**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ТАВРІЙСЬКИЙ ДЕРЖАВНИЙ АГРОТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ДМИТРА МОТОРНОГО**

**Факультет економіки та бізнесу**

**Кафедра маркетингу**

**ГНУЧКЕ УПРАВЛІННЯ ПРОЕКТАМИ**

**Agile project management**

**Частина 1**

КОНСПЕКТ ЛЕКЦІЙ

для здобувачів освітнього рівня «Магістр»

за спеціальністю ***075 «Маркетинг»***

**Мелітополь, 2022р.**

**Гнучке управління проектами.** Конспект лекцій для здобувачів освітнього рівня «Магістр» за спеціальністю 075 «Маркетинг». – Мелітополь, ТДАТУ, 2022 р. – 176 с.

**Розробники:**

**Мінц О. Ю.,** д-р екон. наук, доцент, ПДТУ

**Укладачі:**

**Кюрчев В.М.,** д.т.н., професор кафедри експлуатації та технічного сервісу машин, ТДАТУ

**Куліш Т.В.** к.е.н., доцент кафедри маркетингу, ТДАТУ

**Сокіл Я.С.** к.е.н., доцент кафедри маркетингу, ТДАТУ

**Рецензенти:**

**Žaneta Simanavičienė** prof.habil.dr.,Mykolas Romeris University

**Хаджинова О. В.** д.е.н., професор, директор ННІЕМ ПДТУ

Розглянуто та рекомендовано до друку на засіданні кафедри маркетингу

Протокол № *10* від « *09*» травня 2022 року

Завідувач кафедри «Маркетинг»

д.е.н, професор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дар’я ЛЕГЕЗА

Схвалено методичною комісією факультету економіки та бізнесу для здобувачів освітнього рівня «Магістр» за спеціальності 075 «Маркетинг»

Протокол № *7* від «*20*»  *червня* 2022 року

Голова, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_ Анна КОСТЯКОВА

Схвалено Навчально-методичною радою Таврійського державного агротехнологічного університету імені Дмитра Моторного для здобувачів освітнього рівня «Магістр» за спеціальності 075 «Маркетинг»

Протокол № *1* від «*26*»  *серпня* 2022 року

Голова, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_ Олександр ЛОМЕЙКО

© Мінц О.Ю., 2022 рік

**ЗМІСТ**

[ВСТУП 7](#_heading=h.30j0zll)

[Лекція 1: ВСТУП ДО AGILE 8](#_heading=h.1fob9te)

[1.1 Опис курсу 8](#_heading=h.3znysh7)

[1.2 Актуальність Agile 8](#_heading=h.2et92p0)

[1.3 Що таке Agile? 10](#_heading=h.3dy6vkm)

[1.4 Трохи з історії 14](#_heading=h.2s8eyo1)

[1.5 Про авторів Agile 16](#_heading=h.17dp8vu)

[1.6 Як розвивалася Agile 20](#_heading=h.3rdcrjn)

[1.7 Agile Manifesto 27](#_heading=h.1ksv4uv)

[1.8 Застосування в різних предметних областях 28](#_heading=h.2jxsxqh)

[1.9 Короткі висновки 30](#_heading=h.3j2qqm3)

[Лекція 2: ПРОЦЕССНЫЙ ПІДХІД ЯК КОНКУРЕНТНА ПЕРЕВАГА 31](#_heading=h.1y810tw)

[2.1. Процесний підхід 31](#_heading=h.4i7ojhp)

[2.2. Значущість процессного офісу у впровадженні процессного підходу. Можливі варіанти 39](#_heading=h.1ci93xb)

[2.3. Рух у напрямі гнучкості. Пілотні процеси і управління змінами 43](#_heading=h.qsh70q)

[2.4. Регламент як основа процессного підходу 46](#_heading=h.3as4poj)

[2.5. Короткі висновки 48](#_heading=h.1pxezwc)

[Лекція 3: ПЕРЕДУМОВИ ВИНИКНЕННЯ AGILE 50](#_heading=h.49x2ik5)

[3.1. Стан області процесів розробки програмного забезпечення 50](#_heading=h.2p2csry)

[3.2. Порівняння каскадного/ітераційного/Agile процесів 54](#_heading=h.147n2zr)

[3.3 Ефективна пігулка від хвороб? 59](#_heading=h.3o7alnk)

[3.4 Для кого підходить, а для кого ні? 60](#_heading=h.23ckvvd)

[3.5. Короткі висновки 63](#_heading=h.ihv636)

[Лекція 4: ФІЛОСОФІЯ РОБОЧОГО ПРОЦЕСУ 65](#_heading=h.32hioqz)

[4.1. Типи Agile- методологій і їх поширеність 65](#_heading=h.1hmsyys)

[4.2. Scrum - гнучкий управлінський процес 69](#_heading=h.2grqrue)

[4.3. Як виховувати співробітників? 72](#_heading=h.vx1227)

[4.4. Як управляти опором? 75](#_heading=h.3fwokq0)

[4.5. Чим треба управляти в Scrum 78](#_heading=h.4f1mdlm)

[4.6. Значущість дотримання процесу 81](#_heading=h.3tbugp1)

[4.7. Інженерні практики 82](#_heading=h.28h4qwu)

[4.8. Короткі висновки 84](#_heading=h.37m2jsg)

[Лекція 5: РОЛІ SCRUM 86](#_heading=h.1mrcu09)

[5.1 Команда 86](#_heading=h.46r0co2)

[5.2 Етапи командотворення 88](#_heading=h.111kx3o)

[5.3 Розробник 91](#_heading=h.4k668n3)

[5.4 Scrum-майстер 93](#_heading=h.2zbgiuw)

[5.5 Власник продукту 95](#_heading=h.1egqt2p)

[5.6 Самоорганізація членів команди 96](#_heading=h.3ygebqi)

[5.7 Інші члени команди 98](#_heading=h.2dlolyb)

[5.8 Колектив, що самоорганізується 99](#_heading=h.sqyw64)

[5.9 Короткі висновки 101](#_heading=h.1rvwp1q)

[Лекція 6: ПЛАНУВАННЯ 104](#_heading=h.4bvk7pj)

[6.1 Принцип швидкого планування 104](#_heading=h.2r0uhxc)

[6.2 Поетапне уточнення планів 106](#_heading=h.1664s55)

[6.3 Техніка Planning Poker 108](#_heading=h.3q5sasy)

[6.4 Діаграма згорання робіт 111](#_heading=h.kgcv8k)

[6.5. Короткі висновки 113](#_heading=h.1jlao46)

[Лекція 7: ЕТАПИ І ЗАХОДИ SCRUM 115](#_heading=h.43ky6rz)

[7.1. Sprint 115](#_heading=h.2iq8gzs)

[7.2 Щоденні зустрічі (daily) 119](#_heading=h.1x0gk37)

[7.3 Грумінг бізнес-завдань 122](#_heading=h.2w5ecyt)

[7.4. Грумінг технічних завдань 125](#_heading=h.1baon6m)

[7.5 Огляд Спринта 126](#_heading=h.3vac5uf)

[7.6 Ретроспектива 128](#_heading=h.2afmg28)

[7.7 Короткі висновки 131](#_heading=h.pkwqa1)

[Лекція 8: АТРИБУТИ SCRUM 133](#_heading=h.39kk8xu)

[8.1 Story mapping 133](#_heading=h.1opuj5n)

[8.2. Користувацькі історії (User story) 136](#_heading=h.2nusc19)

[8.3. Визначення пріоритетів користувачів 139](#_heading=h.1302m92)

[8.4. Дошка завдань 142](#_heading=h.2250f4o)

[8.5 Беклог продукту 144](#_heading=h.319y80a)

[8.6 Беклог спринту 147](#_heading=h.1gf8i83)

[8.7 Інкремент продукту 148](#_heading=h.40ew0vw)

[8.8. Принцип прототипування 149](#_heading=h.2fk6b3p)

[8.9. Короткі висновки 150](#_heading=h.upglbi)

[Лекція 9: ОЦІНКА 152](#_heading=h.3ep43zb)

[9.1 PERT - оцінка термінів 152](#_heading=h.1tuee74)

[9.2 Перехід від оцінки до зобов'язань 155](#_heading=h.2szc72q)

[9.3 Збалансована система показників Scrum- команди 157](#_heading=h.184mhaj)

[9.4 Напрацьована статистика результатів - фундамент прогнозування і перемог 159](#_heading=h.3s49zyc)

[9.5 Короткі висновки 161](#_heading=h.279ka65)

[Лекція 10: ПІДСУМКИ І ПЕРСПЕКТИВИ 162](#_heading=h.meukdy)

[10.1 Співіснування з альтернативними процесами послідовної розробки програмного забезпечення 162](#_heading=h.36ei31r)

[10.2 Забезпечення відповідності кращим практикам і стандартам 164](#_heading=h.1ljsd9k)

[10.3 Використання Lean- методології в Scrum- процесі 167](#_heading=h.45jfvxd)

[10.4 Продуктивність Scrum для цифрової трансформації 169](#_heading=h.2koq656)

[10.5 Сучасна критика Agile 171](#_heading=h.zu0gcz)

[10.6 Що далі 173](#_heading=h.3jtnz0s)

[СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ 175](#_heading=h.1yyy98l)

# ВСТУП

Agile – це не просто модне віяння, яке є послідовністю певних кроків, а дієвий спосіб управління людським ресурсом, в якому враховані недоліки його попередників і закладені достоїнства, що дозволяють власникам організацій розраховувати на досягнення їх очікувань від застосування програмних продуктів. Гнучкі процеси - це ступінь в розвитку підходів до управління проектами, від яких залежатиме кожна сучасна успішна компанія.

Знання і уміння оперувати поняттями і атрибутами Agile дозволить менеджерам і профільним фахівцям бути ефективним у світі створення і реалізації проектів будь-якого об'єму і міри складності.

Курс адресований широкому колу фахівців, які зацікавлені в застосуванні гнучких методологій, але акцент зроблений на менеджерів і професіоналів, задіяних у сфері розробки проектів інформаційних систем(тобто в якості прикладів приведена реалізація проектів розробки програмного забезпечення). В той же час курс не претендує на звання "єдиного" джерела знань по Agile.

# Лекція 1: ВСТУП ДО AGILE

План

1.1 Опис курсу

1.2 Актуальність Agile

1.3 Що таке Agile?

1.4 Трохи з історії

1.5 Про авторів Agile

1.6 Як розвивалася Agile

1.7 Agile Manifesto

1.8 Застосування в різних предметних областях

1.9 Короткий висновок

## 1.1 Опис курсу

У цьому курсі лекцій йдеться про те, що дозволить досягти поставлених показників тим організаціям і співробітникам, які зацікавлені зрештою, а точніше, в його відтворюваності в поставлених умовах. Адже саме здатність дотримуватися погодженого плану у рамках наявних ресурсів відрізняє компанії-одноденки від тих підприємств, які зацікавлені в стабільному і тривалому існуванні у своєму сегменті бізнесу. Ми поговоримо про спеціалізовану процессной методологію Agile.

На сьогодні видана достатня кількість літератури по Agile, але більшість авторів своєю цільовою аудиторією обрали майбутніх адептів гнучких процесів, які здатні з головою зануритися в їх деталі. Більшість курсів рясніють технічними і технологічними деталями.

Цей курс лекцій переслідує мету об'єктивно освітити місце і суть Agile у сучасному світі. Акцент зроблений на доступність і наочність викладу матеріалу не стільки для майбутніх учасників гнучких процесів, але значною мірою на тих, кому необхідно розуміти, як бути ефективним, використовуючи Agile у своїх професійних цілях.

## 1.2 Актуальність Agile

Актуальність Agile обумовлена безліччю різних технологічних, організаційних і соціальних чинників.

Поступово відбувається глобальне зміщення парадигми управлінського мислення з наслідування ваговитих і тривалих стратегій на ситуаційно-тактичну поведінку. Саме такий тип організації і управління робіт дозволить успішно лавірувати в сучасних динамічних реаліях навколишнього світу.

Розробка тривалого і грунтовного плану може так і не закінчитися. В процесі його створення станеться така кількість змін, облік і відображення яких займе тривалий період часу.

Саме для подолання криз, пов'язаних з дефіцитом думок, ідей, часу, треба використати гнучкі процеси, суть яких полягає в адаптації до поточних умов, акценті на роботу з людьми і сприйнятті змін як неминучого блага, а не страшної проблеми.

Менеджери і фахівці, які вивчили Agile і використовують її у своїй професійній діяльності, безумовно, є компетентнішими і ціннішими кадрами для будь-якої організації.

Чим Agile може бути корисна для вашої компанії? Не варто думати, що Agile є вузькоспеціалізованою методологією, мета якої зводиться до того, щоб створювати потрібні і ефективні програмні продукти. Безумовно, саме так і замислювалося, але після того, як результати її застосування виправдали усі можливі очікування, багато відгалужень Agile було адаптовано під інші види діяльності, і на даний момент можна констатувати, що гнучкі процессные методології є певного виду управлінський framework.

У зв'язку з цим ми сподіваємося, що кожен менеджер, який приступив до вивчення цього курсу, вивчаючи його, не просто буде намагатися сприйняти викладену тут інформацію, але зможе застосувати прийоми і методи, які визнає оптимальними для умов своєї компанії.

Окрім теорії ви знайдете безліч конкретних прийомів і практичних методів, що довели свою ефективність в операційній і проектній діяльності.

У сфері інформаційних технологій питання, пов'язані з процесами розробки програмних продуктів, - одні з самих спірних і складних. Починаючи з моменту широкого поширення комп'ютерних технологій, в 60-і роки XX століття, фахівці намагаються відшукати універсальний засіб, що іменується "срібна куля", за допомогою якого вдасться вирішити завдання створення складних програмних продуктів швидко, дешево і якісно. Його шукають досі і, мабуть, шукатимуть ще дуже довго. Скажіть, а в інших компаніях, де відбувається створення і реалізація проектної діяльності, не теже мети і завдання?

Процессные методології розробки інформаційних систем змінюють один одного, але при цьому зберігається загальна закономірність, пов'язана з фокусуванням на оптимізації самого процесу і максимізацією кінцевого результату.

На даний момент склалися два основні протидіючі табори.

Адепти першого – це консерватори, що вважають за краще грунтовніші, "важчі" методології, в яких чітко і однозначно визначена структура і послідовність виконуваних етапів робіт. Типовим прикладом таких методологій є водоспадна модель розробки інформаційних систем.

Прибічники другого – це новатори, що виділилися з популярного раніше напряму ітераційної розробки, які структурі і послідовності робіт вважають за краще ефективна професійна взаємодія і гнучкість у відповідь на мінливість зовнішнього світу. Саме про такі методології, які прийнято називати гнучкими, і піде мова в нашому курсі.

Усе сімейство гнучких методологій на сьогодні знаходяться у безперервному розвитку:

* модернізуються і доповнюються існуючі підходи;
* розробляються нові методології;
* розширюється сфера їх використання.

## 1.3 Що таке Agile?

Процессный підхід, як спосіб організації діяльності, розбурхує уми і викликає безліч різних питань вже досить давно.

Коли мова заходить про "правильне" і ефективне управління діяльністю, безліч практиків приводять різні аргументи на користь того або іншого підходу, який, на їх думку, є оптимальним способом організації роботи.

Проходить час, міняються внутрішні і зовнішні умови, різні чинники, і ринку потрібні нові інструменти, які змогли б дозволити пристосуватися компаніям до дійсності, що змінюється, і при цьому демонструвати усе більш стійкий результат в порівнянні з їх попередниками. Незмінною залишається тільки наявність "людського ресурсу" з усіма його достоїнствами і недоліками. Спроба відноситися до співробітника, як до самозарядного автомата, який повинен в стислі терміни вистрілювати результатом і бути готовим до використання у будь-яких, навіть найжорсткіших умовах, поступово відживає своє.

Значущість зміни подібної парадигми ознаменувалася тим, що в літературі замість варварського терміну "людський ресурс" стали використовувати м'якший - "людський капітал".

Але це не повинно вводити в оману тих, хто ставить основною задачею творче начало і заперечує всяку значущість дисципліни. Співробітник має бути не гвинтиком у відлагодженому механізмі(не так вже багато таких відлагоджених механізмів, як показує практика), а грати роль важливої шестерінки, яка приводить в рух суміжні процеси, що забезпечують життя організації, кожного з професіоналів або тих, хто прагне ним стати.

На сьогодні процессные методології, орієнтовані не лише на кінцевий результат, але і на грамотне управління співробітником як творчою бойовою одиницею, що самовизначається, показують найбільший успіх.

Приміром, можна привести Lean, Six Sigma, ітераційний підхід до розробки і реалізації проектів, які є досить складними до впровадження, подальшої адаптації і розвитку, але їх постійне застосування дозволяє компаніям вирватися в лідери своїх сегментів ринку і зайняти на них стійке положення.

Процес є завжди. Навіть у тому випадку, коли йдеться про стихійне виконання різнорідних етапів, ми все одно говоримо про процес з низьким рівнем зрілості, результат якого у більшості випадків буде непередбачуваним.

Якщо керівництво організації не влаштовує ситуація, в якій воно не може впливати на досягнення результату, то процесами зазвичай прагнуть займатися. Не просто займатися, а робити це ефективно.

Головним помічником в досягненні оптимального управління діяльністю останнім часом все більше і більше виступають інформаційні технології, що привносять певну міру порядку в навколишній організації хаос. Відлагоджена інформаційна система або набір систем допомагають компаніям оптимізувати і удосконалити їх процеси до такої міри ефективності, коли можливий негативний вплив людського чинника зведений до мінімуму.

Головне завдання, яке необхідно вирішувати кожному підприємству, що прийшло до необхідності застосування ІС, - це те, що процесами їх впровадження, розробки і розвитку потрібно також управляти.

У цій області склалося декілька процессных методологій, кожна з яких має певний багаж перемог і поразок.

Перший і основний на сьогодні - послідовний підхід, що полягає в чіткому наслідуванні запланованої структури робіт, суть яких визначена заздалегідь.

Другий, найбільш перспективний і амбітний, - гнучкий підхід, що є послідовником ітераційного підходу, який не заперечує першого, але мета якого полягає в зниженні складності необхідної структури етапів при реалізації проекту, підвищенні прозорості виконуваного набору робіт і використанні творчої ініціативи людей як засоби підвищення результативності. Такими методологіями є процеси сімейства Agile.

Agile - процессная методологія, що складається з серії підходів(Scrum, Kanban і ін.) до розробки програмного забезпечення або інформаційного продукту, орієнтована на використання ітеративної розробки, динамічне формування вимог і забезпечення їх реалізації, в результаті постійної взаємодії усередині робітників груп, що складаються з фахівців різного профілю, що самоорганизующихся.

Вище приведено основне і загальноприйняте визначення Agile, але, як це часто буває в мейнстрімних напрямах, яким Agile стала на сьогодні, єдиного визначення не існує і є безліч альтернативних тлумачень цього терміну.

Приведемо ще декілька специфічних тлумачень, які доповнять ваше уявлення про його суть.

Agile - це framework для розробки і підтримки функціонально складних продуктів.

Agile - це framework, у рамках якого можливо вирішувати складні комплексні завдання і в той же час продуктивно і креативно розробляти продукти задовільної для замовника якості.

Agile - це framework, використовуваний для комплексного управління процесами.

Agile - це процессная методологія, що активно розвивається(з початку 90 - x), мета якої полягає в оптимальній організації робочого наукомісткого процесу, створенні ефективних інструментів моніторингу і контролю за його виконавцями і вироблюваними результатами.

Як ви помітили, приводячи визначення, ми використали поняття framework, яке служить для позначення професійного підходу до рішення всіляких завдань, що є ступенем до досягнення поставленого результату.

У сучасному світі це поняття стало поширюватися не лише на професійну діяльність, але і на усі проблеми, які стоять перед фахівцем. У нашому курсі ми доповнимо і розширимо це поняття, оскільки, говорячи про Agile, поступово стає зрозуміло, що це не просто підхід до організації процесів, а ефективна життєво-професійна філософія.

На завершення скажемо про те, що хоч Agile і народилася як спосіб організації спеціалізованих процесів в області інформаційних технологій, але при цьому її з успіхом використовують у багатьох областях, де:

* вплив людського чинника на досягнення кінцевого результату дуже високо;
* продукт, який виходить в результаті діяльності, є складним і комплексним артефактом;
* в роботі над продуктом задіяна команда професіоналів, об'єднана спільними цілями і завданнями.

У твердженні, що Agile - це гнучкий управлінський framework, немає лукавства. Це дійсно так, що підтверджено на практиці. Багато компаній, основними процесами яких є маркетинг, консалтинг і ін., вже успішно "живуть" по Agile.

У нашому курсі ми приділимо основну увагу найпопулярнішому і авторитетнішому відгалуженню Agile - Scrum. Засадничі принципи і артефакти, про які ми вестимемо мову, використовуються в усіх відгалуженнях сімейства Agile.

## 1.4 Трохи з історії

У лютому 2001 року на гірськолижному курорті в гірському хребті Васатч, штат Юта(США), 17 чоловік, шанованих фахівців в області розробки програмного забезпечення, зустрілися поспілкуватися на загальні теми(біль, ідеї, надії), покататися на лижах і, звичайно ж, поїсти.

Саме так, якщо вірити авторитетним джерелам, почалася офіційна історія найпопулярнішого нащадка сімейства Agile - Scrum.

До того моменту виникла необхідність створення дієвої і загальнозастосовної альтернативи ваговитим методам розробки програмного забезпечення. Перші паростки, що іменуються екстремальним програмуванням, адаптивна розробка програмного забезпечення, кристал, FDD, прагматичне програмування вже почали з'являтися на ринку, але для великого поширення були занадто екзотичними(посадити за один комп'ютер двох розробників в один і той же час було просто запаморочливим рішенням для початку 2000-х років).

Починаючи з кінця 80-х ринок інформаційних технологій, як в старенькій Європі, так і в Сполучених Штатах Америки, почав готуватися(а вірніше, його почали готувати саме ті, хто в подальшому стануть авторами Agile) до прориву.

Мирна революція назрівала. Процеси розробки програмного забезпечення повинні були перестати бути долею невеликих псевдо- чи навколонаукових колективів або одинаків, які були гиками свого часу. Потрібний був поштовх, який дозволить вивести розвиток програмного забезпечення на новий виток, ініціювати наступну ітерацію в розвитку ринку програмних продуктів.

Розвиток електронно-обчислювальної техніки повинен був стати додатковим каталізатором, що підштовхнув до буму розвитку комп'ютерно-обчислювальної техніки і процесів розробки програмних продуктів.

Поворот "лицем" до бізнес-замовника, який раніше був далекий від того, що відбувається, і його активне знайомство з програмними продуктами як засобом оптимізації і підвищення ефективності його бізнесу продемонструвало усю потужність інформаційних технологій. Піджак з краваткою, зацікавлений тільки зрештою, який сидить декількома десятками поверхів вище від відділу інформаційних технологій, повинен стати залученим в поточні процеси, зробитися їх активним учасником. Звучить дуже добре і перспективно, але так просто, з наскоку, відкрити усі двері одним рухом ноги ніхто не дозволив би. Аудиторію зацікавлених сторін необхідно було ментально підготувати.

І ця підготовка почалася здалека, спочатку з найбільш певних ділянок людської діяльності, в якій не потрібний широкий і дуже швидкий політ творчості, - бухгалтерський і суміжний з ним типи обліку.

Поступово інформаційні технології стали відвойовувати все більше і більше простору, проникати в місткі і складні види діяльності, займаючи своє місце серед інших видів діяльності.

Експоненціальне зростання темпів розвитку інформаційних технологій привело до того, що терміни на розробку інформаційних систем стали постійно скорочуватися.

Усе менше і менше часу стали відводити на довге і грунтовне планування, подальше проектування, написання повного пакету необхідній документації і подальше усеосяжне тестування. Існуючі процеси розробки програмного забезпечення, а вірніше - їх методологія, повинні були адаптуватися до дійсності, що настає.

Якщо зрушення не сталося б, то усі програмні продукти, що розробляються, застарівали б ще до того, як були повністю готові до комерційного використання. Рухатися у напрямі інформатизації було необхідно, і цей рух має бути продуманим, але не занадто довгим.

Саме це і сталося з гнучкими методологіями. Еволюційно, перетворюючи ітераційний принцип розробки інформаційних систем, спочатку повністю залучаючи до процесів тільки фахівців, розробників, тестувальників і ін., а потім і суміжні спеціальності, ринок готувався до появи методології, в якій одним з головних постулатів буде універсальність, але зі збереженням професійної індивідуалізації її учасників.

Agile з'явилася і почала набирати оберти на тлі швидкорослого ускладнення технологій і загальної інформатизації різних бізнес-напрямів діяльності.

Як говорять усі ті ж авторитетні західні джерела, на тій доленосній зустрічі в "штаті мормонів" з'явився Agile Manifesto - біблія гнучких процесів розробки програмного забезпечення, про яке ми поговоримо трохи пізніше.

## 1.5 Про авторів Agile

До вдосконалення Agile доклали руку і уми багато стовпів напряму розробки програмних продуктів області інформаційних технологій 90-х років. Багато професіоналів не просто брали участь в розвитку цього напряму діяльності, а присвятили цьому свої професійні кар'єри.

Англомовні джерела серед інших виділяють наступних авторів:

 Верд Кенінгхем. Засновник однойменної компанії, є директором в області досліджень і розробок компанії "Вэйт Програмне забезпечення". Перед цим займав пост головного інженера в компанії "Науково-дослідна комп'ютерна лабораторія Тектроникс".

Вэрд добре відомий своїм вкладом в розробку однієї з практик об'єктно-орієнтованого програмування, що називається екстремальним програмуванням, і створеною ним технологією Wiki.

Джим Хенгсміт.

Головний розробник методу "Адаптивної розробки", що входить в сімейство Agile, і автор книги з однойменною назвою. Він одним з перших начал публічно пропагувати використання Agile на міжнародних конференціях з розробки ПО. Джим в співавторстві з Мартіном Фаулером створив статтю-дослідження "The Agile" в популярному журналі "Розробка програмного забезпечення". Джим і Алистер Кокберн працювали над створенням багатьох відгалужень Agile- сімейства, які були опубліковані у видавництві Addison, - Wesley у вигляді серії книг з Agile.

Ендрю Хант. 

Один з авторів книги "The Pragmatic Programmers" і співавтор бестселера "Прагматичний програміст: шлях від новачка до Майстра". Між написанням книг і статей, виголошуванням надихаючих промов і грою на піаніно Ендрю знаходить час для свого консультаційного бізнесу, що спеціалізується на Agile. Енді створює програмне забезпечення професійно з 80-х років і встиг зануритися в такі галузі, як телекомунікації, банківська і фінансова сфери, утилізація, медицина, графічне мистецтво і, звичайно ж, створення сервісів в Інтернеті. Енді задіяний в компанії Raleigh NC і зі своїм співавтором Дэйвом Томасом займається поширенням найбільш ефективних і добре себе методів розробки програмного забезпечення, що зарекомендували, в США. Він також є президентом RTP, відділення незалежної асоціації комп'ютерних консультантів і членом ACM і IEEE.

