системами;
* скорость изменений, влияющих на структуру процессов;
* неудовлетворительные способы описания поведения комплексных информационных систем для дальнейшей поддержки и их развития;
* необходимость обеспечения достаточной гибкости программных продуктов;
* трудности управления процессом разработки.

Проблемы и задачи, которые мы пытаемся решить с помощью программного обеспечения, часто неизбежно содержат элементы, сложность которых обусловлена мультиаспектностью и завышенными ожиданиями от их реализации целевых пользователей информационных систем.

Относительно редко за автоматизируемым функционалом видят предварительную обязательную процессную составляющую, о которой было подробно рассказано во лекции "Процессный подход как конкурентное преимущество" . Только после того как будет отлажена процессная составляющая, можно переходить к ее последующей автоматизации. Но это только в теории. На практике приходится комбинировать организацию разрозненных, кусочных "процессиков" в единый архитектурный монолит, от качества разработки которого будет зависеть успешность автоматизируемого направления деятельности. Но это только одна из задач, которую предстоит учитывать в процессе разработки программного продукта.

Современный информационный мир имеет еще одну сторону, которая оказывает огромное влияние на него. Аспект скорости развития, изменения, дополнения информации, которая лежит в основе предпосылок создания программных продуктов и автоматизации бизнес-процессов.

Высокий темп изменений приводит к тому, что к соответствующим программам предъявляется множество различных, порой взаимоисключающих требований.

Пользователи и разработчики имеют разные взгляды на причины появления проблем. На основе разных предпосылок о сути возникших задач эти группы делают различные выводы о возможных альтернативах их решения. Поэтапная, инкрементальная разработка функционирующих версий системы позволит пользователям лучше понять и яснее сформулировать то, что им действительно нужно.

В то же время итерационный процесс позволит:

* постепенно повышать квалификацию разработчиков в предметной области за счет более тесного и продуктивного сотрудничества со стейкхолдерами разрабатываемой системы;
* увеличивать терпимость бизнес-пользователей за счет того, что они более отчетливо будут представлять процессы разработки и сложность вопросов автоматизации, которые разработчикам приходится решать каждый день.

Еще одна задача, которую постоянно приходится решать группе разработчиков, - это проблема адекватного описания поведения больших информационных систем.

В основе проектирования каждой системы лежит принцип декомпозиции, который постулирует разделение большой системы на части так, чтобы одна минимально воздействовала на другую, но при этом сохранялась общая *целостность* всего продукта. Любое внешнее *по* отношению к системе событие может привести ее в новое состояние. Проблематичность этой ситуации заключается в неопределенности перехода между известным и новым состояниями. Как правило, проблематичность снимается за счет основательного анализа и последующего всеобъемлющего тестирования таких программ, но при неблагоприятных дополнительных условиях небольшое неизвестное внешнее событие может привести к критической ошибке в функционировании системы. Чем сложнее система, тем легче ее полностью развалить. Задача адекватного накопления информации и знаний *по* системе и ее окружению с последующим документированием призвана предупредить возможные "информационные" катастрофы.

Еще одним действенным способом предупреждения проблем является "природная" гибкость информационных систем, заложенная принципами подходов к их разработке.

*Программирование* обладает предельной гибкостью, и каждый разработчик может *по* необходимости сам обеспечить себя всеми элементами, относящимися к любому уровню абстракции. Такая гибкость чрезвычайно соблазнительна и действенна, если использовать ее *по* назначению. Она предписывает разработчику создавать все базовые блоки будущей архитектуры информационной системы, из которых составляются элементы более высоких уровней абстракции, но при этом эта гибкость имеет и свои недостатки, которые выражаются многообразием возможных элементов и порой их дублированием и избыточностью. В отличие от более проработанных направлений деятельности, в которых накоплен *массив* статистических данных, на основе которого целесообразно делать выводы на предмет того, как правильно организовывать *конструирование* и реализацию базовых артефактов, информационным технологиям, несмотря на интеллектуальность и инновационность этой сферы, только предстоит встать на централизованный *путь* стандартизации подходов к конструированию. На текущий момент направление, которое включает в себя *определение* таких подходов (системная инженерия), еще только становится на этот *путь*. Уже накоплены определенные подходы, но считать их конечными и успешными неправильно. Способы взаимодействия в окружающем цифровом мире очень разнообразны. Этот фактор не является катализатором процессов информационной глобализации.

Кроме перечисленных факторов остается еще один аспект, влияние которого на создаваемые программные продукты является подавляющим. Это *процесс управления* разработкой информационных систем.

Основная задача разработчиков состоит в создании иллюзии простоты, в защите пользователей от сложности описываемого предмета или процесса. Сегодня обычными стали программные системы, размер которых исчисляется десятками тысяч или даже миллионами строк на языках высокого уровня. Ни один человек никогда не сможет полностью понять такую систему. Поэтому такой объем *работ* потребует привлечения команды разработчиков. Чем больше разработчиков, тем сложнее связи между ними и тем сложнее *координация*, особенно если участники *работ* географически удалены друг от друга.

На решение и управление перечисленными факторами и направлена Agile. Ее применение позволит решить обозначенные проблемы.

## 3.2. Сравнение каскадного/итерационного/Agile процессов

В первой лекции мы упомянули о двух противодействующих методологических лагерях классического и нового стиля работы. Настало время обсудить их подробнее.

К первому лагерю относятся так называемые тяжелые, классические подходы к разработке программного обеспечения. Мы кратко рассказали о них в ракурсе исторической ситуации возникновения Agile.

Сейчас рассмотрим их суть, преимущества, недостатки, необходимые условия и возможные результаты их применения.

Первой *по* значимости и распространенности на сегодняшний момент для процессов управления информационным технологиями является самая известная на сегодня водопадная модель разработки информационных систем. В литературе часто встречаются *альтернативные* названия этого стиля разработки программных продуктов - каскадный, классический и пр.

*Каскадная модель - это вид процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток последовательно проходящих фаз анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.*

Эта модель разработки была впервые презентована в 1970 году в статье Винстона Ройса. Он описал в виде концепции то, что сейчас принято называть "*каскадная модель*", и обсуждал ее недостатки. В этой же статье он показал, как этот тип процесса разработки программных продуктов может быть доработан до итеративной модели разработки *ПО*. В оригинальной каскадной модели Ройса следующие этапы процесса были представлены в таком порядке:

* Определение требований.
* Проектирование.
* Конструирование/реализация/кодирование.
* Воплощение.
* Тестирование/отладка/верификация.
* Инсталляция.
* Поддержка.

Переход между фазами возможен только после полного и успешного завершения предыдущей. Следуя предложенной структуре, разработчик переходит от одной стадии к другой последовательно. Сначала полностью завершается первый этап ("*Определение* требований"), в результате которого появляется полный и исчерпывающий *список* требований к информационной системе. Только после этого возможен переход к проектированию, в ходе которого разрабатываются документы, подробно, ясно и непротиворечиво описывающие для программистов способ и план реализации зафиксированных требований. Далее начинается этап реализации требований в виде программного кода системы. *По* сути, этот этап и является ядром разработки программных продуктов. Его результаты будут определять дальнейшее качество последующих этапов и информационной системы в целом. Затем происходит *интеграция* отдельных компонентов, разрабатываемых различными способами, в единую систему. После того как перечисленные этапы завершены, производится тестирование и последующая *отладка* продукта. Тут устраняются все недочеты и ошибки, появившиеся ранее. После этого *программный продукт* внедряется и осуществляется его *поддержка* - разработка новой функциональности, устранение ошибок и т. д.

*Каскадная модель* строится на постулате последовательных переходов от одной фазы разработки к другой только после полного и успешного завершения предыдущей. Переходы назад, "перепрыжки" вперед, перекрытия фаз - недопустимы.

Каскадную модель организации процесса разработки программного обеспечения критикуют за недостаточную гибкость, объявление самоцелью формальное *управление проектом* в *ущерб* срокам, стоимости и качеству. Но ее четкая структурированность и формализация являются неоспоримыми ценностями, которые способствуют снижению многих возникающих рисков.

*В PMBOK 3-й версии формально была закреплена только методика каскадной модели и не были предложены альтернативные варианты, известные как итеративное ведение проектов или Agile.*

Следующей *по* значимости и распространению является итеративная модель разработки программного обеспечения.

*Итеративный подход в разработке программного обеспечения - это выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл PDCA: планирование - реализация - проверка - оценка.*

Итеративные (или инкрементальные) модели используют иной подход к организации деятельности. Взамен единой продолжительной последовательности этапов применяется *разбиение* жизненного *цикла* разработки продукта на набор отдельных мини-циклов. Каждый из них включает в себя те же базовые стадии, применяемые в водопадной модели. Мини-цикл - это *итерация*. В каждой итерации происходит разработка отдельного компонента или компонентов системы (инкремента). Далее реализованный *компонент* добавляется к уже существующему функционалу.

Итеративная модель, в противовес классической модели, не предполагает полного объема требований для начала *работ* *по* реализации продукта. Разработка программы должна начинаться с основных требований к базовой части функционала. Далее, на последующих итерациях, реализованные требования будут дополняться, модифицироваться, приводя к расширению функционала системы. Процесс повторяется, обеспечивая создание новой версии продукта для каждой итерации.

*С PMBOK 4-й версии был достигнут компромисс между приверженцами каскадной модели разработки ПО и профессионалами, делающими ставку на итеративные методы. Это привело к тому, что начиная с 2009 года Институтом управления проектами (PMI) предлагается как стандарт гибридный вариант методологии управления проектами, сочетающий в себе плюсы как от водопадной методологии, так и от достижения итеративных методов.*

Залог успешного применения этой модели - четко выстроенные этапы тестирования/отладки/верификации требований и тщательная *валидация* разрабатываемой функциональности в каждой из итераций.

Итеративная модель, *по* сути, является переходной (от каскадной к Agile) моделью разработки программного обеспечения и, *по* мнению многих специалистов, оптимальной моделью разработки программного обеспечения.

При сравнении классических (водопадной и итерационной) методик с Agile необходимо осознание того, что *каскадная модель* - это подход хорошо описанный и детализированный, а Agile - это набор практик и принципов, в которых так или иначе поддерживаются различные методологии гибкой разработки проектов.

*В последней версии PMBOK уже идет речь об итеративно-инкрементальном подходе как об основном рекомендованном. PMI в процессе разработки собственной Agile-сертификации.*

Сравнивая эти подходы, следует отметить, что оба имеют набор преимуществ и недостатков. Возможны ситуации, когда обоими методами можно организовать реализацию необходимого программного обеспечения, но на старте процессов разработки *входные данные* *по* ресурсным параметрам (*стоимость*, время, квалификация персонала), а также требования к качеству информационной системы и т. д. существенно влияют на выбор методологии.

В качестве сильных сторон водопадной модели следует выделить следующие:

* Легкость для понимания и последующего применения.
* Подробная структурированность, что облегчает ее применение к малоопытным командам.
* Модель с самого начала задает стабильные требования к проекту/продукту.
* Проекты легко контролируются, отслеживаются ресурсы, риски, время.
* Качество имеет первоочередной приоритет по сравнению со стоимостью и временем.

Agile, в свою *очередь*, обладает следующими сильными сторонами:

* Итеративный подход к разработке программного обеспечения.
* Использование четких временных рамок.
* Заинтересованные пользователи вовлечены в процесс разработки с самого начала.
* Быстрое получение первой/пробной версии продукта для тестирования.
* Легко воспринимаются корректировки и изменения в процессе разработки.

Слабыми сторонами подходов являются следующие:

"Каскад":

* Требования должны быть определены и детально описаны до начала стадии разработки.
* Высокая цена.
* Медленный темп работы.
* Чувствительность к изменениям.
* Мало возможностей для конечного пользователя повлиять на цели проекта и требования к продукту.
* Зачастую проблемы выявляются только на этапе тестирования.
* Много документации, которая непонятна конечному пользователю или заказчику.

Agile:

* Может привести к низкому качеству продукта.
* Существует риск никогда не достигнуть поставленной цели при инициации процесса.
* Могут возникнуть проблемы с расширяемостью продукта.

На основе приведенных факторов становится возможным сделать целесообразные выводы о ситуациях, когда использование того или иного подхода является оптимальным.

Классические модели предпочтительнее использовать в ситуациях, когда требования к продукту предельно ясны и стабильны, определены используемые технологии и инструменты, речь идет о внедрении большого и сложного программного обеспечения. Примером может служить проекты внедрения ERP-систем.

Agile следует применять в тех случаях, когда *конечный пользователь* вовлечен в проект со старта, определены бизнес-цели проекта/продукта, проект небольшой или средний, относительно короткий *по* времени, состав команды стабильный, с высоким уровнем профессионализма, технические требования приемлемые, связаны с технологиями, которые собираются быть использованными для разработки, а информационная система *по* своей сути является модульной.

Представленные подходы обладают плюсами и минусами, каждый из которых прекрасно подходит для применения в проектах с совершенно разными исходными данными и требованиями.

При выборе методологии необходимо выбрать ту, которая подходит для достижения поставленных целей проекта. Необходимо понимать структуру, принципы, преимущества и недостатки каждой из них. В некоторых случаях это не выбор между методологиями, а правильная комбинация подходов для каждого из этапов конкретного процесса или проекта.

За последнее время изменился рынок потребителей *ПО*. Вместо больших проектов компании стремятся к использованию маленьких и средних проектов. Компании - разработчики *ПО* стремятся к более быстрому, частому, регулярному выпуску информационных систем.

Согласно недавнему опросу, 80% решений о внедрении в компании Agile-методологий и идей, принадлежат менеджерам высшего и среднего звена.

## 3.3 Эффективная таблетка от болезней?

Итеративный и Agile методы разработки программных продуктов подходят не всем. Во всех компаниях существует своя специфика, которая определяется множеством разнообразных факторов различной природы, определяемой организационными, административными и специализированными условиями внешней и внутренней среды. В ряде компаний использование водопадной модели обосновано и экономически целесообразно. Можно привести много причин, почему Agile использовать не стоит.

Разработка качественной информационной системы - дело непростое. Каждый, кто сулит реальность "волшебного зелья", гарантирующего выпуск программного продукта, решающего все проблемы и снимающего "боль" компании, - шарлатан, продающий плацебо.

Консультанты, специализирующиеся во внедрении и продвижении Agile, первым делом заявляют, что Agile - не волшебная пуля, магическим образом убирающая все проблемы, но оставшуюся часть времени вам говорят, что это лучшее изобретение человечества.

Agile - это набор устоявшихся методик, основанный на итеративной разработке программного обеспечения.

Конечно, есть литература, книги, эксперты, конференции и т. д., но следует помнить, что это всего лишь устоявшиеся методики, которые базируются на:

* изменениях;
* приспособляемости;
* людях.

Методология Agile, несмотря на обилие существующей литературы, имеет строго практическую направленность. Не получится просто прочесть несколько статей, изучить книгу, пройти курс и внезапно стать гибким. Важно параллельно с освоением теоретических азов постигать практические приемы работы. Agile дает многочисленные преимущества рабочим группам и небольшим проектам. Это является основой для обсуждения преимуществ, которые они могут предоставлять для разработки крупных модульных систем.

## 3.4 Для кого подходит, а для кого нет?

В сообществе ИТ-менеджмента Agile - одна из самых популярных тем уже на протяжении долгого периода времени. Происходит это из-за волнообразного спроса на повышение эффективности процессов разработки в различных сегментах деятельности. Мнения *по* этому поводу самые разные.

При обсуждении необходимо учитывать предпосылки, которые демонстрируют преимущества и недостатки, основываясь не столько на отдельных оценках уважаемых экспертов, а на фактах, возникающих в процессе эксплуатации Agile различными участниками процессов. Именно взвешенные мнения участников процесса позволят установить истину, рожденную в дебатах.

Agile, несмотря на демократичность, подразумевает наличие жесткой дисциплины. Это может способствовать более эффективной организации труда и, как следствие, максимизации получаемого результата. Но это кое-чего стоит.

Для распространения необходимой информации *по* процессу в Agile предписывается для выполнения ряд необходимых мероприятий. В них на постоянной основе должны участвовать не только специалисты, занятые разработкой программного обеспечения, но и менеджеры, активное участие которых необходимо и во многом предопределяет дальнейший успех создаваемого продукта.

Если *менеджер* не сможет участвовать в предписываемых гибкой методологией встречах, то необходимо задуматься о делегировании соответствующих полномочий в развитии разрабатываемой системы коллеге или подчиненному, который сможет вдумчиво, настойчиво, ясно и с полным осознанием сути задачи донести ее до команды и впоследствии принять созданный результат. В противном случае суть Agile будет выхолощена и принцип "первого руководителя", *по* мнению многих, самый действенный при внедрении и применении различных процессов, будет нивелирован. Как следствие, Agile не достигнет постулируемой эффективности, и *программный продукт* будет создаваться разработчиками для разработчиков.

Код создаваемой системы должен быть простым и читаемым, чтобы его можно было легко менять, причем в любой момент и разным специалистам. Для этого требуется постоянно критически его пересматривать и заниматься рефакторингом разработанной, но устаревшей функциональности. Важно иметь человека, который сможет критически подойти к актуальной ситуации и при необходимости обозначить нужные доработки в целях облегчения дальнейшей сопровождаемости и развития информационной системы

Для того чтобы держать планку заданного качества, требуется создавать юнит-тесты. Это приводит к тому, что на разработчика вешается вдвое больше работы. Но если этого не делать, то повышается *вероятность* выпуска низкокачественного продукта, который будет не соответствовать ожиданиям пользователей.

Для того чтобы разрабатываемый продукт был целостным с точки зрения концепции его развития, необходимо постоянно обсуждать вопросы анализа, проектирования, выполнять *контроль* деятельности менее опытных разработчиков, отыскивать ошибки. Одна из техник, которая может позволить выполнить намеченные задачи, называется парным программированием. Его суть состоит в парной, поочередной разработке кода двух разработчиков в единицу времени за одним рабочим местом. Это очень тяжелое *испытание* для тех, чей уровень коммуникационных способностей находится на достаточно низком уровне. Представьте себе, что кто-то сидит рядом с вами и постоянно указывает на то, как, *по* его мнению, правильно реализовать тот или иной участок создаваемой информационной системы. Более того, иногда (а вернее, очень даже часто) будет требоваться напрямую общаться с заказчиками функционала. Agile позволит создавать работающий продукт с высоким качеством в прогнозируемые сроки. Единственное важное условие - заказчик должен быть заинтересован в создании работающего продукта с высоким качеством в прогнозируемые сроки.

Большинству разработчиков интересно реализовывать сложный, "заковыристый" *модуль*, но неинтересно писать простой, но надежно работающий продукт. Разработчики более заинтересованы в увлекательном исследовательском процессе, а что будет происходить с заказчиком в момент использования разработанной функциональности клиентом - их, как правило, не интересует.

Специалисты ценят свой комфорт за компьютером. Рабочее *место* - родной дом и непререкаемая зона комфорта. *Менеджер*, ответственный за процесс, должен изменить ситуацию, когда собственный комфорт важнее, чем работающий софт, успех компании и т. д. Требуется постоянно доносить до ответственных специалистов необходимость и нужность их работы, ее влияние на процессы компании.

Agile предписывает изменения в подходах не только к работе, но и к устоявшимся привычкам, если они противоречат общим целям. Такой процесс не только постулирует желание и готовность к постоянным переменам и "ломкам" характера и профессиональных устоев. Он привносит мотивацию, которой вряд ли можно ожидать в компаниях, проповедующих классические подходы к процессам разработки программного обеспечения, как, например, техника парного программирования, о которой мы поговорим позже.

Итогом становится то, что для эффективного внедрения Agile необходим ряд факторов, которые смогут обеспечить его оптимальное применение в компании:

* заинтересованность руководства в достижении поставленных результатов;
* желание и способность участвовать в развитии продукта;
* наличие квалифицированной команды разработчиков, способной создавать качественный продукт и анализировать результаты и существующий процесс в целях его оптимизации;
* готовность к открытому диалогу и конструктивной критике всех членов команды;
* желание изменений и перемен.

В последующих главах курса мы дополним, раскроем и детализируем приведенный *список* факторов.

## 3.5. Краткие выводы

Ситуацию, благодаря которой Agile получила право на существование, трудно назвать идеальной. Состояние области разработки и внедрения программного обеспечения сложно назвать стабильной и отвечающей ожиданиям заинтересованных в ее использовании. Agile возникла не как академическая дисциплина, а как ответ на несостоятельность рыночным реалиям ее конкурентов. Не просто возник, а явился результатом вдумчивого процесса размышления признанной группы консультантов и экспертов над проблемами в области управления разработкой программного обеспечения и уже на протяжении долгого периода показывает свою состоятельность и результативность.

Все это говорит о том, что Agile завоевала свою аудиторию и стала популярной. В то же время *по*-прежнему еще слабо представляется, как правильно работать *по* Agile. Приведем слова Майкла Кона: "Мне хочется, чтобы все Agile-бренды в конечном счете исчезли, и осталось просто то, как мы разрабатываем *программное обеспечение*".

Можно и нужно сравнивать различные направления Agile, такие как Scrum или Kanban, сосредоточиваясь на техническом совершенстве каждого из них, или же, напротив, развивать коммуникации и командную работу. Все это в конечном итоге приведет к совершенствованию и развитию Agile, повышению эффективности в зависимости от условий, в которых ее используют.

**Питання до теми:**

# Лекция 4: ФИЛОСОФИЯ РАБОЧЕГО ПРОЦЕССА

План

4.1. Типы Agile-методологий и их распространенность

4.2. Scrum - гибкий управленческий процесс

4.3. Как воспитывать сотрудников?

4.4. Как управлять сопротивлением?

4.5. Чем нужно управлять в Scrum

4.6. Значимость соблюдения процесса

4.7. Инженерные практики

4.8. Краткие выводы

## 4.1. Типы Agile-методологий и их распространенность

После того как Agile завоевала определенную нишу в области процессного управления сферы информационных технологий, стало очевидно, что для дальнейшего развития необходимы конкретные практические подходы, адаптированные под "боевые" условия конкурентной среды функционирования предприятий.

*Agile - набор принципов и методик, которые необходимо встроить в деятельность компании.*

Каждая организация имеет свои неповторимые черты, свой уровень зрелости процессов, свои условия, в которых она добивается поставленных целей. Использовать универсальные принципы для компаний с низким уровнем зрелости и для организаций, занимающих в лестнице *CMMI* самые высокие позиции, было бы недальновидным решением. Agile не подразумевает пошаговых руководств и конкретных рекомендаций, а лишь своими принципами задает рамки, в которых используют гибкие методологии.

В данной главе мы обсудим ряд гибких методологий разработки программных продуктов, использующихся при создании программных продуктов ( рис. 4.1).



Рисунок 4.1 –Популярность Agile-практик, 2015 г.

**Kanban**

Это яркий представитель процессной методологии, которая появилась благодаря японскому послевоенному экономическому чуду в сфере машиностроения и постепенно была заимствована в сферу информационных технологий.

В основе Kanban лежат три базисных принципа:

* *Визуализация (иероглиф "кан").* При иллюстрировании и моделировании процесса он разбивается на отдельные стадии (анализ, проектирование, разработка, тестирование и т. д.), упрощая таким образом его восприятие.
* *Ограничение максимального количества задач на определенном этапе.* Этот принцип позволяет свести потери к минимуму - максимальное сосредоточение на своих задачах.
* *Оптимизация существующего процесса.* Время на выполнение задачи отслеживается, анализируется, и вырабатываются предложения о том, как можно выполнить работу более совершенно. В процессе не должно быть простоев, равно как и не должна выполняться ненужная работа. Kanban характеризуется утверждением: "Уменьшение выполняющейся в данный момент работы". Данный тип методологии является, пожалуй, самым гибким. Это значит, что она является наиболее требовательной к условиям и ресурсам, в рамках которых предполагается ее эксплуатировать. Сотрудники, работающие по Kanban, должны быть готовы к экстремальной гибкости и при этом не должны ломаться.

Уникальность методологии Kanban, в сравнении с конкурентами по семейству Agile, состоит в:

* Способе распределения задач. Каждый специалист, работающий в команде разработчиков, может взять на себя лишь ограниченное количество, при этом выбор задач он осуществляет самостоятельно, а не по чьему-то указанию.
* Отсутствии временных рамок. Kanban не предполагает ограничения на время выполнения поставленных задач.
* Размере задач, которые необходимо реализовать. В сравнении с аналогами задач меньше, но их объем и трудоемкость значительно больше.
* Отсутствии активности оценки и планирования. Оценки сроков на задачу: опциональные или вообще их нет.

Выгода от применения приведенных постулатов весьма спорна. Как выполнять контроль над разработкой, если мы убираем основные инструменты контроля - сроки, скорость работы и т. д.? Ответ на приведенный вопрос мы оставим на более поздние обсуждения, которые приведем в следующих главах.

*Менеджер должен заботиться о контроле и сделать все для его организации. Существует точка зрения на контроль как на фикцию. Обосновывается она тем, что команда должна получать удовольствие от работы и выкладываться на полную катушку. Звучит перспективно, но по факту сотрудников, которым можно было бы доверять полностью, не так много. А вернее сказать, их единицы.*

Kanban ориентирована на задачи. Сотрудники работают над задачами с самого начала и до завершения. Рабочая команда не должна оценивать время на выполнение задачи, ибо это имеет мало смысла и почти всегда ошибочно вначале. Если менеджер верит команде, то зачем иметь оценку времени?

**Lean**

Этот тип методологии подразумевает создание продукта в условиях максимальной экономии ресурсов с целью устранения всех возможных потерь. Изначальный функционал продукта ограничивается до минимально полезного. Таким образом, функционал реализуется путем поэтапного наращивания функционала небольшими порциями, сродни инкрементальному принципу реализации информационных систем. Корни этого подхода относятся к принципам бережливого производства (Lean Manufacturing). Мэри и Том Поппендик, о которых было упомянуто в лекции «Введение в Agile» курса, адаптировали эти принципы для разработки программного обеспечения:

* устранение потерь;
* повышение качества;
* создание знаний;
* отсроченные обязательства;
* быстрая поставка;
* уважение людей;
* полная оптимизация.

Lean постулирует отказ от всего, что не добавляет ценности создаваемой информационной системе. Разрабатывать необходимо только то, в чем есть абсолютная уверенность, что это нужно делать сейчас. Устранение потерь во всех аспектах работы (бесполезные собрания, избыточные задачи, документация, неэффективные способы работы и т. д.). Акцент на то, что называется "системным подходом", то есть сотрудники работают как единое целое, как команда. Необходимо верхнеуровневое осознание того, что выполняемая работа помогает повышать ценность создаваемого продукта, в сравнении с аналогами. Не нужно заставлять людей работать на 150% их рабочего времени. Не нужно кодировать то, что не нужно. Дополнительный функционал создает дополнительные обязательства для пользователей и руководства. Сотрудники нуждаются в уважении как личностном, так и профессиональном. Нужно давать им ту работу, которую они лучше всего знают, как надо сделать. Смысл программной разработки в постоянном обучении. Принятие управленческих решений нужно выполнять в последний возможный момент. К моменту реализации необходимого функционала сотрудники будут знать уже больше и лучше ориентироваться не только в создаваемой системе, но и в бизнес-процессах и данных организации.

**Экстремальное программирование**

Авторами этой методологии являются Кент Бек, Уорд Каннингем, Мартин Фаулер. Именно об этих людях мы говорили, когда в лекции "Введение в Agile" делали историческое отступление об авторах Agile. Методология получила свое название благодаря воплощению идеи применить полезные классические методы и техники разработки программного обеспечения, подняв их на качественно новый "экстремальный" уровень. К примеру, этап ревизии кода, заключающийся в аудите одним программистом кода, написанного другим программистом, в "экстремальном" варианте представляет собой "парное программирование". Таким образом, мы получаем рабочий процесс, в ходе которого один программист занимается кодированием, а его напарник в это же время непрерывно просматривает только что написанный код. Через некоторое время они меняются ролями.

Если подвести локальный итог рассмотренных ответвлений гибких методологий в целом, то главным минусом станет "плавающая" оценка сроков разработки и бюджета. К неоспоримым плюсам стоит отнести малые сроки производства продукта, отсутствие простоев на время согласования проектной документации, оперативное управление изменениями и пр.

Последняя, оставшаяся к рассмотрению и, пожалуй, наиболее распространенная методология из семейства Agile - Scrum.

По результатам актуальных исследований и опросов, именно Scrum является самой популярной техникой управления процессами разработки программного обеспечения

**Scrum**

В Scrum требования разбиваются на небольшие подгруппы, каждая из которых должна быть максимально независима от другой. Это делается для того, чтобы в течение каждого нового спринта (промежутка между выпуском новых версий продукта) команда разработчиков могла реализовать новый функционал. Scrum подразумевает постоянную коммуникацию - будь то планирование очередной итерации работ совместно с клиентом или ежедневный Scrum-митинг, в течение которого команда обсуждает ряд наиболее важных вопросов, ответы на которые непосредственно влияют на создаваемый программный продукт. Именно на примере описания и подробного рассмотрения Scrum в дальнейшем будет вестись речь о Agile.

## 4.2. Scrum - гибкий управленческий процесс

Принципы итеративной разработки программного обеспечения предполагают действия *по* определенным итерациям. Небольшие кусочки, каждый из которых должен представлять собой законченную функциональность, выполняются в соответствии с постепенным наращиванием дополнительных более качественных и детализированных атрибутов автоматизации разрабатываемого продукта должны быть реализованы в определенных временных рамках. Суть заключается в том, что вы постоянно, циклично делаете все необходимое для выкладки очередной версии продукта. Продукт становится качественнее, а *команда* сотрудников, разрабатывающая информационную систему, приобретает *опыт* на всех этапах разработки и с каждой следующей итерацией начинает все лучше понимать истинные потребности бизнес-пользователей. Результатом каждой итерации является *инкремент*.

Scrum - это стандартизированный вариант организации процесса разработки программного обеспечения для итеративной и инкрементальной разработки. Для работы *по* Scrum необходим набор артефактов, с помощью которых и выполняется управление и направление создаваемого продукта.

Scrum с самого начала создавался для управления процессами контроля, планирования и анализа на всех этапах создания информационной системы. Благодаря такому подходу к разработке он пользуется высокой популярностью у команд, занимающихся поддержкой и/или сопровождением программных продуктов.

Среди *множества* других, одной из особенностей этой методологии следует выделить ежедневные короткие совещания, во время которых идет обсуждение результатов предыдущих итераций (спринтов) и ставятся задачи на следующие. Это позволяет заинтересованным менеджерам держать под контролем процесс разработки, при необходимости направляя его в нужное русло.

Достоинства Scrum:

* Scrum ориентирован на клиента, адаптивен.
* Scrum дает клиенту возможность делать изменения в требованиях в любой момент времени, но не гарантирует то, что эти изменения будут выполнены.

Возможность изменения требований привлекательна для многих заказчиков:

* Scrum достаточно прост в изучении, позволяет экономить время за счет исключения некритичных активностей.
* Scrum позволяет получить потенциально рабочий продукт в конце каждой итерации работ.
* Scrum делает упор на самоорганизующуюся многофункциональную команду, способную решить необходимые задачи с минимальной координацией.

Это особенно привлекательно для малых компаний, так как избавляет от необходимости найма или обучения специализированного персонала дорогостоящих руководителей.

Scrum содержит и важные недостатки. Ввиду простоты и минималистичности, Scrum задает небольшое количество довольно жестких правил. Однако это вступает в *конфликт* с идеей "клиент всегда прав", так как заказчикам не важны внутренние правила команды разработки, особенно если они его ограничивают.

Так же общеизвестной проблемой Scrum являются большие издержки на обсуждения, встречи и большие потери времени на стыках спринтов. Как *минимум* день уходит на закрытие очередного спринта, а потом день - на открытие нового. В условиях, когда спринт длится 2 недели, 2 дня составляют 20% рабочего времени. Итог - 30-40% времени при применении Scrum тратится на поддержание самого процесса (ежедневные встречи, планирование и детализирование *работ*, презентация полученных результатов и пр.)

Scrum относится к семейству Agile, в котором не приветствуется создание планов коммуникаций, реагирования на риски, управления сложными техническими изменениями. Итог - сложное, а иногда невозможное формальное (юридическое или административное) противодействие нарушениям правил Scrum. Конечно, нерадивого сотрудника можно просто уволить, но это не является системным способом решения проблемы. Процессы в компании должны быть выстроены так, чтобы, оставляя определенную степень свободы сотруднику, всегда можно было бы контролировать его не только количественно, но и качественно. Для этого есть множество техник и методик, применение которых характеризует зрелость компании в целом и ее процессов в частности. Все лидирующие в различных сегментах компании характеризуются низким уровнем увольняемости. Ротация сотрудников нужна для каждой организации, но ротация не должна быть массовой. Когда уходят ключевые сотрудники, организация ставит себя под удар. В Scrum этот недостаток попытались преодолеть за счет постулата универсализма и упор на самоорганизующуюся многофункциональную команду. При кажущемся снижении затрат на координацию команды это приводит к повышению затрат на отбор персонала, его мотивацию, обучение. В определенных условиях рынка труда, формирование полноценной, эффективной Scrum-команды может быть невозможным. Хорошие сотрудники стоят дорого, и руководство организации должно отдавать себе в этом отчет.

Фокус Scrum - управление продуктом в условиях неопределенных и часто меняющихся требований, которые характеризуют современный неустойчивый информационный мир.

## 4.3. Как воспитывать сотрудников?

После того как менеджментом компании будет принято решение использовать Scrum в операционной деятельности, необходимо организовать его внедрение. Наиболее важным аспектом в вопросе внедрения является *адаптация* сотрудников компании к новому рабочему процессу. *Адаптация* будет эффективной в том случае, когда изменения преподнесены не как новое управленческое веяние, а как необходимость. Организации, в которых подобный подход к процессам и активностям управления изменениями организован на высоком уровне зрелости, имеют значительное преимущество перед конкурентами за счет их гибкости и адаптивности под постоянно меняющиеся условия внешней и внутренней бизнес-среды.

Причины, *по* которым сотрудники становятся лояльными к организации, ее целям и задачам, продолжают трудиться на ее благо, крайне разнообразны и определяются многими факторами. Продолжать рассуждать о важности разделения сотрудниками миссии компании нецелесообразно.

*О важности организационного менеджмента написано достаточно толковой литературы. Сосредоточимся на личности работника.*

Если компания заинтересована в сотрудничестве с определенными работниками, ей нужно уметь продумать, иногда индивидуально, как удерживать того или иного "уникума". Для этого можно выделить *сегменты* респондентов в соответствии с длительностью работы, проанализировать ответы ветеранов для установления возможных закономерностей. *По* результатам опроса обсуждают пути продолжения работы, выявляют возможные проблемы.

Общие рекомендации, используя которые можно учесть 90% возможных проблем, заключаются в управлении следующими факторами:

* *"Воспитание" эффективности.* Каждый сотрудник должен иметь перед собой трудные, но достижимые цели, выраженные в виде совершенствования или развития навыков полезных для компании и интересных самому работнику. Эти цели подкрепляются в виде четких и понятных планов развития, декомпозированных на конкретные задачи (расширить компетенции в бизнес-процессах компании, освоить новый инструмент по работе с БД и т. д.), которые должны иметь четкий срок и метрику.
* *Планирование сложных рабочих заданий.* Рабочие задачи следует планировать так, чтобы они были понятны сотруднику и практически подкрепляли развитие его навыков. Таким образом, мы получаем четкую зависимость между его эффективностью, способностью и желанием решать задачи, необходимые для достижения целей компании.
* *Синхронизация понимания между руководством и сотрудниками о задачах развития и пути достижений результата конкретных задач.* Руководство и работники должны находиться в состоянии открытой, честной коммуникации. Доверие приводит к осознанию безопасности. Безопасность - это один из первейших инстинктов, удовлетворение которого приводит к созданию комфортной рабочей обстановки. Выстраивание доверия и согласованности понимания в действиях, необходимых для достижения поставленных целей, - одна из основных задач руководства на пути получения долгосрочного результата от деятельности сотрудников.
* *Командная работа.* Работники могут чувствовать себя изолированно и неудовлетворенно, если они не являются частью сплоченной команды или если ими тайно "жертвуют" по тем или иным причинам. Данная проблема решается за счет создания команд различной величины, которые должны быть саморегулируемы. Саморегуляция достигается, если члены команды понимают значимость командной работы и необходимость достижения результатов.
* *Философия лидерства как новый тип управления командной работой.* При внедрении изменений важным движущим фактором является наличие лидера, который собственным примером сможет продемонстрировать преимущества, которые будут получены командой от применения этих изменений. Этот лидер должен иметь опыт, желательно успешный, работы по внедряемым процессам. Можно очень долго и красиво рассказывать о том, как будет хорошо, но сотрудникам важно иметь коллегу, который сможет подсказать, как правильно и оптимально поступить в тех или иных рабочих условиях. Это является катализатором быстрого достижения результатов от внедрения. В противном случае риск "блуждания" по лабиринтам процессов и увеличения времени на адаптацию увеличивается в несколько раз.

В итоге следует, что для воспитания рабочей философии рекомендуется следовать следующим основным принципам:

* *Выбирайте сотрудников "точечно".* Выбор конкретных работников должен быть взаимосвязан с задачами, которые стоят перед компанией сейчас и должны возникнуть в ближайшей перспективе.
* *Воспитывайте в сотрудниках эффективность или расставайтесь с ними.* Сотрудники и компания должны расти и развиваться вместе. Если сотрудник стремится осваивать новые навыки и при этом использует их в работе, то именно это тот тип работника, который должен стать частью локомотива будущих побед организации. Грамотные исполнители, которые качественно выполняют свои функциональные обязанности, тоже нужны, но они не будут способствовать развитию компании, они будут способствовать ее стабилизации в уже занятой рыночной нише.
* *Сотрудникам необходима возможность обучаться как на рабочих местах, так и в процессе работы пилотных проектов.* После принятого решения об изменениях в процессах компании следует начать с пилотных проектов оптимизации тех активностей, в которых есть лидеры, за которыми смогут начать тянуться остальные. Способствуйте тому, чтобы ваши сотрудники имели возможность получить необходимые знания и информацию об изменениях.
* *Необходим опыт и знания коллег, уже внедривших Scrum.* Лидеры изменения должны на практике знать то, о чем говорят в теории.
* *Необходимо постоянно осведомлять людей о произошедших переменах.* В компании должна быть организована система оповещений и атмосфера доверительной коммуникации о происходящих изменениях. Не пренебрегайте "сарафанным" радио, которое можно организовать через кадровый департамент или аналогичную службу. Доносите информацию даже о самых незначительных победах, обозначайте возникшие проблемы и пути их решения.

Подводя локальный итог темы управления и развития кадров в Scrum, нужно сказать сокровенное: "Кадры решают все". Именно сотрудники и их навыки формируют основную ценность любой компании. А когда речь заходит о гибких методологиях, то *значимость* утверждения, приведенного выше, возрастает в несколько раз. На потенциале и трудолюбии команды будет держаться *качество программного продукта*. Кадрами предстоит заниматься очень много и кропотливо, но эта работа окупается.

## 4.4. Как управлять сопротивлением?

Предпочтения и профессиональные "устои" каждого специалиста формируются в соответствии с широтой его взглядов на мир и шаблонностью профессионального мышления. Как правило, работники стараются бессознательно сопротивляться переменам. Любым. Чем серьезнее и глобальнее перемены, тем более ожесточенные сопротивления. Если речь идет о коренном изменении рабочей философии и переходе на гибкие методологии разработки программного обеспечения, то перестройки кардинальные.

Сопротивление принято снимать за счет жесткой политики перехода, называемой взрывом. Вкратце, она заключается в навязывании сотрудникам новой реальности. "Навязывание" определяется конкретной культурой управления организацией. Руководитель должен быть осторожным и внимательным при принятии любых решений, связанных с проведением изменений, но при этом необходимо решительно придерживаться принятых правил. Даже если они кому-то не нравятся.

Для того чтобы понять, как реагировать на возможное сопротивление, полезно понять, на каком уровне возникают сопротивления:

* на организационном уровне;
* на уровне группы;
* на уровне индивида.

От понимания того, на каком уровне возникает сопротивление и чем оно характеризуется, возникает понимание способов устранения проблем.

На каждом из приведенных уровней есть свои особенности сопротивления и свои приемы воздействия с целью уменьшения сопротивления.

На организационном уровне структурные и культурные факторы могут способствовать широкому распространению сопротивления. Существующие структура и культура не могут оперативно приспособиться к новым требованиям и измениться. Причины заключаются в том, что для приспособления к изменениям в масштабе предприятия необходим продолжительный *интервал* времени и большие ресурсы.

Один из способов уменьшения сопротивления - системный подход. Однако сложность заключается в том, что для понимания поведения организации как системы необходимо учитывать поведение всех взаимосвязанных в организации групп. Системный подход предусматривает рассмотрение организации как единого целого, выявление взаимосвязей между различными частями системы, например, путем изменения иерархического порядка *принятия решений* или обеспечения некоего равновесия между социальной, организационной, технической и прочими частями системы. Если у компании нет ресурсов на применение системного подхода к изменениям, самым действенным способом является лозунг: "Можешь - начинай работать, не можешь - ищи для себя работу". Если речь идет о сотрудниках с невысокой квалификацией, то, возможно, эта стратегия управления изменениями имеет право на существование. Если же речь идет о сотрудниках, которые своими действиями формируют интеллектуальную собственность компании, выраженную в виде автоматизированных процессов и уникальных данных, адаптированных под конкретные условия функционирования организации, то этот подход может привести к катастрофе.

При осуществлении изменений необходимо иметь в виду, что компания как система включает в себя не только формальные группы (управления, отделы, секторы и т. д.), но и неформальные, например группы "ветеранов". Широкое освещение стратегического замысла и консультации перед осуществлением стратегии (в идеале - на стадии проектирования изменений) могут помочь уменьшить сопротивление со стороны групп и выявить, что же действительно беспокоит людей в предложенной ситуации. Для этого может потребоваться:

* информирование о результатах организационной диагностики тех подразделений и групп, которые непосредственно затрагивает изменение;
* проведение семинаров и дискуссий, в которых бы участвовала конкретная группа сотрудников;
* организация информационной сети, чтобы каждый мог узнать о том, что происходит, и имел возможность выразить свои сомнения;
* привлечение на сторону изменений членов влиятельных и авторитетных неформальных групп в организации.

Формальные и неформальные группы, к которым принадлежат сотрудники, придерживающиеся определенных взглядов относительно изменений. Эти взгляды решающим образом влияют на позицию сотрудника - члена группы, которую он будет занимать.

Некоторые сотрудники, несмотря на принадлежность к группе, могут иметь личную обеспокоенность относительно влияния изменения на их будущее положение в организации, возможностей карьеры, реализации устремлений и перспектив.

При управлении ценными сотрудниками важно правильно понять возможные причины сопротивления конкретных групп работников (консерваторы, прагматики, новаторы), и только после этого можно будет выбрать правильную стратегию управления сопротивлением. Чтобы помочь сотруднику приобрести новое понимание происходящего и пересмотреть свое *отношение* к изменению, чаще всего требуется индивидуальная работа с ним *по* разъяснению выгод и преимуществ, которые он лично получит в результате реализации изменений.

Такая работа должна привести к изменению поведения сотрудника. Сопротивляющийся сотрудник - не проблема, а человек, которого необходимо понять и предоставить ему объективную информацию о необходимости выполняемых изменений. Обычно после осознания значимости проводимых изменений сотрудники постепенно переходят на их сторону. При этом важно различать, что причины сопротивления у менеджеров и специалистов могут быть разными *по* причине разницы выполняемых функциональных обязанностей (табл. 4.1).

Таблица 4.1. Основные причины сопротивления

|  |  |  |
| --- | --- | --- |
| **№** | **Специалисты** | **Менеджеры** |
| 1 | Недостаточная информированность | Опасение потери контроля и власти |
| 2 | Страх перед неизвестностью | Нехватка времени |
| 3 | Опасение потерять работу | Устраивает существующее положение вещей |
| 4 | Недостаточная поддержка со стороны руководства | Непонятно, что лично я буду иметь от этих перемен |
| 5 | Невозможность участвовать в принятии решений |  |

Успешная реализация изменений в организации всегда характеризуется умелым применением целого ряда перечисленных подходов, часто в самых различных сочетаниях, и отличается двумя особенностями - менеджеры используют эти подходы с учетом их достоинств и недостатков и реалистично оценивают ситуацию.

Наиболее распространенной ошибкой руководства является использование только одного или ограниченного числа подходов независимо от ситуации. Это касается и сурового начальника, который часто прибегает к принуждению, и менеджера, ориентированного на своих сотрудников, который постоянно пытается привлекать и поддерживать своих людей, и начальника-циника, всегда манипулирующего своими сотрудниками и часто прибегающего к манипуляции, и интеллигентного менеджера, который в большой степени полагается на образование и общение, и, наконец, менеджера типа адвоката, который всё время старается вести переговоры.

Основная задача руководства - принимать верные и своевременные решения, используя различные подходы не только поодиночке, но и в сочетании друг с другом.

## 4.5. Чем нужно управлять в Scrum

Основная цель использования управленческой методологии Scrum - *эффективное управление* ресурсами с возможностью сосредоточения всех участников гибкого процесса на конечном результате. В основе Scrum лежат принципы, предписывающие полностью сосредоточиться на продукте, разработка и развитие которого является основным результатом деятельности. Суть принципов Scrum унаследована от принципов, перечисленных в *Agile Manifesto* (см. лекцию "Введение в Agile" ).

Основное внимание сосредоточено на команде как средстве разработки качественного продукта. Реализация продукта осуществляется за счет тесного и постоянного взаимодействия с заинтересованными сторонами в целом и владельцем продукта в частности. В ходе этого взаимодействия *команда* разработчиков становится более гибкой и готовой к изменениям. Инструменты, документация, условия работы и планы вторичны, но также очень важны. Таким образом, фокус сосредоточен на команде, которая удовлетворяет потребность заказчика.

*Scrum опирается на людей и взаимодействие между ними. Грамотное управление людьми выходит на первый план.*

**Команда**

Команда - группа людей, взаимодополняющих и взаимозаменяющих друг друга, которые собраны для совместного решения задач создания продукта, посредством которых они поддерживают взаимную ответственность перед владельцем продукта. В идеале специалисты, задействованные в Scrum, должны представлять собой самоуправляемый и самоорганизующийся коллектив, цель которого - реализовать требования к продукту в том объеме и с уровнем качества, удовлетворительным для последующего использования продукта.

Успешность рабочего процесса обеспечивается за счет непрерывного общения друг с другом и со всеми заинтересованными сторонами. На всех этапах работы над продуктом происходит непрерывное взаимодействие, в ходе которого определяется, как можно реализовать конкретную задачу с учетом достоинств и недостатков конкретной реализации. Вся полученная в ходе взаимодействия информация учитывается и используется в дальнейшем. В последующем это помогает оптимизировать процесс разработки наилучшим образом и постепенно совершенствоваться и развиваться. Идеал команды в Scrum - самоуправляемый коллектив профессионалов, объединенных общей целью.

**Требования**

Для работы над продуктом у сотрудников, объединенных в единую команду, есть все необходимое - Product Owner, бэклог продукта…

Scrum продуктивен из-за своей простоты. Члены команды имеют свободу самим определять, каким образом разрабатывать продукт. И в этом скрыта самая большая сложность Scrum. Одним из основных качеств, которые отделяют начинающего от состоявшегося профессионала, является черта самоанализа, которая заключается в постоянном переосмыслении того, что и как он делает. Цель самоанализа - улучшить работу, над которой идет размышление. Необходимо вырваться из рамок "так положено" и понять, что конкретно нужно сделать для достижения результата наименьшими затратами. Гибким быть непросто. Для того чтобы быть эффективным, нужна соответствующая квалификация. Именно уровень этой квалификации во многом предопределяет качество проработки требований, которые необходимо реализовать. Но при этом необходимо учитывать, что инженерия требований для разработки функционала должна выполняться в необходимом объеме. Требования должны создаваться непрерывно при активном сотрудничестве заинтересованных лиц с командой разработки. Важно помнить, что конечная цель сбора требований в том, чтобы облегчить и разделить общее понимание требований для всех тех, кто вовлечен в процесс разработки. Требования могут различаться по детальности и степени проработки и принимать самые разные формы. Для получения общего представления о требованиях полезно иметь какую-нибудь форму или шаблон документации, определяемой командой для каждой конкретной ситуации.

В Scrum в качестве инструмента используются use case. Более подробно о технике проработки требований мы поговорим в лекции "Атрибуты Scrum" .

**Продукт**

Основное внимание Scrum-команды сосредоточено на артефакте, разработка которого является их обязанностью, - продукте. Работа над продуктом должна строиться исходя из актуального понимания существующих потребностей в функционале. Можно долго размышлять о возможных путях развития, но эти размышления должны основываться на стойком базисе действительности. Любые прогнозы меняются в зависимости от тактических данных. Тренд необходим, но это не должно быть определяющим фактором в работе. Важно итеративно разрабатывать "нужную" функциональность, постоянно измерять ее эффективность и, отталкиваясь от ключевых актуальных метрик, продумывать дальнейшее направление развития продукта.

*Продукт - живая субстанция, в развитии которой участвует вся команда, но его "очертания" задает владелец продукта, который "кошельком" отвечает за конечный результат.*

Можно выделить еще несколько ключевых факторов, управление которыми будет влиять на создание продукта (риски, технологии, инженерные практики и пр.), но основными являются именно сам продукт и команда, которая участвует в его разработке. Остальные являются либо производными, либо второстепенными, и в соответствии с их значимостью для внедрения и адаптации Scrum будут рассмотрены далее.

## 4.6. Значимость соблюдения процесса

Во второй лекции были обоснованы важность и необходимость регламентации существующих в компании процессов как средства контроля, мониторинга и поиску возможностей дальнейшего развития деятельности организации. Наличие регламента позволяет гарантировать организационный порядок, но для этого каждая компания, проводя внедрение определенной процессной деятельности, адаптирует эталонные принципы выбранной методологии для реалий своей компании и стремится к наиболее эффективному использованию существующего процесса в рамках имеющихся ресурсов. Условием и гарантом эффективности процесса является регламент, следуя которому можно достичь запланированных результатов в определенные сроки и прогнозировать дальнейшее развитие функционала.

В Scrum-документах зафиксировано, что основной ценностью являются люди, которые за счет оптимального взаимодействия друг с другом должны достичь запланированных результатов. Именно за счет командного взаимодействия должна достигаться рабочая гармония. *Значимость* "записанного" порядка взаимодействия снижается, во главу угла выходит *человеческий фактор*. При создании и следовании процессу необходимо держаться принципа золотой середины, то есть нужно руководствоваться основными постулатами методологии, но корректировать тактику в зависимости от конкретных условий окружающей среды и в уме держать необходимость внедрения принципа гибкости. Командный процесс в идеале должен получиться:

* прозрачным;
* инспективным;
* адаптивным.