Рон Джефріс. 

Засновник порталу XProgramming.com, консультант компанії Object Mentor і автор(спільно з Ганною Эндерсон і Чотом Хендриксоном) Extreme Programming, Рон був першим тренером за методологією Extreme Programming.

Джон Керн

Джон живить пристрасть до надання успішної допомоги клієнтам в досягненні ними цінності від використання розробленого ним програмного забезпечення. Його різноманітна кар'єра розтягнулася з дослідницької діяльності по створенню двигуна для реактивних літаків до розробки літального симулятора, він став євангелістом(прибічником-розповсюджувачем) об'єктно-орієнтованого підходу c 90-х років, розпочавши з використання C++ і потім приступивши до Java. Він першим опублікував опис використання ітераційної розробки при створенні Lotus Notes 4.5 і 5.0. Він був мотивований мантрою, яку вимовляв його друг Пітер Код : "Часто, відчутно, що дає результат". Він отримав ступінь доктора наук, заснував власну компанію(Lightship, Inc.). У 1999 році він приєднався до Пітера Коду в його стартапе "TogetherSoft", де він створив професійну групу консультантів і працював над розробкою програмних продуктів. Джон - співавтор реалізації Java, працював разом з Пітером і Джефом ДеЛука над FDD. Джон постійно зайнятий пошуком найкращих методів досягнення поставлених цілей з точки зору використовуваної методології роботи і використовуваних технологій. Ви можете знайти його блог за адресою <http://blogs.compuware.com/cs/blogs/jkern/>.

Брайан Марик

Консультант по розробці і тестуванню програмного забезпечення. При розробці Agile виступав як представник співтовариства тестування, який підкреслював необхідність тестування і документування, особливо враховуючи складність, що підвищується, і мінливість реалізації комп'ютерних систем.

Роберт С. Мартін

Фахівець в області розробки програмного забезпечення з 1970 року. Президент і засновник Object Mentor Inc. - компанії, яка займається глибокою експертизою за допомогою гнучких методологій, розробці програмного забезпечення, навчанні і розробки систем і сервісів для великих корпорацій. У 1995 році написав бестселер "Проектування додатків на Object Oriented C++ із застосуванням методу Золить". У 1997 році був головним редактором книги "Шаблони проектування мов програмування". У 1999 році був редактором книги "More C++ Gems". Співавтор книги "XP in Practice", виданої в 2001 році. Безпосередньо працював над книгою "Принципи, шаблони і практика розробки програмного забезпечення по Agile", виданою в 2002 році. З 1996 по 1999 рік був головним редактором звітів по З++. Опублікував безліч статей в різних журналах і виступає постійним доповідачем на міжнародних конференціях.

Майк Бідл

Засновник і виконавчий директор компанії e - Architects Inc. Один з самих ранніх євангелістів філософії Agile. Майк спеціалізується на створенні масштабних інформаційних систем, в роботу над якими, як правило, залучається безліч розподілених команд.

 Арі Ван Беннекум

Розробник, консультант, практикуючий тренер. Один з найактивніших пропагандистів Agile- процесів. Внутрішній двигун Scrum - cообщества.

Алістер Кокберн.

Засновник компанії Humans and Technology, автор багатьох сучасних відгалужень в області ИТ, один з батьків-засновників ядра методології Agile. Спонсорував первинний розвиток і просування Agile. Досі є практикуючим професіоналом, діяльність якого спрямована на створення техноекосистем.

 Мартін Фаулер.

Виконавчий директор по дослідженнях великої компанії в області розробки програмного забезпечення і консалтингу. Автор книг з аналітичних шаблонів, використання Uml, рефакторингу, планування і інших видів активностей у напрямі Agile. Попри те, що Мартін Фаулер безпосередньо не брав участь в розробці і просуванні Agile, професіонали відносять його до числа тих, хто вніс в її розвиток найбільший вклад.

Кен Швайбер

Президент компанії ADM. Розробник, проектний менеджер, консультант, що ініціював масовий перехід від ваговитих методологій убік Agile. Один з авторів відгалуження Scrum(разом з Д. Сазерлендом), яке на сьогодні має найбільшу популярність серед Agile- напрямів.

Джеф Сазерленд

Директор по технологіях великої компанії. Співавтор Scrum. Архітектор великих інформаційних наукомістких систем.

Варто відмітити, що Agile є повністю відкритою методологією, і прийняти участь в її подальшому розвитку і популяризації може будь-який з нас.

## 1.6 Як розвивалася Agile

Опис процесу зародження і розвитку Agile варто розпочати з ситуації, що відбувалася в галузі в середині минулого століття.

Почалася так звана "перша криза програмування", яка полягала в тому, що вартість розробки програмного забезпечення стала наближатися до вартості апаратури, а при аналізі подальших трендів розвитку ситуації були прогнози, що до кінця минулого століття усе людство буде задіяно в процесах розробки інформаційних систем.

Потім стався прорив розвитку електроніки і, як наслідок, збільшилася продуктивність комп'ютерів, а вартість їх стала різко знижуватися. Обмеження на розробку, пов'язані з апаратними засобами, стали зникати або трансформуватися в обмеження на ПО. Таким чином, уміння розробляти нові програми дисонувало з тими вимогами, які замовники пред'являли до цих самих програм. Це привело до того, що розробка програмного забезпечення перестала бути просто "кодуванням", а стала формуватися у вигляді конкретних концепцій, що підтримують повний життєвий цикл програм(формування ідеї, аналіз вимог, проектування функціональності, розробка ПО, тестування ПО, впровадження, підтримка з подальшою утилізацією), що розробляються.

Зміщення фокусу із спеціалізованого процесу розробки у бік повноцінної методології і інструментарію, що підтримує його виконання, привело до появи цілого світу програмної інженерії і, як наслідок, до промислової розробки програмного забезпечення, мета якої полягала в скороченні вартості створення програм.

Усі необхідні напрями, що виконуються в процесі промислової розробки і необхідні для отримання результату, стали формуватися у вигляді конкретних процессных напрямів розробки ПО, кожен з яких надалі переосмислювався, узагальнювався і відповідним чином формалізовувався.

Це привело до появи конкретних практичних методик, кожна з яких підтвердила свою успішність на масиві практичних робіт.

Якщо ми розглянемо будь-яку сучасну методологію управління проектом або моделями життєвого циклу проекту по розробці інформаційних систем, згідно SWEBOK, то виглядати вона буде таким чином:

* традиційна(каскадна, водоспадна) модель;
* спіральна модель;
* ітеративна і інкрементна модель.

SWEBOK - документ, в якому описані еталонні методики по усіх стадіях розробки програмного забезпечення.

SWEBOK є одним з найавторитетніших стандартів в області розробки програмного забезпечення, постійно перевидається і розширюється на предмет поетапної деталізації найбільш складних і важливих напрямів робіт.

Каскадна модель (рис. 1.1) розробки програмного забезпечення є першою і, як наслідок, самою критикованою. При роботі з цією методологією передбачається послідовне виконання етапів розробки. Така структура не дає змінити вимоги до програмного продукту до самого релиза(до виходу зразка готової продукції).

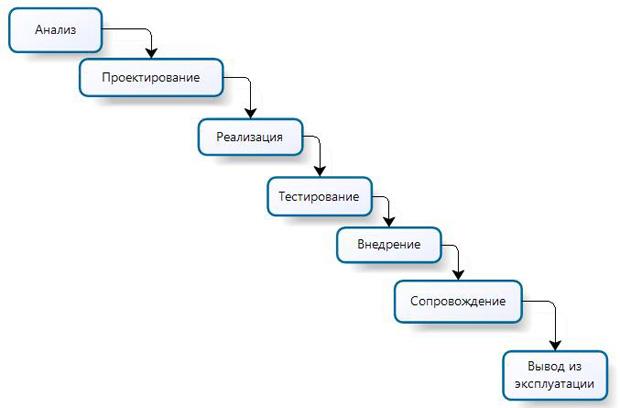


Рисунок 1.1 **–** Каскадна модель розробки

Чим масштабніший проект, тим більше змін накопичується. Реалізація змін в наступній версії продукту іноді стає недоцільною. Продукт необхідно писати з нуля. Таким чином, вартість працездатної версії невиправдано сильно росте. Це призводить до того, що відсоток успішно завершених проектів нікчемно малий в порівнянні з іншими методологіями. Але при цьому для частини проектів, які зачіпають, приміром, безпека життєдіяльності, строго поставлені вимоги і висока міра формалізації є засадничим і необхідним чинником.

Крім того, традиційна модель відіграє важливу роль. Вона накладає на процес розробки вимогу украй необхідної для нього дисциплінованості, за допомогою якої вдається благополучно обходити неструктуровані процеси типу "пишемо і правимо написане".

Традиційна модель внесла фундаментальний вклад в розуміння процесів розробки наступними твердженнями:

* процес повинен підкорятися дисципліні, розумному плануванню і управлінню;
* повна реалізація продукту має бути відкладена до повного розуміння цілей цієї реалізації.

Наступною за часом виникнення і еволюційному розвитку стала спіральна модель (рис. 1.2). Як це часто буває з послідовником, в ній спробували виправити усі основні проблеми її попередника.

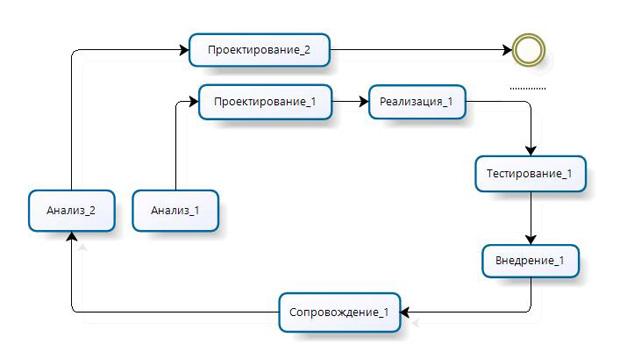


Рисунок 1.2 –Спіральна модель розробки

Набір фаз і їх структура в спіральній моделі відповідають водоспадній моделі, але в спіральній методології кожна фаза завершується етапом прототипирования і управління рисками. Етап прототипирования після кожної фази проекту дозволяє визначити, наскільки поточний стан проекту відповідає первинному плану. За підсумками прототипирования виконується або перехід до наступної фази, або повернення на одну з попередніх фаз для виконання необхідних коригувань.

Наступним витком в розвитку процесів розробки програмного забезпечення стала ітеративна модель. Вона припускає розбиття життєвого циклу на послідовні ітерації, кожна з яких, по суті, є самодостатнім циклом по створенню функціональності програмного продукту.

Витоки концепції ітеративної розробки простежуються в тих, що відносяться до 1930-м рокам роботах експерта по проблемах якості продукції Уолтера Шеварта з Bell Labs. Важливою віхою в історії є здійснений в 50-і роки проект по розробці надзвукового реактивного літака X - 15. На думку учасників цих робіт, застосування спіральної методології значною мірою визначило подальший успіх проекту.

Плавно переходячи від методологій, які були історичними віхами на шляху виникнення Agile(їх детальне порівняння буде проведено в наступних главах), кожна з яких явила собою ступінь в еволюційному розвитку розуміння того, як ефективно на даний момент треба розробляти програмне забезпечення, ми плавно підійшли до теми Agile.

Варто відмітити, що Agile- сімейство, по суті, є відгалуженням ітеративної моделі розробки програмного забезпечення. При описі будь-якої з гнучких методологій згадується принцип розділення на ітерації, який знайшов відображення у багатьох артефактах, вживаних в Agile(task board і ін.).

Наступні віхи в розвитку Agile заслуговують згадки:

* 1992 рік:
  + *Crystal Methods.* Сімейство методологій Crystal послужило основою для розвитку сучасних методологій розробки програмного забезпечення. Agile була створена людьми, які "вийшли" з руху Crystal. Автором Crystal є Алистер Коберн. Методологія може бути застосована до команд, що складаються з шести-восьми учасників, розташованих в одному місці, працюючих над створенням програмних систем, користувачів, що не є критичними для життя.

*Crystal* фокусується на:

* частому постачанню працюючого коду кінцевим користувачам;
* розумних поліпшеннях;
* всепроникній комунікації(osmotic communication) між членами команди, розташованими в одному місці.

Під всепроникною комунікацією мається на увазі безпосередня передача інформації, у тому числі шляхом підслуховування або спостереження за речами, що відбуваються навкруги.

* 1993 рік:
  + *Refactoring.* Термін "рефакторинг" був введений Білом Опдайком в статті під назвою "Creating Abstract Superclasses by Refactoring". У Вікіпедії приведений наступний опис рефакторинга : "Процес зміни внутрішньої структури програми, що не зачіпає її зовнішньої поведінки і має на меті полегшити розуміння її роботи".
* 1994 рік:
  + *Dynamic Systems Development Method (DSDM).* DSDM був розроблений консорціумом, що є об'єднанням постачальників і виробників програмного забезпечення. Мета їх роботи - спільними зусиллями розробити і розповсюдити незалежний framework для швидкої розробки додатків з використанням накопиченого досвіду.

DSDM фокусується на восьми основних принципах:

* враховувати потреби бізнесу;
* поставляти вчасно;
* взаємодіяти;
* ніколи не знижувати якість;
* створювати поступово з самих основ;
* розробляти ітераційно;
* безперервно і ясно спілкуватися;
* демонструвати керованість.
* 1995 рік:
  + *Scrum and Pair Development.* Scrum був розроблений спільно Джефом Сазерлендом і Кеном Швабером, які представили доповідь на конференції OOPSLA'95 в Остіні, штат Техас.
* 1997 рік:
  + *Feature Driven Development.* Методологія Feature Driven Development(FDD) спочатку була розроблена Джефом ДеЛукой.

Основні практики, описані в FDD, наступні:

* моделювання об'єктної моделі домена предметної області;
* розробка на основі поліпшень;
* індивідуальне володіння кодом;
* команди, організовані по напрямах поліпшень;
* інспекції;
* управління конфігурацією;
* регулярні складки продукту;
* видимість прогресу і результатів.
* 1999 рік:
  + *Adaptive Software Development.* Джим Хэнгсмит сформулював концепцію Adaptive System Development і опублікував книгу з такою ж назвою. Ідея виросла з його роботи по методологіях швидкого створення додатків (RAD).
* 2002 рік:
  + *Нові ідеї 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 рік:
  + *Lean Software Development.* o Послідовники Agile вважали, що наступним витком її розвитку буде трансформація з процессной методологією Lean. Час підтвердив обгрунтованість цієї ідеї. Термін такого відгалуження Agile був введений Мері і Томом Поппендик в 2003 році.

Це адаптація бережливих принципів виробництва до розробки програмного забезпечення. Розглядаються сім основних принципів:

* усувати витрати;
* посилювати навчання;
* приймати рішення як можна пізніше;
* поставляти якомога раніше;
* надавати підтримку команді;
* вбудовувати цілісність;
* бачити картину в цілому.

Agile на даний момент свого розвитку - більшою мірою зведення знань по організації роботи з психологічної точки зору. Agile допомагає проявляти творчу складову, уміння працювати в команді, навички комунікації і інше. Технічна сторона організації робіт все більше йде на другий план, але при цьому Agile містить достатню кількість інженерних практик, використовуючи які, можна досягти якісного кінцевого результату. Ці практики з успіхом застосовуються у поєднанні з іншими методологіями.

## 1.7 Agile Manifesto

Після того, як у нас склалося цілісне уявлення про історичний шлях Agile, логічно перейти до розгляду документу, який є засадничим для багатьох процесів, що належать цьому сімейству.

Цим документом є Agile Manifesto. По суті, його і документом назвати складно. Це, швидше, невелика пам'ятка, яка містить 4 цінності і 12 принципів роботи. Далі ми приведемо повний зміст Agile Manifesto, як воно представлене на офіційному сайті(http://agilemanifesto.org/) консорціуму, що розробив його.

**Цінності:**

1. Люди і взаємодія важливіше процесів і інструментів.
2. Працюючий продукт важливіше вичерпної документації.
3. Співпраця із замовником важливіша за узгодження умов контракту.
4. Готовність до змін важливіше дотримується первинного плану.

Таким чином, не заперечуючи важливості того, що справа, все-таки більше цінується те, що ліворуч.

**Основні принципи:**

1. Найвищим пріоритетом є задоволення потреб замовника завдяки регулярному і ранньому постачанню цінного програмного забезпечення.
2. Зміна вимог вітається навіть на пізніх стадіях розробки.
3. Працюючий продукт слід випускати як можна частіше, з періодичністю від пари тижнів до пари місяців.
4. Упродовж усього проекту розробники і представники бізнесу повинні щодня працювати разом.
5. Над проектом повинні працювати мотивовані професіонали. Щоб робота була зроблена, створіть умови, забезпечте підтримку і повністю довіртеся ім.
6. Безпосереднє спілкування є найбільш практичним і ефективним способом обміну інформацією, як з самою командою, так і усередині команди.
7. Працюючий продукт - основний показник прогресу.
8. Інвестори, розробники і користувачі повинні мати можливість підтримувати постійний ритм нескінченно.
9. Постійна увага до технічної досконалості і якості проектування підвищує гнучкість проекту.
10. Простота - мистецтво мінімізації зайвої роботи - украй потрібна.
11. Самі кращі вимоги, архітектурні і технічні рішення народжуються у команд, що самоорганізуються.
12. Команда повинна систематично аналізувати можливі способи поліпшення ефективності і відповідно коригувати стиль своєї роботи.

Викладені цінності і принципи говорять самі за себе і якихось тривалих і просторових міркувань не особливо потребують.

Акцент зроблений на створенні відчутної вигоди для замовника. У Agile Manifesto враховані усі ті складнощі, які є критичними для водоспадної і ітераційної моделей розробки програмного забезпечення, з метою їх можливої мінімізації.

Кожна з цінностей і принципів буде розглянута нами детальніше в подальших лекціях. Головне, на чому хочеться зосередити увагу зараз, - за допомогою Agile Manifesto консорціумом професіоналів була закладена успішна спроба вироблення процессной методології, в якій стало реальним взаємопроникнення інформаційних технологій і сфери діяльності, зацікавлений не просто в застосуванні сучасних високотехнологічних напрацювань, а в їх постійному використанні, розвитку і впровадженні в операційну діяльність.

Бар'єри, які виникали з середини минулого століття між інтелектуалом-розробником і представником бізнесу, пов'язані з нерозумінням один одного, перетворювалися на бар'єри при створенні інформаційних систем, які не демонстрували очікуваній від них корисності і стали поступово рушитися.

## 1.8 Застосування в різних предметних областях

Незважаючи на ту, що здається специализированность Agile, ця методологія розширює сферу свого впливу і поступово стає новою культурою управління компаніями(Toyota) і проникає в різні сфери діяльності.

Одним з основних постулатів використання Agile є розробка ефективного кінцевого продукту.

Результатом практично кожного напряму діяльності є саме продукт, виражений у вигляді результату, втіленого в певну специфіку конкретного виду діяльності.

Навіть якщо йдеться про такий консервативний і "нескінченний" процес, як бухгалтерський облік, то можна провести паралель не з розробкою програмного забезпечення, а з таким етапом його життєвого циклу, як підтримка і розвиток.

Розвиток продукту ітераціями, визначуваними тимчасовим інтервалом, навряд чи підійде, але якщо говорити про ітерації, які визначатимуться способом роботи над конкретним завданням, то все стає цілком логічним.

Останнім часом набуває широкого поширення спосіб управління, званий холакратией. Це підхід до управління і розподілу відповідальності, який дозволяє позбавитися від ієрархії в компанії. Кожен співробітник(холакратия пропонує замість слова "співробітник" використати слово "партнер") потрапляє в умови, які на пострадянському просторі називаються "Працювати, як для себе", безперервно щось змінює і покращує в тій частині бізнесу, за яку він відповідальний. І це не просто слова, це логічне продовження Agile, яке має приклади успішного застосування(проект "Кнопка").

Основне в застосуванні Agile - це бажання вчитися і розвиватися. Професіонали, які прагнуть до використання нових ідей і передових практик, як правило, із задоволенням стають на рейки Agile.

Аgile - підхід інноваційного переосмислення створення нового продукту або послуги. У його основі дуже проста ідея: кожен учасник процесу, кожен співробітник цього "конвеєрного складання" повинні залучатися до процесу переосмислення своїх завдань і загальної справи. Кожен може зупинити конвеєр і внести свої раціональні пропозиції.

## 1.9 Короткі висновки

Перша лекція, в якій приведений короткий огляд історичної ситуації, пов'язаної з виникненням такого напряму діяльності сучасного світу, як розробка програмного забезпечення, добігла кінця. Ми заронили основні розповідні зерна, які пророщуватимемо в подальших главах курсу. Як ви могли зрозуміти, тема гнучких процессных методологій є складною, але при цьому цікавим і доступним розумінню напрямом, який приковує до себе увагу. Очікування від його застосування дуже високі. Але на успіх можуть розраховувати тільки ті, у кого вже склалося цілісне уявлення про те, що процессный підхід до організації діяльності - це складна управлінська парадигма, яка потребує постійного контролю і управління. Інакше очікуваний ефект від Agile навряд чи буде досягнутий.

У лекції "Процессный підхід як конкурентна перевага" буде вивчена процессный підхід, його значущість для сучасних компаній, розглянуті найбільш ефективні варіанти його впровадження і подальшої адаптації і основні артефакти, без яких його використання не буде по-справжньому оптимальним.

**Питання до теми:**

1. Чим Agile може бути корисна для вашої компанії?
2. Що таке Agile?
3. Приведіть умови застосовності Agile.
4. Приведіть цінності Agile.
5. Приведіть основні принципи Agile.
6. Що таке "холакратия"?

# Лекція 2: ПРОЦЕССНЫЙ ПІДХІД ЯК КОНКУРЕНТНА ПЕРЕВАГА

План

2.1. Процессный підхід

2.2. Значущість процессного офісу у впровадженні процессного підходу. Можливі варіанти

2.3. Рух у напрямі гнучкості. Пілотні процеси і управління змінами

2.4. Регламент як основа процессного підходу

2.5. Короткі виcновки

## 2.1. Процесний підхід

**Сенс процессного управління**

Компанії для завоювання лідерства на ринку багатьох років тестують різні підходи до управління, такі як: управління по цілях, управління дорученнями, управління проектами і управління бізнес-процесами. Кожен з управлінських підходів хороший для тієї або іншої ситуації, яка складається з галузевої приналежності організації, її зрілості, конкурентності ринків, на яких функціонує компанія, особи керівника компанії і безлічі інших чинників.

Для підприємств, які працюють на неконкурентних ринках, система управління може бути будь-якою. Наприклад, в українських монополіях частіше можна побачити систему управління дорученнями з відповідною системою контролю їх виконання.

Управління бізнес-процесами найбільше необхідно виробничим компаніям і компаніям з сфери послуг, працюючих на конкурентних ринках. Їм доводиться боротися за свої позиції, покращуючи якість, оптимізуючи вартість або скорочуючи терміни. Тому за довгі роки управління бізнес-процесами стало стандартом де-факто для більшості міжнародних компаній.

У українських умовах процессное управління затребуване комерційними компаніями, працюючими в конкурентному середовищі, - банками, страховими компаніями, компаніями роздрібної торгівлі, сферою надання послуг і багатьма іншими галузями.

Сенс процессного управління гранично простий. Потоки робіт, що проходять через компанію, вимагають участі представників різних підрозділів, що часто мають різні цілі, завдання і показники. При цьому часто ніхто не зацікавлений в ефективності усього наскрізного бізнес-процеса, що проходить через усю компанію. Головне - якось зробити свою частину усередині бізнес-процеса, після чого передати завдання далі. В результаті такого функціонально-орієнтованого підходу в компанії немає відповідального за увесь бізнес-процес цілком, що призводить до серйозної неефективності організації робіт і постійних конфліктів між представниками різних функціональних підрозділів, які ще більше знижують результативність процесу. Фактично потужність бізнес-процеса визначається потужністю найслабкішої ланки у бізнес-процесі, що часто зводить нанівець зусилля інших підрозділів із оптимізації своєї частини бізнес-процеса.

Сенс управління бізнес-процесами полягає у визначенні кросс-функциональных або наскрізних бізнес-процесів, їх формалізації в тому або іншому форматі і подальшому їх аналізі, а головне - їх оптимізації. Чим більше у бізнес-процесі учасників, тим більше в нім нестикувань, і тому особливу увагу треба приділяти саме кросс-функциональным процесам. Процеси, які називаються наскрізними, ще важливіше, вони починаються поза організацією і закінчуються поза організацією, саме по організації наскрізних бізнес-процесів клієнти і інші зацікавлені сторони створюють свою думку про компанію, і тому наскрізні процеси - перші кандидати на управління і оптимізацію.

Якщо говорити про циклю управління бізнес-процесом, то в першу чергу необхідно зробити опис, або, як ще говорять, моделювання бізнес-процеса, яке показує усю неефективність існуючого бізнес-процеса. А аналіз фактичних значень показників бізнес-процеса дозволяє не лише проаналізувати існуючу ситуацію, але і визначити ті показники, які мають бути вдосконалені.

На практиці існує безліч способів оптимізації, але найголовніше, що після їх застосування повинна стати зрозуміла цільова картина бізнес-процеса, тобто той стан бізнес-процеса "як повинно бути", яке і необхідно впровадити в організації.

Далі на підставі цільової картини необхідно почати зміну процесу і його впровадження в діяльність нових правил. Найчастіше це робиться за допомогою нормативних документів(регламентів і стандартів), інформаційних систем, а також навчання учасників бізнес-процеса. Саме впровадження бізнес-процесів "як повинно бути" є найбільш трудомістким завданням, адже треба змінити компанію, її співробітників разом з їх звичками і представленнями, а можливо навіть і звільнити частину не згідних з новими правилами.

**Моделювання, аналіз і оптимізація бізнес-процесів**

Ключовим інструментарієм впровадження процессного підходу до управління на перших етапах являються графічні інструменти моделювання бізнес-процесів, а також текстові і табличні редактори, в яких моделюються або описуються бізнес-процеси, а також створюються регламенти, що закріплюють цільовий стан бізнес-процеса.

На сьогодні часу для графічного моделювання бізнес-процесів MS Visio є найбільш поширеним інструментом, при цьому моделювати бізнес-процеси можна в різних нотаціях. Проте ринок інструментарію не коштує на місці, і для моделювання простих схем бізнес-процесів зараз можливо використати абсолютно безкоштовні хмарні застосування, які дозволяють намалювати бізнес-процес фактично у будь-якій нотації.

Деякі компанії взагалі обходяться без інструментів графічного моделювання бізнес-процесів, використовуючи презентації для графічної ілюстрації і текст або таблицю для опису необхідних деталей бізнес-процесів, що дозволяє їм обмежитися класичними офісними редакторами.

На українському ринку є серйозніший інструментарій, що дозволяє будувати цілісну модель бізнес-процесів зі взаємозв'язками як між окремими моделями процесів, так і з елементами організаційної структури, класифікатором інформаційних систем, стратегічними цілями і ключовими показниками результативності, а також документами в різних форматах, даними в інформаційних системах і іншими артефактами. Найбільше поширення серед серйозних промислових систем для моделювання і аналізу бізнес-процесів, використовуваних в найбільших українських компаніях, зараз має система ARIS. В той же час є приклад імпортозаміщення на українському ринку - система Business Studio.