В этом плане Scrum представляет собой процесс "с человеческим лицом", в котором каждый член команды должен контролировать своего сотоварища, так как за результат деятельности несет ответственность вся *команда*.

*Рабочий процесс* должен находиться под контролем владельца продукта. Это позволяет избежать нестабильности и добиться возможности достоверно прогнозировать результаты посредством управленческих инструментов. Для обеспечения его контролируемости в Scrum предусмотрены все необходимые атрибуты (доска задач, бэклог продукта, бэклог спринта), о которых мы поговорим чуть позже.

## 4.7. Инженерные практики

Под инженерными практиками понимается проверенный временем набор технических решений, связанных непосредственно с разработкой программного продукта. Сложно разработать качественный продукт без внедрения этих практик в повседневную работу.

Немаловажным фактором является понимание преимуществ и возможностей того или иного инструмента, а также сложностей и специфики его внедрения.

Большинство практик, применяемых в Scrum, своими корнями уходят в экстремальное *программирование*. Именно на этой методологии рассматриваемые далее практики доказали свою эффективность.

Для начала поговорим о технике **Code Review**. Это *деятельность*, связанная с ревизией, производимой командой кода. Ревизия должна осуществляться на систематической основе. Она позволяет обнаружить и исправить ошибки, оставшиеся незамеченными автором, но заметные для того, кто проводит ревизию. Оговоримся, что ревизию должен проводить каждый раз следующий член команды.

*Во многих командах ревизию проводит назначенный дежурный, в обязанность которого на определенный временной период включен Code Review.*

Данная практика позволяет разрабатывать более стабильный и качественный *программный продукт*. *Код программы* становится более структурированным и стабильным, плюс ко всему постепенно каждый член команды, прошедший должность ревьюера, начинает более глубоко понимать все детали реализуемой информационной системы.

Далее поговорим о **Unit Testing**. Это техника, в основе которой лежат инструменты для проверки работоспособности отдельных частей программного продукта как *по* отдельности, так и в совокупности для проверки интеграции модулей и отдельных систем. Реализация *Unit Testing* позволяет своевременно реагировать на некорректные изменения логики программного продукта, приводящие к ошибкам и сбоям. Unit-тестирование способствует стабилизации программного продукта и его места в информационном ландшафте заказчика, снижению количества проблем при интеграции.

**Code Refactoring.** Это не просто техника. Благодаря развитию и распространению Agile рефакторинг стал полноценным техническим направлением деятельности. **Refactoring** предписывает осуществление изменений внутренней структуры исходного кода, которые не должны повлечь изменения в поведении программы. За счет применения **Code Refactoring** достигается упрощение структуры программного продукта и повышение скорости его работы. Это способ систематического приведения кода в порядок, при котором шансы появления новых ошибок минимальны. В сущности, при проведении рефакторинга кода вы улучшаете его *дизайн* уже после того, как он создан.

Как следствие получаем экономию на дальнейшем сопровождении и поддержке.

**Build Automation.** Это техника автоматической сборки исходного кода, выполнения тестов и развертывания программы на целевом ландшафте. *Автоматизация* позволяет повысить качество процесса разработки и поставки продукта. Результаты *Build* *Automation* не так очевидны бизнес-заказчикам системы, но за счет внедрения этой техники с технической команды снимается часть работы, после чего они могут направить освободившиеся ресурсы на более значимые для бизнес-пользователей активности.

**Continuous Integration.** Практика непрерывной интеграции направлена на выполнение частых автоматизированных сборок программного продукта. Цель - выявление и решения интеграционных проблем. Переход к непрерывной интеграции позволяет снизить трудоемкость интеграции и сделать ее более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий.

Все перечисленные инженерные практики, безусловно, связаны между собой и дают максимальный эффект лишь при работе в комплексе. Каждая из них направлена на поддержание стабильного рабочего процесса, ведущегося *по* Scrum. Их введение способствует созданию качественных информационных систем быстро и с меньшими рисками.

В промышленной разработке программного продукта самое важное - взаимодействие с заказчиком. Использование инженерных практик и подходов Agile подразумевает регулярную и довольно тесную *связь* с ним для понимания того, как каждая из внедряемых практик влияет на удовлетворенность заказчика. Каждый из этапов работы над продуктом становится более открытым, понятным и обсуждается на каждой стадии. Внедрение и использование каждой инженерной практики возможно только после того, как владелец продукта оценит ее результат. *Команда* разработки и заказчик берут на себя одинаковую ответственность за конечный результат.

## 4.8. Краткие выводы

Agile представляет не столько направление процессной методологии, которое делает акцент на коммуникации взамен избыточному документационному обоснованию выполняемых *работ*, сколько семейство различных техник, которые доказали свою успешность на протяжении долгого периода эксплуатации в различных типах деятельности.

Основным представителем семейства гибких процессов *по* распространенности и эффективности является Scrum. Именно на его рассмотрении будет сосредоточен фокус данного курса. Главной отраслью, в которой Scrum нашел свое применение на сегодня, является сфера информационных технологий. Именно поэтому в Scrum появились специализированные артефакты, которые призваны адаптировать его под реалии данного направления деятельности.

При внедрении любой процессной методологии и Scrum в частности необходимо помнить о том, что основной движущей силой любого изменения являются сотрудники. Поэтому необходимо понимание того, как должно выполняться адаптирование нововведения, как необходимо работать с персоналом и каким аспектам внедрения уделить наибольшее внимание.

"*Ядро*" Scrum - это *команда* квалифицированных мотивированных специалистов, заинтересованных в развитии программного продукта. Коммуникации внутри команды позволяют преодолевать возникающие проблемы и решать задачи за счет общих договоренностей. Кроме коммуникации для успешного достижения результатов целесообразно применять проверенные временем успешные инженерные практики. Они смогут помочь в решении первостепенных технических задач и сосредоточить внимание профессионалов на разработке качественной информационной системы.

В каждой команде необходимо наличие сотрудников, каждый из которых будет обладать определенными навыками и квалификацией, что позволит выполнять им определенные функции. Таким образом, *команда* состоит из работников, выполняющих определенные роли.

О команде и ролях мы поговорим в лекции "Роли Scrum" .

**Питання до теми:**

# Лекция 5: РОЛИ SCRUM

План

5.1 Команда

5.2 Этапы командообразования

5.3 Разработчик

5.4 Scrum-мастер

5.5 Владелец продукта

5.6 Самоорганизация членов команды

5.7 Прочие члены команды

5.8 Самоорганизующийся коллектив

5.9 Краткие выводы

## 5.1 Команда

*Команда* - это *группа* профессиональных работников, занимающиеся различной трудовой деятельностью. Смысл командной работы состоит в производстве высоких результатов за счет кооперирования между сотрудниками высокой квалификации и производства совокупного результата, который должен превосходить сумму результатов членов команды.

Безусловно, есть категории асов, предпочитающих работать в одиночку, но в сегодняшнем высокотехнологичном мире одиночная работа стала уделом немногих. Только при наличии команды можно рассчитывать на производство качественного программного продукта в заданные сроки. Именно время - это то превосходство, которое может предоставить командная работа для компании за счет более быстрого вывода информационной системы на рынок. На командной работе базируется методология Scrum.

Топ-*менеджмент* большинства организаций полагает, что, отдавая предпочтение отдельным специалистам, проще осуществлять *контроль* и управление коллективом в целом. Когда есть необходимость в хороших исполнителях, руководство нацелено на то, чтобы в фирме собрать самых опытных работников. Вопросы качества и хороших результатов становятся решенными априори после того, как нужный работник найден. Не так все просто.

*Ресурс* отдельного, хоть и профессионального исполнителя ограничен многими факторами, которые определяют необходимый результат. Но когда мы начинаем говорить о командной работе, то начинают работать совсем другие принципы. Командный *ресурс* состоит из ресурсов исполнителей, к которому добавляется синергетический эффект, приводящий общий механизм в движение. Эту материю сложно пощупать, но она работает и отделяет успешные команды от сборища заурядных исполнителей.

Характеристиками эффективных команд являются следующие:

* *Стремление к совершенству.* Лучшие группы объединены единой целью. Для достижения поставленных задач они постоянно используют наиболее эффективные варианты, не пользуясь теми, которые плохо зарекомендовали себя в прошлом, благодаря чему переходят в ранг от посредственных до высоких уровней, получают понятие своих возможностей и имеют повышенные требования к получаемым результатам.
* *Независимость.* Обладающие этим качеством штаты работников самоорганизованны и самоуправляемы. У них есть возможность принимать решения, переходить к их реализации. Они научены действовать самостоятельно.
* *Универсальность.* В состав лучших команд включены специалисты профилей (планирования, разработки, продажи и реализации производства), необходимых для самодостаточной работы команды. Главными качествами являются взаимная поддержка, взаимодействие и понимание между членами трудового процесса.

Каким образом удается создать группу людей с целью достижения высоких результатов? Будет ли она самоорганизованной и создавать атмосферу взаимного обучения и информирования? Чтобы найти ответы на эти волнующие вопросы, один из основателей Scrum Д. Сазерленд потратил уйму времени. Как добиться результата, чтобы люди соединились в одно целое, думали о непрерывном усовершенствовании? Точно не методом террора и жесткого контроля, ведь каждый должен прийти к пониманию, что это ему действительно необходимо. Давление даст только отрицательные результаты. Вероятнее всего, есть набор несложных правил, помогающих достичь требуемого результата.

Все зафиксированные правила укладываются в единый лозунг: "Одна *команда* - от старта до финиша". Главное требование к команде, которая должна создавать необходимый продукт, - многофункциональность. Это выражается в наличии всех специалистов, необходимых знаний и практики для успешного выполнения заданий.

Стандартная организационная структура включает в себя отдельные *подразделения* управления, анализа, разработки, тестирования, внедрения и т. д. Они укомплектованы сотрудниками данных специализаций. Подобная структура полностью соответствует классическому подходу к разработке программного обеспечения. Но на сегодня необходимы команды, которые будут состоять из всех необходимых сотрудников и, оперативно реагируя на внешние вызовы, отражать их в конечном продукте. Именно *по* такому принципу и построены Scrum-команды. Все члены подобных команд должны иметь определенные задачи и функции в работе, выполняемые в рабочих интервалах, за которые должен быть сформирован очередной вид рабочего продукта (спринты). Функциональные отделы не выпускают отдельно продукт, а *команда*, в противовес функциональной организации, все делает сообща. Это во многом дублирует принципы процессного подхода, что еще раз подчеркивает, что Scrum является эффективной процессной методологией.

Но *команда* - более сложный организационный элемент, нежели отдельный сотрудник. *Команда* возникает, живет, развивается и заканчивает функционирование *по* своим принципам, которые необходимо знать. Понимание этих принципов поможет оптимально ею управлять и прогнозировать результаты, которые может предоставлять *команда* на различных этапах своего жизненного *цикла*.

## 5.2 Этапы командообразования

Эксперты, занятые вопросами создания эффективной организационной среды, сходятся во мнении, что методика Scrum помогает компаниям достигнуть желаемых результатов производства программного обеспечения за счет внедрения инкрементного подхода производства. Во главе этого лежит базисный принцип разработки информационной системы или сервиса в виде создания минимально необходимой функциональности для конкретной временной даты и предоставления этой версии в пользование клиенту. После чего выполняется использование созданного артефакта, и *команда* приступает к следующей итерации работы над продуктом, который развивается на основе полученных результатов. И так далее, итерационно, шаг за шагом продукт приходит к тому виду, применение которого оптимально для заданных рабочих условий компании.

*"На начальном этапе созданная компания выпускала раз в пару месяцев новый продукт. Мы стремительно росли, но управление происходило каскадным методом. Показатели упали до выпуска одного продукта в год. Пришлось немедленно принимать серьезные меры. Мы решили воспользоваться Scrum, благодаря чему вернули прежний уровень производства. Немногие имеют такой результат. Мы этим гордимся".*

Существует простая методика, которая может позволить проверить "правильность" мышления команды. Нужно в разговоре с любым сотрудником, включенным в Scrum-команду, задать вопрос о состоянии их группы. Оптимальным будет, если член команды ответит о разрабатываемом продукте, а не будет рассуждать об их текущей специализации. В случае, когда сотрудник не связывает себя с общим продуктом, речь идет о недопонимании целей Scrum или же о прочей "проблемной" причине. Если же есть тенденция, что вся *команда* находится в "кризисном" состоянии, тогда необходимо диагностировать ее состояние и принимать соответствующие меры.

Чтобы *группа* людей превратилась в сплоченную и эффективную команду, необходимо пройти несколько этапов.

Наиболее распространенной моделью, описывающей стадии командообразования, является модель Такмана ( рис. 5.1).



Рисунок 5.1 – Этапы командообразования

В своем развитии все команды обязательно проходят несколько этапов.

**Формирование**

Этот этап во многом является отправной точкой для создания команды и постановки целей на разработку или развитие продукта. Как правило, на этом этапе выполняются создание или обновление состава, постановка целей деятельности всего состава команды в целом и отдельных ее членов, не очень понимающих те задачи, выполнение которых поставлено перед ними для достижения общего результата.

**Бурление**

На этом этапе все члены команды отчетливо осознают свои и общие цели, задающие вектор движения коллектива. В период бурления часто возможны конфликты и противостояние между отдельными членами команды, формирование мелких группировок, поэтому особенно возрастает роль Scrum-мастера (SM) как модератора (его обязанности мы обсудим чуть позже). Этап бурления - самое время для определения формальных и неформальных лидеров Scrum-команды.

**Нормализация**

На этапе нормализации происходит "притирание" членов команды друг к другу. Закрепляются лидеры команды, и она начинает демонстрировать стабильные результаты деятельности. Основная задача Scrum-мастера на этом этапе - помощь команде для перехода на следующий этап.

**Функционирование**

Функционирование - это этап, на котором команда переходит к самоуправляемости. На этом этапе команда способна оптимизировать свою производительность, существующие неформальные связи между членами команды закрепляются окончательно. Этот период наибольшей эффективности и производительности.

*Хотя команды являются самоуправляемыми, они не становятся бесконтрольными. Руководство устанавливает контрольные точки, чтобы избежать нестабильности и хаоса. В то же время руководство избегает жесткого микроконтроля, который убивает креативность.*

**Расформирование**

Цели, поставленные перед командой, достигнуты, мотивация большинства участников в этот период убывает. Профессиональный Scrum-мастер должен четко отслеживать рубеж между этапом функционирования и расформирования, и предпринимает шаги, которые не позволят команде сильно понизить их производительность. В качестве примера подобных шагов можно привести изменение состава команды, смену целей, предоставление нового продукта, над развитием которого должна трудиться команда, и т. д.

Чтобы работать эффективно, команда должна как можно дольше находиться на этапе функционирования. Соответственно, главной задачей команды (и, в частности, Scrum-мастера) является максимально быстрый переход между этапами, предваряющими стадию функционирования, и замедление этапов, которые идут после. Но надо отдавать себе отчет в том, что команда также подвержена влиянию организации, ее управленческим процессам, которые протекают в компании, и возможны временные и "сезонные" отклонения, определяемые, скажем, аттестациями, выплатами премий и пр.

Команду нужно держать в тонусе, но необходимо тщательно отслеживать состояние наиболее "заслуженных" работников в целях предотвращения возможных выгораний и срывов.

Организация командной работы - это очень важная часть деятельности Scrum-мастера, который своей работой должен обеспечить комфортное рабочее состояние всем ее членам.

## 5.3 Разработчик

Scrum-*команда* должна состоять из мотивированных профессиональных сотрудников. Это факт, на котором построена *оптимальность* данной гибкой методологии. Члены команды отвечают за разработку программного продукта высокого качества. Они должны обладать множеством различных навыков, необходимых для создания эффективного программного обеспечения:

* Бизнес и системный анализ.
* Проектирование архитектуры программного продукта.
* Программирование.
* Тестирование.
* Настройка баз данных.
* Разработка эргономичных пользовательских интерфейсов.

Члены команды участвуют в планировании *работ* на временной *интервал*, который в Scrum принято называть "спринт". *Команда* может включать опытных разработчиков и новичков, которые в процессе работы должны совершенствоваться при обмене знаниями. Члены команды отвечают за следующие задачи в проекте:

* обязательное выполнение элементов работ, включенных в текущий спринт:
	+ акцент на взаимосвязанных задачах спринта;
* совершенствование личной и командной работы.

Основной движущей силой команды является ее участник, которого принято называть разработчик (*Development* *member*) - специалист, представляющий собой "*ядро*" Scrum. От его навыков, умения, квалификации, опыта будет зависеть дальнейшее развитие продукта и Scrum-команды. Как правило, *команда*, работающая над продуктом, состоит из 4-15 человек. Помимо владения теоретическими знаниями каждый член команды должен уметь практически применить следующие навыки:

* общаться с пользователями для понимания целей производимых разработок;
* собирать "правильные" требования, которые в дальнейшем можно сопоставить с ожиданиями ключевых пользователей;
* планировать дальнейшую работу над продуктом в виде его инкрементного развития;
* документировать в минимальном объеме требования;
* выполнять синтез разрозненных и противоречивых требований в единое комплексное рабочее решение;
* создавать удобный, читаемый, понятный, однозначный код ИС в целях его дальнейшего развития;
* проводить полный цикл разных видов тестирования;
* выполнять установку программного обеспечения;
* поддерживать пользователей и отвечать на любые, даже глупые, вопросы.

Разработчика Scrum-команды целесообразно сравнивать со швейцарским ножом, который виртуозно может решить любую поставленную перед ним задачу. Но неправильно думать, что за счет универсализации теряется спецификация, выражаемая в создании качественного и используемого кода программного продукта. Универсализация приветствуется и всячески "взращивается" в членах команды, но разрабатывать *инкремент* программного продукта - это основная задача разработчиков Scrum-команды.

## 5.4 Scrum-мастер

Scrum-мастер - движущая сила Scrum-процесса. Он призван помогать команде в освоении гибкой методологии. Его роль является определяющей при внедрении Scrum. Для того чтобы Scrum получился успешным, Scrum-мастер должен обладать следующими качествами:

* быть ответственным;
* отдавать должное команде, а не личным достижениям;
* всегда быть готовым к диалогу;
* выкладываться по максимуму;
* обладать умением влиять на других;
* обладать развитым профессиональным кругозором;
* разбираться в профессиональных и технических вопросах.

Scrum-мастер призван помогать команде в использовании Scrum. Его функции сопоставимы с функциями тренера, который помогает спортсменам соблюдать спортивный режим и поддерживать физическую форму. Scrum-мастер создает мотивации и в то же время следит за тем, чтобы члены команды не увиливали от выполнения "тяжелых физических упражнений". Однако его полномочия ограничены. Он не может заставить сотрудника выполнять то или иное упражнение, которое тот не желает выполнять. Вместе с тем он напоминает о целях и о том, как необходимо их достигать. Scrum-мастер обладает властью в той мере, в какой этой властью наделила его *команда*.

Scrum-мастер может сказать команде: "Итак, к окончанию каждого спринта мы должны создавать *программный продукт*, потенциально готовый к установке. На этот раз нам не удастся создать такой продукт. Что мы можем сделать для того, чтобы подобная ситуация не повторилась в следующем спринте?". Это пример проявления власти Scrum-мастера над процессом. Если команде не удалось к окончанию текущего спринта изготовить *программный продукт*, потенциально готовый к установке, значит, произошел какой-то сбой в работе команды. Но поскольку власть Scrum-мастера распространяется только на процесс, то он не имеет *права* сказать: "Поскольку нам не удалось к окончанию текущего спринта изготовить *программный продукт*, потенциально готовый к отправке заказчику, я хочу, чтобы он просмотрел весь код до того, как он будет окончательно принят". Возможно, было бы не так уж плохо, если бы кто-то просмотрел весь код, но Scrum-мастер не имеет *права* принять такое решение, поскольку оно выходит за пределы его полномочий и вторгается в сферу ответственности команды. Scrum-мастер должен лишь контролировать соблюдение процесса командой, его роль может оказаться более трудной, чем роль типичного руководителя проекта, так как руководители проектов зачастую могут сказать: "Делайте так, потому что я так решил".

При назначении *SM* не рекомендуется совершать следующих ошибок:

* назначать на эту роль сотрудника без соответствующих знаний и опыта;
* назначать на эту роль другого члена команды;
* делегировать ему возможность принимать решения от имени команды или ее членов.

Проводя параллель с функциональным или процессным подходом к организации процессов, можно сказать, что обязанности Scrum-мастера соответствуют роли функционального менеджера/администратора, но не владельца процесса.

## 5.5 Владелец продукта

Роль владельца продукта является основной при использовании методологии Scrum. Ожидания и *определение* этой роли очень размыты. С одной стороны, на нее возлагается масса надежд, но обязанности четко не определены.

Для того чтобы быть успешным, сотрудник, выполняющий эту роль, должен сочетать в себе навыки таких должностей, как *менеджер* продукта, *менеджер* проекта, бизнес-аналитик, и при этом быть природным лидером, способным мотивировать и вдохновить окружающую его команду на свершения.

*Бизнес-аналитик - сотрудник, который знает, как получить требования от пользователей и как подготовить требования к команде, чтобы она исполняла их максимально быстро.*

*Менеджер продукта - сотрудник, который знает, чего хотят бизнес и пользователи.*

*Менеджер проекта - сотрудник, который знает, как планировать и отслеживать исполнение проекта.*

*Лидер - сотрудник, который знает, как направить общие усилия в одну сторону, на чем сфокусироваться и как мотивировать команду на достижение этой цели.*

"Готового" для конкретной организации идеального владельца продукта, как ни прискорбно это констатировать, не существует. Владельца продукта необходимо воспитать в соответствии с целями, задачами и условиями, в которых функционирует организация, более того, успех создаваемого продукта зависит именно от того, кто исполняет эту роль и какова его позиция в организации. Этот сотрудник должен занимать достаточно важное *место* в организационной структуре компании. Это поможет отстоять обоснованную точку зрения в дискуссиях руководства и донести до них те выгоды, которые сулит развитие программного продукта.

Владелец продукта (Product *Owner*) - сотрудник, который указывает Scrum-команде ее цель и не позволяет отклоняться от нее. Указание цели происходит за счет:

* четкого определения выполняемых задач;
* установки приоритетов выполнения задач;
* повышения ценности работы, выполняемой командой;
* обеспечения видимости, прозрачности, понятности выполняемых задач, формирования понятных и непротиворечивых требований к ним.

На нем лежит ответственность за то, как члены команды поймут задачи, необходимые для реализации. Для эффективного выполнения роли PO необходимо иметь следующие качества:

* быть доступным для вопросов от членов команды;
* досконально знать профильное направление бизнеса;
* быть коммуникабельным;
* быть решительным;
* иметь определенные властные полномочия в организации.

Подытоживая разговор о владельце продукта, стоит отметить, что эта роль - новая для современной бизнес-среды. Предстоит осознать множество критичных факторов, как технического, так и социального порядка, относящихся к разработке эффективных программных продуктов, для того чтобы требования к этой роли выкристаллизовались.

## 5.6 Самоорганизация членов команды

Самый серьезный вызов для внедрения гибкого подхода - уметь перестроиться с привычных "рельсов" течения рабочих процессов. Каждый член Scrum-команды должен осознавать, что результат (*инкремент*) и эффективность команды будут напрямую зависеть от результатов его деятельности, синхронизированных с результатами деятельности остальных членов команды. Scrum поощряет развитие универсализма каждого из членов команды, но при этом должна сохраняться специализация. Для того чтобы продукт был качественным, члены команды должны быть (или стремиться к тому, чтобы стать) экспертами в профильных областях деятельности. *Самоорганизация* и высокий профессионализм необходимы для того, чтобы:

* команда и продукт не зависели от конкретного специалиста, и в случае необходимости его можно было заменить;
* набор задач, определяемых конкретной бизнес-"болью", Scrum-команда могла решить вне зависимости от имеющихся в наличии ресурсов;
* команда смогла в короткие сроки принять новичка и подтянуть его до общего уровня команды, или подтянуться за ним сама, в зависимости от его уровня владения необходимыми навыками и квалификацией.

После того как задания на ближайший спринт определены, *команда* должна самоорганизоваться для его выполнения. То, как именно будет достигнут результат, целиком зависит от членов команды и результатов их работы. *Самоорганизация* возможна при соблюдении определенных условий. Важно, чтобы среди членов команды были сотрудники, разные не только *по* профессиональной специализации, но и *по* своим командным ролям. К примеру, можно привести следующие роли, которые скорее соответствуют личностным качествам сотрудников:

* критик;
* координатор;
* генератор идей;
* исполнитель;
* прочие.

Каждая подобная роль занимает свою "зону" в команде. Чем больше свободных зон, тем ниже уровень самоорганизации команды, и для ее развития необходимо внедрение в команды сотрудников, соответствующих подобным ролям. Зрелость и развитость команды можно оценить, только понаблюдав за тем, как совершенно разные люди уживаются между собой и достигают консенсуса в коллективном обсуждении критичных вопросов и задач. Следующим важным этапом в процессе самоорганизации является проведение ретроспектив.

*Ретроспектива - собрание, которое проводит проектная команда в конце каждой итерации, чтобы обсудить, чему она научилась как команда, и построить планы на следующие итерации, основываясь на извлеченных уроках.*

*По* окончании спринта *команда* собирается и совместно обсуждает, что прошло хорошо, а что не так, как планировали или хотелось бы. Ретроспективы дают возможность проанализировать и при необходимости подправить правила, *по* которым *команда* работала раньше. Тему ретроспективы мы более подробно обсудим в следующих главах.

Только после того, как будет достигнута атмосфера общего доверия, сотрудники в команде захотят слушать и понимать друг друга. *Самоорганизация* отдельных членов команды необходима для достижения эффективности коллективного труда, как совокупность достигнутых результатов каждого.

## 5.7 Прочие члены команды

Кроме обозначенных основных ролей, в зависимости от типа разрабатываемого продукта или условий, в которых функционирует организация, для команды могут потребоваться сотрудники со следующими профильными навыками и квалификацией:

* *Аналитики.* Разделяют роль владельца продукта. Нужны для крупных продуктов и рассредоточенных команд. Действуют на стратегическую и тактическую перспективу. Их акцент смещается с документирования в сторону вербального общения.
* *Руководители проектов.* Разделяют роль Scrum-мастера. Нужны для организации работ и ресурсного обеспечения команды.
* *Архитекторы.* Разделяют роль технического лидера владельца продукта. Определяют качество технической работы команды. Задают тренд развития команды и создаваемого продукта.
* *Функциональные менеджеры.* Роль не изменяется, но часть обязанностей, связанных с участием в разработке программного продукта, переходит к команде. Должны трансформироваться во владельцев процессов.
* *Программисты.* Трансформируются в разработчиков.
* *Администраторы базы данных.* Как правило, их не переводят в Scrum-команды из-за очень узкой, но необходимой для поддержки и развития продукта профессиональной специализации.
* *Тестировщики.* Трансформируются в разработчиков/аналитиков. Им необходимо учиться действовать не в режиме выявления, а в режиме упреждения проблемы.
* *Проектировщики пользовательского интерфейса.* Их роль аналогична роли аналитиков, но с акцентом на продукты, в которых требуется usability.

При внедрении в команду новых ролей важно помнить, чтобы все сотрудники разделяли общие ценности и правила инкрементности и итеративности *работ*, а также связанные с ними принципы работы в Scrum-команде. Смысл Scrum состоит в том, что член команды должен выйти за пределы своей специализации.

Если того требуют общие интересы, люди должны быть готовы работать, не ограничиваясь лишь своей узкой специализацией.

## 5.8 Самоорганизующийся коллектив

После того как будет достигнута *самоорганизация* всех членов команды, следующей ступенькой на пути совершенствования командной работы и, как следствие, самого процесса Scrum, является достижение состояния самоорганизующегося коллектива.

Основной вопрос, который необходимо решить для эффективного внедрения в управленческую практику компании, заключается в том, что может дать *самоорганизация* компании, ведь то, что все организовались и заняты, может в итоге ни к чему не привести. Если цель самоорганизации команды должна последовательно привести к успеху продукта, то помимо перечисленных условий важна мотивация - сотрудники должны быть мотивированы самоорганизовываться.

Культура компании, в которой разрабатывается *программный продукт*, неизбежно влияет на самоорганизацию: может способствовать ее распространению, а может и нет. Если инициатива в организации хотя бы не наказуема, то *самоорганизация* команды возможна.

Директивное вмешательство определяется типом управления организацией.

Если директивное вмешательство состоит в задании цели и задач реализации информационной системы, то важно, чтобы это делалось непротиворечиво и последовательно. Вмешательство, направленное на прояснение ожиданий, доступных ресурсов и того, что можно и чего нельзя делать, поощряет и способствует формированию самоорганизованности Scrum-команд.

Если директивное вмешательство определяет операционное управление, то распространение в такой управленческой культуре высоких самоорганизованных Scrum-команд достаточно затруднительно.

Чтобы организация смогла встать на рельсы самоорганизованности, помимо необходимости наличия определенной управленческой культуры важно, чтобы гибкие методологии разработки и принцип командной работы опоясывали и проходили "красной нитью" через всю организацию. Подобная организационная структура в последнее время начала внедряться в *деятельность* немногих компаний *по* всему миру. Но это только начало. Эта "мета"-методология называется холократия.

*Холократия - социальная технология, система управления, в которой полномочия и ответственность за принятие решений распределяются по самоорганизующимся единицам вместо управленческой иерархии.*

Холократия *по* своей сути - это способ децентрализации управления, который позволяет выстроить процессы таким образом, чтобы сотрудник мог влиять на жизнь компании и обладал полной властью в рамках своей роли. Эта технология управляется не менеджментом, а общей целью компании, процессом, ожиданиями и метриками, определенными управленческой командой, состоящей из выделенных представителей отдельных команд. В Scrum принципы холократии на сегодняшний день наибольшее распространение получили при организации деятельности отдельных команд и при организации взаимодействия между несколькими командами.

В теории это выглядит перспективно, но вопросов пока больше, чем ответов:

* Как осуществить переход к такому принципу управления? (в частности, крупным и иерархическим организациям, в которых любая управленческая ломка может закончиться глубоким кризисом).
* Где взять достаточное количество мотивированных и квалифицированных сотрудников?
* Как осуществлять мотивацию отдельных команд?
* И прочее.

Только самые отчаянные инноваторы могут решиться на внедрения подобного типа управленческой структуры, но есть успешные примеры и в России, и не только в сфере информационных технологий.

Но, несмотря на блестящую идею о самоорганизованности, важно понимать, что такое *самоорганизующаяся* *команда*, а также что не входит в ее полномочия.

* *Команда не может сама себе поставить цель.* Ею всегда должны руководить сверху. Как правило, задачи спускаются от руководства, и в дальнейшем контролируется ход их исполнения.
* *Команда не определяет состав сама.* Она может повлиять на свой состав, но основная задача по ее формированию и обеспечению ресурсами лежит на руководстве.

На определенном этапе, когда *команда* может объективно оценивать вклад участников в формирование конечного продукта и процесс Scrum, ей можно доверить самостоятельное формирование состава команды в рамках заданных ресурсов.

*Команда* должна определять максимально эффективный способ, с помощью которого будет достигать результата. Эффективность в подобных случаях измеряется сроком, составом и ценой *работ*. При этом *команда* должна быть способной адаптироваться к ограничениям и условиям окружающей среды.

Самоорганизованность и способность адаптироваться под изменчивые условия свойственны многим природным колониям живых существ. Возьмите, к примеру, пчел или муравьев. В том случае, когда человек в своих стремлениях создать что-то эффективное наиболее точно копирует природные аналоги, получаются оптимальные решения.

В решениях, из которых складывается конечный результат, принимает участие каждый член команды, разделяя ответственность и вознаграждения. Самоорганизованность команды - это действенный этап в работе *по* Scrum. Для его достижения и последующего удержания требуется управленческое участие, неподдельный интерес к производимым результатам и желание к постоянному развитию.

## 5.9 Краткие выводы

Scrum - действенная методология, в основе которой лежат принципы командной работы, состоящей из сотрудников, каждый из которых должен выполнять определенную основную роль:

* владелец продукта;
* Scrum-мастер;
* члены команды разработки.

Эти сотрудники работают совместно, чтобы создать рабочий *инкремент* программного продукта. Разработка проходит относительно короткими интервалами, которые принято называть "спринт". *Инкремент* должен удовлетворять заданным критериям и требованиям, обозначенным владельцем продукта, перед началом каждого спринта. Сотрудники, выполняющие различные роли, скомпонованы в единую команду. В зависимости от личных и профессиональных качеств каждого из сотрудников в развитие Scrum привносятся определенные качества. От того, насколько они зрелые, зависит совокупная профессиональная зрелость команды в целом и производимого ими продукта.

Сотрудники, играющие роли, которые являются основными для функционирования Scrum, *по* мере работы проходят становление и развитие как *команда*. Каждый из этапов этого становления характеризуется определенными характеристиками. Для того чтобы на каждом этапе *деятельность* команды удовлетворяла владельца продукта, Scrum-мастер должен отслеживать состояние этих характеристик и при необходимости корректировать ход гибкого процесса.

Наивысшая степень развития сотрудника - состояние самоорганизованности. В этом состоянии сотрудник полностью отвечает за производимые им результаты и готов к постоянным изменениям в целях осознания и последующего совершенствования.

Наивысшая степень развития команды - состояние самоорганизованности. Для достижения этого состояния каждый из членов команды должен быть самоорганизованным, *команда* в целом должна разделять общие цели, постоянно отслеживать отклонения от заданного рабочего процесса, искать способы для последующего развития и совершенствования как командного взаимодействия, так и отдельных ее индивидов.

В лекции "Планирование" мы поговорим о наиболее сложном и критичном этапе деятельности в процессах разработки программного обеспечения, который определяет ожидания заинтересованных пользователей от создаваемой информационной системы и закладывает прочный фундамент дальнейшего взаимодействия между заказчиками и разработчиками программного продукта, - этапе планирования *работ*.

В Scrum этот этап достаточно сильно отличается от классических подходов к планированию и имеет свои детали, которыми также нужно эффективно управлять для получения оптимального рабочего продукта.

**Питання до теми:**

# Лекция 6: ПЛАНИРОВАНИЕ

План

6.1 Принцип быстрого планирования

6.2 Поэтапное уточнение планов

6.3 Техника Planning Poker

6.4 Диаграмма сгорания работ

6.5. Краткие выводы

## 6.1 Принцип быстрого планирования

Планирование, вопреки всеобщему убеждению, - это фундаментальный аспект деятельности Scrum-команд.

Принципы инкрементальной разработки программного обеспечения налагают дополнительные требования на руководство компании, управляющей и контролирующей процесс и результаты от использования разработанного программного продукта. *По* сравнению с классическими методологиями разработки информационных продуктов происходит смещение акцента с поставки достаточно целостного и концептуально завершенного продукта, полностью готового к полноценной эксплуатации, в сторону разработки "полуфабрикатной" системы, закрывающей потребности, необходимые "здесь и сейчас". Эта трансформация избавляет бизнес-пользователей и профильных аналитиков от необходимости продумывать всевозможные требования к функциональности реализуемой информационной системы, но повышает важность тактического управления процессом разработки, его своевременного снабжения необходимыми детализациями ранее обозначенных требований и измерении эффекта от их внедрения.

Таким образом, при работе *по* Scrum появляется необходимость в планировании и реализации только наиболее значимых для бизнеса и владельца продукта функциональностей информационной системы. В ответ на это Scrum-*команда* должна не просто стремиться разработать вовремя рамки обозначенного инкремента, а взять на себя обязательство следовать намеченным планам внедрения. А это значит, что Scrum-*команда* должна стремиться работать над реализацией функциональных возможностей, имеющих наивысшую ценность. *Команда* и владелец продукта должны оценивать *стоимость* разработки той или иной функциональной возможности. В противном случае приоритизация *работ* осуществляется *по* субъективному критерию желательности. Также важно оценить продолжительность разработки той или иной функциональности в желаемом виде. Функциональность, которая не попадает в критичное "рыночное окно", будет обладать гораздо меньшей ценностью. Scrum-*команда*, достигшая состояния самоорганизованности, взявшая на себя обязательство работать в порядке приоритетности, должна соблюдать важность планирования. При этом *команда* получит достаточное *количество информации* для спокойной и продуктивной работы в течение одного или нескольких спринтов, а владелец продукта будет четко понимать, что и как реализует *команда*.

В итоге планирования должны быть известны:

* цель спринта;
* участники команды и степень их занятости;
* набор задач на текущий спринт;
* дата демонстрации полученных результатов.

Для Scrum подходит вид планирования, при котором работа, которую надо будет выполнить в ближайшей перспективе, подробно планируется с глубоким раскрытием структуры необходимых *работ*. Далеко отстоящая работа планируется с относительно неглубоким раскрытием иерархии *работ*. *По* мере выполнения глубоко раскрытой и понятной работы производится более подробное планирование последующих задач с возможной корректировкой общих целей.

Но тут появляется проблема: не всегда есть возможность прогнозировать даже ближайшее будущее, даже в процессах и командах с постоянным прогрессом. Поэтому планирование выполняется "волнами" или этапами, где действия ближнего этапа детально планируются, а действия в далеком будущем отложены. Необходимо выполнять столько волн планирования, сколько итераций работы необходимо провести для достижения конечного и удовлетворяющего потребностям результата. В частности, если четкий подход или ресурсы зависят от ближайших событий.

Такой подход к планированию называется планированием методом "набегающей волны".

Он наиболее полно соответствует принципам Scrum и поддерживает процесс поэтапной разработки программного продукта, в ходе которого удается эффективно управлять ожиданиями максимального количества заинтересованных сторон.

## 6.2 Поэтапное уточнение планов

После того как работа на ближайшую итерацию спланирована и зафиксированы определенные желаемые результаты, следует задуматься о метриках мониторинга за отслеживанием выполнения хода процесса. Но что более важно, продумать *деятельность* над созданием систематического этапа в процессе планирования, на основе которого будет возможно уточнять намеченные планы и синхронизировать ожидания между владельцем процесса и реализуемой на практике функциональности.

Внедрение этой деятельности в этап планирования позволит:

* минимизировать затраты времени;
* принимать решения в нужное для продукта время;
* уточнять команде направление необходимых действий;
* не оказаться в плену у намеченного первоначально плана.

Эта *деятельность* прекрасно сочетается с техникой планирования методом набегающей волны и в литературе называется поэтапным уточнением ранее намеченных планов. Суть метода состоит в том, что при итерационной работе возникает множество непредвиденных факторов различной природы возникновения, которыми необходимо управлять и обрабатывать для достижения намеченных результатов. Для этого команде необходимо "нащупать" общий ритм и следовать ему. Следование общему ритму работы поможет гарантировать устойчивую скорость работы, в которую закладываются временные интервалы на устранение внезапно возникающих проблем операционного характера, связанных с устранением неопределенностей. Но для того чтобы найти свой ритм, *команда* должна соблюдать следующие принципы:

* избегание постоянной сверхурочной работы;
* накопление статистических параметров по выполнению итераций (время выполнения, количество возникающих проблем в работе над определенным типом задач, природа проблем (организационная, техническая и т. д.) и пр.);
* извлечение системных выводов из полученных параметров и их дальнейшее применение в процессе.

После того как *команда* выйдет на устойчивый ритм работы, необходимо зафиксировать условия внутренней и внешней среды, касающиеся производительности команды. Подобная фиксация на достаточно длительном интервале времени позволит:

* оставить дополнительный потенциал команды на форс-мажоры;
* высвободить часть ресурсов команды для творческого развития продукта;
* экспериментировать с производительностью и составом команды;
* постепенно выйти на новый уровень производительности с последующим повторением цикла стабилизации процесса.

Устойчивый ритм командной работы позволит заложить прочный фундамент производительности не только для текущей операционной деятельности Scrum, а поэтапно заниматься ее совершенствованием. Но для того чтобы перейти к систематической работе с этапом совершенствования, необходимо, чтобы методика поэтапного уточнения планов заняла прочное *место* в деятельности отдельно взятой Scrum-команды.

Кроме всего прочего, поэтапное уточнение планов способствует выработке задела для прогнозирования показателей разработки. Детальный *анализ* результатов процесса поэтапного уточнения планов делает разработку более прозрачной и понятной для постороннего взгляда.

Это особенно критично, когда речь идет о работе над сложным функционалом. Обычно задачи, выгоду от которых можно посчитать только в случае, если реализован ряд смежных доработок, не очевидны *по* своей сложности и трудоемкости для ключевых бизнес-пользователей. Процесс поэтапного уточнения планов позволяет показать эту сложность за счет многоитеративности работы над задачей и впоследствии более адекватно оценивать суммарную оценку задач.

Прогнозирование производительности труда на предстоящий период производится на основе:

* оцененного объема работ для реализации требуемого функционала;
* сравнения результатов работ по статистике подобных разработок, проведенных с использованием наиболее эффективных инженерных техник и методик;
* экспертной оценки реализации данной наиболее авторитетными членами команды разработки.

После того как была определена важность и необходимость внедрения поэтапного уточнения планов в стадию планирования Scrum, имеет смысл поговорить о наиболее эффективных методиках оценки *работ*, применяемых в гибких методологиях разработки программного обеспечения. Одной из таких является техника *Planning* Poker.

## 6.3 Техника Planning Poker

Если сказать наиболее просто, то *Planning* Poker (PP) - это коллективное обсуждение задачи или проблемы с целью выработки единой оценки *по* ее решению. Это вид техники оценки, который основан на достижении договоренности между всеми участниками команды, участвующими в процессе планирования. Он используется для оценки сложности предстоящей работы.

Методика PP имеет явные преимущества в сравнении со стандартными обсуждениями проблем, применяемыми в классических подходах к разработке программного обеспечения, которые сводились к следующим активностям:

* Инициатор обсуждения (руководитель / аналитик / team lead) собирал команду, состоящую из экспертов или людей, знакомых с объектом обсуждения.
* Проблема озвучивалась, начиналось активное обсуждение.
* Искалось разовое решение, которое должно было устраивать всех причастных к этой проблеме на протяжении всего периода разработки/эксплуатации программного продукта.

Подобный подход жизнеспособен только в рамках классических подходов к разработке программного обеспечения, когда ответственность за принятие определенного технического или бизнес-решения сводилась к указанию конкретного компетентного сотрудника, который мог быть не в курсе всей необходимой информации или возможных изменений. Отметим, что он характеризуется следующими недостатками:

* Не все люди готовы публично высказывать свое мнение относительно обсуждаемого вопроса.
* Высказывания и мнения первых участников обсуждения прямым или косвенным образом могут влиять на мнения остальных.
* Попытка в подобном обсуждении найти универсальное средство решения проблемы, как правило, заканчивается последующими переработками, как только условия, на которых основано решение, изменяются.

Для нивелирования указанных недостатков была разработана обозначенная методика PP. Суть подхода состоит в том, что оценка задач выполняется не в виде часов или альтернативных шкал, представляющих трудозатраты, а в виде сравнительной оценки, которая показывает относительный "размер" требований. Можно встретить Scrum-команды, которые оценивают работы в виде "пунктов", "попугаев", "маек", "грибов" и т. д. Важны относительные значения, и не может быть абсолютного эталона. Единицы измерения не имеют физического эквивалента.

В процессе проведения PP вся *команда* должна прийти к единому пониманию размеров, чтобы *представление* о том, сколько сил затратить на реализацию задачи, у всех было одинаковым.

Вот тут и придет на помощь *статистика* результатов, которые *команда* достигала за последние несколько спринтов. Каждую выполненную задачу нужно идентифицировать, с тем чтобы каждый член команды понял, о чем идет речь. После этого необходимо выполнять сравнение запросов между собой для оценки их актуального размера. Затем задачи выкладываются в ряд *по* возрастанию достигнутых оценок *по* каждой задаче, и напротив каждой кладется соответствующая карта с оценкой. Теперь нужно сгруппировать требования вокруг чисел. Описанное мероприятие является действенным способом достижения общего понимания трудоемкости выполняемых задач. Его можно проводить периодически, когда *команда* "буксует" с выполнением оценки задач на PP.

*Planning* Poker обладает значимыми преимуществами *по* сравнению с аналогичными техниками планирования. К ним стоит отнести следующие:

* В планировании участвует вся команда.
* У каждого члена команды есть возможность высказаться, не испытав влияния более авторитетных коллег.
* Все члены команды берут на себя ответственность за сроки.
* Оценки, полученные PP, более точные в сравнении с оценками, полученными с помощью альтернативных методов оценок.

Правила, которым подчинена техника *Planning* Poker, достаточно просто запоминаются и легки в применении:

* Scrum-мастер не участвует, а только ведет общее собрание.
* Владелец продукта представляет краткие обзоры каждой из задач.
* Члены команды задают вопросы и ведут дискуссию о предложениях и рисках.
* Итог обсуждения записывается Scrum-мастером.



Рисунок 6.1 **–**Атрибуты техники Planning Poker

* Каждый раз разыгрывается определенная задача.
* Участники выбирают карту.
* Кладут ее рубашкой вверх.
* Числовые достоинства - оценка трудозатрат на задачу.
* Участникам с высокими и низкими оценками дается возможность высказаться и обосновать свою оценку.
* Процесс обсуждения продолжается до тех пор, пока не будет достигнут консенсус.

Критичные для проведения процесса замечания:

* Не рекомендуется использовать числа больше 13 или 20.
* Не верьте в то, что бывает работа, не требующая никаких усилий.

Если речь идет о команде, которая раньше не работала вместе, или разрабатывается новый продукт, нет возможности использовать статистику, накопленную за определенный период. Размер затрат должен оценить наиболее авторитетный *участник команды*, обладающий схожим опытом работы и выложить в виде линейной последовательности.

Это будет наиболее эффективным способом оценки задач и старта активности PP, даже если потом окажется, что в начальном расчете были допущены незначительные ошибки.

Несмотря на то что практика *Planning* Poker все чаще входит в жизнь большинства современных Scrum-команд, этого оказывается недостаточно, чтобы команды научились делать оценки легко и быстро. Иногда, если людям дать только колоду карт со стандартными инструкциями, этого оказывается мало, и со временем *команда* перестает использовать этот механизм оценки. Роль и *значимость* Scrum-мастера для этой конкретного мероприятия, так же как и для большинства мероприятий, проводимых в Scrum, сложно переоценить.

## 6.4 Диаграмма сгорания работ

Один из самых важных артефактов, которые использует Scrum-*команда* для контроля за запланированными показателями выполнения *работ* в течение спринта, - *диаграмма* сгорания *работ* (Burndown *Chart*) ( рис. 6.2).



Рисунок 6.2 – Диаграмма сгорания работ

Эта *диаграмма* показывает, сколько задач осталось до завершения запланированного периода времени на выполнение *работ* (спринта):

* по вертикали - количество задач;
* по горизонтали - время;
* идеальная диаграмма сгорания, которая отображает запланированный ход работ.

Цель каждой Scrum-команды - "сжечь" все взятые в спринт задачи до того, как приблизится конец намеченного срока. Если фактическая кривая отличается от идеальной, то *по* ситуации необходимо корректировать действия команды.

Оперативный *анализ* процесса сгорания задач должен проводиться *по* количеству оставшихся "пунктов" путем сравнения реального *графика* с идеальным:

* Если реальный график выше идеального - значит, команда отстает от плана.
* Если реальный график ниже идеального - команда опережает план.

*Анализ* результатов хода выполнения процесса целесообразно проводить в одно и то же время. Лучше всего распределить эти активности равномерно *по* времени выполнения спринта. Именно это позволит оперативно отслеживать динамику выполнения задач. Лучше всего, если на ежедневной основе автоматический сбор информации *по* процессу позволит строить метрики эффективности и принимать необходимые командные решения.

Если намечается отклонение от плана в сторону отставания от *графика*, то *команда* должна на очередном собрании обсудить складывающуюся ситуацию, принять решение и выработать *путь* ее оперативного исправления. В *список* самых распространенных причин, которые приводят к отставанию, входят:

* ошибка в планировании и последующей оценке задачи;
* болезнь или иная причина отсутствия одного или нескольких членов команды;
* недооценивание и реализация рисков различного характера.

Об отставании необходимо максимально оперативно сообщить владельцу продукта. Если *команда* сама не сможет выработать пути исправления ситуации, нужно, чтобы владелец продукта уменьшил объем спринта за счет задач пользователя с минимальным приоритетом.

Если же речь идет об опережении *графика* выполняемых *работ*, команде следует внести в журнал пожеланий спринта одну или несколько дополнительных задач с высшим приоритетом, не вошедших в спринт ранее.

Отметим еще несколько важных пунктов, которые необходимо учитывать при работе с диаграммой сгорания *работ*.

* Диаграмма должна обновляться каждый день, по факту сделанной работы. В простой форме должна постоянно демонстрироваться динамика выполнения работ в спринте.
* График должен быть общедоступен. Любой член команды или заинтересованное лицо должны иметь к нему доступ.
* Любой член команды может сделать предложение по поводу оптимизации выполнения задач. Любой член команды может сделать предложение, которое не должно быть отвергнуто "просто так". Любое предложение воспринимается командой и обсуждается при первой возможности. Только команда в целом может принять конечное решение по каждому конкретному предложению.

## 6.5. Краткие выводы

В данной главе был подробно рассмотрен наиболее значимый и критичный для управления ожиданиями пользователей этап планирования процесса разработки программного обеспечения. Мы уделили внимание методам и профильным техникам планирования для работы с гибкими процессами. В частности, рассмотрели планирование методом "набегающей волны" и последующее поэтапное уточнение планов. Применение этих методик в Scrum позволит выстроить работу, начиная с "правильной" работы с менеджментом, на предмет однозначного понимания объема необходимых ресурсов для достижения поставленных результатов и заканчивая техниками оценки трудоемкости задач, к примеру, техникой *Planning* Poker.

Также мы поговорили о таком артефакте, как *диаграмма* сгорания *работ*, которая *по* своей сути является контрольной панелью для отслеживания результатов командной работы, выполняемой в рамках спринта. Все эти атрибуты направлены на повышение прозрачности выполняемых *работ* в Scrum и являются простыми в использовании.

В лекции "Этапы и мероприятия Scrum" мы более подробно поговорим о контрольных мероприятиях, используемых в Scrum для управления процессом разработки программного обеспечения.