Перевагою таких систем є забезпечення зв'язності моделі опису діяльності, а також можливість отримання в автоматичному режимі регламентів бізнес-процесів, положень про підрозділи, а також будь-яку іншу інформацію на підставі побудованих моделей бізнес-процесів.

Недоліком промислових систем моделювання і аналізу бізнес-процесів є складність їх освоєння користувачами з бізнес-підрозділів, що часто призводить до локалізації робіт по моделюванню бізнес-процесів в процессном офісі - підрозділі, супроводжуючим впровадження процессного підходу в компанії.

Застосування простих інструментів з комплекту MS Office або навіть безкоштовних хмарних інструментів, таких як draw.io і bpmn.io, дозволяє розширити круг співробітників, що описують бізнес-процеси усередині компанії, за рахунок представників бізнес-підрозділів.

Зараз можна вибирати між різними формами моделювання бізнес-процесів, але так чи інакше лідерство доки за текстовою і табличною формою опису бізнес-процесів, в цьому форматі процес може описати будь-який керівник або співробітник. У графічному форматі частіше малюють бізнес-аналітики, використовуючи стандартні нотації моделювання бізнес-процесів, такі як BPMN і EPC.

На перших етапах опису і аналізу бізнес-процеса дуже важливо правильно виділити бізнес-процес, описавши усі його складові, зібравши проблематику і фактичні значення ключових показників результативності по процесу.

І вже після того, як первинні дані зібрані, ключовою стає необхідність піддати існуючий бізнес-процес оптимізації або через скорочення часу, або через скорочення вартості, або через підвищення якості.

Можна займатися оптимізацією процесу за двома параметрами з трьох, але ось оптимізація за трьома параметрами рідкісна, оскільки якщо буде скорочена тривалість процесу, то збільшиться вартість процесу або впаде його якість - проектний трикутник доки ще ніхто не відміняв.

У рамках аналізу і оптимізації бізнес-процесів дуже корисно використати різні референтні моделі, а також аналізувати приклади реалізації аналогічного бізнес-процеса в компаніях тієї ж галузі.

Проте найскладніше - це не пошук цільового стану бізнес-процеса, а його впровадження в діяльність компанії. Адже треба не просто описати нову схему роботи, але і переконати учасників процесу наслідувати нові правила. У більшості російських компаній найчастіше складається регламент, контроль над виконанням якого покладається на того або іншого керівника, проте в століття інформаційних технологій вже з'явилися інформаційні системи, які допомагають закріпити цільовий бізнес-процес в організації і контролювати дотримання закріплених правил. Ці системи з'явилися разом з технологією процессного управління і спочатку називалися workflow.

**Автоматизація бізнес-процесів**

Як і безліч інформаційних систем, workflow- системи стали реакцією ИТ-ринку на появу процессного підходу до управління, саме в workflow- системах з'явився такий об'єкт, як бізнес-процес, який настроювався розробником або системним аналітиком при впровадженні workflow- системи, і відповідно до якого маршрутизувалися завдання у рамках виконання бізнес-процеса. Згодом разом з розширенням функціонала workflow- системи придбали назву BPMS(Business Process Management Suite), зараз ці системи називаються iBPM(intelligent Business Process Management).

На думку багатьох керівників, автоматизація бізнес-процесів часто являється єдиним результативним способом впровадити бізнес-процес "як повинно бути" в діяльність в точній відповідності з цільовою моделлю. І саме тому в деяких компаніях стали використовуватися системи iBPM для автоматизації деяких кросс-функциональных бізнес-процесів, а деякі компанії навіть змогли замкнути цикл управління для ключових бізнес-процесів, впровадивши окрім систем iBPM ІТ-рішення класу BI(Business Intelligence) для аналізу процессных показників.

Серйозною віхою в застосуванні систем автоматизації бізнес-процесів стала можливість не програмувати бізнес-процес, а настроювати його через формування графічної нотації у форматі BPMN, що забезпечило швидкість внесення змін до автоматизованих бізнес-процесів.

При цьому треба розуміти, що стандартизація бізнес-процесів із застосуванням інформаційних технологій може зажадати рішень різного класу, тут і CRM- системи, і ERP, і SCM, при цьому різноманітність існуючих IT-рішень на ринку вимагає вибору інформаційної системи під особливості того або іншого бізнес-процеса. При цьому добре відомо, що більшість типових IT-рішень мають усередині жорстко зашитий бізнес-процес, і тому автоматизація за допомогою даних IT-рішень часто типізує бізнес-процеси відповідно до запрограмованих алгоритмів, що жорстко закріплює бізнес-процес в інформаційних системах.

Проте не усі бізнес-процеси можна автоматизувати типовими ІТ-решеннями, для багатьох процесів можна не знайти IT-рішень, і тут може використовуватися або власна розробка, або різні системи-конструктори, які дозволяють створити спеціальне рішення для автоматизації саме цього бізнес-процеса.

Але існує ще складніший випадок, коли бізнес-процес настільки специфічний, що під нього не знайти типового рішення, і в доповненні бізнес-процес необхідно регулярно змінювати. Типові системи в даному випадку не підходять, а використовуючи власну розробку, доведеться постійно переписувати код створеної інформаційної системи, що негативно позначиться на швидкості внесення змін у бізнесі. Саме для цих бізнес-процесів, які, як правило, і є наскрізними або кросс-функциональными, і застосовуються системи класу iBPM, які спочатку і були призначені для автоматизації взаємодії співробітників відповідно до правил, зашитих у бізнес-процесі. Фактично iBPM- системи дозволяють закріпити логіку взаємодії і час виконання окремих операцій, а також забезпечують високу швидкість внесення змін до автоматизованого бізнес-процеса, що дозволяє створити регламент в електронному форматі зі вбудованими функціями моніторингу і контролю ключових параметрів бізнес-процеса.

Приклади проектів показують, що iBPM- система цілком може замінити керівника при маршрутизації завдань усередині одного бізнес-процеса, враховуючи при призначенні завдання на виконавця, безліч параметрів, починаючи від завантаження співробітників і закінчуючи їх статистикою по операційній ефективності, розрахованій на базі історичних даних.

Можна сказати, що за допомогою iBPM- систем автоматизується управлінське рутинне завдання маршрутизації завдань між виконавцями з контролем термінів їх виконання і можливістю перебудовувати бізнес-процес без виснажливого програмування. iBPM- системи в автоматичному режимі эскалируют порушення по термінах, що дозволяє будувати досить жорсткі системи мотивування, залежні від виконання окремих завдань усіма учасниками бізнес-процеса.

**Контроллинг бізнес-процесів**

Не завжди вдається використати iBPM- системи для автоматизації бізнес-процеса, оскільки часто процес виконується в декількох типових інформаційних системах, які в тому або іншому вигляді інтегровані між собою. Тоді для контролю бізнес-процеса можливо використати рішення класу Process Intelligence(PI), в якому можна зібрати і проаналізувати дані з різних інформаційних систем, що підтримують виконання бізнес-процеса.

Окрім систем класу PI для аналізу бізнес-процесів можуть бути використані спеціалізовані ИТ-рішення. Такі як Process Mining - в рішеннях цього класу на підставі даних про фактичне виконання бізнес-процеса вибудовується графічна модель бізнес-процеса, а також системи класу Business Activity Monitor(BAM), які збирають в оперативному режимі статистику по виконуваному процесу.

За допомогою систем класу PI і BAM можна проводити аналіз як тимчасових, так і частотних показників бізнес-процеса, при цьому для аналізу фактичної логіки взаємодії учасників бізнес-процеса можна використати інструментарій Process Mining, за допомогою якого із статистики виконання окремих дій можна зібрати фактичну модель виконаного процесу для подальшого аналізу.

При організації контроллинга бізнес-процеса найчастіше аналізуються показники часу виконання тієї або іншої операції, а також середнього часу виконання екземпляра бізнес-процеса за той або інший період.

Скорочення часу виконання окремої операції досягається нормуванням і контролем роботи учасників бізнес-процеса, тоді як секрет скорочення середнього часу виконання бізнес-процеса полягає в пошуку "пляшкових шийок", в яких завдання по процесу знаходиться в черзі, чекаючи, коли буде доступний ресурс для її виконання.

Окремою групою показників для аналізу бізнес-процеса є показники конверсії, які показують ефективність окремих операцій процесу, наприклад, скільки комерційних пропозицій перейшло в договір або скільки кандидатів на роботу пройшла співбесіда з керівником підрозділу. Ці частотні показники найчастіше аналізуються в масових бізнес-процесах, де відбувається обробка великого числа заявок і замовлень.

Окремим напрямом контроллинга бізнес-процеса є його вартість. На підставі середнього часу виконання операції з урахуванням вартості одиниці робочого часу виконавця, а також витрат на організацію робочого місця можна підрахувати вартість однієї операції. І вже на цій підставі розрахувати вартість випадку одиничного виконання бізнес-процеса і потім отримати можливість розрахунку вартості бізнес-процеса за період.

Аналіз вартості бізнес-процеса дозволяє правильно перерозподілити витрати по послугах, оцінити ефективність заходів по автоматизації бізнес-процеса і навіть проаналізувати результативність автоматизації бізнес-процеса. При цьому практика проектів за розрахунком вартості бізнес-процесів показує, що керівники часто не бачать фактичну вартість бізнес-процесів, що призводить до ухвалення неефективних з економічної точки зору рішень.

Так або інакше, організація контроллинга бізнес-процеса дозволяє організувати зворотний зв'язок з бізнес-процеса, що дозволяє контролювати в регулярному режимі його результативність, а головне - оцінювати ефективність заходів по його оптимізації.

## 2.2. Значущість процессного офісу у впровадженні процессного підходу. Можливі варіанти

**Від центру компетенції до процессному офісу**

Процессный підхід до управління вимагає кросс-функциональной відповідальності за бізнес-процеси. У деяких компаніях функціональні керівники домовляються між собою, синхронізуючи наскрізні або кросс-функциональные бізнес-процеси на межах переходу між підрозділами. Але така ситуація складається не скрізь.

У багатьох компаніях існують бізнес-процеси, в яких бере участь безліч підрозділів з пересічними інтересами, при цьому при спробі оптимізувати такий бізнес-процес менеджерам компанії не завжди вистачає часу або управлінських повноважень для його оптимізації.

Саме неузгодженість дій декількох функціональних підрозділів призводить до затримок у виконанні робіт, помилкам, бюрократичним процедурам, що не додають якості у бізнес-процес, а що лише з'їдає додаткові ресурси.

Таким чином, впровадження процессного підходу в організації необхідно підтримати організаційними заходами і створенням спеціалізованого центру компетенції або навіть окремого підрозділу для визначення бізнес-процесів і подальшого їх удосконалення.

На першому етапі центр компетенції може полягати в теці на сервері, де складені повчальні матеріали по управлінню бізнес-процесами і регулярної інформаційної розсилки по зацікавлених особах. Надалі така розсилка може перерости в окремий підрозділ, що займається впровадженням процессного підходу в організації.

Такі підрозділи в теорії називаються процессными офісами, хоча на практиці можуть зустрічатися і інші назви, наприклад, департамент по оптимізації бізнес-процесів або департамент стандартизації бізнес-процесів, або навіть департамент опису бізнесу і регламентації стосунків.

Незалежно від назви процессный офіс є підрозділом, який сприяє в описі, регламентації і оптимізації бізнес-процесів в компанії. Для забезпечення підтримки впровадження процессного підходу на рівні усієї компанії окрім процессного офісу має бути організований спеціалізований комітет, в завдання якого входить постановка завдань на опис і оптимізацію бізнес-процесів, а також розбір кейсів, що сталися, і аналіз кращих практик з метою запустити процес безперервного вдосконалення діяльності організації.

Частота засідань цього комітету може відрізнятися залежно від організації, десь це щорічне засідання з доповіддю про зміни, зроблені у бізнесі, а десь це може бути щотижневе засідання, на якому розглядаються кейси, затверджуються регламенти і даються завдання на стандартизацію або оптимізацію тих або інших процедур.

Мета цього комітету - зробити вдосконалення існуючих бізнес-процесів регулярним, дати керівникам підрозділів і власникам бізнес-процесів завдання на оптимізацію тих або інших процесів, а також в оперативному режимі розглядати конфліктні ситуації, які виникають у бізнес-процесах із-за тієї, що розузгодила дій учасників бізнес-процеса, і старатися на системному рівні виключити повторення конфліктних ситуацій в майбутньому.

При цьому завдання оптимізації бізнес-процесів необхідно ставити функціональним керівникам на рівні заступників генерального директора або директора департаменту, і саме вони повинні розглядати свої бізнес-процеси і пропонувати заходи по їх удосконаленню. Процессный офіс в даному випадку виступає як внутрішній ресурс, який може навчити моделюванню бізнес-процесів, формуванню регламентів, а також методам аналізу і оптимізації бізнес-процесів. Якщо завдання опису, аналізу і оптимізації бізнес-процесів повністю передати в процессный офіс, то з великою вірогідністю сама технологія процессного управління не буде впроваджена у бізнес-підрозділи, які з радістю віддадуть усі завдання по оптимізації і регламентації бізнес-процесів процессному офісу, самі спостерігаючи з боку, як їх діяльність намагатимуться удосконалити.

У українських компаніях існує безліч процессных офісів, які не мають організаційних важелів для оптимізації бізнес-процесів, і тоді вони зосереджуються на описі існуючих бізнес-процесів, закріплюючи їх поточний стан у вигляді графічних моделей або регламентів.

Найбільш ефективна схема, в якій бізнес підрозділу самостійно формують бачення існуючого бізнес-процеса, спільно з процессным офісом і ИТ-підрозділом створюють бачення процесу "як повинно бути", після чого активно беруть участь у впровадженні цільового бізнес-процеса. Саме така структура випробувана у більшості міжнародних компаній. Адже при активній участі бізнес-підрозділів в описі, аналізі і оптимізації бізнес-процесів процессный офіс можна зробити досить компактним за штатною чисельністю або взагалі вивести його на аутсорсинг.

Адже завдання процессного офісу полягає не в описі, аналізі і оптимізації існуючих бізнес-процесів замість менеджерів, а в просуванні культури і інструментарію процессного управління серед менеджменту компанії при активній підтримці верхнього керівника.

**Відповідальність за бізнес-процес**

Одним з ключових питань в процессном управлінні є визначення відповідальності за бізнес-процеси. Кожен функціональний підрозділ відповідає за свою частину бізнес-процеса, але як тільки ставиться завдання знайти відповідального за увесь бізнес-процес, серед функціональних керівників виникає конфлікт, що вимагає втручання керівника, що знаходиться за управлінською ієрархією вище їх. Деякі компанії вирішують питання конфліктів, що виникають у рамках кросс-функциональных процесів, нарадами, на яких розбираються деталі ведення операційної діяльності, а також розглядаються ті або інші кейси. Якщо у рамках наради виробляти загальні правила і фіксувати їх у форматі моделей або текстового опису, то з часом необхідність таких нарад відпаде, а між підрозділами виникне розуміння, закріплене у формалізованому виді, як необхідно взаємодіяти по бізнес-процесу.

Якщо самостійної взаємодії на рівні керівників і виконавців підрозділів у рамках бізнес-процеса не складається, то кількість конфліктів росте, а на нарадах конструктиву усе менше. У такому разі співробітники процессного офісу можуть виконувати функцію рефері по розмежуванню позицій сторін з одночасною формалізацією досягнутих домовленостей.

Є варіант, коли у рамках кросс-функционального бізнес-процеса може бути визначений старший керівник, який стає "головніше", ніж інші учасники бізнес-процеса, - це власник процесу. Його завдання - не лише добитися синхронізації дій учасників процесу, але і забезпечити контроль результатів, а також ініціацію заходів по оптимізації бізнес-процесів.

Фактично власник процесу відповідає за результат процесу, незважаючи на те що в процесі беруть участь співробітники інших підрозділів. У рамках управління процесом з'являється рамкова відповідальність - за якість виконання окремої операції відповідає учасник бізнес-процеса - співробітник функціонального підрозділу, тоді як за формати і своєчасність передачі і прийому результатів робіт по усьому бізнес-процесу між підрозділами відповідає власник процесу. Саме власник процесу аналізує організаційні і інформаційні розриви, що виникають на межах підрозділів, а також визначає заходи по їх усуненню.

Розрахунок фактичних значень показників по процесу теж входить в зону відповідальності власника, адже саме він зацікавлений в організації контроллинга бізнес-процеса для розуміння точок неефективності, які негативно впливають на цільові показники процесу. На практиці власником процесу може бути керівник рівня директора департаменту або навіть вище. При цьому над власником процесу є ще одна роль - це куратор процесу, який визначає цільові показники, формує стратегічні установки, а також розпоряджається бюджетом по бізнес-процесу.

Призначення власників процесу дуже схоже з призначенням керівника проекту, різниця лише в тому, що керівник проекту призначається на разову активність, в якій беруть участь декілька виконавців від функціональних підрозділів, а власник процесу відповідає за регулярну діяльність. При цьому керівник проекту орієнтований на планування майбутнього плану робіт, тоді як власник процесу зосереджений на аналізі минулого з метою надалі зробити його ефективніше.

**Варіанти організації процессного офісу**

У великих компаніях часто виникає питання, як має бути побудований процессный офіс і які мають бути його функції. Існує три способи організації процессного офісу : перша має на увазі централізацію на рівні штаб-квартири, друга - повну децентралізацію по підрозділах, і третя, така, що має на увазі гібридну організацію, частина процессного офісу знаходиться в штаб-квартирі, а частина розосереджена по підрозділах. У разі повної централізації процессный офіс часто настільки віддаляється від реальних бізнес-процесів, що втрачає компетенцію у бізнесі і перетворюється на даремну структуру, що займається лише стандартизацією бізнес-процесів. І хоча стандартизація теж корисна, проникнення процессных технологій у бізнес-підрозділи ускладнене.

У разі повної децентралізації процессного офісу не існує єдиного координаційного центру, що призводить до несумісності отримуваних результатів і дублювання робіт. Прикладами незручності децентрализованной форми організації процессного офісу можуть бути використання різних нотацій для моделювання бізнес-процесів, закупівля різних інструментів дли опису бізнес-процесів, різні шаблони регламентуючої документації і багато що інше.

Гібридна форма побудови процессного офісу визнана найбільш ефективною, оскільки вгорі знаходиться мінімальна структура, що відповідає за методологію і загальні правила, а на рівні підрозділів зосереджені основні ресурси процессного офісу, що допомагають бізнесу займатися описом, аналізом і оптимізацією бізнес-процесів.

Чисельність процессного офісу може варіюватися від однієї людини до декількох сотень, при цьому чим більше процессных завдань розташовані усередині основних бізнес-підрозділів, тим процессный офіс компактніший, а поширеність процессных технологій ширша.

## 2.3. Рух у напрямі гнучкості. Пілотні процеси і управління змінами

Одне з непростих питань в застосуванні процессного підходу полягає в наданні побудованим бізнес-процесам гнучкості. У компаніях дуже часто можна побачити, що побудований бізнес-процес "костеніє" з часом і перестає відповідати вимогам споживачів, що змінилися. Якщо процес автоматизований в інформаційній системі, в яку складно внести зміну, то процес стає таким, що не просто "закостенів", а "скам'янілим", адже для його зміни треба переписати існуючу інформаційну систему, на що у ИТ-підрозділу, як правило, немає ні часу, ні ресурсів.

Саме тому у випадках впровадження процессного підходу необхідно особливу увагу приділяти не лише підтримці гнучкості існуючих бізнес-процесів на організаційному рівні, але і забезпеченню гнучкості бізнес-процеса на рівні ІТ-систем.

Окрім інформаційних систем гнучкості процесів заважає небажання його учасників мінятися самим і змінювати існуючий бізнес-процес. Людській психіці властиво використати перевірені варіанти дій, і тому переконати учасників бізнес-процеса в тому, що їм тепер потрібно працювати по-іншому - швидше і якісніше, та ще і меншим складом, - буває досить непросто. І тут необхідно шукати прибічників - тих, хто добре сприймає зміни, робити їх агентами змін і, користуючись підтримкою власника процесу, навчати усіх інших учасників новим алгоритмам роботи.

Напевно, 70% ресурсів в проекті оптимізації бізнес-процеса доводиться саме на впровадження нової версії бізнес-процеса в діяльність організації. При цьому якщо процес критичний або зачіпає безліч учасників, то необхідно перед впровадженням бізнес-процеса його ретельно протестувати - наприклад, у форматі ділової гри або використовуючи засоби імітаційного моделювання, а вже потім випробувати процес на пілотній ділянці або регіоні. Як правило, у рамках пілотного впровадження бізнес-процеса виявляються нюанси, про які забули подумати при розробці цільового стану бізнес-процеса "як повинно бути", а при впровадженні вони стають видні.

Українська специфіка, що полягає в управлінні бізнес-процесами через розсилку паперових регламентів, що описують бізнес-процес "як повинно бути", часто нівелює проблему впровадження нових бізнес-процесів, адже регламент висланий, і керівники сподіваються, що люди стали працювати по-іншому. У деяких компаніях це спрацьовує, а в деяких співробітники відразу видаляють лист з регламентом, навіть не читаючи його.

Технологія впровадження через регламент стає нерезультативна, якщо новий бізнес-процес і його представлення на папері робляться без участі керівників підрозділів, або керівники підрозділів самі відстороняються від участі в проектуванні бізнес-процесів. Якщо вони не брали участь в створенні бізнес-процеса і відповідного регламенту, то і при його впровадженні будуть складнощі. Швидше за все, виникне думка, що "ми тут працюємо, а ви нам своїми бізнес-процесами і правилами тільки заважаєте". Така корпоративна культура, швидше за все, не дозволить змінити існуючі бізнес-процеси.

Для забезпечення впровадження нових бізнес-процесів в корпоративну практику треба залучати увесь керівний склад і операційних співробітників до роботи над бізнес-процесами "як повинно бути", адже інакше новий процес не відбиватиме усіх нюансів і не буде прийнятий співробітниками в якості нормативу для роботи, якщо вони не братимуть участь в розробці нового бізнес-процеса.

На практиці складно щось змінити в існуючих бізнес-процесах, якщо співробітникам згори спускається регламент або стандарт, який вироблений без їх участі і обговорення. Саме тому так важливо розвивати корпоративну культуру, спрямовану на безперервне вдосконалення бізнес-процесів на рівні силами усіх співробітників компанії, і тут процессный офіс має бути підтриманий керівництвом в частині необхідності перенесення робіт по тому, що перепроектувало процесів всередину операційних підрозділів, а не створювати окремий підрозділ, який у відриві від бізнесу розроблятиме бізнес-правила і стандарти, часто без участі представників бізнесу.

Одним з інструментів залучення співробітників компанії у безперервне вдосконалення бізнес-процесів є "банк ідей" - це бізнес-процес, у рамках якого відбувається генерація ідей по оптимізації бізнес-процесів, ранжирування пропозицій, їх фільтрація і відбір на підставі тих або інших параметрів.

Фактично до оптимізації бізнес-процесів залучаються усі співробітники компанії, а спеціально створена система мотивації сподвигает їх не лише на генерацію ідей, але і на їх подальше впровадження. Організація "банку ідей" дозволяє показати усю неефективність існуючих бізнес-процесів, піднявши її з рівня виконавців на рівень керівництва. І хоча при генерації ідей велика їх частина може виявитися "сміттям", навіть 10% ідей, придатних до впровадження, можуть дати серйозний ривок оптимізації бізнес-процесів в компанії.

## 2.4. Регламент як основа процессного підходу

Як вже говорилося, при впровадженні процессного підходу необхідно спочатку визначити існуючий бізнес-процес, а після його зміни впровадити його через регламент або інформаційну систему в діяльність.

Фактично одним з інструментів впровадження процессного управління на практиці є розробка, узгодження, твердження, розсилка і контроль виконання регламентів бізнес-процесів, адже саме регламент містить опис бізнес-процеса "як повинно бути" і є нормативним документом, відповідно до якого учасники бізнес-процеса повинні взаємодіяти між собою.

І хоча опис бізнес-процеса у вигляді графічної моделі є найбільш якісним, якщо дотримуватися певних правил і структури, то можна описувати бізнес-процеси і в простому текстовому редакторові, іноді додаючи графічну схему бізнес-процеса у вигляді додатка до регламенту.

Існує безліч прикладів різних регламентів, деякі з яких досягають сотні листів. Проте чим більше за об'ємом регламент бізнес-процеса, тим менше шансів, що його прочитають, і ще менше - що стануть виконувати. Саме тому краще відразу розробляти гранично короткий і "сухий" регламент об'ємом не більше п'яти листів тексту без урахування додатків.

На вичитування і узгодження регламенту піде немало часу управлінців, і тому чим "сухіше" буде текст документу, тим менше часу піде на його вичитування і обговорення.

На початку документу треба вказати мету регламенту, наприклад, встановлення порядку взаємодії при розрахунку заробітної плати або скорочення часу обробки заявки клієнта. Далі необхідно зробити розділ з термінами і скороченнями, які використовуватимуться в регламенті бізнес-процеса : наприклад, ДУП - Департамент управління персоналом або ЦФЗ - центр фінансового забезпечення.

Після термінів і скорочень необхідно розташувати загальну частину регламенту, куди розмістити усі положення, що стосуються бізнес-процеса в цілому, наприклад, система мотивування розробляється для керівників рівня директора департаменту. Окремим рядком у кінці загальної частини регламенту бажано прописати, для кого призначений цей регламент: наприклад, регламент призначений для співробітників ДУП і керівників ЦФЗ.

Ну і найголовніша частина, в якій міститиметься опис бізнес-процеса, що регламентується. Цю частину необхідно заповнювати максимально короткими фразами по наступній структурі. Хто і коли(фахівець з оплати праці, щотижня, до 18.00 п'ятниці). Потім, перевівши рядок і зрушивши нумерацію углиб, можна написати перелік дій, які виконуються фахівцем (погоджує систему мотивації для нових співробітників, передає систему мотивації директорові ДУП).

Таким чином крок за кроком описується увесь бізнес-процес з вказівкою переліку документів, які передаються по процесу, і інформаційних систем, які використовуються для виконання тієї або іншої операції. Якщо необхідно вказати комплект документів, що виникає в процесі, то це можна зробити або в загальних станах, або в тексті опису бізнес-процеса.

Як правило, навіть дуже масштабні бізнес-процеси можуть бути описані в такій структурі. При цьому треба мати на увазі, що короткий опис сценаріїв виконання бізнес-процесів, що рідко зустрічаються, можна винести в додаток або в загальні стани: наприклад, у разі потреби підготувати пропозицію в терміновому порядку керівник ЦФЗ формує робочу групу в складі. Такий підхід дозволить зосередитися на основному сценарії протікання бізнес-процеса.

Після узгодження і затвердження регламенту він стає нормативним документом, по якому повинні діяти учасники бізнес-процеса. Фактично регламент містить правила взаємодії підрозділів при роботі в тому або іншому бізнес-процесі.