**Питання до теми:**

# Лекция 7: ЭТАПЫ И МЕРОПРИЯТИЯ SCRUM

План

7.1. Sprint

7.2 Ежедневные встречи (daily)

7.3 Груминг бизнес-задач

7.4. Груминг технических задач

7.5 Обзор Спринта

7.6 Ретроспектива

7.7 Краткие выводы

## 7.1. Sprint

В Scrum *итерация* работы команды называется "спринт" (sprint). Ее длительность определяется конкретными условиями и требованиями к процессам разработки программного обеспечения и создаваемому продукту. Оптимальной длительностью спринта считается *интервал* 2 недели, максимально возможной - 6 недель. Не рекомендуется делать его более полутора месяцев, иначе могут возникнуть проблемы с создаваемым инкрементом.



Рисунок 7.1 – Диаграмма спринта

Результатом спринта является *инкремент* готового продукта (*build*), который можно передать заказчику для установки на продуктивный информационный ландшафт. Именно в реализации готового инкремента в короткие сроки и с минимальными трудозатратами, который заказчик может использовать и получать от этого определенную бизнес-ценность, заключается вся прелесть и уникальность Scrum в сравнении с альтернативными видами разработки программного обеспечения.

За счет коротких спринтов обеспечивается оперативный ответ Scrum-команде от заказчика. При этом заказчик получает возможность гибко управлять объемами *работ* и направлять разработку в нужном для себя направлении, оперативно оценивая результат спринта и предлагая улучшения создаваемого продукта.

Улучшения должны фиксироваться и попадать в бэклог продукта. После этого они оцениваются в соответствии с существующими бизнес-приоритетами во всем объеме существующих задач. В следующий спринт попадают задачи, которые имеют самый высокий приоритет.

Каждый спринт представляет собой маленький "водопад". На протяжении спринта делаются все работы по сбору требований, разработке и тестированию инкремента продукта. Объем *работ*, включенных в спринт (бэклог спринта), должен быть фиксированным. Это позволяет команде давать обязательства на тот объем *работ*, который должен быть сделан в спринте. Это означает, что бэклог спринта не может быть изменен никем, кроме команды.

Спринт имеет свой *жизненный цикл*, который, как было сказано выше, похож на классический подход к разработке программного обеспечения в миниатюре:

1. Сначала выполняется комплексное планирование спринта в два этапа.

В начале каждого спринта проводится его планирование. В этом мероприятии участвуют владелец продукта, ключевые пользователи, Scrum-мастер и команда разработки.

Планирование спринта состоит из двух последовательных митингов:

|  |  |
| --- | --- |
| а) | *Первый митинг. Определение целей спринта.* Участники: команда, владелец продукта, Scrum-мастер. Цель этой встречи - определить цель спринта и наполнить его бэклог. |
| б) | *Второй митинг.* Определение способа реализации задач, взятых в бэклог спринта. Участники: команда, Scrum-мастер. Цель этого митинга - определить, как именно будет разрабатываться функциональность для того, чтобы достичь цели спринта и синхронизировать понимание всех разработчиков о том, что именно будет сделано. Именно на этой встрече применяется техника Planning Poker для оценки трудоемкости и продолжительности задач. Если в ходе реализации задач выясняется, что члены команды могут не успеть сделать запланированное, отслеживаемое по диаграмме сгорания задач, то Scrum-мастер, владелец продукта и команда встречаются и выясняют, как можно сократить работы и при этом достичь поставленной цели спринта. |

1. Разработка инкремента спринта.

Активные участники: команда. Пассивные участники: Scrum-мастер, владелец продукта. Цель - разработка продукта силами команды. Это основной процесс, в рамках которого команда занимает активную позицию, от которой будет зависеть результат их деятельности. Scrum-мастер и владелец продукта готовы помогать и отвечать на вопросы при необходимости.

1. Остановка спринта.

Остановка спринта производится только в самых исключительных ситуациях. Спринт может быть остановлен до того, как закончится отведенное на него время. Остановку должна выполнять команда, если обоснованно понимает, что не может достичь цели спринта в отведенное на спринт время. Спринт может остановить и владелец продукта, если необходимость в достижении поставленных целей исчезла.

После того как произошла остановка спринта, должен проводиться командный митинг, на котором обсуждаются причины остановки. После этого жизненный цикл спринта должен повториться начиная с планирования.

В период проведения спринта каждый день необходимо проводить короткую встречу, называемую "дэйли", для того чтобы все члены команды понимали, кто и чем занимается в данное конкретное время. Продолжительность дэйли - *максимум* 15 минут. Его проведение предназначено для решения явных и потенциальных проблем. Ведущим этого митинга является Scrum-мастер, который по кругу задает каждому члену команды три вопроса:

* Что сделано c момента предыдущего дэйли?
* Что планируется делать сегодня?
* С какими проблемами вы столкнулись?

Все остальные вопросы и обсуждения должны быть вынесены за пределы митинга и обсуждаться между заинтересованными в его решении лицами.

Если *команда* достигла достаточного уровня самоорганизации, то участие в этом митинге Scrum-мастера не требуется.

После того как спринт достиг своего завершения, необходимо проводить демонстрацию реализованного функционала и последующий *анализ* спринта. *Активность* демонстрации называется "Обзор Спринта" (Sprint *review*), а *анализ* спринта - "ревью" или "Ретроспектива Спринта" (Sprint Retrospective).

В назначенное время на Обзор Спринта приходят *команда*, владелец продукта, ключевые пользователи. В процессе этой активности члены *команда* рассказывают о поставленных задачах, о том, как они были решены, какие препятствия возникали у них на пути, какие были приняты решения, какие проблемы остались нерешенными, и демонстрируют реализованный функционал, а владелец продукта и пользователи оценивают достигнутый результат.

На Ретроспективе Спринта *команда* и Scrum-мастер обсуждают ход выполнения процесса, отмечают его достоинства и недостатки, принимают решения о возможной корректировке процесса в целях его улучшения. Scrum-мастер является непосредственным организатором этих мероприятий и отвечает за общую подготовку к нему. *Команда* помогает ему составить адженду и запланировать расписание выступления. Подготовка к митингу не должна занимать у команды много времени (правило - не более двух часов). Для проведения демонстрации запрещено использовать инструменты типа *Power* *Point*. Функционал должен демонстрироваться непосредственно в рабочем программном продукте. Подготовка к митингу также не должна занимать у команды более 2 часов. В следующих главах мы продолжим обсуждение, связанное с рассмотрение отдельных этапов и мероприятий, более подробно. Здесь же необходимо акцентировать внимание на таком важном понятии процессов гибкой разработки программного обеспечения, как цель спринта.

Большинство современных команд сконцентрировано на выполнении запланированных задач спринта, выполнение которых было согласовано с владельцем продукта. Таким образом, возникает потогонная система, основной критерий успеха которой - полностью выполненные и продемонстрированные задачи.

Если какая-то история не сделана - вечный позор разработчикам, которые отвечали за нее, полный провал спринта, катастрофа вселенского масштаба и непрекращающаяся скорбь, инициированные заказчиками *работ*.

В ответ на это появляется экстремистская *реакция* самой Scrum-команды. Есть мнение, что в объем спринта нельзя вносить изменения после того, как спринт начался, а это, соответственно, ограничивает заказчика. Но если убрать рамки спринта по времени и объему задач, то мы "скатываемся" к классической, водопадной модели разработки информационных систем.

И первая, и вторая ситуации - это совсем не "гибко". *Истина*, как это часто бывает в 100% ситуаций, где-то посередине. Для достижения этой середины в Scrum появилось понятие - цель спринта. Это командная установка, определяющая, зачем мы собираемся потратить время и ресурсы спринта, чего хотим достичь с точки зрения инкремента ценности для заказчика. И это не просто набор разрозненных требований. Цель позволяет и предписывает нам быть более гибкими и толерантными, как к небольшому изменению объема спринта, так и к неожиданным причинам, не позволившим нам реализовать то или иное требование в спринте. Успешный спринт - спринт, достигший своей цели. Если мы реализовали ожидания заказчика, пришедшего на демо, то это успех.

Цель спринта - инструмент, дающий нам гибкость в управлении ожиданиями заказчиков и работы команды внутри спринта. Единственное условие - цель спринта не должна меняться в рамках самого спринта.

## 7.2 Ежедневные встречи (daily)

Теперь более подробно поговорим о мероприятии, которое, по мнению большинства экспертов, является самой полезной практикой гибких процессов. Дэйли, как было отмечено выше, - это ежедневные встречи для синхронизации *работ* членов команды в единую согласованную командную работу ( рис. 7.2).



Рисунок 7.2 –Процесс проведения StandUp

*Знание* о том, чем занимаются коллеги по команде, помогает в:

* минимизации затрат времени, связанных с неосведомленностью членов команды о наилучшем пути решения задач, которые уже решались Scrum-командой;
* понимании общей динамики работ по спринту;
* понимании потенциальных проблем, возникающих у членов команды, и путей их решения.

Длительность дэйли строго ограничена и не должна превышать 15 минут. Эта встреча не предназначена для решения проблем. Все требующие специального обсуждения вопросы должны быть вынесены за ее пределы.

Встреча проводится каждый день всегда в одно и то же время, чаще всего это утро, но в распределенных командах это может быть день или вечер.

*Scrum-мастер ведет эту встречу, задавая каждому члену команды по три вопроса:*

* Что сделано c момента предыдущего дэйли?
* Что планируется делать сегодня?
* С какими проблемами вы столкнулись?

Каждый член команды должен отвечать на необходимые вопросы.

Важно, чтобы все, отвечая на вопросы, не вдавались глубоко в детали и, боже упаси, уж точно не пытались тут же их решать.

С одной стороны, это кажется не такой уж и большой проблемой, но по факту оказывается, что отвлечение сотрудников от плана встречи приводит к тому, что через какое-то время многие не видят в ней необходимости. В тот момент, когда кто-то углубляется в повествование о деталях, большинство незаинтересованных начинает скучать, погружаться в серфинг социальных сетей посредством мобильных гаджетов и т. д.

*Цель митинга - сосредоточить внимание всех членов команды на аспектах процесса. Именно это позволяет быть эффективным в Scrum и иметь дополнительную мотивацию на успех. Если перегрузить всех ненужной на сейчас информацией, то интерес к происходящему будет постепенно ослабевать.*

Для того чтобы перебороть неправильное развитие Scrum-процесса и направить дэйли в нужное русло, нужно:

* *Соблюдать установленные правила.* Правила не должны устанавливаться "сверху". Принципы работы и решения о применении правил должна принимать команда, но не Scrum-мастер и не владелец продукта. Визуализировать принятые правила полезно. Актуализировать правила, синхронизируясь с состоянием команды и окружающих условий, также очень важно.
* *Думать о развитии.* Scrum - полезная практика. С течением времени члены команды учатся правильно проводить встречи и быть эффективными в обстоятельствах постоянно меняющихся условий. Количество проблем, если соблюдать правила и заниматься совершенствованием рабочих практик, будет постепенно уменьшаться. Через некоторое время Scrum трансформируется в полностью операционную деятельность. Появится соблазн немного его разнообразить. Дайте волю этим стремлениям. Это поможет команде сплотиться и стать более производительной.

## 7.3 Груминг бизнес-задач

Следующим по важности мероприятием является груминг - продолжительная встреча, на которой Scrum-*команда* вместе с владельцем продукта рассматривает существующий бэклог продукта, а также требования, которые планируется включить в ближайший спринт, добавляет необходимые технические и бизнес-задачи, чтобы зафиксировать направления проектирования или решить необходимые архитектурные вопросы развития продукта и его инкремента. Без регулярного груминга планируемой работы спринт становится громоздким и очень запутанным.

*Груминг - это практика "причесывания" беклога (по-английски "grooming"), одна из тех активностей, без которых не обходятся продуктивные Agile-команды.*

Груминг бизнес-задач - это согласование объема *работ*, которые нужно выполнить для того, чтобы владелец продукта на ближайшем обзоре был удовлетворен ценностью от реализованного командой инкремента.

Эту *активность* по возможности необходимо проводить в очном режиме присутствия всех ее участников. В противном случае велика *вероятность*, что она разрастется во времени.

*Груминг - это не еще один тип митинга в Scrum. Это активность, которая делается на протяжении спринта для подготовки бэклога к следующему спринт-планированию.*

Груминг стоит проводить до или в начале спринта, с тем чтобы детально разобрать подготавливаемые задачи к реализации разработчиками. *Периодичность* его проведения должна быть один раз в спринт. Время, затрачиваемое на груминг, должно составлять 10% от общей продолжительности спринта.

Сначала, для вновь собранной команды, на груминг может уходить немало времени. Это связано с "притиранием" разработчиков и владельца продукта друг к другу, более глубинным пониманием бизнес-смысла разрабатываемого программного обеспечения, осознанием деталей и взаимосвязей влияния автоматизируемых процессов и т. д. Но это нужно делать. Как результат появится прогресс в работе над задачами, члены команды уйдут от "слепого" анализа требований к осознанному совершенствованию системы.

После того как количества "перегрумленных" историй хватит на два-три спринта вперед, стоит сбавить темп и уделять грумингу чуть меньше времени. Два-три спринта - это оптимальный *интервал* для обеспечения полной декомпозиции выполняемых задач. Не больше и не меньше. Если "перегрумить" слишком много задач, то велика *вероятность* того, что к моменту начала реализации наиболее отдаленных задач их решение будет уже не совсем актуально, а возможно, даже устареет и *команда* опять потратит время на то, чтобы погрузиться в актуальные условия задачи и принять наиболее правильное техническое решение для ее реализации. Все разобранные на груминге задачи стоит поместить в самую "верхушку" бэклога продукта и постепенно наполнять из него бэклоги спринтов:

* эти задачи понятны всем членам команды;
* они имеют такой объем, что несколько из них могут быть сделаны за спринт;
* они оценены командой.

Чтобы груминг прижился в операционной деятельности каждой команды, необходимо следовать нескольким простым правилам:

* сделайте груминг частью вашего процесса;
* выработайте понятие готового бэклога спринта - "Definition of Done" (DoD).

Эти характеристики, в соответствии с которыми задачу можно брать в работу на спринт.

* Установите понятие "текущего" и "следующего" инкремента, что позволит владельцу продукта управлять объемом работ, выполняемых командой.
* Встречайтесь "вживую" всей Scrum-командой с владельцем продукта не реже раз одного раза в спринт для груминга и планирования релиза.
* Во время груминга работайте с бэклогом продукта, который был подготовлен и приоритизирован владельцем продукта.

Способ *принятия решений* (DoD) при проведении груминга о возможности взять задачу в работу - это не догма (в Scrum их практически нет). Этот способ выбирает *команда* и может изменить его в зависимости от своей "развитости". Могут быть использованы следующие подходы:

* *Классический подход - консенсус.* Преимущество этого метода состоит в том, что решение принимается всеми членами единогласно (это повышает мотивацию всех членов команды и осознание разделения ее ценностей, но, как правило, происходит очень медленно.
* *Большинство.* Это менее демократичный способ по сравнению с классическим, что может привести к снижению мотивации отдельных членов команды, но он является более быстрым. Если голоса разделились поровну, то можно либо продлить обсуждение, пока один человек не изменит мнение, либо всегда использовать правило выбора меньшей (вызов) или большей (запас) оценки.
* *Оценка будущим исполнителем.* Это метод, который наименее вписывается в дух Scrum. Решение принимает конкретный исполнитель, который будет далее реализовывать этот функционал, и команда остается за бортом "командной" работы. Он наиболее быстрый для принятия решения о работе, но в дальнейшем при злоупотреблении им команда рискует разбиться на индивидуумов.

Для того чтобы повысить профессионализм проведения груминга, целесообразно использовать методы оценки задач, о которых было сказано в лекции "Планирование" , - к примеру, метод *Planning* Poker.

Среди основных ценностей гибкой разработки важную роль играют взаимное *доверие* и *самоорганизация*, к развитию и работе над которыми должны стремиться все члены команды. Если люди обижаются, когда другие не соглашаются с их мнением, это означает, что они ставят свое профессиональное эго выше интересов команды. Так бывает в командах, которые стали таковой вынужденно, по управленческому указанию или иной причине. Члены команды используют основные принципы и совершают процедуры, но никто не думает об их основном смысле. Внедрение ценностей Scrum - основная задача Scrum-мастера. Он постоянно должен объяснять и воспитывать, в первую *очередь* на собственном примере. После того как будут воспитаны несколько лидеров, остальные начнут тянуться за ними.

Груминг бизнес-задач - очень важный этап на пути осознания командой ценности разрабатываемого ими программного продукта, но часто для того, чтобы правильно провести этот тип груминга, необходимо основательно подготовиться с технической точки зрения, то есть разложить по шагам *алгоритм* реализации сложных требований. А для этого существует груминг технических задач.

## 7.4. Груминг технических задач

Теперь мы добрались до рассмотрения специфичного вида активности, чья *значимость*, однако, не меньше предыдущих.

После того как становится понятен набор бизнес-задач и способ их реализации, которые должна выполнить *команда* в рамках текущего и следующих спринтов, целесообразно выполнить декомпозицию наиболее сложных и комплексных из них для того, чтобы все члены команды однозначно представляли себе способ и объем планируемой автоматизации, которая должна быть произведена в рамках реализации.

Если провести параллель с классическим подходом к разработке информационных систем, то эта стадия соответствует стадии технического проектирования. Разница состоит в том, что в водопадной и инкрементальной модели за эту стадию отвечает отдельно выделенный специалист - как правило, технический лидер или системный *архитектор*, а в Agile ответственность за технический груминг ложится на команду в целом.

В Scrum над проектированием работает *группа* разработчиков, каждый из которых отвечает за область взятых на себя задач и за их гармоническую интеграцию с задачами коллег.

С одной стороны, это повышает риски создания концептуально целостного продукта, но, с другой стороны, снижает *вероятность* "остаться вдали" от критически важных изменений, которые происходят постоянно при реализации отдельных задач, выполняемых разными разработчиками из Scrum-команды. Совместное взаимодействие членов коллектива позволяет достичь нужного уровня восприятия информации и во многом закладывает *базис* активности управления знаниями на нужном уровне понимания.

Правила и советы проведения технического груминга соответствует правилам проведения бизнес-груминга. Очень важно, чтобы на техническом груминге присутствовал владелец продукта или кто-то из его представителей. Любое решение, принимаемое в процессе декомпозиции и принятия технического решения, связано с условиями и факторами автоматизируемых бизнес-процессов.

Чем более глубоко происходит *интеграция* продукта, команды, владельца продукта и других бизнес-пользователей, тем глубже становится понимание того, что нет требований, которые не оказывают влияние на качество результата их совместной работы. Не важно, о каком виде требований идет речь: функциональных или нефункциональных, их решения в любом случае сводятся к качеству создаваемого программного продукта.

## 7.5 Обзор Спринта

В самом конце каждого спринта наступает момент, когда все члены команды должны продемонстрировать то, что было ими выполнено. Этот этап называется Обзор Спринта.

Эта встреча проводится в заранее запланированном месте, в заранее запланированное время. Цель обзора - показать заинтересованным лицам все, что *команда* сделала за период спринта. Кажущаяся легкость организации этого мероприятия не должна вводить в заблуждение. Для эффективной демонстрации реализованных задач необходимо потратить немало времени и сил. Отсутствие ясности и прозрачности в организации обзора приводит к тому, что участники этого процесса, кроме членов команды, не понимают статус разработки информационной системы, не понимают ценность реализуемых задач, не понимают, что именно им показывают и для чего это разрабатывается. Чтобы избежать подобных проблем, Scrum-мастер должен с большой ответственностью отнестись к организации этого мероприятия.

Подготовка к обзору не должна занимать у команды более 2 часов. В начале обзора необходимо четко и однозначно озвучить цель прошедшего спринта. Если на обзоре приглашены сотрудники, которые имеют лишь отдаленное *представление* о разрабатываемом продукте, то следует уделить пару минут, чтобы ввести их в курс дела.

Члены команды рассказывают:

* о поставленных задачах;
* о том, как они решались;
* о том, какие проблемы возникали;
* о том, какие были приняты решения для управления проблемами;
* о том, какие проблемы остались нерешенными.

Если в компании есть проблемы с менеджментом встреч, тогда начинать имеет смысл тогда, когда пришли ключевые участники. Не имеет смысла ждать всех. После того как несколько встреч начнутся вовремя с участием лишь одного или двух пользователей, остальные участники поймут, насколько важно приходить вовремя.

Следует соблюдать назначенный план обзора, который необходимо прописать в описании к встрече. Внезапно возникающие хаотичные дискуссии следует пресекать и выносить за рамки встречи так же, как это делается на дэйли.

Обзор - это не только одно из необходимых мероприятий Scrum. Оно несет в себе множество посылов, управление которыми поможет во внедрении и развитии гибких методологий в вашей компании:

* Положительная оценка работы воодушевляет и мотивирует команду.
* Бизнес-пользователи постепенно начнут более подробно понимать, чем занимается Scrum-команда.
* На обзоре заинтересованные стороны обмениваются жизненно важными отзывами о продукте между собой и с командой.
* Обзор проходит в дружеской атмосфере, поэтому разные команды могут свободно общаться между собой и обсуждать насущные вопросы.

На обзоре необходимо соблюдать следующие принципы:

* Обзор должно происходить быстро и быть сфокусировано на идее показа реализованной функциональности. Сконцентрируйтесь на создании не столько красивого, сколько динамичного демо. Это поможет приучить всех участвующих к соблюдению заданного тайминга и не отвлекаться на посторонние вещи.
* Обзор должно быть ориентированным на бизнес-пользователей и подготовленной командой ценностями. Сфокусируйтесь на том, "что мы сделали", а не на том, "как мы это делали". Технические детали очень важны, но смысл демонстрации в том, чтобы показать реализованную ценность. Технические детали команда должна обсуждать в процессе работы.
* Аудитория обзора может сама попробовать разработанный продукт. Возможность "поиграть" с продуктом действует на пользователей вдохновляюще.
* Не нужно показывать кучу ошибок и то, как они были исправлены. О них нужно упомянуть, но демонстрировать их не стоит, потому что это заберет у вас много времени и снизит внимание к более важным задачам.

Если команду заставлять проводить обзор, когда у них ничего толком не работает, им будет некомфортно. *Команда* будет запинаться и спотыкаться, показывая "сырую" функциональность. Это неприятно. Но это действует как горькая пилюля.

В следующий раз *команда* постарается все доделать к сроку! Команду необходимо приучить к мысли, что обзор придется проводить несмотря ни на что. Если обзор выполнено хорошо, то это оказывает положительное влияние.

## 7.6 Ретроспектива

Ретроспектива - мероприятие, направленное на систематическое инспектирование и адаптацию Scrum к условиям функционирования конкретной команды. На ретроспективе должна собираться вся *команда* и обсуждать, каким образом сделать следующий спринт более продуктивным по сравнению с прошедшим.

*Что такое ретроспектива? Это регулярная встреча, на которой команда обсуждает свой рабочий процесс и что-то в нем меняет.*

В основе ретроспективы лежит концепция *цикла* деминга, PDCA (англ. *Plan*-Do-Check-Act). Цель ретроспективы - к ее окончанию получить практический план эффективных изменений, но это не план окончательных изменений в процессе - это эскиз эксперимента на ближайший период. Цикл деминга состоит из следующих этапов:

* Plan - запланируй.
* Do - выполни.
* Check - проверь.
* Act - прими какие-то дальнейшие решения, реши, что дальше делать.

Собственно, сама ретроспектива - это *Plan*.

Целями проведения ретроспективы являются:

* инспекция того, насколько успешно прошел спринт в отношении людей, отношений между ними, процессов и инструментов;
* определение и упорядочение того, что прошло успешно, и того, что нуждается в улучшении;
* разработка плана по внедрению улучшений в Scrum.

Ретроспектива, как каждое Scrum-мероприятие, должна иметь четкую и однозначную цель. Получить план процессного эксперимента - самая подходящая цель для любой команды, однако многие этого не понимают. На ретроспективе команды порой пытаются придумать какие-то глобальные решения своих проблем и сталкиваются с тем, что не получается это сделать разом. В итоге они застревают в промежуточном состоянии полной неудовлетворенности. Если вы столкнулись с такой проблемой, то необходимо понимание того, что двигаться нужно маленькими шагами, пробуя и проверяя разные подходы к решению проблем, впоследствии выбирая лучший из них. В процессе проведения ретроспективы нужно выработать не план окончательных изменений, а план эксперимента на ближайший период, в ходе которого будут проверены принятые решения в совершенствовании Scrum.

*Ретроспектива - это мероприятие, на котором команда решает свои проблемы.*

Решение проблем в Scrum - это не праздный вопрос. Невозможно сделать так, чтобы организационный руководитель или эксперт пришел, посмотрел и сказал, что команде надо делать, а что в рабочем процессе стоит изменить.

Во-первых, приходить к команде с готовыми решениями чревато отторжением предлагаемых решений. Этот феномен принято называть "*not* invented here" (не изобретено здесь). Тут срабатывает природная гордость и упертость человеческой натуры. Даже если разработчики понимают, что предлагаемое решение верное, у них нет уверенности в его работоспособности для их команды, плюс отсутствует "собственническое" *отношение* к нему. Подобные, не "выстраданные" командой решения имеют низкие шансы на реализацию. Члены команды должны набить собственные шишки проблем и определиться с решением, которое будет оптимальным для всех.

Во-вторых, *разработка программного обеспечения* - сложная дисциплина. Более того, в каждой компании, которая занимается данной активностью, процессы разработки разные или, по крайней мере, имеют уникальные для их окружения детали. Вряд ли найдется консультант, который, не зная условий и факторов организации конкретного процесса, сможет расписать, как должна работать Scrum-*команда* при решении определенной задачи в конкретной компании. Только проведя несколько рабочих экспериментов и поняв суть операционных проблем, можно понять, хороша или не очень та или иная практика для данной команды.

*Существует такое понятие, как "best practice". Это тип практик, которые подтвердили свою эффективность на ряде примеров при решении конкретных задач. Несмотря на кажущийся универсализм, каждая подобная практика поможет не в любом случае. Это происходит потому, что рассматриваемую практику можно оценить только в контексте определенной ситуации.*

Ведущий ретроспективу (поначалу эту роль должен выполнять Scrum-мастер) должен привести команду к конкретным рабочим принципам и договоренностям, которые нужно соблюдать и придерживаться их при работе в спринтах.

Идеи о развитии команды и совершенствовании Scrum рождаются в процессе обсуждения недостатков прошедшего спринта. Ведущий должен следить за тем, чтобы ход обсуждения минусов не перерастал в *поиск* конкретных виновников. Для достижения целей Scrum более важно понимать, как не допустить подобную ситуацию в будущем и что с ней надо сделать, чтобы минимизировать возможные потери. Споры по поводу идей, несмотря на устоявшееся мнение об их необходимости, на данном этапе бесполезны. В план будут включены только те идеи, с которыми согласны все участники митинга.

В результате обсуждения достоинств, недостатков и предлагаемых идей *команда* должна приступить к составлению плана, куда должны быть включены не просто итоги обсуждения, а конкретные действия, которые должна выполнить *команда*, или ее отдельные роли:

* написать …;
* принять …;
* Задачу Х выполнять с использованием подхода N.

Однако не стоит пытаться на конкретной встрече решить все проблемы сразу - для эффективной работы в следующем спринте достаточно плана из 3-6 пунктов. Слишком объемный план может в итоге оказаться невыполнимым, демотивирует команду и точно изменится через короткий *интервал* времени. В процессе проведения ретроспективы могут возникнуть проблемы, связанные как с Scrum, так и с его участниками:

* *Команда полагает, что у нее нет проблем, Scrum-процесс хорош, и она не видит смысла в его улучшении.* Ошибка. Но команде этого так просто не объяснить. Чтобы сдвинуть ее с мертвой точки, полезно пригласить на ретроспективу кого-то из менеджеров компании - заказчика или пользователей, которые имеют конкретные претензии к команде или продукту. Пользователи очень редко полностью удовлетворены. Они могут быть удовлетворены до определенной степени, но у них все равно есть какие-то мысли на тему того, что можно сделать лучше. Если такой заказчик приходит на ретроспективу и рассказывает это команде, то она вынуждена обсуждать направления для дальнейшего развития.
* *На ретроспективе говорит в основном один или 2-3 человека.* Ошибка. Людям всегда есть что сказать. Если внимание забирает негласный лидер, то своим доминированием он подавляет остальных членов команды. Если каждый будет высказывать свое мнение, то вероятность найти лучшие решения намного возрастет.

Групповая дискуссия побуждает всех участников высказываться. Это помогает посмотреть на проблему с разных точек зрения и придумать лучшее решение. Часто справиться с этой проблемой помогает ведущий ретроспективы, который будет следить за тем, чтобы каждый из присутствующих обязательно высказал свою точку зрения.

Стоит еще раз отметить, что формат проведения ретроспективы может быть различным. Ретроспективы - это не единичное мероприятие. Они проводятся регулярно, и по результатам каждого такого собрания выполняется основная цель - создается план на ближайшую итерацию. Если отнестись к этой процедуре профессионально, заранее проанализировать наиболее типичные проблемы, возникающие в ходе ретроспективы, можно создать благоприятные условия развития настоящей самоорганизующейся команды.

## 7.7 Краткие выводы

В этой главе мы обсудили понятие спринта, которое является центральным для методологии Scrum.

Спринт - это временной *интервал*, в рамках которого перед командой стоит задача реализовать согласованное количество требований, объединенных общей целью. Спринт, в отличие от обычного рабочего *графика*, подчинен особым принципам и в своем составе имеет ряд мероприятий, каждое из которых направлено на то, чтобы эффективным образом скоординировать процесс работы над разрабатываемыми командой задачами. Состав *работ*, длительность спринта фиксированы. Он повторяется до тех пор, пока есть необходимость в разработке программного обеспечения для конкретной организации.

Кроме того, в спринте есть специфические активности, которые направлены на повышение качества процесса и развитие членов команды как активных участников Scrum.

В итоге необходимо отметить, что Scrum требует не просто формального соблюдения предписанных принципов, но постоянного участия в процессе менеджеров, которые своими решениями могут повлиять на направление развития команды.

В лекции "Атрибуты Scrum" мы приступим к рассмотрению ключевых артефактов, используемых при работе в Scrum.

**Питання до теми:**

# Лекция 8: АТРИБУТЫ SCRUM

План

8.1 Story mapping

8.2. Пользовательские истории (User story)

8.3. Определение приоритетов пользователей

8.4. Доска задач

8.5 Бэклог продукта

8.6 Бэклог спринта

8.7 Инкремент продукта

8.8. Принцип прототипирования

8.9. Краткие выводы

## 8.1 Story mapping

*Story* *mapping* (англ. "карта историй") - техника визуального и физического представления последовательности действий, которые должны быть реализованы в разрабатываемом командой программном продукте.

*Story mapping - инструмент, помогающий в осмыслении функциональности ПО и "правильном" проектировании способов его использования.*

Карта историй - это источник информации, используемый для визуализации требований к продукту в контексте использования его функциональности и их приоритетов. *Story* *mapping* помогает выполнить декомпозицию задачи с ее высокоуровневого представления до уровня конкретных задач, а выполненная *декомпозиция* обеспечивает эволюционное понимание цельного продукта, начиная с полного охвата всех потребностей и завершая погружением до детальных требований пользователей. Но обычно с погружением в детальные требования пользователей возникают самые большие проблемы.

Как правило, чем крупнее компания, тем более разветвленным и кросс-функциональным является автоматизируемый процесс. Соответственно и требование будет более высокоуровневым и комплексным. Для небольшой компании реализация небольшого требования занимает не более недели, а вот в крупных организациях разработка подобного требования займет не менее нескольких месяцев.

Причин, как у каждой проблемной ситуации в области разработки программного обеспечения, много. Из них следует выделить:

* "Неделимость" требований. Миф о том, что только часть высокоуровневых требований можно поделить на несколько соподчиненных задач, родился во многом благодаря низкому доверию к специалистам в области разработки программного обеспечения. Особенно это касается процессов внедрения программных продуктов и "внешних" аутсорсинговых и консалтинговых компаний, которым выгодно "вбивать" этот миф в головы заказчиков и тем самым эксплуатировать это понимание для получения дополнительных контрактов на продолжение сотрудничества. На то, чтобы преодолеть подобный миф, направлена технология инкрементальной разработки. В ее ходе владелец процесса и Scrum-команда добиваются корректного разбиения высокоуровневой идеи на поэтапность выполняемых задач, каждая из которых приносит определенную ценность бизнес-заказчикам.
* Нежелание понимать значимые детали процесса разработки программного обеспечения. Заказчик, особенно если это топ-менеджер компании, как правило, не склонен погружаться в детали автоматизируемого процесса. Это понятно. У топ-менеджеров другие цели и задачи. Поэтому желательно, чтобы владельцем продукта был пользователь, который не только понимает автоматизируемый бизнес, но имеет навыки работы с информационными системами и понимание общих принципов их устройства и функционирования.
* Недоверие к области коммерческой разработки информационных систем. Область коммерческой разработки программного обеспечения - не новая область, и уже многие организации с ней сталкивались не единожды. Учитывая состояние в этой области, описанное в лекции "Введение в Agile" , у многих пользователей накоплен определенный, как правило негативный, опыт взаимодействия с компаниями разработчиками программного обеспечения. Поэтому заказчики вынуждены говорить "сделайте всё и сразу" просто потому, что это единственный способ получить требуемые доработки в сложившихся условиях. Потому что если не попросят максимум и сейчас, то вряд ли получат это потом.

Выходов из подобной ситуации несколько. Основное решение, которое предлагает Scrum, - разрабатывать результат максимально быстро. Для этого и организована "спринтовая" модель поставки функциональности маленькими порциями, в соответствии с наиболее актуальными требованиями. А это возможно только в том случае, когда спроектирован общий каркас функциональности через декомпозицию требований.



Рисунок 8.1 –Представление карты историй

Карта историй представляется в виде двух последовательностей зависящих друг от друга активностей. Таким образом, мы получаем матричный вид карты историй.

Она представляет собой последовательные стадии, выполняющиеся в автоматизируемом процессе/продукте. На самом верхнем уровне располагается наиболее верхнеуровневое *представление* стадий или компонентов разрабатываемого продукта. Далее, *по* мере снижения, каждый *компонент* раскладывается на составляющие, и так до самого низкоуровневого представления автоматизируемого продукта.

Процесс построения карты историй:

1. Сначала выделяют ключевые виды деятельности. Каждый вид фиксируется на отдельной карточке.
2. Они располагаются в функциональном порядке использования слева направо.
3. После этого определяются отдельные задачи, определяющие каждую активность, и также фиксируются на карточках.
4. Все задачи располагаются в логическом порядке.

Карта историй содержит каркас из задач, представляющих собой "кирпичики", составляющие конечный продукт. Этот каркас покрывает крупный набор функций, реализуемых в соответствии с приоритетами пользователей.

*Story mapping - это архитектурное представление задач пользователя.*

Под каркасом находятся подробные требования, которые описывают конкретные функциональные части для выполнения задач. В Scrum, когда речь заходит о работе с требованиями, применяют определенную технику работы с ними, которая называется *User* *Story* (пользовательские истории).

## 8.2. Пользовательские истории (User story)

В практике подготовки требований, которые в дальнейшем будут составлять каркас разрабатываемого программного продукта, в Scrum принято использовать один из стандартизированных подходов для их описания - Use Cases ("пользовательские истории").

*Use Case ("пользовательская история", "юскейс" и т. д.) - это сценарная пошаговая техника описания взаимодействия двух или более участников, задействованных в автоматизации. С помощью Use Case может быть описано и пользовательское требование, и требование к взаимодействию систем, и описание взаимодействия людей и компаний в реальной жизни. В общем случае с помощью Use Case может описываться взаимодействие двух или большего количества участников, имеющее конкретную цель.*

В разработке программного обеспечения этот способ разработки и управления требованиями часто применяют для проектирования и описания взаимодействия пользователя и системы, поэтому название Use *Case* часто воспринимают как *синоним* требования человека-пользователя к решению определенной задачи в системе.

Пользовательские истории формулируются как одно или более предложений на "повседневном" языке пользователя. Они получаются относительно небольшие *по* объему, что удобно как для составления, так и для их обсуждения, приоритизации, планирования, оценки и дальнейшей работы с ними. Пользовательские истории получаются в виде алгоритма действий пользователя с реализуемым программным продуктом. Таким образом, каждый квадратик в *Story* *Mapping* можно представить в виде Use *Case*. Если владелец продукта отнесется к приоритизации пользовательских требований со всей необходимой серьезностью и значимостью, то Scrum-*команда* может сконцентрироваться на наиболее значимых и важных Use *Case*, что будет влиять на инкрементальность создаваемой информационной системы. Адекватная оценка трудоемкости историй позволяет планировать сроки ее реализации, тем самым управляя содержимым спринтов. Разработать оптимальную пользовательскую историю не так просто, нужен определенный навык, который позволит создать качественное описание требований пользователей к реализуемому программному продукту, но в награду за это получаются следующие преимущества:

* *Краткость.* Пользовательская история описывает небольшую часть бизнес-ценности, которую возможно реализовать за период спринта.
* *"Незатратность" создания и сопровождения.* За счет своей "компактности" пользовательские требования достаточно просто создать и сопровождать их изменения на всем протяжении жизненного цикла продукта.
* *Вовлечение ключевых пользователей в процесс создания продукта.* За счет своей доступности бизнес-требования смогут стать реальным "мостиком" между пользователями и Scrum-командой. Это позволит более адекватно управлять ожиданиями пользователей и вовлечь их на нужную степень погружения в процесс разработки продукта.
* *Облегчают оценку заданий.* Формат пользовательских историй способствует более точной оценке необходимых системных разработок/доработок.

Чем более компактный объем имеет пользовательская история, тем проще выполнять ее оценку. Это приводит к более верным оценкам сроков реализации программного продукта и к более понятному планированию выполняемых *работ*. При этом важно знать меру в уменьшении и дроблении Use cases. Если сделать их слишком много, то у вас получится огромный *список* задач в бэклоге продукта, и *процесс управления* ими и фиксации на карте историй усложнится.

Для пользователей требования, выраженные в виде пользовательских историй, являются основным инструментом влияния на разрабатываемый *программный продукт*. Пользовательские истории определяют формат, в котором у пользователей есть возможность отразить все те важные факторы, которые, *по* их мнению, должны быть учтены в процессе автоматизации.

Важно отдавать себе отчет в том, что пользовательские истории не обеспечивают полноту всех функциональных требований и имеют ряд недостатков:

* *Не масштабируются для больших программных продуктов.* Пользовательские истории хорошо себя зарекомендовали, когда речь идет о создании небольших или средних по объемам и сложности программным продуктам. За счет того, что этот вид работы с требованиями ориентирован на непосредственную работу с пользователями и поддерживается "бизнес-лексиконом", он неприменим в работе над крупными информационными системами, когда на первый план выходит организационная структура проекта/процесса, и важны формальные признаки сдачи/приемки работ.
* *Требовательны к квалификации разработчиков.* Разработчики, работающие с пользовательскими историями, должны обладать высокой квалификацией и неплохими коммуникативными навыками, которые позволят им получить необходимые уточнения уже изложенных требований. Как правило, таких разработчиков немного. И это еще один неоспоримый недостаток.
* *Не являются средством документирования.* Пользовательские истории - это небольшое и удобное представление информации. Они сформулированы на ежедневном языке пользователя и содержат небольшие детали, оставаясь открытыми для интерпретации. Они помогают понимать, что должна делать система, но при этом пользовательских требований недостаточно, чтобы понять, как будет организована логика системы. Пользовательские требования являются необходимой "верхушкой" для понимания назначения информационной системы, но для реализации системы разработчику приходится додумывать множество значимых деталей.

Инженерия требований - важный этап в создании каждого программного продукта. Scrum - гибкая методология, в которой работе с требованиями отводится не центральное, но при этом значимое *место*. Функциональные обязанности *по* извлечению, разработке и управлению требованиями ложатся на всех членов команды, именно поэтому в процессе работы с ними задействованы такие артефакты, как карта историй и пользовательские истории, назначение которых - облегчить получение информации, необходимой для качественной реализации программного продукта.

## 8.3. Определение приоритетов пользователей

После того как процесс работы с требованиями в Agile-методологиях стал более очевиден, упрощен и понятен в сравнении со стандартными, классическими подходами к созданию информационных систем, целесообразно обсудить техники, которые помогут облегчить процесс определения приоритетов отдельных задач для их последующего включения в спринты Scrum-команды.

На сегодня существует множество различных техник расстановки приоритетов. Лишь небольшая их часть может быть применена для приоритизации пользовательских историй как единицы инкремента для разрабатываемого продукта в Scrum.

Рассмотрим наиболее эффективные из них:

* *Принцип Эйзенхауэра.* Принцип, или, как еще называют в литературе, "матрица Эйзенхауэра" - техника тайм-менеджмента для определения приоритетов задач. Выглядит матрица как четыре квадрата, которые получаются при пересечении осей "Важно - не важно" по вертикали и "Срочно - не срочно" по горизонтали ( рис. 8.2). При использовании этой матрицы по ее "частям" распределяются задачи в соответствии с их важностью и срочностью.



Рисунок 8.2 –Матрица Эйзенхауэра

* 1. *Важные и срочные задачи.* Это те задачи, которые важны, и выполнить их необходимо срочно. Без них все порушится, ничего работать не будет, и сделать их завтра - будет уже поздно.
	2. *Важные, но не срочные задачи.* Это те, которые срочными станут в скором времени. Успешные команды и сотрудники в первую очередь обращают свое внимание именно на этот тип задач, чтобы они постепенно не перешли в разряд "Важные и срочные задачи".
	3. *Не важные, но срочные.* Это тип задач, которые никак не приближают к достижению необходимого результата, которые надо делать, но исключительно для того, чтобы их делать.

* 1. *Не важные и не срочные.* Эти задачи не важны, они не срочны, но именно их хочется делать. Это пожиратели времени, и от них необходимо избавляться.

*В момент усталости многие разработчики начинают заниматься сторонними делами, чтобы отдохнуть. Это неправильно. Правильно -запланировать качественный отдых в соответствии с индивидуальными особенностями каждого разработчика. Это задача из категории "Важно и не срочно".*

Принцип Эйзенхауэра - очень эффективная техника расстановки приоритетов. Но ее особенности больше направлены на расстановку приоритетов отдельного сотрудника, а не команды в целом.

* *Методика "АБВ".* В соответствии с этой методикой задачи делятся на три категории: жизненно важное, важное, приятное. Эта методика является логическим (более "глубинным") продолжением принципа Эйзенхауэра.

Применение принципа Парето конкретизируется, если все задачи проанализировать в соответствии с их долей в итоговом результате и затем распределить по категориям важности. АБВ основывается на следующих трех закономерностях, подтвержденных опытом:

* + Самые критичные задачи (категория "А") составляют ~ 20% всех задач, которыми занят руководитель. Значимость этих задач, вклада в достижение конечного результата составляет ~ 60%.
	+ Менее важные задачи (категория "Б") составляют ~ 20 % от общего числа. Их значимость - также 20% значимости задач и дел руководителя.
	+ Неважные и несущественные задачи (категория "В") составляют 60% от общего числа задач. Они имеют незначительную долю - 15% в общей ценности всех задач.

Методику "АБВ" можно применять для оценки приоритетов задач, необходимых для разработки продукта, но для того, чтобы грамотно ее использовать, владелец продукта и команда должны иметь определенный опыт и навыки по расстановке приоритетов.

* *Метод MoSCoW (Oracle).* Один из консультантов Oracle (Dai Clegg) предложил однозначный, логичный и понятный метод приоритизации требований для анализа и задач разработки программного обеспечения - MoSCoW. Этот метод используется при фиксированных сроках на реализацию функциональности, когда всё внимание должно быть обращено на самые приоритетные требования. Приоритизация задач методом MoSCoW позволяет сосредоточить фокус внимания как владельца продукта, так и Scrum-команды на задачах конкретного спринта.

Все имеющиеся задачи в бэклоге продукта следует сгруппировать в четыре приоритетные корзины по следующим правилам:

* + M (must) - задачи должно быть реализованы в первую очередь, без них программный продукт не имеет смысла. От этих задач нельзя отказаться. В них заключен залог успеха. Задачи, отмеченные как must, должны быть включены в текущий спринт.
	+ S (should) - следовало бы иметь, но можно отложить на более позднее время. Задачи этого типа также критичны для успеха разрабатываемого продукта, но не необходимы в ближайшее время. Такие задачи, как правило, имеют альтернативные пути решения.
	+ C (could) - можно было бы иметь, но если нет возможности их разработать сейчас, то можно и отложить. Задачи этого типа менее критичны. Они обычно относятся к типу "хотелось бы иметь".
	+ W (would) - в этот раз стоит отказаться от этих задач, но в следующий раз можно их сделать. Наименее критичные. Задачи данной категории не планируются к разработке в ближайшие спринты, но при будущих поставках их статус может быть пересмотрен.

Рассмотренные нами техники позволяют выполнять адекватную оценку задач, включение которых необходимо для разработки качественного программного продукта.

## 8.4. Доска задач

После того как мы определились с "спринтовой" моделью организации *работ* в Scrum и разобрались с тем, как необходимо работать с требованиями и задачами, настало время рассмотреть еще один очень важный *артефакт* каждой Scrum-команды, цель которого - визуализировать состояния задач в Scrum. Речь идет о Scrum-доске.

Каждый рабочий день все члены команды собираются на 15-минутное daily, на котором они обмениваются информацией *по* состоянию текущих дел ("Что сделано с момента предыдущей встречи?", "Чем планирую заниматься сегодня?", "Какие есть препятствия?").

На доске команды развешены задачи, каждой из которых выделен бумажный стикер, на котором написано, в чем состоит суть задачи ( рис. 8.3).



Рисунок 8.3 –Доска задач

Доска задач должна *минимум* делиться на три колонки:

* запланировано (To Do);
* в работе (In Progress);
* завершено (Done).

На момент планирования задач, которые предстоит взять в спринт, все карточки помещаются в колонку "Запланировано". Каждый день, когда член команды берет в работу определенную задачу, он говорит: "Я начал работать над…" и перемещает карточку в столбец "In Progress". После того как задача выполнена, *исполнитель* говорит о том, что работа над задачей закончена, и карточка перемещается в соответствующую колонку "Завершено".

Использование доски задач полностью соответствует принципам прозрачности *работ*, которые являются одним из главных преимуществ Agile в сравнении с альтернативными подходами к разработке программного обеспечения. Участники daily каждый день собственными глазами видят, как идет прогресс в работе над тем или иным типом задачи. Они не только погружаются в значимые детали *работ*, но и получают ценный *опыт*. Неоспоримыми выгодами, которые получает *команда* от использования доски задач, являются:

* наглядность спринтов;
* прозрачность состояния задач и проблемы;
* простой контроль за "загрузом" разработчиков и прочее.

Стоит сказать о том, что в начале своей деятельности каждая *команда* должна попробовать именно "материальную" доску, *по* которой каждый сможет перемещать свои задачи.

Во-первых, это приучает всех членов команды к определенной дисциплине. Каждый должен отчитаться за свои задачи и как именно над ними ведется работа.

Во-вторых, доска, которая постоянно висит на стене в одном месте, приучает всех относящихся к команде, что именно на ней можно увидеть актуальное состояние дел и задач.

## 8.5 Бэклог продукта

В теории Agile-процессов работа с требованиями и их трансформация в пользовательские истории, карту процессов и набор отдельных задач - это автономный процесс, за который ответственен владелец продукта и выделенные бизнес пользователи.

На практике же этот процесс требует творчества, навыков и немало ресурсов. Задать высокий темп выполнения Scrum-процесса за счет прозрачности поставляемых для команды требований - кропотливый труд, известный как инженерия требований. Не обязательно, чтобы этой сферой заведовал определенный специалист, более того, в Scrum-команде этот функционал должен быть рассредоточен между всеми членами коллектива.

Инженерия требований представляет собой набор техник, методов и принципов, связанных друг с другом.

*Инженерия требований содержит огромный потенциал для поднятия уровня технических навыков как разработчиков, так и владельца продукта, с тем чтобы создать правильный продукт.*

В последнее время стала распространенной точка зрения, что требования не играют значимой роли в гибких процессах разработки. На это можно возразить, что *знание* требований необходимо, чтобы привнести их суть в создаваемый продукт.

Ключевые моменты инженерии требований можно найти, сравнивая сходства в работе *по* управлению требованиями в традиционной и Agile-среде. Это ведет к осознанию, какие техники могут подойти конкретному типу процесса.

Основное заблуждение относительно управления требованиями заключается в том, что оно касается только надлежащего документирования. Конечная цель процесса сбора требований в том, чтобы облегчить и донести до всех заинтересованных общее понимание требований к создаваемому продукту. На практике подтверждено, что важно как *минимум* проговорить требования, чтобы убедиться, что все понимают их одинаково. Способы инженерии требований могут различаться при традиционном и гибком подходе, но конечная цель остается той же, и важно держать ее в уме, чтобы не запутаться, применяя конкретные технологии.

Техники сбора требований схожи при традиционной и гибкой работе, как можно увидеть на следующих примерах:

* Задача --> Решение.
* Общее --> Частное.
* Учет влияния различных факторов --> Улучшение качества продукта.
* Прочее.

Если все требования будут указаны разом во всех необходимых для процесса разработки деталях, то это точно не Agile. Спринт содержит ограниченный набор требований, с которыми работает Scrum-*команда*. Каждый спринт синхронизирует понимание владельца процесса и команды о конечном результате, к которому должна стремиться *команда*. В Scrum спринт содержит только те задачи, реализация которых необходима в краткосрочной перспективе. Их детально рассматривают и реализуют командными усилиями. Это приводит к тому, что владельцу продукта быстро показывают результат и могут получить от него эффективную обратную *связь*, которая позволяет больше узнать о требованиях к продукту и оптимальным образом скорректировать *рабочий процесс*.

Техники гибких процессов разработки в отношении требований основаны на идее совместных усилий относительно набора необходимых требований, в отличие от классического подхода, при котором специальные сотрудники сразу разрабатывают исчерпывающее описание. Команды гибкой разработки предпочитают собирать требования способом, который поддерживает взаимодействие и подвижность.

Пользовательские истории, создаваемые на основе требований, разрабатываются таким образом, что главным в них выступает *пользователь*. При традиционном подходе главным субъектом является система. Кроме того, традиционный подход часто делит функциональные и нефункциональные требования, обычно разделяя их на разные главы спецификации. Это *деление* предназначено обеспечить полноту сбора требований. В гибком подходе и функциональные, и нефункциональные аспекты собираются вместе.

Это ведет к лучшему пониманию всеми вовлеченными сторонами, выявлению новых деталей.

После того как необходимые требования к продукту собраны, их необходимо зафиксировать и "взять на учет" для дальнейшей реализации. Для этого в Scrum есть *артефакт*, который называется бэклог продукта (Product Backlog, PB).