Є у регламентів в українських умовах одне слабке місце - це необов'язковість їх виконання, викликана складністю контролю виконання усіх правил регламенту. Перевірити, наскільки співробітники працюють за регламентом, можна двома способами. У першому випадку це аудит з аналізом діяльності і її перевіркою на відповідність закладеним в регламенті процедурам. Другий варіант - це розбір конфліктів, що виникають у бізнес-процесі, інформація про яких передається власникові процесу від учасників, якщо один з них починає порушувати регламент на шкоду іншим. Другий варіант - це система самоконтролю виконання процесу, коли виконавець, одержуючий завдання у рамках бізнес-процеса, контролює якість роботи попереднього виконавця і відповідність його дій алгоритмам і тимчасовим обмеженням, закріпленим в регламенті.

Якщо в компанії немає ніякої культури роботи відповідно до затверджених регламентів, то доводиться притягати на допомогу інформаційні технології. І в цьому випадку доводиться "примушувати" людей працювати за правилами, прописаними в регламенті, автоматизуючи процес в інформаційній системі, щоб вона в автоматичному режимі контролювала відповідність усіх дій закладеним правилам.

Цей спосіб вимагає великих витрат, оскільки для автоматизації необхідно закупити софтвер і силами розробника створити ИТ-рішення, яке автоматизуватиме процес, не дозволяючи порушити його логіку і контролюючи терміни виконання окремих операцій і ефективність кожного з учасників процесів. Але після автоматизації бізнес-процеса власник отримає потужний інструментарій, який дозволить управляти бізнес-процесом з мінімальними втратами часу.

## 2.5. Короткі висновки

Процессное управління вирішує проблеми, що виникають в організаційних структурах, побудованих за функціональним принципом, за рахунок визначення і оптимізації наскрізних і кросс-функциональных бізнес-процесів в організації.

Підрозділом, який супроводжує впровадження процессного підходу в організації, є процессный офіс.

Співробітники бізнес-підрозділів повинні за підтримки процессного офісу описувати, аналізувати і удосконалювати свої бізнес-процеси.

Найбільш трудомістким є впровадження бізнес-процесів в практику діяльності компаній, що вимагає зміни не лише самих процедур, але і корпоративної культури в компанії.

Інструментом впровадження нових бізнес-процесів в діяльність організації є або організаційно-розпорядливі документи, такі як регламенти і стандарти, або інформаційні системи, в яких закріплюються усі правила маршрутизації завдань по бізнес-процесу, а головне - забезпечуються засоби контролю над параметрами виконання процесу.

**Питання до теми:**

1. Які підходи до управління використовуються в сучасних компаніях? Що таке процессное управління?
2. Які інструменти графічного моделювання бізнес-процесів ви знаєте?
3. Що таке iBPM- системи, для чого вони використовуються, наведіть приклади?
4. Які системи контролю бізнес-процесів існують, чим вони розрізняються?
5. Які зміни необхідно провести в організації з впровадженням нових бізнес процесів?

# Лекція 3: ПЕРЕДУМОВИ ВИНИКНЕННЯ AGILE

План

3.1. Стан області процесів розробки програмного забезпечення

3.2. Порівняння каскадного/ітераційного/Agile процесів

3.3 Ефективна пігулка від хвороб?

3.4 Для кого підходить, а для кого немає?

3.5. Короткі висновки

## 3.1. Стан області процесів розробки програмного забезпечення

Про те, наскільки все невтішно в області впровадження, розробки і супроводу програмних продуктів, не говорив тільки зовсім ледачий консультант.

Найбільш актуальні огляди і зібрана статистика авторитетних вітчизняних і західних видань демонструють, що визнані і найбільш відомі методології розробки програмного забезпечення не виправдовують покладені на них очікування.

Як приклад приведемо дослідження Standish group, проведене в 2015 році. Воно було виконане за результатами аналізу даних про впровадження програмних продуктів за період з 2001 по 2010 рік. Фундаментом дослідження послужила інформація по більш ніж 10 000 проведених проектів різної міри складності ( рис. 3.1).

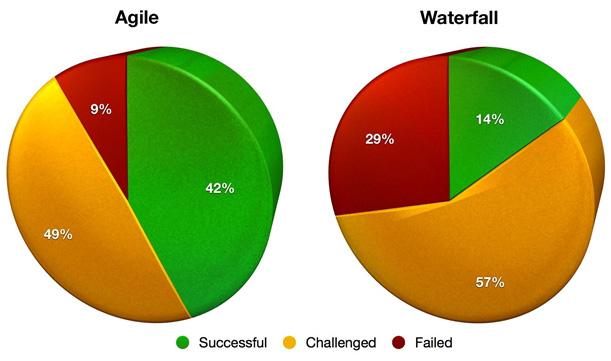


Рисунок 3.1 – Статистика проектів в розрізі процессных методологій за період з 2001 по 2010 рр.

На приведеній діаграмі продемонстровано, що задоволеність від використання класичного водоспадного підходу значно нижче порівняно із застосуванням Agile.

Загальний тренд очевидний. Сучасні компанії знайшли в Agile успішну альтернативу класичному підходу управління інформаційними технологіями.

Причини такого вимогливого відношення до самої сфери, а головне - до її результатів, досить повно розкриті в лекції "Введення в Agile" . На додаткову увагу заслуговує факт інтеграції, що підвищується, між інформаційними системами і усіма іншими напрямами життєдіяльності сучасного світу.

Скрутно представити напрям діяльності, в якому залежність від інформаційних систем була б незначна. Технологічні продукти, що автоматизують операційні процеси, виконують усі ті функції, які покладали на них в найфантастичнішому фільмі, створеному не більше 10 років тому. Фантастика стає реальністю. Інформатизація матеріалізується не лише у вигляді социотехнических систем, але і у вигляді всюди використовуваних сервісів.

Не варто сильно упиватися локальними перемогами. Належить ще попрацювати над багатьма глобальними питаннями, не вирішеними до сьогоднішнього моменту і виникаючими по ходу ери інформатизації, що настає.

Основні проблеми і складнощі, які явно можна виділити, породжуються наступними причинами:

* постійне ускладнення предметних областей, що автоматизуються інформаційними системами;
* швидкість змін, що впливають на структуру процесів;
* незадовільні способи опису поведінки комплексних інформаційних систем для подальшої підтримки і їх розвитку;
* необхідність забезпечення достатньої гнучкості програмних продуктів;
* труднощі управління процесом розробки.

Проблеми і завдання, які ми намагаємося вирішити за допомогою програмного забезпечення, часто неминуче містять елементи, складність яких обумовлена мультиаспектностью і завищеними очікуваннями від їх реалізації цільових користувачів інформаційних систем.

Відносно рідко за функціоналом, що автоматизується, бачать попередню обов'язкову процессную складову, про яку було детально розказано в лекції "Процессный підхід як конкурентна перевага" . Тільки після того, як буде відлагоджена процессная складова, можна переходити до її подальшої автоматизації. Але це тільки в теорії. На практиці доводиться комбінувати організацію розрізнених, шматкових "процессиков" в єдиний архітектурний моноліт, від якості розробки якого залежатиме успішність напряму діяльності, що автоматизується. Але це тільки одне із завдань, яке належить враховувати в процесі розробки програмного продукту.

Сучасний інформаційний світ має ще одну сторону, яка робить величезний вплив на нього. Аспект швидкості розвитку, зміни, доповнення інформації, яка лежить в основі передумов створення програмних продуктів і автоматизації бізнес-процесів.

Високий темп змін призводить до того, що до відповідних програм пред'являється безліч різних, іноді взаємовиключних вимог.

Користувачі і розробники мають різні погляди на причини появи проблем. На основі різних передумов про суть виниклих завдань ці групи роблять різні висновки про можливі альтернативи їх рішення. Поетапна, инкрементальная розробка функціонуючих версій системи дозволить користувачам краще зрозуміти і ясніше сформулювати те, що їм дійсно потрібне.

В той же час ітераційний процес дозволить:

* поступово підвищувати кваліфікацію розробників в предметній області за рахунок тіснішої і продуктивнішої співпраці із стейкхолдерами системи, що розробляється;
* збільшувати терпимість бізнес-користувачів за рахунок того, що вони виразніше представлятимуть процеси розробки і складність питань автоматизації, які розробникам доводиться вирішувати щодня.

Ще одне завдання, яке постійно доводиться вирішувати групі розробників, - це проблема адекватного опису поведінки великих інформаційних систем.

У основі проектування кожної системи лежить принцип декомпозиції, який постулював розділення великої системи на частини так, щоб одна мінімально впливала на іншу, але при цьому зберігалася загальна цілісність усього продукту. Будь-яка зовнішня по відношенню до системи подія може привести її в новий стан. Проблематичність цієї ситуації полягає в невизначеності переходу між відомим і новим станами. Як правило, проблематичність знімається за рахунок грунтовного аналізу і подальшого усеосяжного тестування таких програм, але за несприятливих додаткових умов невелика невідома зовнішня подія може привести до критичної помилки у функціонуванні системи. Чим складніше система, тим легше її повністю розвалити. Завдання адекватного накопичення інформації і знань по системі і її оточенню з подальшим документуванням покликане попередити можливі "інформаційні" катастрофи.

Ще одним дієвим способом попередження проблем є "природна" гнучкість інформаційних систем, закладена принципами підходів до їх розробки.

*Програмування має граничну гнучкість, і кожен розробник може з потреби сам забезпечити себе усіма елементами, що відносяться до будь-якого рівня абстракції.* Така гнучкість надзвичайно спокуслива і дієва, якщо використати її за призначенням. Вона пропонує розробникові створювати усі базові блоки майбутньої архітектури інформаційної системи, з яких складаються елементи більш високих рівнів абстракції, але при цьому ця гнучкість має і свої недоліки, які виражаються різноманіттям можливих елементів і іноді їх дублюванням і надмірністю. На відміну від напрямів діяльності, що більше пропрацювали, в яких накопичений масив статистичних даних, на основі якого доцільно робити висновки на предмет того, як правильно організовувати конструювання і реалізацію базових артефактів, інформаційним технологіям, незважаючи на інтелектуальність і инновационность цієї сфери, тільки належить встати на централізований шлях стандартизації підходів до конструювання. На даний момент напрям, який включає визначення таких підходів(системна інженерія), ще тільки стає на цей шлях. Вже накопичені певні підходи, але вважати їх кінцевими і успішними неправильно. Способи взаємодії у навколишньому цифровому світі дуже різноманітні. Цей чинник не є каталізатором процесів інформаційної глобалізації.

Окрім перерахованих чинників залишається ще один аспект, вплив якого на створювані програмні продукти є таким, що пригнічує. Це процес управління розробкою інформаційних систем.

Основне завдання розробників полягає в створенні ілюзії простоти, в захисті користувачів від складності описуваного предмета або процесу. Сьогодні звичайними стали програмні системи, розмір яких обчислюється десятками тисяч або навіть мільйонами рядків на мовах високого рівня. Жодна людина ніколи не зможе повністю зрозуміти таку систему. Тому такий об'єм робіт зажадає залучення команди розробників. Чим більше розробників, тим складніше зв'язки між ними і тим складніше координація, особливо якщо учасники робіт географічно віддалені один від одного.

На рішення і управління перерахованими чинниками і спрямована Agile. Її застосування дозволить вирішити позначені проблеми.

## 3.2. Порівняння каскадного/ітераційного/Agile процесів

У першій лекції ми згадали про два протидіючі методологічні табори класичного і нового стилю роботи. Настав час обговорити їх детальніше.

До першого табору відносяться так звані важкі, класичні підходи до розробки програмного забезпечення. Ми коротко розповіли про них в ракурсі історичної ситуації виникнення Agile.

Зараз розглянемо їх суть, переваги, недоліки, необхідні умови і можливі результати їх застосування.

Першою за значимістю і поширеністю на сьогоднішній момент для процесів управління інформаційним технологіями являється найвідоміша на сьогодні водоспадна модель розробки інформаційних систем. У літературі часто зустрічаються альтернативні назви цього стилю розробки програмних продуктів - каскадний, класичний і ін.

Каскадна модель - це вид процесу розробки програмного забезпечення, в якій процес розробки виглядає як потік фаз аналізу вимог, проектування, реалізації, тестування, інтеграції і підтримки, що послідовно проходять.

Ця модель розробки була уперше презентована в 1970 році в статті Винстона Ройса. Він описав у вигляді концепції те, що зараз прийнято називати "каскадна модель", і обговорював її недоліки. У цій же статті він показав, як цей тип процесу розробки програмних продуктів може бути допрацьований до ітеративної моделі розробки ПО. У оригінальній каскадній моделі Ройса наступні етапи процесу були представлені в такому порядку:

* Визначення вимог.
* Проектування.
* Конструювання/реалізація/кодування.
* Втілення.
* Тестування/відладка/верифікація.
* Інсталяція.
* Підтримка.

Перехід між фазами можливий тільки після повного і успішного завершення попередньою. Наслідуючи запропоновану структуру, розробник переходить від однієї стадії до іншої послідовно. Спочатку повністю завершується перший етап("Визначення вимог"), в результаті якого з'являється повний і вичерпний список вимог до інформаційної системи. Тільки після цього можливий перехід до проектування, в ході якого розробляються документи, детально, ясно і що несуперечливий, що описують для програмістів спосіб і план реалізації зафіксованих вимог. Далі починається етап реалізації вимог у вигляді програмного коду системи. По суті, цей етап і є ядром розробки програмних продуктів. Його результати визначатимуть подальшу якість подальших етапів і інформаційної системи в цілому. Потім відбувається інтеграція окремих компонентів, що розробляються різними способами, в єдину систему. Після того, як перераховані етапи завершені, робиться тестування і подальша відладка продукту. Тут усуваються усі недоліки і помилки, що з'явилися раніше. Після цього програмний продукт впроваджується і здійснюється його підтримка - розробка нової функціональності, усунення помилок і т.і.

Каскадна модель будується на постулаті послідовних переходів від однієї фази розробки до іншої тільки після повного і успішного завершення попередньої. Переходи назад, повторні "стрибки" вперед, перекриття фаз - недопустимі.

Каскадну модель організації процесу розробки програмного забезпечення критикують за недостатню гнучкість, оголошення самоціллю формальне управління проектом на шкоду термінам, вартості і якості. Але її чітка структурованість і формалізація є безперечними цінностями, які сприяють зниженню багатьох виникаючих ризиків.

У PMBOK 3-ої версії формально була закріплена тільки методика каскадної моделі і не були запропоновані альтернативні варіанти, відомі як ітеративне ведення проектів або Agile.

Наступною за значимістю і поширенням є ітеративна модель розробки програмного забезпечення.

Ітеративний підхід в розробці програмного забезпечення - це виконання робіт паралельне з безперервним аналізом отриманих результатів і коригуванням попередніх етапів роботи. Проект при цьому підході в кожній фазі розвитку проходить цикл PDCA, що повторюється : планування - реалізація - перевірка - оцінка.

Ітеративні(чи инкрементальные) моделі використовують інший підхід до організації діяльності. Замість єдиної тривалої послідовності етапів застосовується розбиття життєвого циклу розробки продукту на набір окремих міні-циклів. Кожен з них включає ті ж базові стадії, вживані у водоспадній моделі. Міні-цикл - це ітерація. У кожній ітерації відбувається розробка окремого компонента або компонентів системи(інкремента). Далі реалізований компонент додається до вже існуючого функціонала.

Ітеративна модель, на противагу класичної моделі, не припускає повного об'єму вимог для початку робіт по реалізації продукту. Розробка програми повинна розпочинатися з основних вимог до базової частини функціонала. Далі, на подальших ітераціях, реалізовані вимоги доповнюватимуться, модифікуватимуться, призводячи до розширення функціонала системи. Процес повторюється, забезпечуючи створення нової версії продукту для кожної ітерації.

З PMBOK 4-ої версії був досягнутий компроміс між прибічниками каскадної моделі розробки ПО і професіоналами, що роблять ставку на ітеративні методи. Це привело до того, що починаючи з 2009 року Інститутом управління проектами(PMI) пропонується як стандарт гібридний варіант методології управління проектами, що поєднує в собі плюси як від водоспадної методології, так і від досягнення ітеративних методів.

Запорука успішного застосування цієї моделі - чітко побудовані етапи тестування/відладки/верифікації вимог і ретельна валидация функціональності, що розробляється, в кожній з ітерацій.

Ітеративна модель, по суті, є перехідною(від каскадної до Agile) моделлю розробки програмного забезпечення і, на думку багатьох фахівців, оптимальною моделлю розробки програмного забезпечення.

При порівнянні класичних(водоспадною і ітераційною) методик з Agile потрібне усвідомлення того, що каскадна модель - це підхід добре описаний і деталізований, а Agile - це набір практик і принципів, в яких так чи інакше підтримуються різні методології гнучкої розробки проектів.

У останній версії PMBOK вже йде мова про итеративно-инкрементальном підхід як про основний рекомендований. PMI в процесі розробки власної Agile- сертифікації.

Порівнюючи ці підходи, слід зазначити, що обоє мають набір переваг і недоліків. Можливі ситуації, коли обома методами можна організувати реалізацію необхідного програмного забезпечення, але на старті процесів розробки вхідні дані за ресурсними параметрами(вартість, час, кваліфікація персоналу), а також вимоги до якості інформаційної системи і т. д. істотно впливають на вибір методології.

В якості сильних сторін водоспадної моделі слід виділити наступні:

* легкість для розуміння і подальшого застосування;
* детальна структурованість, що полегшує її застосування до малодосвідчених команд;
* модель із самого початку задає стабільні вимоги до проекту/продукту;
* проекти легко контролюються, відстежуються ресурси, риски, час;
* якість має першочерговий пріоритет в порівнянні з вартістю і часом.

Agile, у свою чергу, має наступні сильні сторони:

* ітеративний підхід до розробки програмного забезпечення;
* використання чітких часових рамок;
* зацікавлені користувачі залучені в процес розробки із самого початку;
* швидке отримання першої/пробної версії продукту для тестування;
* легко сприймаються коригування і зміни в процесі розробки.

Слабкими сторонами підходів є наступні:

"Каскад":

* вимоги мають бути визначені і детально описані до початку стадії розробки;
* висока ціна;
* повільний темп роботи;
* чутливість до змін;
* мало можливостей для кінцевого користувача вплинути на цілі проекту і вимоги до продукту;
* частенько проблеми виявляються тільки на етапі тестування;
* багато документації, яка незрозуміла кінцевому користувачеві або замовникові.

Agile:

* може привести до низької якості продукту.
* існує ризик ніколи не досягти поставленої мети при ініціації процесу;
* можуть виникнути проблеми з розширюваністю продукту.

На основі приведених чинників стає можливим зробити доцільні висновки про ситуації, коли використання того або іншого підходу є оптимальним.

Класичні моделі прийнятніше використати в ситуаціях, коли вимоги до продукту гранично ясні і стабільні, визначені використовувані технології і інструменти, йдеться про впровадження великого і складного програмного забезпечення. Прикладом може служити проекти впровадження ERP- систем.

Agile слід застосовувати в тих випадках, коли кінцевий користувач залучений в проект із старту, визначені бизнес-цели проекту/продукту, проект невеликий або середній, відносно короткий за часом, склад команди стабільний, з високим рівнем професіоналізму, технічні вимоги прийнятні, пов'язані з технологіями, які збираються бути використаними для розробки, а інформаційна система за своєю суттю є модульною.

Представлені підходи мають плюси і мінуси, кожен з яких прекрасно підходить для застосування в проектах з абсолютно різними початковими даними і вимогами.

При виборі методології необхідно вибрати ту, яка підходить для досягнення поставлених цілей проекту. Необхідно розуміти структуру, принципи, переваги і недоліки кожної з них. В деяких випадках це не вибір між методологіями, а правильна комбінація підходів для кожного з етапів конкретного процесу або проекту.

За останній час змінився ринок споживачів ПО. Замість великих проектів компанії прагнуть до використання маленьких і середніх проектів. Компанії - розробники ПО прагнуть до швидшого, частому, регулярному випуску інформаційних систем.

Згідно з недавнім опитуванням, 80% рішень про впровадження в компанії Agile- методологій і ідей, належать менеджерам вищої і середньої ланки.

## 3.3 Ефективна пігулка від хвороб?

Ітеративний і Agile методи розробки програмних продуктів підходять не усім. У усіх компаніях існує своя специфіка, яка визначається безліччю різноманітних чинників різної природи, визначуваної організаційними, адміністративними і спеціалізованими умовами зовнішнього і внутрішнього середовища. У ряді компаній використання водоспадної моделі обгрунтоване і економічно доцільно. Можна привести багато причин, чому Agile використати не варто.

Розробка якісної інформаційної системи - справа непроста. Кожен, хто обіцяє реальність "чарівного зілля", що гарантує випуск програмного продукту, вирішального усі проблеми і знімаючого "біль" компанії, - шарлатан, що продає плацебо.

Консультанти, що спеціалізуються у впровадженні і просуванні Agile, насамперед заявляють, що Agile - не чарівна куля, магічним чином що прибирає усі проблеми, але частину часу, що залишилася, вам говорять, що цей кращий винахід людства.

Agile - це набір сталих методик, заснований на ітеративній розробці програмного забезпечення.

Звичайно, є література, книги, експерти, конференції і т. д., але слід пам'ятати, що це усього лише сталі методики, які базуються на:

* змінах;
* пристосовності;
* людях.

Методологія Agile, незважаючи на велику кількість існуючої літератури, має строго практичну спрямованість. Не вийде просто прочитати декілька статей, вивчити книгу, пройти курс і несподівано стати гнучким. Важливо паралельно з освоєнням теоретичних азів осягати практичні прийоми роботи. Agile дає численні переваги робочим групам і невеликим проектам. Це є основою для обговорення переваг, які вони можуть надавати для розробки великих модульних систем.

## 3.4 Для кого підходить, а для кого ні?

У співтоваристві ИТ-менеджменту Agile - одна з найпопулярніших тим вже упродовж довгого періоду часу. Відбувається це із-за хвилеподібного попиту на підвищення ефективності процесів розробки в різних сегментах діяльності. Думки із цього приводу самі різні.

При обговоренні необхідно враховувати передумови, які демонструють переваги і недоліки, грунтуючись не стільки на окремих оцінках шанованих експертів, а на фактах, що виникають в процесі експлуатації Agile різними учасниками процесів. Саме зважені думки учасників процесу дозволять встановити істину, народжену в дебатах.

Agile, незважаючи на демократичність, має на увазі наявність жорсткої дисципліни. Це може сприяти ефективнішій організації праці і, як наслідок, максимізації отримуваного результату. Але це дечого коштує.

Для поширення необхідної інформації по процесу в Agile пропонується для виконання низка необхідних заходів. У них на постійній основі повинні брати участь не лише фахівці, зайняті розробкою програмного забезпечення, але і менеджери, активна участь яких потрібна і багато в чому зумовлює подальший успіх створюваного продукту.

Якщо менеджер не зможе брати участь в пропонованих гнучкою методологією зустрічах, то необхідно замислитися про делегування відповідних повноважень в розвитку системи, що розробляється, колезі або підлеглому, який зможе вдумливо, наполегливо, ясно і з повним усвідомленням суті завдання донести її до команди і згодом прийняти створений результат. Інакше суть Agile буде вихолощена і принцип "першого керівника", на думку багатьох, найдієвіший при впровадженні і застосуванні різних процесів, знівелює. Як наслідок, Agile не досягне постулируемой ефективності, і програмний продукт створюватиметься розробниками для розробників.

Код створюваної системи має бути простим і читаним, щоб його можна було легко міняти, причому у будь-який момент і різним фахівцям. Для цього вимагається постійно критично його переглядати і займатися рефакторингом розробленої, але застарілої функціональності. Важливо мати людину, яка зможе критично підійти до актуальної ситуації і при необхідності позначити потрібні доопрацювання в цілях полегшення подальшої сопровождаемости і розвитку інформаційної системи

Для того, щоб тримати планку заданої якості, вимагається створювати юнит-тесты. Це призводить до того, що на розробника вішається удвічі більше роботи. Але якщо цього не робити, то підвищується вірогідність випуску низькоякісного продукту, який не відповідатиме очікуванням користувачів.

Для того, щоб продукт, що розробляється, був цілісним з точки зору концепції його розвитку, необхідно постійно обговорювати питання аналізу, проектування, виконувати контроль діяльності менш досвідчених розробників, відшукувати помилки. Одна з техніки, яка може дозволити виконати намічені завдання, називається парним програмуванням. Його суть полягає в парній, почерговій розробці коду двох розробників в одиницю часу за одним робочим місцем. Це дуже важке випробування для тих, чий рівень комунікаційних здібностей знаходиться на досить низькому рівні. Уявіть собі, що хтось сидить поряд з вами і постійно вказує на те, як, на його думку, правильно реалізувати ту або іншу ділянку створюваної інформаційної системи. Більше того, іноді(а вірніше, дуже навіть часто) вимагатиметься безпосередньо спілкуватися із замовниками функціонала. Agile дозволить створювати працюючий продукт з високою якістю в прогнозовані терміни. Єдина важлива умова - замовник має бути зацікавлений в створенні працюючого продукту з високою якістю в прогнозовані терміни.

Більшості розробників цікаво реалізовувати складний, "хитромудрий" модуль, але нецікаво писати простій, але надійно працюючий продукт. Розробники зацікавленіші в захоплюючому дослідницькому процесі, а що відбуватиметься із замовником у момент використання розробленої функціональності клієнтом - їх, як правило, не цікавить.

Фахівці цінують свій комфорт за комп'ютером. Робоче місце - рідний будинок і незаперечна зона комфорту. Менеджер, відповідальний за процес, повинен змінити ситуацію, коли власний комфорт важливіший, ніж працюючий софтвер, успіх компанії і т. д. Вимагається постійно доносити до відповідальних фахівців необхідність і потрібність їх роботи, її вплив на процеси компанії.

Agile пропонує зміни в підходах не лише до роботи, але і до сталих звичок, якщо вони суперечать спільним цілям. Такий процес не лише постулював бажання і готовність до постійних змін і "ломок" характеру і професійних засад. Він привносить мотивацію, якою навряд чи можна чекати в компаніях, що проповідують класичні підходи до процесів розробки програмного забезпечення, як, наприклад, техніка парного програмування, про яку ми поговоримо пізніше.

Підсумком стає те, що для ефективного впровадження Agile потрібний ряд чинників, які зможуть забезпечити його оптимальне застосування в компанії:

* зацікавленість керівництва в досягненні поставлених результатів;
* бажання і здатність брати участь в розвитку продукту;
* наявність кваліфікованої команди розробників, здатної створювати якісний продукт і аналізувати результати і існуючий процес в цілях його оптимізації;
* готовність до відкритого діалогу і конструктивної критики усіх членів команди;
* бажання змін і змін.

У подальших лекціях ми доповнимо, розкриємо і деталізуємо приведений список чинників.

## 3.5. Короткі висновки

Ситуацію, завдяки якій Agile отримала право на існування, важко назвати ідеальною. Стан області розробки і впровадження програмного забезпечення складно назвати стабільною і такою, що відповідає очікуванням зацікавлених в її використанні. Agile виникла не як академічна дисципліна, а як відповідь на неспроможність ринковим реаліям її конкурентів. Не просто виник, а став результатом вдумливого процесу роздуму визнаної групи консультантів і експертів над проблемами в області управління розробкою програмного забезпечення і вже упродовж довгого періоду показує свою спроможність і результативність.