Бэклог продукта - это упорядоченный *список* задач, которые должны быть реализованы в конечном продукте.

*PB является единственным источником требований для любых изменений, которые может потребоваться внести в ПО. Ответственность за бэклог продукта несет владелец продукта, включая его содержимое, доступность, упорядочение и приоритизацию.*

PB никогда не является полным. В начале (сверху) - только первоначально известные и наиболее понятные требования. Бэклог постоянно обновляется *по* мере обновления самого продукта и окружающей его среды, поэтому PB "живет" вместе с продуктом. PB является динамическим, постоянно изменяющимся для соответствия требованиям продукта, его конкурентоспособности и пригодности. PB существует ровно до тех пор, пока существует и сам продукт:

1. PB содержит все "фичи", функции, требования, усовершенствования и информацию по исправлению дефектов, то есть те данные, которые и определяют изменения, необходимые в следующих релизах продукта. Каждому элементу PB присваивается описание, порядковый номер, оценка объема работы и ценность.
2. Элементы бэклога продукта, расположенные сверху, должны быть более понятными и содержать больше деталей, чем те, которые расположены ниже. Более точные оценки даются требованиям, которые являются более четкими и содержат больше дополнительной информации.
3. Чем ниже находятся требования, тем меньше деталей. Бэклог спринта представляет собой репозиторий задач, реализация которых позволит инкрементально двигаться к достижению желаемого конечного результата от разработки конкретного программного продукта.

## 8.6 Бэклог спринта

Если бэклог продукта содержит полный *список* существующих задач, связанных с разработкой программного продукта, то в Scrum есть еще один *артефакт*, который содержит *список* задач, которые необходимо выполнить в спринте.

Бэклог спринта (Sprint Backlog SB) - это набор элементов Product Backlog, выбранных для выполнения в текущем спринте.

*SB - это прогноз и обязательства Scrum-команды относительно функциональности, которая станет частью разрабатываемого инкремента за время спринта.*

Бэклог спринта:

* наполняется во время планирования работ на спринт;
* визуализируется на доске задач;
* определяет тот объем работы, которую Scrum-команда должна выполнить, чтобы превратить требования и задачи в готовый для использования в бизнес-процессах инкремент.

Бэклог спринта должен быть достаточно "глубоко" детализирован *по* сравнению с задачами в Product Backlog, чтобы прогресс работы над ним можно было видеть ежедневно, не уделяя дополнительное время сбору необходимых деталей.

Scrum-*команда* работает над бэклогом спринта на всем его протяжении, и он постоянно изменяется вместе с прогрессом команды.

Изменения происходят потому, что в процессе работы возникают всё новые и новые задачи, которые нужно выполнить для достижения конечного результата.

Если возникает необходимость в дополнительном объеме работы, то эти задачи добавляются в бэклог продукта и планируются к выполнению в следующих спринтах.

После того как они выполнены, оценки оставшегося объема *работ* обновляются.

Если некоторые задачи считаются уже неактуальными, то их удаляют.

Только Scrum-*команда* может изменять свой бэклог во время спринта. *По* решению команды и ее участников задачи могут добавляться или удаляться, но необходимо четко обосновать, почему это необходимо.

## 8.7 Инкремент продукта

Цель команды Scrum в отдельно взятом спринте - предоставлять работающий *инкремент* продукта.

*Работающий "прирост" продукта (инкремент) - обязательный результат каждого спринта Scrum.*

В составе бэклога спринта могут находиться совершенно разные задачи, но главное для команды - создание инкремента программного продукта. Отдельный *инкремент* может включать в себя недостаточно функциональности для принятия владельцем продукта решения о его вводе в эксплуатацию, но *команда* должна убедиться, что поставляемое ими качество достаточно для поставки и запуска его в продуктивную эксплуатацию.

*Инкремент* *по* своей сути - это продукт, который будет приносить бизнес-пользу ( рис. 8.4) в момент своей разработки после того, как результаты определенного спринта реализованы Scrum-командой и приняты заказчиком.

|  |  |
| --- | --- |
| https://intuit.ru/EDI/02_07_20_3/1592432413-28243/tutorial/824/objects/8/files/08_04a.jpg | https://intuit.ru/EDI/02_07_20_3/1592432413-28243/tutorial/824/objects/8/files/08_04b.jpg |

Рисунок 8.4 – Наглядное изображение инкремента в строительстве

*На* рис. 8.4 *приведены две иллюстрации.*

*Первая (левая) иллюстрирует неинкрементальный процесс организации постройки дома. До тех пор, пока дом в целом не будет готов, в нем нельзя будет жить.*

*На второй (правая) продемонстрирован инкрементальный процесс строительства. После того как первый этаж будет готов, в доме можно начинать жить и при этом продолжать строительство второго этажа.*

Цель спринта - сформировать, спланировать и разработать самодостаточный, готовый для применения продукт или его часть, которая будет дополнять уже имеющиеся разработки и повышать суммарную *стоимость* разрабатываемого программного продукта.

*По* окончании спринта *инкремент* должен быть пригодным к использованию и приносить реальную и ощутимую *прибыль* PO.

*Архитектура* системы или дорожная карта продукта могут быть построены таким образом, что поставить *инкремент* продукта в сроки спринта - это очень сложная задача, а порой - невыполнимая. Таким образом, мы должны вернуться к самой идее наполнения бэклога продукта однозначными и понятными задачами и пересмотреть имеющиеся требования в сторону итерационной и инкрементальной работы.

Инкрементальная модель разработки программного обеспечения является наиболее успешной и перспективной моделью создания и внедрения программных продуктов.

## 8.8. Принцип прототипирования

Scrum основывается на "итерационно-инкрементальной" разработке программных продуктов, которые будут с момента своего "зачатия" приносить компании *прибыль*. Шаг за шагом.

А для этого каждый отдельный реализуемый *инкремент* должен иметь определенную ценность сам *по* себе, при этом добавляя ценности к ранее реализованным инкрементам. В связи с этим работа с требованиями и как следствие с архитектурой создаваемого программного продукта трансформируется в процесс постоянного прототипирования решения *по* актуальным требованиям. В связи с этим повышается важность отдельных активностей самих *по* себе и их дальнейшей интеграции друг с другом.

* *Работа с требованиями.* Их анализ, синтез и последующее проектирование имеют первостепенную значимость. Необходимо так выстроить процесс разработки, чтобы каждый последующий разрабатываемый инкремент интегрировался в существующий продукт и при этом оптимизировал его с точки зрения повышения ценности производимых информационной системой результатов.
* *Требования должны быть взаимосвязаны, трассируемы по уже реализованному функционалу.* В каждый момент времени рабочий продукт должен быть самодостаточен с точки зрения ценности. Поэтому важно правильно проводить анализ влияния выполняемых разработок с точки зрения существующего функционала и места в информационном ландшафте заказчика создаваемого программного продукта.

С учетом обозначенных факторов повышается *значимость* проведения постоянного рефакторинга программного продукта *по* причине его постоянной изменчивости. При этом цель и направление *работ* должны быть четко определены, и задача Scrum-команды и владельца продукта - постоянно отслеживать *тренд* развития продукта.

Подобный принцип организации и управления *работ* получил ранее распространение в области разработки программного обеспечения и называется принципом прототипирования. Этот принцип наилучшим образом подходит к проектированию информационных систем, разрабатываемых в соответствии со Scrum.

## 8.9. Краткие выводы

В этой главе внимание было уделено тем атрибутам, которые выделяют Scrum в наиболее формализованный вид гибких методологий. Именно они позволяют говорить о том, что Scrum является самой понятной и эффективной методологией разработки программного обеспечения, внедрение которой является относительно несложным процессом.

Использование в операционной работе доски задач, бэклога продукта и спринта позволяет сделать прозрачным процесс работы над пользовательскими историями и картой задач, в конечном виде выражающей результат в виде готового к использованию инкремента продукта.

**Питання до теми:**

# Лекция 9: ОЦЕНКА

План

9.1 PERT - оценка сроков

9.2 Переход от оценки к обязательствам

9.3 Сбалансированная система показателей Scrum-команды

9.4 Наработанная статистика результатов - фундамент прогнозирования и побед

9.5 Краткие выводы

## 9.1 PERT - оценка сроков

Адекватная оценка трудоемкости выполняемых *работ* - мастерство управления, которого достигают только специалисты, постоянно сталкивающиеся с необходимостью профессионально оценивать ресурсы, необходимые для реализации задач.

На текущий момент накоплено достаточно много различных методик, успешно зарекомендовавших себя в различных условиях эксплуатации. Существующие техники различаются по следующим, важным для применения в операционной деятельности группам характеристикам:

* Научная обоснованность применения.
* Необходимость наличия статистического репозитория данных по выполнению в альтернативных процессах/проектах.
* Возможность оперативного использования и получения результата.
* Наличие определенного инструментария.
* Прочие.

После того как сравнению было подвергнуто большинство методик, стало очевидно, что для использования в Scrum подходят совсем не многие.

С одной стороны, используемая методика должна иметь научный *базис*, на котором было бы возможно проводить обоснованные вычисления трудоемкости *работ*, необходимые для каждой конкретной задачи, но, с другой стороны, она не должна быть громоздкой для применения в плане времени вычисления и сложности используемого инструментария. Таким образом, часть высокоточных техник, основанных на сложном математическом аппарате и продолжительных вычислениях, отпала сама собой, так же как и техники, которые в своей основе используют преимущественно экспертные методы оценки трудоемкости. В лидеры вырвался комбинированный метод, называемый *PERT*, применяемый в проектном управлении.

*PERT (Program Evaluation and Review Technique) - техника проверки и оценки трудоемкости выполняемых работ.*

После того как пользовательские истории и задачи детализированы на необходимом уровне, следует иметь адекватную оценку трудоемкости их выполнения. Под трудоемкостью понимают совокупность следующих значений:

* *Срок разработки.* Под сроком разработки понимается дата к которой необходимо подготовить необходимый программный продукт, его функциональность или отдельную задачу. Срок разработок в реалиях сегодняшнего рынка производства программных продуктов во многом предопределяет качество выполняемых работ и их объем. Когда же мы говорим о гибких процессах разработки программного обеспечения, то оценка сроков работ отходит на второй план, уступая место объему работ и их качеству. Срок разработки замещается понятием спринтов. Заказчик, увидев на демо "работоспособный" результат, готовый к применению, сам определяет, сколько еще спринтов понадобится, чтобы достичь желаемого им качества.
* *Качество разработки.* Способность поддержать заданное качество программного продукта - качество только самоорганизованных Scrum-команд. Качество программного продукта, заданное командой, зависит от множества различных факторов, но определяется квалификацией членов команды. Однако необходимо помнить, что заказчик, повышая требования к качеству, будет платить за это увеличением сроков работ (количества спринтов).
* *Объем выполненных работ.* Эта характеристика напрямую связана с показателями качества разрабатываемой информационной системы. Чем выше уровень требуемого качества, тем больший объем работ необходимо реализовать для достижения заданных показателей, а соответственно, больше времени потратить на то, чтобы достичь реализации конечного продукта.

Приведенные характеристики, оценка которых необходима для разработки программного продукта или его инкремента в рамках Scrum, не противоречит, а только подтверждает верность утверждения из проектного управления о "треугольнике ресурсов", используя которые, всегда придется пожертвовать одним ради достижения максимальных показателей двух оставшихся.

*В основе методологии Scrum лежит утверждение о том, что "конечные сроки" разработки программного продукта - рудимент, от которого необходимо избавляться. Scrum имеет в своей основе "спринтовую модель".*

*По результатам каждого спринта заказчик должен получать готовый к использованию инкремент функциональности программного продукта. Заказчик сам регулирует необходимое количество спринтов для достижения конечного, удовлетворительного для использования результата от разработки программного продукта.*

На практике подтверждено, что наиболее точная оценка лежит в рамках относительной погрешности ±5%. Этот допуск является вполне приемлемым для работы по Scrum, но "попасть" в него необходимо как можно более точно. Для этого предлагается к использованию методика *PERT*, *адекватность* и обоснованность которой подтверждены не только теоретически, но и на массиве конкретных практических примеров. Метод *PERT* часто в литературе называют еще и "методом трех точек".

В основе метода лежит оценка трудоемкости *работ* определенной задачи, полученная от нескольких экспертов (как *минимум* три). Все полученные оценки экспертов классифицируются по следующим группам:

* оптимистичная (a);
* ожидаемая (m);
* пессимистичная (b).

И в каждой группе вычисляется среднее *значение*, отражающее групповую оценку планируемых *работ*, выраженную в абстрактной или конкретной величине. После того как оценки получены, выполняется их усреднение по достаточно простой формуле ( рис. 9.1).



Рисунок 9.1 – Формула усреднения

В формуле учтена каждая из заявленных "групповых" оценок, но при этом основной вклад вносит ожидаемая оценка. Методика *PERT* является универсальным инструментом оценки любой из характеристик "проектного треугольника". В ней сочетаются как экспертные мнения, так и математическая основа, которая позволяет нивелировать возможные риски крайних оценок.

После того как будут предварительно оценены все задачи, предполагаемые к включению в бэклог спринта, становится возможным влиять на длительность спринта или его объем задач. В том случае, когда компания в состоянии достоверно оценить объем необходимых *работ*, спринты могут быть сформированы в соответствии с оцененной длительностью *работ*. Еще раз сделаем акцент на том факте, что гибкие процессы разработки - это способ организации процессов, и если компания заинтересована в его применении, то возможно Scrum "вписать" в любую организационную структуру и наладить тесное сосуществование различных методик управления процессной деятельностью.

## 9.2 Переход от оценки к обязательствам

Оценки, даваемые отдельными членами или командой в целом, и обязательства - это не одно и то же. Фундаментальная и, пожалуй, основная проблема многих специалистов и организаций состоит в том, что даваемые оценки и обязательства, которые они при этом принимают на себя, в глазах менеджмента считаются одним и тем же.

Разработчики и команды, какой бы методологии они ни придерживалась, прикидывают, что реализация желаемой совокупности инкремента - при наличии у команды определенных ресурсов - займет определенное время. Члены команды сообщают это руководству или Scrum-мастеру, после чего она транслируется заказчику.

В некоторых случаях в процессе таких передач критичной для разработки информации от одних лиц к другим первоначальная оценка изменяется до неузнаваемости, в результате чего перед командой ставится нереальная цель.

Проблема в подобных случаях заключается не в том, что оценка команды оказалась правильной или неправильной, а в том, что оценка трансформировалась в обязательства, которые приходится брать на себя команде. Важны и оценка продолжительности выполнения проекта, и обязательство, принятое на себя командой, однако их необходимо рассматривать как разные виды деятельности.

Scrum-*команда* и современные организации должны уметь отделять оценки от обязательств. Оценка всегда первична, а уже после, основываясь на степени уверенности, опыте, навыках, мы можем преобразовать ее в определенное обязательство.

Но без надежной первоначальной оценки любое обязательство команды будет лишено всякого смысла. Чтобы получить надежную оценку, владелец и Scrum-*команда* должны располагать всеми необходимыми для этого данными:

* объем работы, которую предстоит выполнить;
* факторы, влияющие на темп выполнения командой работы, и, как следствие, динамика выполнения задач в спринте.

Первоначально при внедрении Scrum практикуется оценка в виде диапазона, затем, по мере зрелости, *команда* будет в состоянии перейти к точечной оценке, которая будет использоваться в качестве обязательства.

Подобный переход на более высокий уровень самоорганизации в тактической перспективе сулит команде проблемы с выполнением обязательств, но в стратегической перспективе это поможет снизить риск в нарушения оценок и обязательств, даваемых командой.

Альтернативная ситуация, когда перед командой сразу ставится ответственность по реализации определенного объема к заданной дате, более рискованна и в отдаленной перспективе не принесет желаемых результатов.

*При внедрении Scrum в организациях более эффективно двигаться инкрементально, по шагам, постоянно повышая заданную планку результатов.*

Оценки и обязательства по своей сути различаются степенью ответственности за формирование конечного результата. В случае, если *команда* готова брать на себя обязательство по реализации программного продукта и его функциональности за определенный срок, это свидетельствует о зрелости ее членов и высоком уровне качества продукта, который они создают.

Качественный переход от оценки *работ* к обязательствам свойственен высококвалифицированным командам, которые хорошо знакомы с тем продуктом, который они разрабатывают.

Для того чтобы помочь командам нащупать свой оптимальный темп работы и раскрыть их лучшие качества, имеет смысл задуматься о применении к Scrum сбалансированных системных показателей.

## 9.3 Сбалансированная система показателей Scrum-команды

Вариант комплексного взгляда на процессы современной компании привел к созданию сбалансированной системы показателей (*ССП*).

Ее идея заключается в следующем - чтобы как можно лучше уяснить функционирование любой организации, нельзя ограничиваться только анализом показателей о прибылях и убытках и финансовых отчетов. Они представляют собой лишь ограниченное видение того, как функционирует соответствующее предприятие.

Цель *ССП* состоит в анализе не столько финансовых метрик функционирования предприятия, сколько в рассмотрении четырех процессных направлений (финансы, клиенты, процессы, обучение и рост), в совокупности представляющих основу деятельности каждого предприятия.

Если попытаться установить Scrum-команде *KPI* только по их специализированному направлению разработки программного обеспечения, то руководство организации столкнется с тем, что *деятельность* команды будет направлена не на достижение целей деятельности компании, а на оптимизацию обозначенного показателя.

Известно и не подвергается сомнению, что, если мы вводим какой-либо показатель и сообщаем команде о том, что их работа будет оцениваться по этому показателю, они постараются адаптировать свою *деятельность* таким образом, чтобы оптимизировать именно этот показатель.

Если работа будет оцениваться, скажем, по количеству обнаруженных системой отслеживания дефектов, соответствующий показатель резко начнет снижаться. Возможно, из-за внедрения разумных усовершенствований, возможно потому, что команде удастся неформально информировать друг друга о дефектах или представлять их в несколько ином качестве. Если бы было возможно разработать показатель, который невозможно "обойти" любым способом, все равно один показатель не отражал бы полную картину деятельности. Требуется более сбалансированный взгляд, чем тот, который обеспечивается представлением одного показателя.

Система сбалансированных показателей должна оптимизироваться под условия деятельности конкретной Scrum-команды, но направления оценки, как правило, совпадают с классическими метриками *ССП*, и они, как правило, общие для всех успешных команд:

* Операционное совершенство:
	+ Создание продукта с максимальной производительностью.
* Ориентация на пользователя:
	+ Фокус на разработке приоритетных функций.
* Экономическая ценность:
	+ Обеспечение ценности в форме экономии затрат, повышения доходов и т. д.
* Ориентация на будущее:
	+ Создание новых продуктов также способствует росту и повышению квалификации членов команды.

Выше представлены метрики, которые могут быть взяты за основу при построении системы сбалансированных показателей отдельной команды, учитывающих конкретное окружение Scrum-коллектива.

Если создание сбалансированной системы показателей синхронизировать с созданием отдельной команды, и *команда*, отдел или организация затем периодически будет оцениваться по этим показателям, то прогресс неизбежен. *Команда*, которая действует успешно в условиях использования Scrum, должна проявить способность к одновременному улучшению по каждой из перечисленных выше точек зрения. Сбалансированная система показателей переключает внимание команды с полной сосредоточенности на гибкой методологии разработки на достижение целей, которые привели конкретную организацию к освоению гибкой методологии разработки.

Стоит обратить внимание на то, что сбор метрик, поддерживающих внедренные показатели, даже если речь идет об очень простых показателях, связан с затратами сил и времени и явно не относится к числу достижений, которыми могут похвастаться компании. Необходимо понимать, что на сбор метрик тратится достаточно много сил и внимания. Этот процесс должен стать частью операционной деятельности организации.

Следует указать три основных преимущества, которые дает сбор показателей:

* *Преодоление возможной организационной гравитации.* Существуют определенные причины сохранения ситуации, имевшейся до начала преобразований. "Организационная гравитация" стремится вернуть нас к тому состоянию, в котором она находилась на протяжении длительного периода времени. Периодические оценки и показатели, демонстрирующие выгоды перехода к Scrum, окажутся наиболее сильным средством, которое поможет преодолеть эту силу.
* *Пропаганда Scrum.* Система операционного оценивания помогает не только преодолевать "силу гравитации", но и явно демонстрировать преимущества Scrum по всей организации. Чтобы привить интерес к переходу на Scrum, очень важно разумно рекламировать основные успехи на этом пути.
* *Обоснованный тренд развития Scrum.* Показатели помогают понять, на чем следует сосредоточить усилия для дальнейшего совершенствования. Показатели должны приводить к определенным действиям. Если собираемые данные не приводят к каким-то действиям и решениям, нужно прекратить сбор этих данных и начать сбор других данных.

Собранные метрики и показатели будут способствовать выявлению "отстающих" процессов и активностей в целях их дальнейшего совершенствования и развития. Также постоянный *мониторинг* показателей поможет держать команду в тонусе, анализируя возможные отклонения в ее деятельности и управляя ими в целях достижения конечного результата.

Сбалансированная система показателей должна быть не "мертвой" системой показателей, а живым и динамичным процессным framework, результаты которого, накопленные в виде массива информации, позволят прогнозировать с высокой долей вероятности точные результаты деятельности команды, в зависимости от определенного количества имеющихся в наличии ресурсов.

## 9.4 Наработанная статистика результатов - фундамент прогнозирования и побед

Высокая скорость работы команды должна сложиться в соответствии с историческими предпосылками работы команды.

Как только становится необходимым донести эту информацию до лиц, принимающих решение, обычно в ответ мы слышим множество обоснованных возражений, таких как нехватка ресурсов, а главное, отсутствие возможности оперативно наработать необходимый *массив* данных. Но сделать это достаточно просто, даже с учетом таких распространенных проблем, как "новая *команда*" и фактор постоянных изменений как команды, так и объема необходимой функциональности.

Если речь идет о вновь сформированной команде, то оптимальным решением в таком случае будет предоставить членам команды возможность выполнить несколько спринтов вместе, и только после этого у них появится возможность принять на себя какие-то обязательства.

Для того чтобы задать скорость работы для вновь сформированной команды и заложить основу фундамента оценки, целесообразно собрать фокус-группу из специалистов, обладающих схожим опытом работы с теми, кто включен в Scrum-команду, и задействовать их в оценивании тех задач, которые нужно выполнить. Это поможет направить работу команды в нужное русло. При этом, когда пройдет несколько спринтов, результаты постепенно будут увеличиваться. Таким образом можно преодолеть проблему "втягивания" команды в Scrum.

Но когда численный состав постоянно меняется, возникают проблемы иного рода. Этот фактор можно рассматривать с многих точек зрения. С одной стороны - *стабильность* идет на пользу. С другой стороны, длительная *стабильность* приводит к застою и постепенной профессиональной и личностной деградации.

Важно при работе с изменениями придерживаться срединной позиции, избегая сильных перегибов, при этом собирая данные о том, как те или иные изменения влияют на *деятельность* коллектива в тактическом и стратегическом приближении. Это позволит быть готовым к любым изменениям в составе команды и предвидеть и управлять возможными влияниями осуществляемых изменений. Изменения нужно отслеживать постоянно, учитывая различные факторы, после которых нарушается устоявшаяся *деятельность* команды.

*В основе Scrum лежит простая идея - организация постоянного контроля деятельности, выполняемой мотивированными профессионалами, адаптированного под конкретные условия работы, с учетом фактора постоянных изменений как во внутренней, так и во внешней среде.*

## 9.5 Краткие выводы

Умение составлять эффективные планы - весьма важное качество для любой Scrum-команды. В этой главе мы рассмотрели способы, с помощью которых *команда*, освоившая основы планирования и оценки трудоемкости спринтов, может улучшить свои результаты. Кроме того, мы пришли к выводу о том, что понятия оценки и обязательств, даваемые командой и воспринимаемые менеджментом компании, различаются степенью ответственности. Все команды, достигшие нужного уровня зрелости и самоорганизации, должны стремиться к тому, чтобы давать обоснованные обязательства и в дальнейшем соблюдать их.

Мы поговорили о системе сбалансированных показателей в приложении к гибким процессам производства программных продуктов. Если оптимально адаптировать технику построения показателей, используемую в *ССП*, под *деятельность* Scrum-команды, то можно прогнозировать *увеличение производительности* команд и их постепенное развитие и совершенствование.

Фундамент, выраженный в виде собираемой статистики и метрик процесса Scrum, наработанный за период внедрения и применения гибких методологий, не должен оставаться без внимания. Он должен быть положен в основу системы прогнозирования и развития Scrum.

**Питання до теми:**

# Лекция 10: ИТОГИ И ПЕРСПЕКТИВЫ

План

10.1 Сосуществование с альтернативными процессами последовательной разработки программного обеспечения

10.2 Обеспечение соответствия лучшим практикам и стандартам

10.3 Использование Lean-методологии в Scrum-процессе

10.4 Продуктивность Scrum для цифровой трансформации

10.5 Современная критика Agile

10.6 Что дальше

## 10.1 Сосуществование с альтернативными процессами последовательной разработки программного обеспечения

В третьей лекции мы рассмотрели и достаточно подробно сравнили классический, итерационный и гибкий подходы к разработке программного обеспечения, а также постарались обоснованно развенчать мифы о несовместимости совместного использования различных подходов.

Во многом такие сторонние точки зрения обусловлены противоположным опытом и ситуацией, в которых осуществлялось и осуществляется *эксплуатация* данных подходов. Но, несмотря на этот дуализм, многие современные исследователи и специалисты констатируют, что можно приобрести много преимуществ от одновременного использования обоих методов в организации.