Усе це говорить про те, що Agile завоювала свою аудиторію і стала популярною. В той же час як і раніше ще слабо представляється, як правильно працювати по Agile. Приведемо слова Майкла Кона : "Мені хочеться, щоб усе Agile- бренди кінець кінцем зникли, і залишилося просто те, як ми розробляємо програмне забезпечення".

Можна і треба порівнювати різні напрями Agile, такі як Scrum або Kanban, зосереджуючись на технічній досконалості кожного з них, або ж, навпаки, розвивати комунікації і командну роботу. Усе це зрештою приведе до вдосконалення і розвитку Agile, підвищення ефективності залежно від умов, в яких її використовують.

**Питання до теми:**

1. Які сильні і слабкі сторони водоспадної моделі ви можете назвати?
2. Які переваги і недоліки має Agile?
3. Що таке ітеративний підхід?
4. Дайте визначення каскадній моделі.
5. Які чинники потрібні для ефективного впровадження Agile?

# Лекція 4: ФІЛОСОФІЯ РОБОЧОГО ПРОЦЕСУ

План

4.1. Типи Agile- методологій і їх поширеність

4.2. Scrum - гнучкий управлінський процес

4.3. Як виховувати співробітників?

4.4. Як управляти опором?

4.5. Чим треба управляти в Scrum

4.6. Значущість дотримання процесу

4.7. Інженерні практики

4.8. Короткі висновки

## 4.1. Типи Agile- методологій і їх поширеність

Після того, як Agile завоювала певну нішу в області процессного управління сфери інформаційних технологій, стало очевидно, що для подальшого розвитку потрібні конкретні практичні підходи, адаптовані під "бойові" умови конкурентного середовища функціонування підприємств.

Agile - набір принципів і методик, які необхідно вбудувати в діяльність компанії.

Кожна організація має свої неповторні риси, свій рівень зрілості процесів, свої умови, в яких вона домагається поставлених цілей. Використати універсальні принципи для компаній з низьким рівнем зрілості і для організацій, що займають в сходах CMMI найвищі позиції, було б недалекоглядним рішенням. Agile не має на увазі покрокового керівництва і конкретних рекомендацій, а лише своїми принципами задає рамки, в яких використовують гнучкі методології.

У цій лекції ми обговоримо ряд гнучких методологій розробки програмних продуктів, що використовуються при створенні програмних продуктів і не лише ( рис. 4.1).

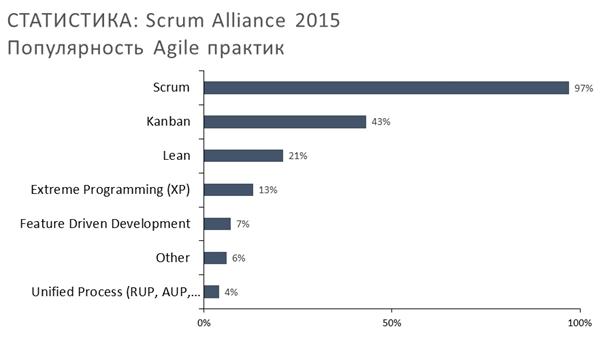


Рисунок 4.1 –Популярность Agile-практик, 2015 г.

**Kanban**

Це яскравий представник процессной методології, яка з'явилася завдяки японському післявоєнному економічному диву у сфері машинобудування і поступово була запозичена в сферу інформаційних технологій.

У основі Kanban лежать три базисні принципи:

* Візуалізація(ієрогліф "кан"). При ілюстрації і моделюванні процесу він розбивається на окремі стадії(аналіз, проектування, розробка, тестування і т. д.), спрощуючи таким чином його сприйняття.
* Обмеження максимальної кількості завдань на певному етапі. Цей принцип дозволяє звести втрати до мінімуму - максимальне зосередження на своїх завданнях.
* Оптимізація існуючого процесу. Час на виконання завдання відстежується, аналізується, і виробляються пропозиції про те, як можна виконати роботу більше абсолютно. У процесі не повинно бути простоїв, так само як і не повинна виконуватися непотрібна робота. Kanban характеризується твердженням: "Зменшення роботи", що виконується в даний момент. Цей тип методології являється, мабуть, найгнучкішим. Це означає, що вона є найбільш вимогливою до умов і ресурсів, у рамках яких передбачається її експлуатувати. Співробітники, працюючі по Kanban, мають бути готові до екстремальної гнучкості і при цьому не повинні ламатися.

Унікальність методології Kanban, порівняно з конкурентами по сімейству Agile, полягає в:

* способі розподілу завдань. Кожен фахівець, працюючий в команді розробників, може узяти на себе лише обмежену кількість, при цьому вибір завдань він здійснює самостійно, а не за чиєюсь вказівкою;
* відсутності часових рамок. Kanban не припускає обмеження на час виконання поставлених завдань;
* розмірі завдань, які необхідно реалізувати. Порівняно з аналогами завдань менше, але їх об'єм і трудомісткість значно більше;
* відсутності активності оцінки і планування. Оцінки термінів на завдання: опциональные або взагалі їх немає.

Вигода від застосування приведених постулатів дуже спірна. Як виконувати контроль над розробкою, якщо ми прибираємо основні інструменти контролю - терміни, швидкість роботи і т. д.? Відповідь на приведене питання ми залишимо на пізніші обговорення, які приведемо в наступних лекціях.

Менеджер повинен піклуватися про контроль і зробити все для його організації. Існує точка зору на контроль як на фікцію. Обгрунтовується вона тим, що команда повинна отримувати задоволення від роботи і викладатися на повну котушку. Звучить перспективно, але за фактом співробітників, яким можна було б довіряти повністю, не так багато. А вірніше сказати, їх одиниці.

Kanban орієнтована на завдання. Співробітники працюють над завданнями із самого початку і до завершення. Робоча команда не повинна оцінювати час на виконання завдання, бо це має мало сенсу і майже завжди помилково спочатку. Якщо менеджер вірить команді, то навіщо мати оцінку часу?

**Lean**

Цей тип методології має на увазі створення продукту в умовах максимальної економії ресурсів з метою усунення усіх можливих втрат. Первинний функціонал продукту обмежується до мінімально корисного. Таким чином, функціонал реалізується шляхом поетапного нарощування функціонала невеликими порціями, схожий на инкрементальному принцип реалізації інформаційних систем. Корені цього підходу відносяться до принципів бережливого виробництва(Lean Manufacturing). Мері і Том Поппендик, про яких було згадано в лекції "Введення в Agile" курсу, адаптували ці принципи для розробки програмного забезпечення:

* усунення втрат;
* підвищення якості;
* створення знань;
* відстрочені зобов'язання;
* швидке постачання;
* повага людей;
* повна оптимізація.

Lean постулював відмова від всього, що не додає цінності створюваній інформаційній системі. Розробляти необхідно тільки те, в чому є абсолютна упевненість, що це треба робити зараз. Усунення втрат в усіх аспектах роботи(даремні збори, надмірні завдання, документація, неефективні способи роботи і т. д.). Акцент на те, що називається "Системним підходом", тобто співробітники працюють як єдине ціле, як команда. Потрібне верхнеуровневое усвідомлення того, що виконувана робота допомагає підвищувати цінність створюваного продукту, порівняно з аналогами. Не треба примушувати людей працювати на 150% їх робочого часу. Не треба кодувати те, що не треба. Додатковий функціонал створює додаткові зобов'язання для користувачів і керівництва. Співробітники потребують поваги як особовому, так і професійному. Треба давати їм ту роботу, яку вони краще всього знають, як потрібно зробити. Сенс програмної розробки в постійному навчанні. Ухвалення управлінських рішень треба виконувати в останній можливий момент. До моменту реалізації необхідного функціонала співробітники знатимуть вже більше і краще орієнтуватися не лише в створюваній системі, але і у бізнес-процесах і даних організації.

**Екстремальне програмування**

Авторами цієї методології є Кент Бек, Уорд Каннингем, Мартін Фаулер. Саме про цих людей ми говорили, коли в лекції "Введення в Agile" робили історичний відступ про авторів Agile. Методологія дістала свою назву завдяки втіленню ідеї застосувати корисні класичні методи і техніку розробки програмного забезпечення, піднявши їх на якісно новий "екстремальний" рівень. Приміром, етап ревізії коду, що полягає в аудиті одним програмістом коду, написаного іншим програмістом, в "екстремальному" варіанті є "парним програмуванням". Таким чином, ми отримуємо робочий процес, в ході якого один програміст займається кодуванням, а його напарник в цей же час безперервно переглядає тільки що написаний код. Через деякий час вони міняються ролями.

Якщо підвести локальний підсумок розглянутих відгалужень гнучких методологій в цілому, то головним мінусом стане "плаваюча" оцінка термінів розробки і бюджету. До безперечних плюсів варто віднести малі терміни виробництва продукту, відсутність простоїв на час узгодження проектної документації, оперативне управління змінами і ін.

Остання, така, що залишилася до розгляду і, мабуть, найбільш поширена методологія з сімейства Agile - Scrum.

За результатами актуальних досліджень і опитувань, саме Scrum є найпопулярнішою технікою управління процесами розробки програмного забезпечення.

**Scrum**

У Scrum вимоги розбиваються на невеликі підгрупи, кожна з яких має бути максимально незалежна від іншої. Це робиться для того, щоб впродовж кожного нового спринту(проміжку між випуском нових версій продукту) команда розробників могла реалізувати новий функціонал. Scrum має на увазі постійну комунікацію - будь то планування чергової ітерації робіт спільне з клієнтом або щоденний Scrum- мітинг, впродовж якого команда обговорює низку найбільш важливих запитань, відповіді на які безпосередньо впливають на створюваний програмний продукт. Саме на прикладі опису і детального розгляду Scrum надалі вестиметься мова про Agile.

## 4.2. Scrum - гнучкий управлінський процес

Принципи ітеративної розробки програмного забезпечення припускають дії з певних ітерацій. Невеликі шматочки, кожен з яких має бути закінченою функціональністю, виконуються відповідно до поступового нарощування додаткових якісніших і деталізованих атрибутів автоматизації продукту, що розробляється, мають бути реалізовані в певних часових рамках. Суть полягає в тому, що ви постійно, циклічно робите усе необхідне для викладення чергової версії продукту. Продукт стає якісніше, а команда співробітників, розробляюча інформаційну систему, набуває досвіду на усіх етапах розробки і з кожною наступною ітерацією починає все краще розуміти істинні потреби бізнес-користувачів. Результатом кожної ітерації є інкремент.

Scrum - це стандартизований варіант організації процесу розробки програмного забезпечення для ітеративної і инкрементальной розробки. Для роботи по Scrum потрібний набір артефактів, за допомогою яких і виконується управління і напрям створюваного продукту.

Scrum із самого початку створювався для управління процесами контролю, планування і аналізу на усіх етапах створення інформаційної системи. Завдяки такому підходу до розробки він користується високою популярністю у команд, що займаються підтримкою і/або супроводом програмних продуктів.

Серед множини інших, однією з особливостей цієї методології слід виділити щоденні короткі наради, під час яких йде обговорення результатів попередніх ітерацій(спринтів) і ставляться завдання на наступні. Це дозволяє зацікавленим менеджерам тримати під контролем процес розробки, при необхідності направляючи його в потрібне русло.

Достоїнства Scrum:

* Scrum орієнтований на клієнта, адаптивний.
* Scrum дає клієнтові можливість робити зміни у вимогах у будь-який момент часу, але не гарантує те, що ці зміни будуть виконані.

Можливість зміни вимог приваблива для багатьох замовників:

* Scrum досить простий у вивченні, дозволяє економити час за рахунок виключення некритичних активностей.
* Scrum дозволяє отримати потенційно робочий продукт у кінці кожної ітерації робіт.
* Scrum робить упор на багатофункціональну команду, що самоорганизующуюся, здатну вирішити необхідні завдання з мінімальною координацією.

Це особливо привабливо для малих компаній, оскільки позбавляє від необхідності найму або навчання спеціалізованого персоналу дорогих керівників.

Scrum містить і важливі недоліки. Зважаючи на простоту і минималистичности, Scrum задає невелику кількість досить суворих правил. Проте це вступає в конфлікт з ідеєю "клієнт завжди прав", оскільки замовникам не важливі внутрішні правила команди розробки, особливо якщо вони його обмежують.

Так само загальновідомою проблемою Scrum є великі витрати на обговорення, зустрічі і великі втрати часу на стиках спринтів. Як мінімум день йде на закриття чергового спринту, а потім день - на відкриття нового. В умовах, коли спринт триває 2 тижні, 2 дні складають 20% робочого часу. Підсумок - 30-40% часу при застосуванні Scrum витрачається на підтримку самого процесу(щоденні зустрічі, планування і деталізація робіт, презентація отриманих результатів і ін.)

Scrum відноситься до сімейства Agile, в якому не вітається створення планів комунікацій, реагування на риски, управління складними технічними змінами. Підсумок - складна, а іноді неможлива формальна(юридичне або адміністративне) протидія порушенням правил Scrum. Звичайно, недбалого співробітника можна просто звільнити, але це не є системним способом вирішення проблеми. Процеси в компанії мають бути побудовані так, щоб, залишаючи певний ступінь свободи співробітникові, завжди можна було б контролювати його не лише кількісно, але і якісно. Для цього є безліч техніки і методик, застосування яких характеризує зрілість компанії в цілому і її процесів зокрема. Усі компанії, що лідирують в різних сегментах, характеризуються низьким рівнем увольняемости. Ротація співробітників потрібна для кожної організації, але ротація не має бути масовою. Коли йдуть ключові співробітники, організація ставить себе під удар. У Scrum цей недолік спробували здолати за рахунок постулату універсалізму і упор на багатофункціональну команду, що самоорганизующуюся. При уявному зниженні витрат на координацію команди це призводить до підвищення витрат на відбір персоналу, його мотивацію, навчання. У певних умовах ринку праці, формування повноцінної, ефективної Scrum- команди може бути неможливим. Хороші співробітники коштують дорого, і керівництво організації повинне віддавати собі в цьому звіт.

Фокус Scrum - управління продуктом в умовах невизначених і часто таких, що міняються вимог, які характеризують сучасний нестійкий інформаційний світ.

## 4.3. Як виховувати співробітників?

Після того, як менеджментом компанії буде прийнято рішення використати Scrum в операційній діяльності, необхідно організувати його впровадження. Найбільш важливим аспектом в питанні впровадження є адаптація співробітників компанії до нового робітника процесу. Адаптація буде ефективною у тому випадку, коли зміни піднесені не як нове управлінське віяння, а як необхідність. Організації, в яких подібний підхід до процесів і активностям управління змінами організований на високому рівні зрілості, мають значну перевагу перед конкурентами за рахунок їх гнучкості і адаптивності під умови зовнішнього і внутрішнього бізнес-середовища, що постійно міняються.

Причини, по яких співробітники стають лояльними до організації, її цілей і завдань, продовжують працювати на її благо, украй різноманітні і визначаються багатьма чинниками. Продовжувати міркувати про важливість розділення співробітниками місії компанії недоцільно.

Про важливість організаційного менеджменту написано досить тямущої літератури. Зосередимося на особі працівника.

Якщо компанія зацікавлена в співпраці з певними працівниками, їй треба уміти продумати, іноді індивідуально, як утримувати того або іншого "унікуму". Для цього можна виділити сегменти респондентів відповідно до тривалості роботи, проаналізувати відповіді ветеранів для встановлення можливих закономірностей. За результатами опитування обговорюють шляхи продовження роботи, виявляють можливі проблеми.

Загальні рекомендації, використовуючи які можна врахувати 90% можливих проблем, полягають в управлінні наступними чинниками:

* "Виховання" ефективності. Кожен співробітник повинен мати перед собою важкі, але досяжні цілі, виражені у вигляді вдосконалення або розвитку навичок корисних для компанії і цікавих самому працівникові. Ці цілі підкріплюються у вигляді чітких і зрозумілих планів розвитку, декомпозированных на конкретні завдання(розширити компетенції у бізнес-процесах компанії, освоїти новий інструмент по роботі з БД і т. д.), які повинні мати чіткий термін і метрику.
* Планування складних робочих завдань. Робочі завдання слід планувати так, щоб вони були зрозумілі співробітникові і практично підкріплювали розвиток його навичок. Таким чином, ми отримуємо чітку залежність між його ефективністю, здатністю і бажанням вирішувати завдання, необхідні для досягнення цілей компанії.
* Синхронізація розуміння між керівництвом і співробітниками про завдання розвитку і шляху досягнень результату конкретних завдань. Керівництво і працівники повинні знаходитися в стані відкритої, чесної комунікації. Довіра призводить до усвідомлення безпеки. Безпека - це один з щонайперших інстинктів, задоволення якого призводить до створення комфортної робочої обстановки. Вибудовування довіри і узгодженості розуміння в діях, необхідних для досягнення поставлених цілей, - одне з основних завдань керівництва на шляху отримання довгострокового результату від діяльності співробітників.
* Командна робота. Працівники можуть почувати себе ізольовано і незадоволено, якщо вони не є частиною згуртованої команди або якщо ними таємно "жертвують" з тих або інших причин. Ця проблема вирішується за рахунок створення команд різної величини, які мають бути саморегулируемы. Саморегуляція досягається, якщо члени команди розуміють значущість командної роботи і необхідність досягнення результатів.
* Філософія лідерства як новий тип управління командною роботою. При впровадженні змін важливим рушійним чинником являється наявність лідера, який власним прикладом зможе продемонструвати переваги, які будуть отримані командою від застосування цих змін. Цей лідер повинен мати досвід, бажано успішний, роботи по впроваджуваних процесах. Можна дуже довго і красиво розповідати про те, як буде добре, але співробітникам важливо мати колегу, який зможе підказати, як правильно і оптимально поступити в тих або інших робочих умовах. Це є каталізатором швидкого досягнення результатів від впровадження. Інакше ризик "блукання" по лабіринтах процесів і збільшення часу на адаптацію збільшується у декілька разів.

У результаті витікає, що для виховання робочої філософії рекомендується наслідувати наступні основні принципи:

* Вибирайте співробітників "точково". Вибір конкретних працівників має бути взаємозв'язаний із завданнями, які стоять перед компанією зараз і повинні виникнути в найближчій перспективі.
* Виховуйте в співробітниках ефективність або розлучайтеся з ними. Співробітники і компанія повинні рости і розвиватися разом. Якщо співробітник прагне освоювати нові навички і при цьому використовує їх в роботі, то саме це той тип працівника, який повинен стати частиною локомотиву майбутніх перемог організації. Грамотні виконавці, які якісно виконують свої функціональні обов'язки, теж потрібні, але вони не сприятимуть розвитку компанії, вони сприятимуть її стабілізації у вже зайнятій ринковій ніші.
* Співробітникам потрібна можливість навчатися як на робочих місцях, так і в процесі роботи пілотних проектів. Після прийнятого рішення про зміни в процесах компанії слід розпочати з пілотних проектів оптимізації тих активностей, в яких є лідери, за якими зможуть почати тягнутися інші. Сприяйте тому, щоб ваші співробітники мали можливість отримати необхідні знання і інформацію про зміни.
* Потрібний досвід і знання колег, вже Scrum, що впровадили. Лідери зміни повинні на практиці знати те, про що говорять в теорії.
* Необхідно постійно інформувати людей про зміни, що сталися. У компанії має бути організована система сповіщень і атмосфера довірчої комунікації про зміни, що відбуваються. Не нехтуйте "сарафановим" радіо, яке можна організувати через кадровий департамент або аналогічну службу. Доносьте інформацію навіть про самі незначні перемоги, означайте виниклі проблеми і шляхи їх рішення.

Підводячи локальний підсумок теми управління і розвитку кадрів в Scrum, треба сказати сокровенне: "Кадри вирішують все". Саме співробітники і їх навички формують основну цінність будь-якої компанії. А коли мова заходить про гнучкі методології, то значущість твердження, приведеного вище, зростає у декілька разів. На потенціалі і працьовитості команди триматиметься якість програмного продукту. Кадрами належить займатися дуже багато і копітко, але ця робота окупається.

## 4.4. Як управляти опором?

Переваги і професійні "засади" кожного фахівця формуються відповідно до широти його поглядів на світ і шаблонності професійного мислення. Як правило, працівники намагаються несвідомо чинити опір змінам. Будь-ким. Чим серйозніше і глобальніший зміни, тим запекліші опори. Якщо йдеться про корінну зміну робочої філософії і перехід на гнучкі методології розробки програмного забезпечення, то перебудови кардинальні.

Опір прийнято знімати за рахунок жорсткої політики переходу, що називається вибухом. Коротко, вона полягає в нав'язуванні співробітникам нової реальності. "Нав'язування" визначається конкретною культурою управління організацією. Керівник має бути обережним і уважним при ухваленні будь-яких рішень, пов'язаних з проведенням змін, але при цьому необхідно рішуче дотримуватися прийнятих правил. Навіть якщо вони комусь не подобаються.

Для того, щоб зрозуміти, як реагувати на можливий опір, корисно зрозуміти, на якому рівні виникають опори:

* на організаційному рівні;
* на рівні групи;
* на рівні індивіда.

Від розуміння того, на якому рівні виникає опір і чим воно характеризується, виникає розуміння способів усунення проблем.

На кожному з приведених рівнів є свої особливості опору і свої прийоми дії з метою зменшення опору.

На організаційному рівні структурні і культурні чинники можуть сприяти широкому поширенню опору. Існуюча структура і культура не можуть оперативно пристосуватися до нових вимог і змінитися. Причини полягають в тому, що для пристосування до змін в масштабі підприємства потрібний тривалий інтервал часу і великі ресурси.

Один із способів зменшення опору - системний підхід. Проте складність полягає в тому, що для розуміння поведінки організації як системи необхідно враховувати поведінка усіх взаємозв'язаних в організації груп. Системний підхід передбачає розгляд організації як єдиного цілого, виявлення взаємозв'язків між різними частинами системи, наприклад, шляхом зміни ієрархічного порядку ухвалення рішень або забезпечення деякої рівноваги між соціальною, організаційною, технічною і іншими частинами системи. Якщо у компанії немає ресурсів на застосування системного підходу до змін, найдієвішим способом являється гасло: "Можеш - починай працювати, не можеш - шукай для себе роботу". Якщо йдеться про співробітників з невисокою кваліфікацією, то, можливо, ця стратегія управління змінами має право на існування. Якщо ж йдеться про співробітників, які своїми діями формують інтелектуальну власність компанії, виражену у вигляді автоматизованих процесів і унікальних даних, адаптованих під конкретні умови функціонування організації, то цей підхід може привести до катастрофи.

При здійсненні змін необхідно мати на увазі, що компанія як система включає не лише формальні групи(управління, відділи, сектори і т. д.), але і неформальні, наприклад групи "ветеранів". Широке освітлення стратегічного задуму і консультації перед здійсненням стратегії(у ідеалі - на стадії проектування змін) можуть допомогти зменшити опір з боку груп і виявити, що ж дійсно турбує людей в запропонованій ситуації. Для цього може знадобитися:

* інформування про результати організаційної діагностики тих підрозділів і груп, які безпосередньо зачіпає зміна;
* проведення семінарів і дискусій, в яких би брала участь конкретна група співробітників;
* організація інформаційної мережі, щоб кожен міг дізнатися про те, що відбувається, і мав можливість виразити свої сумніви;
* залучення на сторону змін членів впливових і авторитетних неформальних груп в організації.

Формальні і неформальні групи, до яких належать співробітники, що дотримуються певних поглядів відносно змін. Ці погляди вирішальним чином впливають на позицію співробітника - члена групи, яку він займатиме.

Деякі співробітники, незважаючи на приналежність до групи, можуть мати особисту стурбованість відносно впливу зміни на їх майбутнє положення в організації, можливостей кар'єри, реалізації спрямувань і перспектив.

При управлінні цінними співробітниками важливо правильно зрозуміти можливі причини опору конкретних груп працівників(консерватори, прагматики, новатори), і тільки після цього можна буде вибрати правильну стратегію управління опором. Щоб допомогти співробітникові придбати нове розуміння того, що відбувається і переглянути своє відношення до зміни, найчастіше потрібно індивідуальну роботу з ним по роз'ясненню вигод і переваг, які він особисто отримає в результаті реалізації змін.

Така робота повинна привести до зміни поведінки співробітника. Співробітник, що чинить опір, - не проблема, а людина, яку необхідно зрозуміти і надати йому об'єктивну інформацію про необхідність виконуваних змін. Зазвичай після усвідомлення значущості змін, що проводяться, співробітники поступово переходять на їх сторону. При цьому важливо розрізняти, що причини опору у менеджерів і фахівців можуть бути різними унаслідок різниці виконуваних функціональних обов'язків (табл. 4.1).

Таблиця 4.1 – Головні причини опору

| **№** | **Фахівці** | **Менеджери** |
| --- | --- | --- |
| 1 | Недостатня інформованість | Побоювання втрати контролю і влади |
| 2 | Страх перед невідомістю | Нестача часу |
| 3 | Побоювання втратити роботу | Влаштовує існуючий стан речей |
| 4 | Недостатня підтримка з боку керівництва | Незрозуміло, що особисто я матиму від цих змін |
| 5 | Неможливість брати участь в ухваленні рішень |  |

Успішна реалізація змін в організації завжди характеризується умілим застосуванням цілого ряду перерахованих підходів, часто в найрізноманітніших поєднаннях, і відрізняється двома особливостями - менеджери використовують ці підходи з урахуванням їх достоїнств і недоліків і реалістично оцінюють ситуацію.

Найбільш поширеною помилкою керівництва є використання тільки одного або обмеженого числа підходів незалежно від ситуації. Це стосується і суворого начальника, який часто прибігає до примусу, і менеджера, орієнтованого на своїх співробітників, який постійно намагається притягати і підтримувати своїх людей, і начальника-циніка, що завжди маніпулює своїми співробітниками і часто прибігає до маніпуляції, і інтелігентного менеджера, який великою мірою покладається на освіту і спілкування, і, нарешті, менеджера типу адвоката, який увесь час намагається вести переговори.

Основне завдання керівництва - приймати вірні і своєчасні рішення, використовуючи різні підходи не лише поодинці, але і в поєднанні один з одним.

## 4.5. Чим треба управляти в Scrum

Основна мета використання управлінської методології Scrum - ефективне управління ресурсами з можливістю зосередження усіх учасників гнучкого процесу на кінцевому результаті. У основі Scrum лежать принципи, приписуючі повністю зосередитися на продукті, розробка і розвиток якого є основним результатом діяльності. Суть принципів Scrum успадкована від принципів, перерахованих в Agile Manifesto(див. лекцію "Введення в Agile" ).

Основна увага зосереджена на команді як засобі розробки якісного продукту. Реалізація продукту здійснюється за рахунок тісної і постійної взаємодії із зацікавленими сторонами в цілому і власником продукту зокрема. В ході цієї взаємодії команда розробників стає гнучкішою і готовою до змін. Інструменти, документація, умови роботи і плани вторинні, але також дуже важливі. Таким чином, фокус зосереджений на команді, яка задовольняє потребу замовника.

Scrum спирається на людей і взаємодію між ними. Грамотне управління людьми виходить на перший план*.*

**Команда**

Команда - група людей, взаємодоповнюючих і таких, що взаємозамінюють один одного, які зібрані для спільного вирішення завдань створення продукту, за допомогою якого вони підтримують взаємну відповідальність перед власником продукту. У ідеалі фахівці, задіяні в Scrum, мають бути самоуправляемый колективом, мета якого, що самоорганизующийся, - реалізувати вимоги до продукту в тому об'ємі і з рівнем якості, задовільним для подальшого використання продукту.

Успішність робочого процесу забезпечується за рахунок безперервного спілкування один з одним і з усіма зацікавленими сторонами. На усіх етапах роботи над продуктом відбувається безперервна взаємодія, в ході якої визначається, як можна реалізувати конкретне завдання з урахуванням достоїнств і недоліків конкретної реалізації. Уся отримана в ході взаємодії інформація враховується і використовується надалі. У подальшому це допомагає оптимізувати процес розробки якнайкраще і поступово удосконалюватися і розвиватися. Ідеал команди в Scrum - самоуправляемый колектив професіоналів, об'єднаних спільною метою.

**Вимоги**

Для роботи над продуктом у співробітників, об'єднаних в єдину команду, є усе необхідне - Product Owner, беклог продукту.

Scrum продуктивний із-за своєї простоти. Члени команди мають свободу самим визначати, яким чином розробляти продукт. І в цьому прихована найбільша складність Scrum. Однією з основних якостей, які відділяють початківця від професіонала, що відбувся, є риса самоаналізу, яка полягає в постійному переосмисленні того, що і як він робить. Мета самоаналізу - поліпшити роботу, над якою йде роздум. Необхідно вирватися з рамок "так належить" і зрозуміти, що конкретно треба зробити для досягнення результату найменшими витратами. Гнучким бути непросто. Для того, щоб бути ефективним, потрібна відповідна кваліфікація. Саме рівень цієї кваліфікації багато в чому зумовлює якість опрацювання вимог, які необхідно реалізувати. Але при цьому необхідно враховувати, що інженерія вимог для розробки функціонала повинна виконуватися в необхідному об'ємі. Вимоги повинні створюватися безперервно при активній співпраці зацікавлених осіб з командою розробки. Важливо пам'ятати, що кінцева мета збору вимог в тому, щоб полегшити і розділити загальне розуміння вимог для усіх тих, хто залучений в процес розробки. Вимоги можуть розрізнятися по детальності і мірі опрацювання і приймати самі різні форми. Для отримання загального уявлення про вимоги корисно мати яку-небудь форму або шаблон документації, визначуваної командою для кожної конкретної ситуації.

У Scrum як інструмент використовуються use case. Детальніше про техніку опрацювання вимог ми поговоримо в лекції "Атрибути Scrum".

**Продукт**

Основна увага Scrum- команди зосереджена на артефакті, розробка якого є їх обов'язком, - продукті. Робота над продуктом повинна будуватися виходячи з актуального розуміння існуючих потреб у функціоналі. Можна довго роздумувати про можливі шляхи розвитку, але ці роздуми повинні грунтуватися на стійкому базисі дійсності. Будь-які прогнози міняються залежно від тактичних даних. Тренд потрібний, але це не повинно бути визначальним чинником в роботі. Важливо ітеративно розробляти "потрібну" функціональність, постійно вимірювати її ефективність і, відштовхуючись від ключових актуальних метрик, продумувати подальший напрям розвитку продукту.

Продукт - жива субстанція, в розвитку якої бере участь уся команда, але його "контури" задає власник продукту, який "гаманцем" відповідає за кінцевий результат.

Можна виділити ще декілька ключових чинників, управління якими впливатиме на створення продукту(риски, технології, інженерні практики і ін.), але основними є саме сам продукт і команда, яка бере участь в його розробці. Інші є або похідними, або другорядними, і відповідно до їх значущості для впровадження і адаптації Scrum будуть розглянуті далі.

## 4.6. Значущість дотримання процесу

У другій лекції були обгрунтовані важливість і необхідність регламентації існуючих в компанії процесів як засоби контролю, моніторингу і пошуку можливостей подальшого розвитку діяльності організації. Наявність регламенту дозволяє гарантувати організаційний порядок, але для цього кожна компанія, проводячи впровадження певної процессной діяльності, адаптує еталонні принципи вибраної методології для реалій своєї компанії і прагне до найбільш ефективного використання існуючого процесу у рамках наявних ресурсів. Умовою і гарантом ефективності процесу є регламент, наслідуючи яке можна досягти запланованих результатів в певні терміни і прогнозувати подальший розвиток функціонала.

У Scrum- документах зафіксовано, що основною цінністю є люди, які за рахунок оптимальної взаємодії один з одним повинні досягти запланованих результатів. Саме за рахунок командної взаємодії повинна досягатися робоча гармонія. Значущість "записаного" порядку взаємодії знижується, на перше місце виходить людський чинник. При створенні і наслідуванні процесу необхідно триматися принципу золотої середини, тобто треба керуватися основними постулатами методології, але коригувати тактику залежно від конкретних умов довкілля і в думці тримати необхідність впровадження принципу гнучкості. Командний процес в ідеалі повинен вийти:

* прозорим;
* інспективним;
* адаптивним.

У цьому плані Scrum є процесом "з людською особою", в якій кожен член команди повинен контролювати свого співтовариша, оскільки за результат діяльності несе відповідальність уся команда.

Робочий процес повинен знаходитися під контролем власника продукту. Це дозволяє уникнути нестабільності і добитися можливості достовірно прогнозувати результати за допомогою управлінських інструментів. Для забезпечення його контрольованості в Scrum передбачені усі необхідні атрибути(дошка завдань, беклог продукту, беклог спринту), про які ми поговоримо трохи пізніше.

## 4.7. Інженерні практики

Під інженерними практиками розуміється перевірений часом набір технічних рішень, пов'язаних безпосередньо з розробкою програмного продукту. Складно розробити якісний продукт без впровадження цих практик в повсякденну роботу.

Важливим чинником є розуміння переваг і можливостей того або іншого інструменту, а також складнощів і специфіки його впровадження.

Більшість практик, вживаних в Scrum, своїми коренями йдуть в екстремальне програмування. Саме на цій методології практики, що розглядаються далі, довели свою ефективність.

Спершу поговоримо про техніка Code Review. Це діяльність, пов'язана з ревізією, вироблюваною командою коду. Ревізія повинна здійснюватися на систематичній основі. Вона дозволяє виявити і виправити помилки, що залишилися непоміченими автором, але помітні для того, хто проводить ревізію. Обмовимося, що ревізію повинен проводити кожного разу наступний член команди.

У багатьох командах ревізію проводить призначений черговий, в обов'язок якого на певний часовий період включений Code Review*.*

Ця практика дозволяє розробляти стабільніший і якісніший програмний продукт. Код програми стає більше структурованим і стабільним, плюс до всього поступово кожен член команди, що пройшов посаду ревьюера, починає більш глибоко розуміти усі деталі інформаційної системи, що реалізовується.

Далі поговоримо про Unit Testing. Це техніка, в основі якої лежать інструменти для перевірки працездатності окремих частин програмного продукту як окремо, так і в сукупності для перевірки інтеграції модулів і окремих систем. Реалізація Unit Testing дозволяє своєчасно реагувати на некоректні зміни логіки програмного продукту, що призводять до помилок і збоїв. Unit- тестування сприяє стабілізації програмного продукту і його місця в інформаційному ландшафті замовника, зниженню кількості проблем при інтеграції.

**Code Refactoring.** Це не просто техніка. Завдяки розвитку і поширенню Agile рефакторинг став повноцінним технічним напрямом діяльності. Refactoring пропонує здійснення змін внутрішньої структури початкового коду, які не повинні спричинити зміни в поведінці програми. За рахунок застосування Code Refactoring досягається спрощення структури програмного продукту і підвищення швидкості його роботи. Це спосіб систематичного приведення коду в порядок, при якому шанси появи нових помилок мінімальні. По суті, при проведенні рефакторинга коду ви покращуєте його дизайн вже після того, як він створений.

Як наслідок отримуємо економію на подальшому супроводі і підтримці.

**Build Automation.** Це техніка автоматичного складання початкового коду, виконання тестів і розгортання програми на цільовому ландшафті. Автоматизація дозволяє підвищити якість процесу розробки і постачання продукту. Результати Build Automation не так очевидні бізнес-замовникам системи, але за рахунок впровадження цієї техніки з технічної команди знімається частина роботи, після чого вони можуть направити ресурси, що звільнилися, на значиміші для бізнес-користувачів активності.

**Continuous Integration.** Практика безперервної інтеграції спрямована на виконання частих автоматизованих складок програмного продукту. Мета - виявлення і рішення інтеграційних проблем. Перехід до безперервної інтеграції дозволяє понизити трудомісткість інтеграції і зробити її більше передбачуваним за рахунок найбільш раннього виявлення і усунення помилок і протиріч.

Усі перераховані інженерні практики, безумовно, пов'язані між собою і дають максимальний ефект лише при роботі в комплексі. Кожна з них спрямована на підтримку стабільного робочого процесу, що ведеться по Scrum. Їх введення сприяє створенню якісних інформаційних систем швидко і з меншими рисками.

У промисловій розробці програмного продукту найважливіше - взаємодія із замовником. Використання інженерних практик і підходів Agile має на увазі регулярний і досить тісний зв'язок з ним для розуміння того, як кожна з впроваджуваних практик впливає на задоволеність замовника. Кожен з етапів роботи над продуктом стає відкритішим, зрозумілішим і обговорюється на кожній стадії. Впровадження і використання кожної інженерної практики можливе тільки після того, як власник продукту оцінить її результат. Команда розробки і замовник беруть на себе однакову відповідальність за кінцевий результат.

## 4.8. Короткі висновки

Agile представляє не стільки напрям процессной методології, який робить акцент на комунікації натомість надмірному документаційному обгрунтуванню виконуваних робіт, скільки сімейство різної техніки, яка довела свою успішність упродовж довгого періоду експлуатації в різних типах діяльності.

Основним представником сімейства гнучких процес сов по поширеності і ефективності є Scrum. Саме на його розгляді буде зосереджений фокус цього курсу. Головною галуззю, в якій Scrum знайшов своє застосування на сьогодні, являється сфера інформаційних технологій. Саме тому в Scrum з'явилися спеціалізовані артефакти, які покликані адаптувати його під реалії цього напряму діяльності.

При впровадженні будь-якої процессной методології і Scrum зокрема необхідно пам'ятати про те, що основною рушійною силою будь-якої зміни є співробітники. Тому потрібне розуміння того, як повинна виконуватися адаптація нововведення, як необхідно працювати з персоналом і яким аспектам впровадження приділити найбільшу увагу.

"Ядро" Scrum - це команда кваліфікованих мотивованих фахівців, зацікавлених в розвитку програмного продукту. Комунікації усередині команди дозволяють долати виникаючі проблеми і вирішувати завдання за рахунок загальних домовленостей. Окрім комунікації для успішного досягнення результатів доцільно застосовувати перевірені часом успішні інженерні практики. Вони зможуть допомогти в рішенні первинних технічних завдань і зосередити увагу професіоналів на розробці якісної інформаційної системи.

У кожній команді потрібна наявність співробітників, кожен з яких матиме певні навички і кваліфікацію, що дозволить виконувати їм певні функції. Таким чином, команда складається з працівників, що виконують певні ролі.

Про команду і ролі ми поговоримо в лекції "Ролі Scrum".

**Питання до теми:**

1. Дайте короткий опис методології Kanban.
2. Дайте короткий опис методології Lean.
3. Дайте короткий опис методології екстремального програмування.
4. Яка основна мета використання управлінської методології Scrum?
5. Назвіть головні причини опору фахівців і менеджерів?
6. Розкрийте поняття самоуправляемого колективу, що самоорганизующегося.
7. Що таке продукт в методології Scrum?
8. Перерахуйте інженерні практики Scrum. Дайте їх коротку характеристику.

# Лекція 5: РОЛІ SCRUM

План

5.1 Команда

5.2 Етапи командотворення

5.3 Розробник

5.4 Scrum- майстер

5.5 Власник продукту

5.6 Самоорганізація членів команди

5.7 Інші члени команди

5.8 Колектив, що самоорганізується

## 5.1 Команда

Команда - це група професійних працівників, що займаються різною трудовою діяльністю. Сенс командної роботи полягає у виробництві високих результатів за рахунок кооперації між співробітниками високої кваліфікації і виробництва сукупного результату, який повинен перевершувати суму результатів членів команди.

Безумовно, є категорії асів, що вважають за краще працювати самостійно, але у сьогоднішньому високотехнологічному світі поодинока робота стала долею небагатьох. Тільки за наявності команди можна розраховувати на виробництво якісного програмного продукту в задані терміни. Саме час - ця та перевага, яка може надати командна робота для компанії за рахунок швидшого виведення інформаційної системи на ринок. На командній роботі базується методологія Scrum.

Топ-менеджмент більшості організацій вважає, що, віддаючи перевагу окремим фахівцям, простіше здійснювати контроль і управління колективом в цілому. Коли є необхідність в хороших виконавцях, керівництво націлене на те, щоб у фірмі зібрати найдосвідченіших працівників. Питання якості і добрих результатів стають вирішеними апріорі після того, як потрібний працівник знайдений. Не так все просто.

Ресурс окремого, хоч і професійного виконавця обмежений багатьма чинниками, які визначають необхідний результат. Але коли ми починаємо говорити про командну роботу, то починають працювати зовсім інші принципи. Командний ресурс складається з ресурсів виконавців, до якого додається синергетичний ефект, що приводить загальний механізм в рух. Цю матерію складно помацати, але вона працює і відділяє успішні команди від зборища звичайних виконавців.

Характеристиками ефективних команд є наступні:

* + - Прагнення до досконалості. Кращі групи об'єднані єдиною метою. Для досягнення поставлених завдань вони постійно використовують найбільш ефективні варіанти, не користуючись тими, які погано зарекомендували себе у минулому, завдяки чому переходять в ранг від посередніх до високих рівнів, отримують поняття своїх можливостей і мають підвищені вимоги до отримуваних результатів.
    - Незалежність. Що мають цю якість штати працівників самоорганизованны і самоуправляемы. У них є можливість приймати рішення, переходити до їх реалізації. Вони навчені діяти самостійно.
    - Універсальність. До складу кращих команд включені фахівці профілів(планування, розробки, продажі і реалізації виробництва), необхідних для самодостатньої роботи команди. Головними якостями є взаємна підтримка, взаємодія і розуміння між членами трудового процесу.

Яким чином вдається створити групу людей з метою досягнення високих результатів? Чи буде вона такою, що самоорганізовується і створювати атмосферу взаємного навчання і інформування? Щоб знайти відповіді на ці питання, що хвилюють, один із засновників Scrum Д. Сазерленд витратив силу-силенну часу. Як добитися результату, щоб люди з'єдналися в одне ціле, думали про безперервне удосконалення? Точно не методом терору і суворого контролю, адже кожен повинен прийти до розуміння, що це йому дійсно необхідно. Тиск дасть тільки негативні результати. Найімовірніше, є набір нескладних правил, що допомагають досягти необхідного результату.

Усі зафіксовані правила укладаються в єдине гасло: "Одна команда - від старту до фінішу". Головна вимога до команди, яка повинна створювати необхідний продукт, - багатофункціональність. Це виражається в наявності усіх фахівців, необхідних знань і практики для успішного виконання завдань.

Стандартна організаційна структура включає окремі підрозділи управління, аналізу, розробки, тестування, впровадження і т. д. Вони укомплектовані співробітниками цих спеціалізацій. Подібна структура повністю відповідає класичному підходу до розробки програмного забезпечення. Але на сьогодні потрібні команди, які складатимуться з усіх необхідних співробітників і, оперативно реагуючи на зовнішні виклики, відбивати їх в кінцевому продукті. Саме за таким принципом і побудовані Scrum- команди.

Усі члени подібних команд повинні мати певні завдання і функції в роботі, що виконуються в робочих інтервалах, за які має бути сформований черговий вид робочого продукту(спринти). Функціональні відділи не випускають окремо продукт, а команда, на противагу функціональної організації, все робить спільно. Це багато в чому дублює принципи процессного підходу, що ще раз підкреслює, що Scrum є ефективною процесною методологією.

Але команда - складніший організаційний елемент, ніж окремий співробітник. Команда виникає, живе, розвивається і закінчує функціонування по своїх принципах, які необхідно знати. Розуміння цих принципів допоможе оптимально нею управляти і прогнозувати результати, які може надавати команда на різних етапах свого життєвого циклу.

## 5.2 Етапи командотворення

Експерти, зайняті питаннями створення ефективного організаційного середовища, сходяться на думці, що методика Scrum допомагає компаніям досягти бажаних результатів виробництва програмного забезпечення за рахунок впровадження інкрементного підходу виробництва. На чолі цього лежить базисний принцип розробки інформаційної системи або сервісу у вигляді створення мінімально необхідної функціональності для конкретної тимчасової дати і надання цієї версії в користування клієнтові. Після чого виконується використання створеного артефакту, і команда приступає до наступної ітерації роботи над продуктом, який розвивається на основі отриманих результатів. І так далі, ітераційно, крок за кроком продукт приходить до того виду, застосування якого оптимальне для заданих робітників умов компанії.

"На початковому етапі створена компанія випускала раз в пару місяців новий продукт. Ми нестримно росли, але управління відбувалося каскадним методом. Показники впали до випуску одного продукту в рік. Довелося негайно вживати серйозні заходи. Ми вирішили скористатися Scrum, завдяки чому повернули колишній рівень виробництва. Небагато мають такий результат. Ми цим гордимося".

Існує проста методика, яка може дозволити перевірити "правильність" мислення команди. Треба в розмові з будь-яким співробітником, включеним в Scrum- команду, поставити питання про стан їх групи. Оптимальним буде, якщо член команди відповість про продукт, що розробляється, а не міркуватиме про їх поточну спеціалізацію. У разі, коли співробітник не зв'язує себе із загальним продуктом, йдеться про нерозуміння цілей Scrum або ж про іншу "проблемну" причину. Якщо ж є тенденція, що уся команда знаходиться в "кризовому" стані, тоді необхідно діагностувати її стан і вживати відповідні заходи.

Щоб група людей перетворилася на згуртовану і ефективну команду, необхідно пройти декілька етапів.

Найбільш поширеною моделлю, що описує стадії командообразования, являється модель Такмана (рис. 5.1).

У своєму розвитку усі команди обов'язково проходять декілька етапів.

**Формування**

Цей етап багато в чому є відправною точкою для створення команди і постановки цілей на розробку або розвиток продукту. Як правило, на цьому етапі виконуються створення або оновлення складу, постановка цілей діяльності усього складу команди в цілому і окремих її членів, що не дуже розуміють ті завдання, виконання яких поставлене перед ними для досягнення загального результату.



Рисунок 5.1 – Етапи командообразования

**Вирує**

На цьому етапі усі члени команди виразно усвідомлюють свої і спільні цілі, задаючі вектор руху колективу. В період того, що вирує часто можливі конфлікти і протистояння між окремими членами команди, формування дрібних угрупувань, тому особливо зростає роль Scrum- майстра(SM) як модератора(його обов'язки ми обговоримо трохи пізніше). Етап того, що вирує - самий час для визначення формальних і неформальних лідерів Scrum- команди.

**Нормалізація**

На етапі нормалізації відбувається "притирання" членів команди один до одного. Закріплюються лідери команди, і вона починає демонструвати стабільні результати діяльності. Основне завдання Scrum- майстра на цьому етапі - допомога команді для переходу на наступний етап.

**Функціонування**

Функціонування - це етап, на якому команда переходить до самоуправляемости. На цьому етапі команда здатна оптимізувати свою продуктивність, існуючі неформальні зв'язки між членами команди закріплюються остаточно. Цей період найбільшої ефективності і продуктивності.

Хоча команди є самоуправляемыми, вони не стають безконтрольними. Керівництво встановлює контрольні точки, щоб уникнути нестабільності і хаосу. В той же час керівництво уникає суворого мікроконтролю, який вбиває креативність*.*

**Розформування**

Цілі, поставлені перед командою, досягнуті, мотивація більшості учасників в цей період убуває. Професійний Scrum- майстер повинен чітко відстежувати рубіж між етапом функціонування і розформування, і робить кроки, які не дозволять команді сильно знизити їх продуктивність. Як приклад подібних кроків можна привести зміну складу команди, зміну цілей, надання нового продукту, над розвитком якого повинні працювати команда, і т.і.

Щоб працювати ефективно, команда повинна як можна довше знаходитися на етапі функціонування. Відповідно, головним завданням команди(і, зокрема, Scrum- майстра) є максимально швидкий перехід між етапами, що упереджають стадію функціонування, і уповільнення етапів, які йдуть після. Але потрібно усвідомлювати те, що команда також схильна до впливу організації, її управлінських процесів, які протікають в компанії, і можливі тимчасові і "сезонні" відхилення, визначувані, скажімо, атестаціями, виплатами премій і ін.

Команду треба тримати в тонусі, але необхідно ретельно відстежувати стан найбільш "заслужених" працівників в цілях запобігання можливим вигоранням і зривам.

Організація командної роботи - це дуже важлива частина діяльності Scrum- майстра, який своєю роботою повинен забезпечити комфортний робочий стан усім її членам.

## 5.3 Розробник

Scrum- команда повинна складатися з мотивованих професійних співробітників. Це факт, на якому побудована оптимальність цієї гнучкої методології. Члени команди відповідають за розробку програмного продукту високої якості. Вони повинні володіти безліччю різних навичок, необхідних для створення ефективного програмного забезпечення:

* бізнес і системний аналіз;
* проектування архітектури програмного продукту;
* програмування;
* тестування;
* проектування і налаштування баз даннях;
* розробка ергономічних призначених для користувача інтерфейсів.

Члени команди беруть участь в плануванні робіт на часовий інтервал, який в Scrum прийнято називати "спринт". Команда може включати досвідчених розробників і новачків, які в процесі роботи повинні удосконалюватися при обміні знаннями. Члени команди відповідають за наступні завдання в проекті:

* обов'язкове виконання елементів робіт, включених в поточний спринт:
  + акцент на взаємозв'язаних завданнях спринту;
* вдосконалення особистої і командної роботи.

Основною рушійною силою команди є її учасник, якого прийнято називати розробник(Development member) - фахівець, що є "ядром" Scrum. Від його навичок, уміння, кваліфікації, досвіду залежатиме подальший розвиток продукту і Scrum- команди. Як правило, команда, працююча над продуктом, складається з 4-15 чоловік. Окрім володіння теоретичними знаннями кожен член команди повинен уміти практично застосувати наступні навички:

* спілкуватися з користувачами для розуміння цілей вироблюваних розробок;
* збирати "правильні" вимоги, які надалі можна зіставити з очікуваннями ключових користувачів;
* планувати подальшу роботу над продуктом у вигляді його інкрементного розвитку;
* документувати в мінімальному об'ємі вимоги;
* виконувати синтез розрізнених і суперечливих вимог в єдине комплексне робоче рішення;
* створювати зручний, читаний, зрозумілий, однозначний код ІС в цілях його подальшого розвитку;
* проводити повний цикл різних видів тестування;
* виконувати установку програмного забезпечення;
* підтримувати користувачів і відповідати на будь-які, навіть безглузді, питання.

Розробника Scrum- команди доцільно порівнювати зі швейцарським ножем, який віртуозно може вирішити будь-яке поставлене перед ним завдання. Але неправильно думати, що за рахунок універсалізації втрачається специфікація, що виражається в створенні якісного і використовуваного коду програмного продукту. Універсалізація вітається і всіляко "вирощується" в членах команди, але розробляти інкремент програмного продукту - це основне завдання розробників Scrum- команди.

## 5.4 Scrum-майстер

Scrum- майстер - рушійна сила Scrum- процесу. Він покликаний допомагати команді в освоєнні гнучкої методології. Його роль являється визначальною при впровадженні Scrum. Для того, щоб Scrum вийшов успішним, Scrum- майстер повинен мати наступні якості:

* бути відповідальним;
* віддавати належне команді, а не особистим досягненням;
* завжди бути готовим до діалогу;
* викладатися по максимуму;
* мати уміння впливати на інших;
* мати розвинений професійний кругозір;
* розбиратися в професійних і технічних питаннях.

Scrum- майстер покликаний допомагати команді у використанні Scrum. Його функції порівнянні з функціями тренера, який допомагає спортсменам дотримуватися спортивного режиму і підтримувати фізичну форму. Scrum- майстер створює мотивації і в той же час стежить за тим, щоб члени команди не ухилялися від виконання "важких фізичних вправ". Проте його повноваження обмежені. Він не може змусити співробітника виконувати те або інша вправа, яку той не бажає виконувати. В той же час він нагадує про цілі і про те, як необхідно їх досягати. Scrum- майстер має владу в тій мірі, в якою цією владою наділила його команда.

Scrum- майстер може сказати команді: "Отже, до закінчення кожного спринту ми повинні створювати програмний продукт, потенційно готовий до установки. Цього разу нам не вдасться створити такий продукт. Що ми можемо зробити для того, щоб подібна ситуація не повторилася в наступному спринті"?. Це приклад прояву влади Scrum- майстра над процесом. Якщо команді не вдалося до закінчення поточного спринту виготовити програмний продукт, потенційно готовий до установки, значить, стався якийсь збій в роботі команди. Але оскільки влада Scrum- майстра поширюється тільки на процес, то він не має права сказати : "Оскільки нам не вдалося до закінчення поточного спринту виготовити програмний продукт, потенційно готовий до відправки замовникові, я хочу, щоб він проглянув увесь код до того, як він буде остаточно прийнятий". Можливо, було б не так вже погано, якби хтось проглянув увесь код, але Scrum- майстер не має права прийняти таке рішення, оскільки воно виходить за межі його повноважень і вторгається в сферу відповідальності команди. Scrum- майстер повинен лише контролювати дотримання процесу командою, його роль може виявитися важчою, ніж роль типового керівника проекту, оскільки керівники проектів частенько можуть сказати: "Робіть так, тому що я так вирішив".

При призначенні Scrum- майстер не рекомендується здійснювати наступних помилок:

* призначати на цю роль співробітника без відповідних знань і досвіду;
* призначати на цю роль іншого члена команди;
* делегувати йому можливість приймати рішення від імені команди або її членів.

Проводячи паралель з функціональним або процессным підходом до організації процесів, можна сказати, що обов'язку Scrum- майстра відповідають ролі функціонального менеджера/адміністратора, але не власника процесу.

## 5.5 Власник продукту

Роль власника продукту являється основній при використанні методології Scrum. Очікування і визначення цієї ролі дуже розмиті. З одного боку, на неї покладається маса надій, але обов'язки чітко не визначені.

Для того, щоб бути успішним, співробітник, що виконує цю роль, повинен поєднувати в собі навички таких посад, як менеджер продукту, менеджер проекту, бізнес-аналітик, і при цьому бути природним лідером, здатним мотивувати і надихнути команду, що оточує його, на звершення.