Одна из наиболее авторитетных организаций в области процессов *SEI* (организация - *автор* модели *CMMI*) опубликовала технические заметки, в которых излагается, почему можно использовать оба метода одновременно.

Различные причины инициировали рост рассматриваемых методов. Во многих организациях, которые пришли к идее внедрения Scrum или любой другой гибкой методики, он приходит на смену классическому подходу. Это происходит не одномоментно, а в течение определенного интервала времени, за который организация должна осуществлять свою полноценную *деятельность* (приносить доход, развиваться и т. д.), поэтому возникают пересечения этих методов:

* *Водопад в начале.* Используется и эффективен в том случае, когда речь идет о запуске проекта или внедрении нового продукта в рамках жестких сроков. Иначе возникает угроза срыва проекта.
* *Водопад в конце.* Обычно происходит на фазе приемочного тестирования функциональности разработанного продукта.
* *Водопад в тандеме.* Самый сложный способ взаимодействия. Возникает в случае, если команды используют разные методологии разработки ПО. Решается в том случае, если ряд членов команды, использующей тяжеловесные технологии, принимают участие в рабочих встречах Scrum-команд.

Молодые специалисты, практикующие гибкие методологии, поднимаются *по* карьерной лестнице и видят влияние водопада на гармоничный процесс гибкой разработки. К примеру, на уровне проектов внедрения классический подход фокусируется на том, чтобы продукты разрабатывались в полном объеме, в то время как гибкая модель концентрируется на том, как его разрабатывать.

Многие современные компании поставлены перед необходимостью сочетать основные процессы, выстроенные *по* классическому принципу разработки программного обеспечения, с наличием эффективных команд разработчиков, использующих гибкие методологии. С другой стороны, есть прецеденты, когда в преимущественно "гибкой" среде используется ограниченный *по* времени итеративный подход.

Нет процесса, который одинаково эффективен при различных условиях функционирования.

*Agile делает акцент на мобильности, изменениях и коммуникации, а классический подход многие обвиняют в излишней требовательности к рационализированной среде использования.*

Организации совершают непоправимую ошибку, если являются адептами лишь одной технологии. Они ограничивают себя в рассмотрении наиболее эффективных веяний и постепенно приходят в состояние полной зашоренности.

Текущие веяния говорят о том, что наиболее эффективные организации со временем все больше внимания будут уделять совмещению гибких и классических методологий разработки программного обеспечения.

## 10.2 Обеспечение соответствия лучшим практикам и стандартам

Не каждая Scrum-*команда* может позволить себе "матриархально" владеть полным циклом управления используемым ею процессом разработки. В качестве примера можно привести разработку программных продуктов с участием субподрядчиков, к которым в большинстве случаев выдвигаются требования *по* подтверждению уровня зрелости их собственных процессов области информационных технологий.

*По* сути это означает, что разработчики программного обеспечения должны использовать сходный набор передовых методов и применять в практике создания программных продуктов такие стандарты, как *ISO 9001* (процессные стандарты), *ISO* 13458 (стандарт медицинской отрасли), требования закона Сарбейниса-Оксли (акционерные компании на территории США) и пр. Таким образом, одной из задач Scrum-команды, направленной на разработку качественного программного продукта, является постоянная забота о том, как интегрировать профильные стандарты в создаваемый ими продукт.

Scrum-*команда* поставлена в условия, когда сама отвечает за конечный результат и при этом минимальным образом зависит от внешнего окружения. В таких условиях возрастает необходимость постоянного совершенствования используемых процессов в плане соответствия отраслевым и международным практикам для целей наименее затратной синхронизации получаемых результатов с результатами деятельности возможных партнеров и коллег. В случае, если этого не будет происходить, *команда* рискует отстать от реальности рынка как процессно, так и технологически, и разрабатывать "отсталый" продукт.

Членам Scrum-команд необходимо заботиться о том, чтобы соблюсти набор правил и принципов, предписываемый "best *practice*" и профильным стандартам. Для этого следует:

* *Организовать процесс документирования требований к продукту в согласованном командой формате.* Система документирования требований позволит описать набор автоматизированных функций и иметь постоянный доступ к площадке оперативного обсуждения последующих доработок и развития функционала продукта в соответствии с актуальными требованиями бизнеса.
* *Создать базу знаний команды/продукта.* База знаний команды/продукта - это своего рода интеллектуальный компонент команды, который позволяет организовать процессы решения возникающих проблем путем документирования в различных форматах (руководства, описания, статьи и т. д.) наиболее важной информации. Базы знаний позволяют накапливать информацию и преобразовывать ее в соответствии с актуальными нуждами команды. Основное назначение - оказать помощь наименее опытным сотрудникам при решении определенных проблем.
* *Управлять изменениями.* О важности процесса управления изменениями было сказано в предыдущих главах. Здесь же еще раз укажем на то, что управлять изменениями нужно на постоянной основе не только в разрезе разработки продукта, но и в разрезе организации "гибких" процессов, оптимальным образом адаптируя их под нужды организации.
* *Документировать актуальное состояние процессов в команде.* Для того чтобы на постоянной основе заниматься развитием и совершенствованием процессов по передовым практикам и стандартам, необходимо иметь актуальное состояние процессов задокументированным. Только тогда будет возможность оперативно рассмотреть их и принять управленческие решения.
* *Применять передовые технологии для автоматизации процессов (тестирование, сбор требований и т. д.).* Члены команды, используя процедуры ретроспективы или механизмы самоанализа выполняемой ими работы, должны постоянно задумываться над используемыми ими техниками, инструментами, профессиональными привычками. Наиболее "отсталые" из них, те, эффективность которых низкая, необходимо модифицировать в соответствии с актуальными условиями Scrum, повышая их ценность и значимость, или же отказываться от них в пользу новых. Описанное так же верно и для автоматизации процессов тестирования, работы с требованиями и пр.
* *Искать, тестировать и использовать инновационные инструменты, которые оптимизируют вашу деятельность.* Члены команды должны постоянно находиться в технологическом поиске. Качество постоянного несогласия с факторами, влияние которых негативно отражается как на командной, так и на личной работе, - отличное качество разработчика. Разработчик может, а порой и должен смириться с недостатками конкретной ситуации, но при этом ему следует постоянно искать пути решения системных проблем. Он должен постоянно искать, пробовать, изменяться, предлагать удачные и отказываться от неудобных рабочих инструментов. Это, по сути, и есть Agile.
* *Приглашать профессиональных аудиторов.* Роль аудитора в реалиях Agile не нужно недооценивать. У команды, как правило, всегда все хорошо. Недостатки и несовершенства могут "замыливаться" за операционной деятельностью коллектива. Человеческая натура так устроена, что она постепенно привыкает ко всему. К хорошему быстро, к плохому чуть медленнее. Аудитор - сотрудник, который должен непредвзято провести валидацию процессов. С помощью его отчетов можно объективно сопоставить, как у команды получается двигаться в желаемом направлении развития, выделить наиболее слабые места, на работе которых стоит сделать акцент. Аудитор может быть как внешний, так и внутренний, главное в его работе - видеть процесс и роли, а не ситуацию и отдельных личностей.
* *Приглашать квалифицированных консультантов.* "Вариться в собственном соку" очень важно. "Не вариться" сопоставимо с "не развиваться". В силу того что члены команды не всегда знают, как наиболее оптимально решать возникающие проблемы и чем принятые решения могут "грозить" в дальнейшем, важно привлекать консультантов, которые обладают практическим опытом решения проблем, с которыми столкнулась или может столкнуться команда. Перед тем как их приглашать, следует убедиться в том, что консультант сможет помочь, а не придет "отсидеть" свою стоимость.

Роль аудитора и консультанта могут выполнить сотрудники организации. При этом они должны обладать нужным уровнем квалификации, который позволил бы успешно выполнять возложенные на них функции и обязанности, а также соответствующими чертами характера.

Подытоживая, следует отметить, что Scrum, как и любое другое процессное направление деятельности, должен развиваться за счет принятия и адаптации наиболее успешных практик, доказавших свою эффективность. Кроме всего прочего, важно отметить необходимость постоянного совершенствования профильного направления деятельности. В нашем случае речь идет о разработке программного обеспечения, которая является одной из самых динамичных областей и консерватизм в которой, как правило, оборачивается потерей скорости разработки и внедрения новых продуктов.

## 10.3 Использование Lean-методологии в Scrum-процессе

Методология *Lean* направлена на совершенствование и процессное развитие. Она не подходит для тех процессов, которые находятся на "зачаточном" и начальном уровнях зрелости. *По* сути, о необходимости внедрения методологии *Lean* задумываются те компании, которые достигли определенного качественного уровня.

*Lean - это маршрут для тех организаций, которые уже уверенно стоят на процессных рельсах. Она подразумевает меньше конкретных методик. Ее применяют в контексте собственной организации, полностью адаптируя под свои условия.*

В *Lean*, так же как и в Scrum, работа разбивается на небольшие пакеты (сравнимо с понятием бэклога спринта), которые должны реализовываться отдельно и независимо. *Lean*, в целях разработки предсказуемого результата, содержит определенный *поток* операций (*workflow*) с этапами. В Scrum существуют свои специфичные процессы и процедуры, которые подтвердили свою эффективность. *Lean* добавляет к "гибким" принципам схему потока операций, для того чтобы каждая из итераций выполнялась одинаково качественно.

Оба направления в своей основе имеют системный подход, основанный на постоянном обучении и совершенствовании. В данном случае Scrum является базисом, на основе которого станет возможным "надстраивать" специфичные методы и методики, которые будут способствовать повышению эффективности процессов компании и, как следствие, увеличивать/улучшать результат.

*Lean-методы направлены на "расширение узких горлышек" и минимизацию избыточной сложности для повышения эффективности Scrum. Это позволит не только изменить внутренние процессы Scrum-команды, но и даст возможность менять процессы снаружи.*

Организация методологии *Lean* и ее этапов позволяют быть уверенными в том, что каждая часть проекта/продукта/процесса реализуется так, как этого требует результат. В *Lean*, как, впрочем, и в Agile, нет четких границы этапов, но, как и в Scrum, прописаны ограничения спринтов.

*Lean позволяет параллельно выполнять несколько задач на разных этапах, что повышает гибкость и увеличивает скорость исполнения проектов.*

Как и гибкие процессы, *Lean* более похожа на концепцию, образ мышления, нежели четко регламентированную методологию. Используя принципы *Lean*, можно создать адаптивную систему, удовлетворяющую конкретным требованиям. Для этого необходимо следовать следующим принципам:

* *Постоянно определять ценность конкретного продукта.* Разработчик должен исследовать реализуемый продукт и наблюдать за ним. Более того, по мере функционирования команды и разработки зрелого продукта каждый разработчик должен научиться оценивать то, какими параметрами необходимо обладать, чтобы стать ценностью.
* *Определить и контролировать поток создания ценности.* Поток создания ценности - это последовательность действий, совершаемая для преобразования исходного материала в конечный продукт, приносящий ценность. Если конкретный этап не создает ценности, целесообразно провести его изменение или удаление из процесса.
* *Обеспечить непрерывность потока создания ценности.* Вместо дискретной организации перемещения продукта по стадиям процесса необходимо рассмотреть возможность создания потока, в котором происходит непрерывное движение продукта по стадиям через специализированные точки (вехи), в которых будет возможность организовать операционный контроль как стадии, так и качества продукта путем измерения заранее разработанных метрик.
* *Позволить пользователю/владельцу продукта "вытягивать" продукт.* Пользователи, находящиеся "выше" по процессу, ничего не должны делать до тех пор, пока конечный потребитель, расположенный "ниже", этого не потребует. Должны создаваться все условия для того, чтобы потребитель "вытягивал" продукт, то есть производство продукта осуществлялось под конкретный заказ. Таким образом, именно конечный пользователь будет запускать "вытягивание" по всем этапам производства.
* *Постоянно стремиться к развитию и совершенству.* Разработку продукта в Lean можно совершенствовать бесконечно, максимально приближая его к "идеалам" ожиданий заказчика. Можно сколь угодно уменьшать трудозатраты, время на разработку и пр.

Подытоживая, стоит сказать, что сильной стороной *Lean* является разнообразный набор инструментов и подходов к работе для удовлетворения разнообразных требований. *Lean* сочетает гибкость и структурированность, но немного отличается в этом от Scrum.

Его недостатком является слишком детальная и дотошная проработка каждой стадии работы. *Lean* предполагает такой подход к каждой задаче и этапу. Это основной минус применения *Lean*, когда речь идет о средних и мелких продуктах. В отличие от Scrum, *Lean* не предлагает четкого рабочего процесса для реализации "кусочков", что способствует растягиванию сроков проекта. Эта проблема может быть решена при помощи эффективного руководства и четких коммуникаций.

Внедрение и последующее использование *Lean* должно стать целью для тех организаций, в которых Agile уже завоевала свое *место* и на ряде проектов подтвердила свою эффективность.

## 10.4 Продуктивность Scrum для цифровой трансформации

Необходимость цифровой трансформации (ЦТ) на сегодня осознана уже многими компаниями. Под ЦТ понимается прежде всего создание гибкой процессной инфраструктуры, способной поддержать любые начинания *по* развитию бизнеса и способствовать достижению намеченных результатов в как можно более сжатые сроки.

То, как сегодня работают специалисты и как они взаимодействуют с клиентами, определяет возможность предприятия стать лидером профильного сегмента рынка и добиться успеха.

Не для кого уже не секрет, что роль и влияние информационных технологий в этих процессах очень велики. Для конечного потребителя уже недостаточно информационных систем, которые могут удовлетворить только минимальные требования к автоматизации.

Пользователи ждут продуктов, в которых успешно сочетались бы удобство в получении необходимой информации с ее последующим анализом и обработкой, как на своем рабочем месте, так и вне его. *Интерфейс* должен быть понятен без длительного изучения различных инструкций, быстрое исправление проблемных ситуации в случае сбоя, а в идеале предвосхищение таких сбоев, - все это неотъемлемые характеристики рабочей среды успешной "цифровой" компании. Метаморфоза, которая позволит компаниям приобрести подобные черты, и называется цифровой трансформацией.

Решение проблемы метаморфозы многие видят в переходе к гибкому стилю работы не только в процессах разработки, внедрения и последующей эксплуатации ИТ-продуктов, но и в компании в целом. Scrum позволит организовать стабильные, предсказуемые поставки рабочего инкремента корпоративных информационных систем. Более того, *бизнес-процессы*, организованные *по* гибким принципам, позволят достичь максимальной загрузки сотрудников компании и найти ресурсы для стратегического или инновационного развития организации.

Кроме того, в качестве основных характеристик компании, прошедшей стадию цифровой трансформации, можно выделить следующие:

* *Фокус на "понимании" клиентов.* Результат от внедрения цифровой трансформации, как правило, начинается с плодов выделения основных категорий клиентов компании в тех или иных регионах и рыночных сегментах. Большинство компаний обращается к социальным сетям, чтобы понять предпочтения клиентов и их основные потребительские ценности.
* *Реклама через социальные медиа.* Компании учатся быть более эффективными в социальных сетях и прочих медиа. Для этих целей создаются сообщества, чтобы консультировать клиентов и формировать у них лояльность.
* *Акцент на измеримых результатах.* Предприятия используют инструментарий аналитических систем для более подробного изучения своих явных и потенциальных клиентов, изучения их поведенческих моделей и пр. Проводимые эксперименты позволяют динамически регулировать цены на продукцию в зависимости от спроса и других условий.
* *Трансформация точек контактов с клиентами.* Качество предоставляемых клиентам продуктов/сервисов/услуг будет повышаться за счет инноваций по использованию цифровых технологий. К примеру, стали создавать аккаунты в социальных сетях для приема жалоб. Этот подход позволяет оперативно решать вопросы, привлекать различные сообщества для решения разнообразных задач. Кроме того, компании, использующие мультиканальное обслуживание, испытывают затруднения при обеспечении комплексной поддержки каналов. Переход к омниканальности (единому каналу общения) поможет оптимизировать затраты и сконцентрировать компетенции и опыт на наиболее критичном виде услуг.
* *Преобразование операционных процессов.* Цифровая трансформация - один из наиболее интересных и сложных процессов преобразования компании, но при этом он, как правило, приводит к необходимости трансформации связанных с ним внутренних процессов.
* *Рост выручки.* Компании заинтересованы в цифровой трансформации и технологиях для того, чтобы повысить свою выручку, аккумулируют информацию о клиентах для обеспечения более точных продаж и персонализированного обслуживания. Таким образом, компания может в режиме реального времени отслеживать реакцию на свои продукты, что поможет в дальнейшем корректировать взаимоотношения с клиентами.

В цифровом бизнесе продуктивность каждого участника бизнес-процесса определяется качеством работы информационных систем. Цифровая трансформация компании требует сильного руководства - только оно может быть драйвером серьезных изменений. Требуется четкое понимание того, какие части компании и каким образом необходимо преобразовать. Цифровая трансформация во многом является экспериментом, плоды успешности которого позволят компании перейти на другой уровень зрелости. При этом абсолютно не важен профиль компании - *информационные технологии* в силах предоставить нужный инструмент с требуемым уровнем настройки, а если взять за основу основные постулаты и принципы Scrum, то эта настройка будет осуществляться наиболее подходящим для конкретных условий образом и в максимально сжатые сроки.

## 10.5 Современная критика Agile

Новички, познакомившиеся с Agile, уверены в том, что, применив гибкие методики к любому типу процесса и организации, можно добиться высоких результатов. Это не так. Это ложная *реклама*. Agile в целом и Scrum в частности требуют многих усилий и осмысленного применения для того, чтобы добиться результатов.

Если вам заявляют, что стоит пригласить внешнего Agile coach или Scrum-мастера и они смогут выполнить всю необходимую работу, то это тоже неправда. Сегодня на рынке есть множество компаний, которые не программировали и не имеют опыта руководства людьми. Подобное "обучение" с помощью таких специалистов не принесет значимых плодов. Люди, занимающиеся пропагандированием и внедрением гибких практик, должны уметь программировать, анализировать, руководить, тестировать и обладать в этих навыках достаточно высоким уровнем компетенции. Если подобного опыта у "всезнающего" консультанта нет, то как с ним можно обсуждать специфические ситуации, связанные с продуктом или Scrum? Консультант *по* внедрению гибких процессов, который получил высшее образование вчера и не написал ни строчки кода, приносит меньше пользы, чем члены команды. Нельзя обучать тому, чего ты ни разу не делал. И "делать" означает "делать многократно и постоянно", а не иметь в послужном списке один *пилотный проект*.

При этом найти причины неудач очень просто. Их и искать не надо - они, как правило, лежат на поверхности. Принципы Scrum предписывают, как нужно действовать, но в них не описан *алгоритм* работы. Эта концептуальность критикуется, но, если бы гибкие методологии претендовали на роль конкретных техник, тогда их использование не нашло бы такого широкого распространения. *Алгоритм* работы должен быть определен в конкретной организации с учетом условий ее деятельности и его поэтапной корректировки в целях совершенствования. Гибкие методологии не должны навязываться. Они должны быть добровольно приняты и одобрены коллективом разработчиков, которые заинтересованы в его применении. Попытка навязать его применение приведет к подавлению роли команды и последующему отторжению.

Это лишь малых перечень проблем, которые приводят к критике культуры Agile. Компаниям, которые пришли к необходимости внедрения и использования этой методологии, рано или поздно придется эти проблемы решать. Игнорирование и замалчивание приведет к "тихим" или явным революциям. Подытоживая, выделим следующие постулаты:

* *Догматичность.* Agile предполагает адаптивный, гибкий и креативный framework, который рождает фанатиков, "повернутых" на его использовании. Без профессионального запала сложно постоянно совершенствовать любую методологию, но чрезмерное увлечение ею отпугивает потенциальных пользователей.
* *Сектантство.* Не стоит замыкаться на Scrum, следует обращать внимание и на другие подходы к разработке.
* *Доминирование консультантов.* Преобладание бывших "гуру", которые стали теоретиками и оторваны от реальной практики работы.
* *Люди.* Расчет на жесткую самодисциплину. Психологическое, эмоциональное, профессиональное выгорание членов команды. Зависимость от слаженной командной работы. Зависимость от ресурсов команды и отдельных "лидерских" персоналий. Необходимость постоянного участия в команде компетентного руководства, заинтересованного в достижении высоких результатов.

*Agile - очень требовательная культура. Меняться и гнуться предстоит всем.*

Когда *менеджмент* приходит к необходимости внедрения гибких методологий, то необходимо осознать, что быть гибкими предстоит всем. Если команде в рамках ее полномочий поручено выполнять ряд действий, принимать решения и нести за них ответственность, то это должно быть действительно так. Если *команда* не принимает решений, то это разрушает ее изнутри. *Команда* должна принять решение о том, что и как она будет развивать и что для этого необходимо. Если решения принимаются не командой, а спускаются "сверху", то в конечном итоге вы получите не коллектив мотивированных и самостоятельных разработчиков, а безынициативную группу послушных сотрудников.

Много прекрасных команд на первых этапах своей деятельности терпят неудачи. Много команд успешны без Scrum. Принимая решение об использовании Scrum, нужно взвесить все за и против и следовать принятому решению.

## 10.6 Что дальше

Вряд ли можно рассчитывать на то, что гибкие методологии будут применяться в идеальной среде, свободной от вмешательства окружающего реального мира. Неидеальность реального мира - это реальный факт. Этот факт необходимо учитывать всегда. Таким образом, очевидна необходимость сопровождения и развития отдельно взятого Scrum конкретной компании. Scrum - гибкая процессная методология, или, *по*-другому, - управленческая дисциплина. И как любую дисциплину, цель которой - приносить предсказуемый результат, ее необходимо контролировать. Но трудоемкость менеджерских активностей можно уменьшить и с самого начала "воспитывать" в сотрудниках, входящих в Scrum-команду, лояльность к изменениям, приверженность к обучению и совершенствованию. Явным результатом этого будет достижение более высокого уровня процессной зрелости и профессиональной культуры.

Путем экспериментирования Scrum-команды могут повести организацию к освоению все более и более эффективных способов работы.

**Питання до теми:**

# СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Scrum. Революционный метод управления проектами / Джефф Сазерленд; пер. с англ. М. Гескиной. – 4-е изд. – М.: Манн, Иванов и Фербер, 2019. – 272 с.
2. Stern, Terra Vanzant. Lean and agile project management: how to make any project better, faster, and more cost effective. CRC Press, 2017.
3. Raharjo, Teguh, and Betty Purwandari. "Agile Project Management Challenges and Mapping Solutions: A Systematic Literature Review." Proceedings of the 3rd International Conference on Software Engineering and Information Management. 2020.
4. Schwaber, Ken; Sutherland, Jeff (2017), The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game
5. Кон Майк. Scrum. Гибкая разработка ПО / Майк Кон. – Изд-во Вильямс. – 2015. – 576 с.
6. Schwaber Ken. Agile Project Management with Scrum. Microsoft Press, 2004 - Computers - 163 pages.
7. Jim Highsmith. Agile Project Management: Creating Innovative Products - 2nd edition. Publisher: Addison-Wesley Longman, Inc.- 2010.
8. Pichler, Roman; Stefan Schulze (2005). "Book Reviews: Agile Project Management: Creating Innovative Products by Jim Highsmith, and Agile Project Management with Scrum by Ken Schwaber". Journal of Product Innovation Management.
9. Архипенков С. Лекции по управлению программными проектами / С. Архипенков. – Москва, 2009. – 127 с.
10. Wysocki, Robert K. Effective project management: traditional, agile, extreme. John Wiley & Sons, 2011.
11. Stellman, Andrew, and Jennifer Greene. Learning agile: Understanding scrum, XP, lean, and kanban. " O'Reilly Media, Inc.", 2014.
12. Руководство к своду знаний по управлению проектами (PMBOK). – Project Management Institute, Four Campus Boulevard, Newtown Square, Pennsylvania 19073-3299 USA / США: пер. с англ. – 3-е изд. – 2004. – 389 с.
13. Морозов В.В. Формування, управління та розвиток команди проекту (поведінкові компетенції): навч. посіб. / В.В. Морозов, А.М. Чередніченко, Т.І. Шпильова; за ред. В.В. Морозова. – К.: Таксон, 2009. – 464 с.
14. Иванов В. Изучение MS Project и методик PMI методом сквозного примера / В. Иванов // Материалы Internet. –www.MicrosoftProject.ru
15. PMBOK® Guide and Standards [Електронний ресурс]. – Режими доступу: https://www.pmi.org/pmbok-guide-standards
16. The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) [Електронний ресурс]. – Режими доступу: https://www.computer.org/web/swebok
17. The home of Scrum [Електронний ресурс]. – Режими доступу: https://www.scrum.org
18. IT Project Management [Електронний ресурс]. – Режими доступу:https://www.smartsheet.com/content-center/best-practices/project-management/projectmanagement-guide/project-management-IT
19. Strength, Weakness, Opportunity, and Threat (SWOT) Analysis [Електронний ресурс]. – Режими доступу: https://www.investopedia.com/terms/s/swot.asp
20. How to Do a SWOT Analysis for Your Small Business (with Examples) [Електронний ресурс]. – Режими доступу: https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis
21. A Full Overview of Business Process Management [Електронний ресурс]. – Режими доступу: https://kissflow.com/bpm/business-process-management-overview/
22. Plan-Do-Check-Act (PDCA). Continually Improving, in a Methodical Way [Електронний ресурс]. – Режими доступу: https://www.mindtools.com/pages/article/newPPM\_89.htm