Бізнес-аналітик - співробітник, який знає, як отримати вимоги від користувачів і як підготувати вимоги до команди, щоб вона виконувала їх максимально швидко.

Менеджер продукту - співробітник, який знає, чого хочуть бізнес і користувачі.

Менеджер проекту - співробітник, який знає, як планувати і відстежувати виконання проекту.

Лідер - співробітник, який знає, як направити загальні зусилля в один бік, на чому сфокусуватися і як мотивувати команду на досягнення цієї мети.

"Готового" для конкретної організації ідеального власника продукту, як ні прикро це констатувати, не існує. Власника продукту необхідно виховати відповідно до цілей, завдань і умов, в яких функціонує організація, більше того, успіх створюваного продукту залежить саме від того, хто виконує цю роль і який його позиція в організації. Цей співробітник повинен займати досить важливе місце в організаційній структурі компанії. Це допоможе відстояти обгрунтовану точку зору в дискусіях керівництва і донести до них ті вигоди, які обіцяє розвиток програмного продукту.

Власник продукту(Product Owner) - співробітник, який вказує Scrum- команді її мета і не дозволяє відхилятися від неї. Вказівка мети відбувається за рахунок:

* чіткого визначення виконуваних завдань;
* установки пріоритетів виконання завдань;
* підвищення цінності роботи, що виконується командою;
* забезпечення видимості, прозорості, зрозумілості виконуваних завдань, формування зрозумілих і несуперечливих вимог до них.

На нім лежить відповідальність за те, як члени команди зрозуміють завдання, необхідні для реалізації. Для ефективного виконання ролі власника продукту необхідно мати наступні якості:

* бути доступним для питань від членів команди;
* досконально знати профільний напрям бізнесу;
* бути комунікабельним;
* бути рішучим;
* мати певні владні повноваження в організації.

Підсумовуючи розмову про власника продукту, варто відмітити, що ця роль - нова для сучасного бізнес-середовища. Належить усвідомити безліч критичних чинників, як технічного, так і соціального порядку, ефективних програмних продуктів, що відносяться до розробки, для того, щоб вимоги до цієї ролі викристалізувалися.

## 5.6 Самоорганізація членів команди

Найсерйозніший виклик для впровадження гнучкого підходу - уміти перебудуватися із звичних "рейок" течії робочих процесів. Кожен член Scrum- команди повинен усвідомлювати, що результат(інкремент) і ефективність команди безпосередньо залежатимуть від результатів його діяльності, синхронізованих з результатами діяльності інших членів команди. Scrum заохочує розвиток універсалізму кожного з членів команди, але при цьому повинна зберігатися спеціалізація. Для того, щоб продукт був якісним, члени команди мають бути(чи прагнути до того, щоб стать) експертами в профільних областях діяльності. Самоорганізація і високий професіоналізм потрібні для того, щоб:

* команда і продукт не залежали від конкретного фахівця, і у разі потреби його можна було замінити;
* набір завдань, визначуваних конкретною бизнес-"болью", Scrum- команда могла вирішити незалежно від ресурсів, що маються в наявності;
* команда змогла в короткі терміни прийняти новачка і підтягнути його до загального рівня команди, або підтягнутися за ним сама, залежно від його рівня володіння необхідними навичками і кваліфікацією.

Після того, як завдання на найближчий спринт визначені, команда повинна самоорганізовуватися для його виконання. Те, як саме буде досягнутий результат, цілком залежить від членів команди і результатів їх роботи. Самоорганізація можлива при дотриманні певних умов. Важливо, щоб серед членів команди були співробітники, різні не лише по професійній спеціалізації, але і по своїх командних ролях. Приміром, можна привести наступні ролі, які швидше відповідають особовим якостям співробітників :

* критик;
* координатор;
* генератор ідей;
* виконавець;
* інші.

Кожна подібна роль займає свою "зону" в команді. Чим більше вільних зон, тим нижче рівень самоорганізації команди, і для її розвитку потрібне впровадження в команди співробітників, що відповідають подібним ролям. Зрілість і розвиненість команди можна оцінити, тільки постеживши за тим, як абсолютно різні люди уживаються між собою і досягають консенсусу в колективному обговоренні критичних питань і завдань. Наступним важливим етапом в процесі самоорганізації є проведення ретроспектив.

Ретроспектива - збори, які проводять проектна команда у кінці кожної ітерації, щоб обговорити, чому вона навчилася як команда, і побудувати плани на наступні ітерації, грунтуючись на витягнутих уроках.

Після закінчення спринту команда збирається і спільно обговорює, що пройшло добре, а що не так, як планували або хотілося б. Ретроспективи дають можливість проаналізувати і при необхідності підправити правила, по яких команда працювала раніше. Тему ретроспективи ми детальніше обговоримо в наступних лекціях.

Тільки після того, як буде досягнута атмосфера загальної довіри, співробітники в команді захочуть слухати і розуміти один одного. Самоорганізація окремих членів команди потрібна для досягнення ефективності колективної праці, як сукупність досягнутих результатів кожного.

## 5.7 Інші члени команди

Окрім позначених основних ролей, залежно від типу продукту, що розробляється, або умов, в яких функціонує організація, для команди можуть знадобитися співробітники з наступними профільними навичками і кваліфікацією:

* *Аналітики.* Розділяють роль власника продукту. Потрібні для великих продуктів і розосереджених команд. Діють на стратегічну і тактичну перспективу. Їх акцент зміщується з документування у бік вербального спілкування.
* *Керівники проектів.* Розділяють роль Scrum- майстра. Потрібні для організації робіт і ресурсного забезпечення команди.
* *Архітектори*. Розділяють роль технічного лідера власника продукту. Визначають якість технічної роботи команди. Задають тренд розвитку команди і створюваного продукту.
* *Функціональні менеджери*. Роль не змінюється, але частина обов'язків, пов'язаних з участю в розробці програмного продукту, переходить до команди. Повинні трансформуватися у власників процесів.
* *Програмісти*. Трансформуються в розробників.
* *Адміністратори бази даних*. Як правило, їх не переводять в Scrum- команди із-за дуже вузької, але необхідної для підтримки і розвитку продукту професійної спеціалізації.
* *Тестувальники*. Трансформуються в розробників/аналітиків. Їм необхідно вчитися діяти не в режимі виявлення, а в режимі попередження проблеми.
* *Проектувальники* призначеного для користувача інтерфейсу. Їх роль аналогічна ролі аналітиків, але з акцентом на продукти, в яких потрібно usability.

При впровадженні в команду нових ролей важливо пам'ятати, щоб усі співробітники розділяли загальні цінності і правила інкрементної і итеративности робіт, а також пов'язані з ними принципи роботи в Scrum- команді. Сенс Scrum полягає в тому, що член команди повинен вийти за межі своєї спеціалізації.

Якщо того вимагають загальні інтереси, люди мають бути готові працювати, не обмежуючись лише своєю вузькою спеціалізацією.

## 5.8 Колектив, що самоорганізується

Після того, як буде досягнута самоорганізація усіх членів команди, наступної сходинкою на шляху вдосконалення командної роботи і, як наслідок, самого процесу Scrum, являється досягнення стану колективу, що самоорганізується.

Основне питання, яке необхідно вирішити для ефективного впровадження в управлінську практику компанії, полягає в тому, що може дати самоорганізація компанії, адже те, що усі організувалися і зайняті, може у результаті ні до чого не привести. Якщо мета самоорганізації команди повинна послідовно привести до успіху продукту, то окрім перерахованих умов важлива мотивація - співробітники мають бути мотивовані самоорганізовуватися.

Культура компанії, в якій розробляється програмний продукт, неминуче впливає на самоорганізацію: може сприяти її поширенню, а може і немає. Якщо ініціатива в організації хоч би не карана, то самоорганізація команди можлива.

Директивне втручання визначається типом управління організацією.

Якщо директивне втручання полягає в завданні мети і завдань реалізації інформаційної системи, то важливо, щоб це робилося несуперечливий і послідовно. Втручання, спрямоване на прояснення очікувань, доступних ресурсів і того, що можна і чого не можна робити, заохочує і сприяє формуванню самоорганизованности Scrum- команд.

Якщо директивне втручання визначає операційне управління, то поширення в такій управлінській культурі високих Scrum- команд, що самоорганізовуються, досить скрутно.

Щоб організація змогла встати на рейки самоорганизованности, окрім необхідності наявності певної управлінської культури важливо, щоб гнучкі методології розробки і принцип командної роботи опоясували і проходили "червоною ниткою" через усю організацію. Подібна організаційна структура останнім часом почала впроваджуватися в діяльність небагатьох компаній по всьому світу. Але цей тільки початок. Ця "мета"-методология називається холократия.

Холократия - соціальна технологія, система управління, в якій повноваження і відповідальність за ухвалення рішень розподіляються по одиницях, що самоорганизующимся, замість управлінської ієрархії*.*

Холократия за своєю суттю - це спосіб децентралізації управління, який дозволяє збудувати процеси так, щоб співробітник міг впливати на життя компанії і мав повну владу у рамках своєї ролі. Ця технологія управляється не менеджментом, а спільною метою компанії, процесом, очікуваннями і метриками, визначеними управлінською командою, що складається з виділених представників окремих команд. У Scrum принципи холократии на сьогодні найбільше поширення отримали при організації діяльності окремих команд і при організації взаємодії між декількома командами.

У теорії це виглядає перспективно, але питань доки більше, ніж відповідей:

* як здійснити перехід до такого принципу управління? (зокрема, великим і ієрархічним організаціям, в яких будь-яка управлінська ломка може закінчитися глибокою кризою).
* де узяти достатню кількість мотивованих і кваліфікованих співробітників?
* як здійснювати мотивацію окремих команд?
* і інше.

Тільки найвідчайдушніші інноватори можуть зважитися на впровадження подібного типу управлінської структури, але є успішні приклади і в Україні, і не лише у сфері інформаційних технологій.

Але, незважаючи на блискучу ідею про самоорганизованности, важливо розуміти, що таке команда, що самоорганизующаяся, а також що не входить в її повноваження.

* Команда не може сама собі поставити мету. Нею завжди повинні керувати згори. Як правило, завдання спускаються від керівництва, і надалі контролюється хід їх виконання.
* Команда не визначає склад сама. Вона може вплинути на свій склад, але основне завдання по її формуванню і забезпеченню ресурсами лежить на керівництві.

На певному етапі, коли команда може об'єктивно оцінювати вклад учасників у формування кінцевого продукту і процес Scrum, їй можна довірити самостійне формування складу команди у рамках заданих ресурсів.

Команда повинна визначати максимально ефективний спосіб, за допомогою якого досягатиме результату. Ефективність в подібних випадках вимірюється терміном, складом і ціною робіт. При цьому команда має бути здатною адаптуватися до обмежень і умов довкілля.

Самоорганизованность і здатність адаптуватися під мінливі умови властиві багатьом природним колоніям живих істот. Візьміть, приміром, бджіл або мурашок. У тому випадку, коли людина у своїх прагненнях створити щось ефективне найточніше копіює природні аналоги, виходять оптимальні рішення.

У рішеннях, з яких складається кінцевий результат, бере участь кожен член команди, розділяючи відповідальність і винагороди. Самоорганизованность команди - це дієвий етап в роботі по Scrum. Для його досягнення і подальшого утримання потрібно управлінську участь, непідробний інтерес до вироблюваних результатів і бажання до постійного розвитку.

## 5.9 Короткі висновки

Scrum - дієва методологія, в основі якої лежать принципи командної роботи, що складається із співробітників, кожен з яких повинен виконувати певну основну роль:

* власник продукту;
* Scrum- майстер;
* члени команди розробки.

Ці співробітники працюють спільно, щоб створити робочий інкремент програмного продукту. Розробка проходить відносно короткими інтервалами, які прийнято називати "спринт". Інкремент повинен задовольняти заданим критеріям і вимогам, позначеним власником продукту, перед початком кожного спринту. Співробітники, що виконують різні ролі, скомпонованы в єдину команду. Залежно від особистих і професійних якостей кожного із співробітників до розвитку Scrum привносяться певні якості. Від того, наскільки вони зрілі, залежить сукупна професійна зрілість команди в цілому і вироблюваного ними продукту.

Співробітники, що грають ролі, які є основними для функціонування Scrum, у міру роботи проходять становлення і розвиток як команда. Кожен з етапів цього становлення характеризується певними характеристиками. Для того, щоб на кожному етапі діяльність команди задовольняла власника продукту, Scrum- майстер повинен відстежувати стан цих характеристик і при необхідності коригувати хід гнучкого процесу.

Найвища міра розвитку співробітника - стан самоорганизованности. У цьому стані співробітник повністю відповідає за вироблювані їм результати і готовий до постійних змін в цілях усвідомлення і подальшого вдосконалення.

Найвища міра розвитку команди - стан самоорганизованности. Для досягнення цього стану кожен з членів команди має бути таким, що самоорганізовується, команда в цілому повинна розділяти спільні цілі, постійно відстежувати відхилення від заданого робітника процесу, шукати способи для подальшого розвитку і вдосконалення як командної взаємодії, так і окремих її індивідів.

У лекції "Планування" ми поговоримо про найбільш складний і критичний етап діяльності в процесах розробки програмного забезпечення, який визначає очікування зацікавлених користувачів від створюваної інформаційної системи і закладає міцний фундамент подальшої взаємодії між замовниками і розробниками програмного продукту, - етапі планування робіт.

У Scrum цей етап досить сильно відрізняється від класичних підходів до планування і має свої деталі, якими також треба ефективно управляти для отримання оптимального робочого продукту.

**Питання до теми:**

1. Приведіть характеристиками ефективних команд.
2. Які етапи проходять команди у своєму розвитку?
3. Дайте пояснення в чому полягає ефективна організація командної роботи Scrum- майстром.
4. Які якості повинен мати Scrum- майстер? Кого не рекомендується призначати на роль Scrum- майстер?
5. Які якості повинен мати власник продукту?
6. Що таке ретроспектива? Що розуміють під самоорганізацією команди?
7. Що таке холократия по суті?

# Лекція 6: ПЛАНУВАННЯ

План

6.1 Принцип швидкого планування

6.2 Поетапне уточнення планів

6.3 Техніка Planning Poker

6.4 Діаграма згорання робіт

6.5. Короткі висновки

## 6.1 Принцип швидкого планування

Планування, усупереч загальному переконанню, - це фундаментальний аспект діяльності Scrum- команд.

Принципи инкрементальной розробки програмного забезпечення накладають додаткові вимоги на керівництво компанії, що управляє і контролюючої процес і результати від використання розробленого програмного продукту. В порівнянні з класичними методологіями розробки інформаційних продуктів відбувається зміщення акценту з постачання досить цілісного і концептуально завершеного продукту, повністю готового до повноцінної експлуатації, у бік розробки системи "напівфабрикату", що закриває потреби, необхідні "тут і зараз". Ця трансформація позбавляє бізнес-користувачів і профільних аналітиків від необхідності продумувати всілякі вимоги до функціональності інформаційної системи, що реалізовується, але підвищує важливість тактичного управління процесом розробки, його своєчасного постачання необхідними деталізаціями раніше позначених вимог і вимірі ефекту від їх впровадження.

Таким чином, при роботі по Scrum з'являється необхідність в плануванні і реалізації тільки найбільш значимих для бізнесу і власника продукту функциональностей інформаційної системи. У відповідь на це Scrum- команда повинна не просто прагнути розробити вчасно рамки позначеного інкремента, а узяти на себе зобов'язання наслідувати намічені плани впровадження. А це означає, що Scrum- команда повинна прагнути працювати над реалізацією функціональних можливостей, що мають найвищу цінність. Команда і власник продукту повинні оцінювати вартість розробки тієї або іншої функціональної можливості. Інакше приоритизация робіт здійснюється за суб'єктивним критерієм бажаності. Також важливо оцінити тривалість розробки тієї або іншої функціональності у бажаному виді. Функціональність, яка не потрапляє в критичне "ринкове вікно", матиме набагато меншу цінність. Scrum- команда, що досягла стану самоорганизованности, узяла на себе зобов'язання працювати в порядку пріоритетності, повинна дотримуватися важливість планування. При цьому команда отримає достатню кількість інформації для спокійної і продуктивної роботи впродовж одного або декількох спринтів, а власник продукту чітко розумітиме, що і як реалізує команда.

У результаті планування мають бути відомі:

* мета спринту;
* учасники команди і міра їх зайнятості;
* набір завдань на поточний спринт;
* дата демонстрації отриманих результатів.

Для Scrum підходить вид планування, при якому робота, яку потрібно буде виконати в найближчій перспективі, детально планується з глибоким розкриттям структури необхідних робіт. Далеко віддалена робота планується з відносно неглибоким розкриттям ієрархії робіт. У міру виконання глибоко розкритої і зрозумілої роботи робиться детальніше планування подальших завдань з можливим коригуванням спільних цілей.

Але тут з'являється проблема: не завжди є можливість прогнозувати навіть найближче майбутнє, навіть в процесах і командах з постійним прогресом. Тому планування виконується "хвилями" або етапами, де дії ближнього етапу детально плануються, а дії в далекому майбутньому відкладені. Необхідно виконувати стільки хвиль планування, скільки ітерацій роботи необхідно провести для досягнення кінцевого і задовольняючого потребам результату. Зокрема, якщо чіткий підхід або ресурси залежать від найближчих подій.

Такий підхід до планування називається плануванням методом "набігаючої хвилі".

Він якнайповніше відповідає принципам Scrum і підтримує процес поетапної розробки програмного продукту, в ході якого вдається ефективно управляти очікуваннями максимальної кількості зацікавлених сторін.

## 6.2 Поетапне уточнення планів

Після того, як робота на найближчу ітерацію спланована і зафіксовані певні бажані результати, слід замислитися про метрики моніторингу за відстежуванням виконання ходу процесу. Але що важливіше, продумати діяльність над створенням систематичного етапу в процесі планування, на основі якого можливо уточнюватиме намічені плани і синхронізуватиме очікування між власником процесу і функціональності, що реалізовується на практиці.

Впровадження цієї діяльності в етап планування дозволить:

* мінімізувати витрати часу;
* приймати рішення в потрібний для продукту час;
* уточнювати команді напрям необхідних дій;
* не опинитися в полоні у наміченого спочатку плану.

Ця діяльність прекрасно поєднується з технікою планування методом набігаючої хвилі і в літературі називається поетапним уточненням раніше намічених планів. Суть методу полягає в тому, що при ітераційній роботі виникає безліч непередбачених чинників різної природи виникнення, якими необхідно управляти і обробляти для досягнення намічених результатів. Для цього команді необхідно "намацати" загальний ритм і наслідувати його. Наслідування загального ритму роботи допоможе гарантувати стійку швидкість роботи, в яку закладаються тимчасові інтервали на усунення несподівано виникаючих проблем операційного характеру, пов'язаних з усуненням неопределенностей. Але для того, щоб знайти свій ритм, команда повинна дотримуватися наступних принципів:

* уникнення постійної наднормової роботи;
* накопичення статистичних параметрів по виконанню ітерацій(час виконання, кількість виникаючих проблем в роботі над певним типом завдань, природа проблем(організаційна, технічна і т. д.) і ін.);
* витягання системних виведень з отриманих параметрів і їх подальше застосування в процесі.

Після того, як команда вийде на стійкий ритм роботи, необхідно зафіксувати умови внутрішнього і зовнішнього середовища, що стосуються продуктивності команди. Подібна фіксація на досить тривалому інтервалі часу дозволить:

* залишити додатковий потенціал команди на форс-мажори;
* вивільнити частину ресурсів команди для творчого розвитку продукту;
* експериментувати з продуктивністю і складом команди;
* поступово вийти на новий рівень продуктивності з подальшим повторенням циклу стабілізації процесу.

Стійкий ритм командної роботи дозволить закласти міцний фундамент продуктивності не лише для поточної операційної діяльності Scrum, а поетапно займатися її вдосконаленням. Але для того, щоб взятися до систематичної роботи з етапом вдосконалення, необхідно, щоб методика поетапного уточнення планів зайняла міцне місце в діяльності окремо взятої Scrum- команди.

Крім усього іншого, поетапне уточнення планів сприяє виробленню заділу для прогнозування показників розробки. Детальний аналіз результатів процесу поетапного уточнення планів робить розробку прозорішою і зрозумілішою для стороннього погляду.

Це особливо критично, коли йдеться про роботу над складним функціоналом. Зазвичай завдання, вигоду від яких можна порахувати тільки у разі, якщо реалізований ряд суміжних доопрацювань, не очевидні по своїй складності і трудомісткості для ключових бізнес-користувачів. Процес поетапного уточнення планів дозволяє показати цю складність за рахунок многоитеративности роботи над завданням і згодом адекватніше оцінювати сумарну оцінку завдань.

Прогнозування продуктивності праці на майбутній період робиться на основі:

* оціненого об'єму робіт для реалізації необхідного функціонала;
* порівняння результатів робіт за статистикою подібних розробок, проведених з використанням найбільш ефективної інженерної техніки і методик;
* експертної оцінки реалізації даної найбільш авторитетними членами команди розробки.

Після того, як була визначена важливість і необхідність впровадження поетапного уточнення планів в стадію планування Scrum, має сенс поговорити про найбільш ефективні методики оцінки робіт, вживаних в гнучких методологіях розробки програмного забезпечення. Однією з таких є техніка Planning Poker.

## 6.3 Техніка Planning Poker

Якщо сказати найпростіше, то Planning Poker(PP) - це колективне обговорення завдання або проблеми з метою вироблення єдиної оцінки по її рішенню. Це вид техніки оцінки, який заснований на досягненні домовленості між усіма учасниками команди, що беруть участь в процесі планування. Він використовується для оцінки складності майбутньої роботи.

Методика PP має явні переваги порівняно із стандартними обговореннями проблем, вживаними в класичних підходах до розробки програмного забезпечення, які зводилися до наступних активностям:

* ініціатор обговорення(керівник / аналітик / team lead) збирав команду, що складається з експертів або людей, знайомих з об'єктом обговорення.
* проблема озвучувалася, починалося активне обговорення.
* шукалося разове рішення, яке повинне було влаштовувати усіх причетних до цієї проблеми упродовж усього періоду розробки/експлуатації програмного продукту.

Подібний підхід життєздатний тільки у рамках класичних підходів до розробки програмного забезпечення, коли відповідальність за прийняття певного технічного або бізнес-рішення зводилася до вказівки конкретного компетентного співробітника, який міг бути не в курсі усієї необхідної інформації або можливих змін. Відмітимо, що він характеризується наступними недоліками:

* не усі люди готові публічно висловлювати свою думку відносно обговорюваного питання.
* висловлювання і думки перших учасників обговорення прямим або непрямим чином можуть впливати на думки інших.
* спроба в подібному обговоренні знайти універсальний кошт вирішення проблеми, як правило, закінчується подальшими переробками, як тільки умови, на яких засновано рішення, змінюються.

Для нівеляції вказаних недоліків була розроблена позначена методика PP. Суть підходу полягає в тому, що оцінка завдань виконується не у вигляді годинника або альтернативних шкал, що представляють трудовитрати, а у вигляді порівняльної оцінки, яка показує відносний "розмір" вимог. Можна зустріти Scrum- команди, які оцінюють роботи у вигляді "пунктів", "папуг", "майок", "грибів" і т. д. Важливі відносні значення, і не може бути абсолютного еталону. Одиниці виміру не мають фізичного еквіваленту.

В процесі проведення PP уся команда повинна прийти до єдиного розуміння розмірів, щоб уявлення про те, скільки сил витратити на реалізацію завдання, у усіх було однаковим.

Ось тут і прийде на допомогу статистика результатів, які команда досягала за останні декілька спринтів. Кожне виконане завдання треба ідентифікувати, з тим щоб кожен член команди зрозумів, про що йде мова. Після цього необхідно виконувати порівняння запитів між собою для оцінки їх актуального розміру. Потім завдання викладаються в ряд за збільшенням досягнутих оцінок по кожному завданню, і навпроти кожної кладеться відповідна карта з оцінкою. Тепер треба згрупувати вимоги навколо чисел. Описаний захід є дієвим способом досягнення загального розуміння трудомісткості виконуваних завдань. Його можна проводити періодично, коли команда "буксує" з виконанням оцінки завдань на PP.

Planning Poker має значимі переваги в порівнянні з аналогічною технікою планування. До них варто віднести наступні:

* в плануванні бере участь уся команда;
* у кожного члена команди є можливість висловитися, не випробувавши впливу авторитетніших колег;
* усі члени команди беруть на себе відповідальність за терміни.
* оцінки, отримані PP, точніші порівняно з оцінками, отриманими за допомогою альтернативних методів оцінок.

Правила, яким підлеглий техніка Planning Poker, досить просто запам'ятовуються і легкі в застосуванні:

* scrum- майстер не бере участь, а тільки ведуть загальні збори;
* власник продукту представляє короткі огляди кожної із завдань;
* члени команди ставлять питання і ведуть дискусію про пропозиції і риски;
* підсумок обговорення записується Scrum- майстром.



Рисунок 6.1 **–**Атрибути техніки Planning Poker

* кожного разу розігрується певне завдання;
* учасники вибирають карту;
* кладуть її сорочкою вгору;
* числові достоїнства - оцінка трудовитрат на завдання;
* учасникам з високими і низькими оцінками дається можливість висловитися і обгрунтувати свою оцінку;
* процес обговорення триває до тих пір, поки не буде досягнутий консенсус.

Критичні для проведення процесу зауваження:

* не рекомендується використати числа більше 13 або 20.
* не вірте в те, що буває робота, що не вимагає ніяких зусиль.

Якщо йдеться про команду, яка раніше не працювала разом, або розробляється новий продукт, немає можливості використати статистику, накопичену за певний період. Розмір витрат повинен оцінити найбільш авторитетний учасник команди, що має схожий досвід роботи і викласти у вигляді лінійної послідовності.

Це буде найбільш ефективним способом оцінки завдань і старту активності PP, навіть якщо потім виявиться, що в початковому розрахунку були допущені незначні помилки.

Незважаючи на те що практика Planning Poker все частіше входить в життя більшості сучасних Scrum- команд, цього виявляється недостатньо, щоб команди навчилися робити оцінки легко і швидко. Іноді, якщо людям дати тільки колоду карт із стандартними інструкціями, цього виявляється мало, і з часом команда перестає використати цей механізм оцінки. Роль і значущість Scrum- майстра для цієї конкретного заходу, так само як і для більшості заходів, що проводяться в Scrum, складно переоцінити.

## 6.4 Діаграма згорання робіт

Один з найважливіших артефактів, які використовує Scrum- команда для контролю за запланованими показниками виконання робіт впродовж спринту, - діаграма згорання робіт(Burndown Chart) ( рис. 6.2).

Ця діаграма показує, скільки завдань залишилося до завершення запланованого періоду часу на виконання робіт(спринту):

* по вертикалі - кількість завдань;
* по горизонталі - час;
* ідеальна діаграма згорання, яка відображає запланований хід робіт.

Мета кожної Scrum-команди - "спалити" всі взяті в спринт задачи до того, як наблизиться кінець наміченого терміну. Якщо фактична крива відрізняється від ідеальної, то по ситуації необхідно коригувати дії команди.

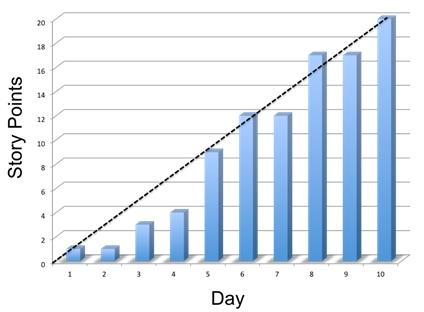


Рисунок 6.2 – Діаграма згорання робіт

Оперативний аналіз процесу згорання завдань повинен проводитися по кількості "пунктів", що залишилися, шляхом порівняння реального графіку з ідеальним:

* якщо реальний графік вищий за ідеальний - значить, команда відстає від плану;
* якщо реальний графік нижчий ідеального - команда випереджає план.

Аналіз результатів ходу виконання процесу доцільно проводити в один і той же час. Краще всього розподілити ці активності рівномірно за часом виконання спринту. Саме це дозволить оперативно відстежувати динаміку виконання завдань. Краще всього, якщо на щоденній основі автоматичний збір інформації по процесу дозволить будувати метрики ефективності і приймати необхідні командні рішення.

Якщо намічається відхилення від плану у бік відставання від графіку, то команда повинна на чергових зборах обговорити ситуацію, що складається, прийняти рішення і виробити шлях її оперативного виправлення. У список найпоширеніших причин, які призводять до відставання, входять:

* помилка в плануванні і подальшій оцінці завдання;
* хвороба або інша причина відсутності одного або декількох членів команди;
* недооцінка і реалізація ризиків різного характеру.

Про відставання необхідно максимально оперативно повідомити власника продукту. Якщо команда сама не зможе виробити шляхи виправлення ситуації, треба, щоб власник продукту зменшив об'єм спринту за рахунок завдань користувача з мінімальним пріоритетом.

Якщо ж йдеться про випередження графіку виконуваних робіт, команді слід внести в журнал побажань спринту одну або декілька додаткових завдань з вищим пріоритетом, що не увійшли до спринту раніше.

Відмітимо ще декілька важливих пунктів, які необхідно враховувати при роботі з діаграмою згорання робіт:

* діаграма повинна оновлюватися щодня, за фактом зробленої роботи. У простій формі повинна постійно демонструватися динаміка виконання робіт в спринті;
* графік має бути загальнодоступний. Будь-який член команди або зацікавлена особа повинні мати до нього доступ;
* будь-який член команди може зробити пропозицію з приводу оптимізації виконання завдань. Будь-який член команди може зробити пропозицію, яка не має бути знехтувана "просто так". Будь-яка пропозиція сприймається командою і обговорюється при першій нагоді. Тільки команда в цілому може прийняти кінцеве рішення по кожній конкретній пропозиції.

## 6.5. Короткі висновки

У цій лекції був детально розглянутий найбільш значимий і критичний для управління очікуваннями користувачів етап планування процесу розробки програмного забезпечення. Ми приділили увагу методам і профільній техніці планування для роботи з гнучкими процесами. Зокрема, розглянули планування методом "набігаючої хвилі" і подальше поетапне уточнення планів. Застосування цих методик в Scrum дозволить збудувати роботу, починаючи з "правильної" роботи з менеджментом, на предмет однозначного розуміння об'єму необхідних ресурсів для досягнення поставлених результатів і закінчуючи технікою оцінки трудомісткості завдань, приміром, технікою Planning Poker.

Також ми поговорили про такий артефакт, як діаграма згорання робіт, яка за своєю суттю є контрольною панеллю для відстежування результатів командної роботи, що виконується у рамках спринту. Усі ці атрибути спрямовані на підвищення прозорості виконуваних робіт в Scrum і є простими у використанні.

У лекції "Етапи і заходи Scrum" ми детальніше поговоримо про контрольні заходи, використовувані в Scrum для управління процесом розробки програмного забезпечення.

**Питання до теми:**

1. Що у результаті швидкого планування повинно бути відомо? Які обов'язкові підсумки планування мають бути відомі усій Scrum- команді?
2. Що має на увазі підхід до планування, який називається методом "набігаючої хвилі"?
3. Що таке поетапне уточненням раніше намічених планів? У чому полягає суть цього методу?
4. Яких принципів повинна дотримуватися команда для того, щоб знайти свій ритм?
5. На основі чого робиться прогнозування продуктивності праці на майбутній період?
6. У чому полягає методика Planning Poker? Які переваги вона дає?
7. Які дії власника продукту у разі відставання від плану при виконанні спринту scrum- командою? Які у разі випередження плану?

# Лекція 7: ЕТАПИ І ЗАХОДИ SCRUM

План

7.1. Sprint

7.2 Щоденні зустрічі(daily)

7.3 Груминг бізнес-завдань

7.4. Груминг технічних завдань

7.5 Огляд Спринту

7.6 Ретроспектива

7.7 Короткі висновки

## 7.1. Sprint

У Scrum ітерація роботи команди називається "спринт"(sprint). Її тривалість визначається конкретними умовами і вимогами до процесів розробки програмного забезпечення і створюваного продукту. Оптимальною тривалістю спринту вважається інтервал 2 тижні, максимально можливі, - 6 тижнів. Не рекомендується робити його більше півтора місяців, інакше можуть виникнути проблеми із створюваним інкрементом.



Рисунок 7.1 – Діаграма спринту

Результатом спринту є інкремент готового продукту(build), який можна передати замовникові для установки на продуктивний інформаційний ландшафт. Саме у реалізації готового інкремента в короткі терміни і з мінімальними трудовитратами, який замовник може використати і отримувати від цього певну бізнес-цінність, полягає уся привабливість і унікальність Scrum порівняно з альтернативними видами розробки програмного забезпечення.

За рахунок коротких спринтів забезпечується оперативна відповідь Scrum- команді від замовника. При цьому замовник дістає можливість гнучко управляти об'ємами робіт і направляти розробку в потрібному для себе напрямі, оперативно оцінюючи результат спринту і пропонуючи поліпшення створюваного продукту.

Поліпшення повинні фіксуватися і потрапляти у беклог продукту. Після цього вони оцінюються відповідно до існуючих бізнес-пріоритетів в усьому об'ємі існуючих завдань. У наступний спринт потрапляють завдання, які мають найвищий пріоритет.

Кожен спринт є маленьким "водоспадом". Упродовж спринту робляться усі роботи по збору вимог, розробці і тестуванню інкремента продукту. Об'єм робіт, включених в спринт(беклог спринту), має бути фіксованим. Це дозволяє команді давати зобов'язання на той об'єм робіт, який має бути зроблений в спринті. Це означає, що беклог спринту не може бути змінений ніким, окрім команди.

Спринт має свій життєвий цикл, який, як було сказано вище, схожий на класичний підхід до розробки програмного забезпечення в мініатюрі:

1. Спочатку виконується комплексне планування спринту в два етапи.

На початку кожного спринту проводиться його планування. У цьому заході беруть участь власник продукту, ключові користувачі, Scrum- майстер і команда розробки.

Планування спринту складається з двох послідовних мітингів:

| а) | *Перший мітинг.* Визначення цілей спринту. Учасники: команда, власник продукту, Scrum- майстер. Мета цієї зустрічі - визначити мету спринту і наповнити його беклог. |
| --- | --- |
| б) | *Другий мітинг.* Визначення способу реалізації завдань, узятих у беклог спринту. Учасники: команда, Scrum- майстер. Мета цього мітингу - визначити, як саме розроблятиметься функціональність для того, щоб досягти мети спринту і синхронізувати розуміння усіх розробників про те, що саме буде зроблене. Саме на цій зустрічі застосовується техніка Planning Poker для оцінки трудомісткості і тривалості завдань. Якщо в ході реалізації завдань з'ясовується, що члени команди можуть не встигнути зробити заплановане, відстежуване по діаграмі згорання завдань, то Scrum- майстер, власник продукту і команда зустрічаються і з'ясовують, як можна скоротити роботи і при цьому досягти поставленої мети спринту. |

1. Розробка інкремента спринту.

Активні учасники: команда. Пасивні учасники: Scrum- майстер, власник продукту. Мета - розробка продукту силами команди. Це основний процес, у рамках якого команда займає активну позицію, від якої залежатиме результат їх діяльності. Scrum- майстер і власник продукту готові допомагати і відповідати на питання при необхідності.

1. Зупинка спринту.

Зупинка спринту робиться тільки в найвинятковіших ситуаціях. Спринт може бути зупинений до того, як закінчиться відведений на нього час. Зупинку повинна виконувати команда, якщо обгрунтовано розуміє, що не може досягти мети спринту у відведений на спринт час. Спринт може зупинити і власник продукту, якщо необхідність в досягненні поставлених цілей зникла.

Після того, як сталася зупинка спринту, повинен проводитися командний мітинг, на якому обговорюються причини зупинки. Після цього життєвий цикл спринту повинен повторитися починаючи з планування.

В період проведення спринту щодня необхідно проводити коротку зустріч, звану "дэйли", для того, щоб усі члени команди розуміли, хто і чим займається в цей конкретний час. Тривалість дэйли - максимум 15 хвилин. Його проведення призначене для вирішення явних і потенційних проблем. Ведучим цього мітингу є Scrum- майстер, який по кругу ставить кожному членові команди три питання:

* Що зроблене c моменту попереднього дэйли?
* Що планується робити сьогодні?
* З якими проблемами ви зіткнулися?

Усі інші питання і обговорення мають бути винесені за межі мітингу і обговорюватися між зацікавленими в його рішенні особами.

Якщо команда досягла достатнього рівня самоорганізації, то участь в цьому мітингу Scrum- майстра не потрібно.

Після того, як спринт досяг свого завершення, необхідно проводити демонстрацію реалізованого функціонала і подальший аналіз спринту. Активність демонстрації називається "Огляд Спринту"(Sprint review), а аналіз спринту - "ревью" або "Ретроспектива Спринту"(Sprint Retrospective).

У призначений час на Огляд Спринту приходять команда, власник продукту, ключові користувачі. В процесі цієї активності члени команди розповідають про поставлені завдання, про те, як вони були вирішені, які перешкоди виникали у них на шляху, які були прийняті рішення, які проблеми залишилися невирішеними, і демонструють реалізований функціонал, а власник продукту і користувачі оцінюють досягнутий результат.

На Ретроспективі Спринту команда і Scrum- майстер обговорюють хід виконання процесу, відмічають його достоїнства і недоліки, приймають рішення про можливе коригування процесу в цілях його поліпшення. Scrum- майстер є безпосереднім організатором цих заходів і відповідає за загальну підготовку до нього. Команда допомагає йому скласти адженду і запланувати розклад виступу. Підготовка до мітингу не повинна займати у команди багато часу(правило - не більше двох годин). Для проведення демонстрації заборонено використати інструменти типу Power Point. Функціонал повинен демонструватися безпосередньо в робочому програмному продукті. Підготовка до мітингу також не повинна займати у команди більше 2 годин. На наступних лекціях ми продовжимо обговорення, пов'язане з розгляд окремих етапів і заходів, детальніше. Тут же необхідно акцентувати увагу на такому важливому понятті процесів гнучкої розробки програмного забезпечення, як мету спринту.

Більшість сучасних команд сконцентрована на виконанні запланованих завдань спринту, виконання яких було погоджене з власником продукту. Таким чином, виникає потогінна система, основний критерій успіху якої - повністю виконані і продемонстровані завдання.

Якщо якась історія не зроблена - вічна ганьба розробникам, які відповідали за неї, повний провал спринту, катастрофа вселенського масштабу і безперервна скорбота, ініційовані замовниками робіт.

У відповідь на це з'являється екстремістська реакція самої Scrum- команди. Є думка, що в об'єм спринту не можна вносити зміни після того, як спринт почався, а це, відповідно, обмежує замовника. Але якщо прибрати рамки спринту за часом і об'єму завдань, то ми "скачуємося" до класичної, водоспадної моделі розробки інформаційних систем.

І перша, і друга ситуації - це зовсім не "гнучко". Істина, як це часто буває в 100% ситуацій, десь посередині. Для досягнення цієї середини в Scrum з'явилося поняття - мета спринту. Це командна установка, визначальна, навіщо ми збираємося згаяти час і ресурси спринту, чого хочемо досягти з точки зору інкремента цінності для замовника. І це не просто набір розрізнених вимог. Мета дозволяє і пропонує нам бути гнучкішими і толерантнішими, як до невеликої зміни об'єму спринту, так і до несподіваних причин, що не дозволили нам реалізувати те або інша вимога в спринті. Успішний спринт - спринт, що досяг своєї мети. Якщо ми реалізували очікування замовника, що прийшов на демо, то це успіх.

Мета спринту - інструмент, що дає нам гнучкість в управлінні очікуваннями замовників і роботи команди усередині спринту. Єдина умова - мета спринту не повинна мінятися у рамках самого спринту.

## 7.2 Щоденні зустрічі (daily)

Тепер детальніше поговоримо про захід, який, на думку більшості експертів, є найкориснішою практикою гнучких процесів. Дэйли, як було відмічено вище, - це щоденні зустрічі для синхронізації робіт членів команди в єдину погоджену командну роботу ( рис. 7.2).



Рисунок 7.2 –Процес проведення StandUp

Знання про те, чим займаються колеги по команді, допомагає в:

* мінімізації витрат часу, пов'язаних з непоінформованістю членів команди про найкращий шлях рішення завдань, які вже вирішувалися Scrum- командою;
* розумінні загальної динаміки робіт по спринту;
* розумінні потенційних проблем, що виникають у членів команди, і шляхів їх рішення.

Тривалість дэйли строго обмежена і не повинна перевищувати 15 хвилин. Ця зустріч не призначена для вирішення проблем. Усі питання, що вимагають спеціального обговорення, мають бути винесені за її межі.

Зустріч проводиться щодня завжди в один і той же час, найчастіше цей ранок, але в розподілених командах це може бути день або вечір.

Scrum- майстер веде цю зустріч, задаючи кожному членові команди по три питання*:*

* Що зроблене c моменту попереднього дэйли?
* Що планується робити сьогодні?
* З якими проблемами ви зіткнулися?

Кожен член команди повинен відповідати на необхідні питання.

Важливо, щоб усі, відповідаючи на питання, не вдавалися глибоко до деталей і, боже, упаси, вже точно не намагалися тут же їх вирішувати.

З одного боку, це здається не такою вже і великою проблемою, але по факту виявляється, що відвернення співробітників від плану зустрічі призводить до того, що через якийсь час багато хто не бачить в ній необхідності. У той момент, коли хтось поглиблюється в оповідання про деталі, більшість незацікавлених починають нудьгувати, занурюватися в серфінг соціальних мереж за допомогою мобільних гаджетів і т. д.

Мета мітингу - зосередити увагу усіх членів команди на аспектах процесу. Саме це дозволяє бути ефективним в Scrum і мати додаткову мотивацію на успіх. Якщо перевантажити усіх непотрібною на зараз інформацією, то інтерес до того, що відбувається поступово слабшатиме.

Для того, щоб перебороть неправильний розвиток Scrum- процесу і направити дэйли в потрібне русло, треба:

* Дотримуватися встановлених правил. Правила не повинні встановлюватися "згори". Принципи роботи і рішення про застосування правил повинна приймати команда, але не Scrum- майстер і не власник продукту. Візуалізувати прийняті правила корисно. Актуалізувати правила, синхронізуючись із станом команди і навколишніх умов, також дуже важливо.
* Думати про розвиток. Scrum - корисна практика. З часом члени команди вчаться правильно проводити зустрічі і бути ефективними в обставинах умов, що постійно міняються. Кількість проблем, якщо дотримуватися правил і займатися вдосконаленням робочих практик, поступово зменшуватиметься. Через деякий час Scrum трансформується в повністю операційну діяльність. З'явиться спокуса трохи його різноманітити. Дайте волю цим прагненням. Це допоможе команді об'єднатися і стати продуктивнішою.

## 7.3 Грумінг бізнес-завдань

Наступним по важливості заходом є груминг - тривала зустріч, на якій Scrum- команда разом з власником продукту розглядає існуючий беклог продукту, а також вимоги, які планується включити в найближчий спринт, додає необхідні технічні і бізнес-завдання, щоб зафіксувати напрями проектування або вирішити необхідні архітектурні питання розвитку продукту і його інкремента. Без регулярного груминга планованої роботи спринт стає громіздким і дуже заплутаним.

Груминг - це практика "причісування" беклога(по-англійськи "grooming"), одна з тих активностей, без яких не обходяться продуктивні Agile- команди.

Груминг бізнес-завдань - це узгодження об'єму робіт, які треба виконати для того, щоб власник продукту на найближчому огляді був задоволений цінністю від реалізованого командою інкремента.

Цю активність по можливості необхідно проводити в очному режимі присутності усіх її учасників. Інакше велика вірогідність, що вона розростеться в часі.

Груминг - це не ще один тип мітингу в Scrum. Це активність, яка робиться упродовж спринту для підготовки беклога до наступного спринт-планированию.

Груминг варто проводити до або на початку спринту, з тим щоб детально розібрати завдання, що готуються, до реалізації розробниками. Періодичність його проведення має бути один раз в спринт. Час, що витрачається на груминг, повинен складати 10% від загальної тривалості спринту.

Спочатку, для знову зібраної команди, на груминг може йти немало часу. Це пов'язано з "притиранням" розробників і власника продукту один до одного, глибиннішим розумінням бізнес-сенсу програмного забезпечення, що розробляється, усвідомленням деталей і взаємозв'язків впливу процесів, що автоматизуються, і так далі. Але це треба робити. Як результат з'явиться прогрес в роботі над завданнями, члени команди підуть від "сліпого" аналізу вимог до усвідомленого вдосконалення системи.

Після того, як кількості "перегрумленных" історій вистачить на два-три спринти вперед, варто зменшити темп і приділяти грумингу трохи менше часу. Два-три спринти - це оптимальний інтервал для забезпечення повної декомпозиції виконуваних завдань. Не більше і не менше. Якщо "перегрумить" надто багато завдань, то велика вірогідність того, що до моменту початку реалізації найбільш віддалених завдань їх рішення буде вже не зовсім актуально, а можливо, навіть застаріє і команда знову згає час на те, щоб зануритися в актуальні умови завдання і прийняти найбільш правильне технічне рішення для її реалізації. Усі розібрані на груминге завдання варто помістити в саму "верхівку" беклога продукту і поступово наповнювати з нього беклоги спринтів:

* ці завдання зрозумілі усім членам команди;
* вони мають такий об'єм, що декілька з них можуть бути зроблені за спринт;
* вони оцінені командою.

Щоб груминг прижився в операційній діяльності кожної команди, необхідно наслідувати декілька простих правил:

* зробіть груминг частиною вашого процесу;
* виробіть поняття готового беклога спринту - "Definition of Done"(DoD).

Ці характеристики, відповідно до яких завдання можна брати в роботу на спринт.

* Встановите поняття "поточного" і "наступного" інкремента, що дозволить власникові продукту управляти об'ємом робіт, що виконуються командою.
* Зустрічайтеся "живцем" усій Scrum- командою з власником продукту не рідше за рази одного разу в спринт для груминга і планування релиза.
* Під час груминга працюйте з беклогом продукту, який був підготовлений і приоритизирован власником продукту.

Спосіб ухвалення рішень (DoD) при проведенні груминга про можливість узяти завдання в роботу - це не догма(у Scrum їх практично немає). Цей спосіб вибирає команда і може змінити його залежно від своєї "розвиненості". Можуть бути використані наступні підходи:

* Класичний підхід - консенсус. Перевага цього методу полягає в тому, що рішення приймається усіма членами одноголосно (це підвищує мотивацію усіх членів команди і усвідомлення розділення її цінностей, але, як правило, відбувається дуже повільно.
* Більшість. Це менш демократичний спосіб в порівнянні з класичним, що може привести до зниження мотивації окремих членів команди, але він є швидшим. Якщо голоси розділилися порівну, то можна або продовжити обговорення, поки одна людина не змінить думку, або завжди використати правило вибору меншої(виклик) або більшої(запас) оцінки.
* Оцінка майбутнім виконавцем. Це метод, який найменш вписується в дух Scrum. Рішення приймає конкретний виконавець, який далі реалізовуватиме цей функціонал, і команда залишається за бортом "командної" роботи. Він найбільш швидкий для ухвалення рішення про роботу, але надалі при зловживанні їм команда ризикує розбитися на індивідуумів.

Для того, щоб підвищити професіоналізм проведення груминга, доцільно використати методи оцінки завдань, про які було сказано в лекції "Планування", - приміром, метод Planning Poker.

Серед основних цінностей гнучкої розробки важливу роль грають взаємна довіра і самоорганізація, до розвитку і роботи над якими повинні прагнути усі члени команди. Якщо люди ображаються, коли інші не погоджуються з їх думкою, це означає, що вони ставлять своє професійне его вище за інтереси команди. Так буває в командах, які стали такими вимушено, за управлінською вказівкою або іншою причиною. Члени команди використовують основні принципи і здійснюють процедури, але ніхто не думає про їх основний сенс. Впровадження цінностей Scrum - основне завдання Scrum- майстра. Він постійно повинен пояснювати і виховувати, в першу чергу на власному прикладі. Після того, як будуть виховано декілька лідерів, інші почнуть тягнутися за ними.

Груминг бізнес-завдань - дуже важливий етап на шляху усвідомлення командою цінності програмного продукту, що розробляється ними, але часто для того, щоб правильно провести цей тип груминга, необхідно грунтовно підготуватися з технічної точки зору, тобто розкласти по кроках алгоритм реалізації складних вимог. А для цього існує груминг технічних завдань.

## 7.4. Грумінг технічних завдань

Тепер ми дісталися до розгляду специфічного виду активності, чия значущість, проте, не менше за попередніх.

Після того, як стає зрозумілий набір бізнес-завдань і спосіб їх реалізації, які повинна виконати команда у рамках поточного і наступних спринтів, доцільно виконати декомпозицію найбільш складних і комплексних з них для того, щоб усі члени команди однозначно уявляли собі спосіб і об'єм планованої автоматизації, яка має бути зроблена у рамках реалізації.

Якщо провести паралель з класичним підходом до розробки інформаційних систем, то ця стадія відповідає стадії технічного проектування. Різниця полягає в тому, що у водоспадній і инкрементальной моделі за цю стадію відповідає окремо виділений фахівець - як правило, технічний лідер або системний архітектор, а в Agile відповідальність за технічний груминг лягає на команду в цілому.

У Scrum над проектуванням працює група розробників, кожен з яких відповідає за область узятих на себе завдань і за їх гармонійну інтеграцію із завданнями колег.

З одного боку, це підвищує риски створення концептуально цілісного продукту, але, з іншого боку, знижує вірогідність "залишитися далеко від критично важливих змін, які відбуваються постійно при реалізації окремих завдань, що виконуються різними розробниками з Scrum- команди. Спільна взаємодія членів колективу дозволяє досягти потрібного рівня сприйняття інформації і багато в чому закладає базис активності управління знаннями на потрібному рівні розуміння.

Правила і ради проведення технічного груминга відповідає правилам проведення бизнес-груминга. Дуже важливо, щоб на технічному груминге був присутнім власник продукту або хтось з його представників. Будь-яке рішення, що приймається в процесі декомпозиції і ухвалення технічного рішення, пов'язане з умовами і чинниками бізнес-процесів, що автоматизуються.

Чим більш глибоко відбувається інтеграція продукту, команди, власника продукту і інших бізнес-користувачів, тим глибше стає розуміння того, що немає вимог, які не роблять вплив на якість результату їх спільної роботи. Не важливо, про який вид вимог йде мова: функціональних або нефункціональних, їх рішення у будь-якому випадку зводяться до якості створюваного програмного продукту.

## 7.5 Огляд Спринта

У самому кінці кожного спринту настає момент, коли усі члени команди повинні продемонструвати те, що було ними виконано. Цей етап називається Огляд Спринту.

Ця зустріч проводиться в заздалегідь запланованому місці, в заздалегідь запланований час. Мета огляду - показати зацікавленим особам все, що команда зробила за період спринту. Уявна легкість організації цього заходу не повинна вводити в оману. Для ефективної демонстрації реалізованих завдань необхідно витратити немало часу і сил. Відсутність ясності і прозорості в організації огляду призводить до того, що учасники цього процесу, окрім членів команди, не розуміють статус розробки інформаційної системи, не розуміють цінність завдань, що реалізовуються, не розуміють, що саме їм показують і для чого це розробляється. Щоб уникнути подібних проблем, Scrum- майстер повинен з великою відповідальністю віднестися до організації цього заходу.

Підготовка до огляду не повинна займати у команди більше 2 годин. На початку огляду необхідно чітко і однозначно озвучити мету минулого спринту. Якщо на огляді запрошені співробітники, які мають лише віддалене уявлення про продукт, що розробляється, то слід приділити пару хвилин, щоб ввести їх в курс справи.

Члени команди розповідають:

* про поставлені завдання;
* про те, як вони вирішувалися;
* про те, які проблеми виникали;
* про те, які були прийняті рішення для управління проблемами;
* про те, які проблеми залишилися невирішеними.

Якщо в компанії є проблеми з менеджментом зустрічей, тоді починати має сенс тоді, коли прийшли ключові учасники. Не має сенсу чекати усіх. Після того, як декілька зустрічей почнуться вчасно за участю лише одного або двох користувачів, інші учасники зрозуміють, наскільки важливо приходити вчасно.

Слід дотримуватися призначеного плану огляду, який необхідно прописати в описі до зустрічі. Несподівано виникаючі хаотичні дискусії слід присікати і виносити за рамки зустрічі так само, як це робиться на дэйли.

Огляд - це не лише один з необхідних заходів Scrum. Воно несе в собі безліч посилань, управління якими допоможе у впровадженні і розвитку гнучких методологій у вашій компанії:

* позитивна оцінка роботи надихає і мотивує команду;
* бізнес-користувачі поступово почнуть детальніше розуміти, ніж займається Scrum- команда;
* на огляді зацікавлені сторони обмінюються життєво важливими відгуками про продукт між собою і з командою;
* огляд проходить в дружній атмосфері, тому різні команди можуть вільно спілкуватися між собою і обговорювати насущні питання.

На огляді необхідно дотримуватися наступних принципів:

* огляд повинно відбуватися швидко і бути сфокусовано на ідеї показу реалізованої функціональності. Сконцентруйтеся на створенні не стільки красивого, скільки динамічного демо. Це допоможе привчити тих, що усіх, що беруть участь до дотримання заданого тайминга і не відволікатися на сторонні речі;
* огляд має бути орієнтованим на бізнес-користувачів і підготовленою командою цінностями. Сфокусуйтеся на тому, "що ми зробили", а не на тому, "як ми це робили". Технічні деталі дуже важливі, але сенс демонстрації в тому, щоб показати реалізовану цінність. Технічні деталі команда повинна обговорювати в процесі роботи;
* аудиторія огляду може сама спробувати розроблений продукт. Можливість "пограти" з продуктом діє на користувачів вдохновляюще;
* не треба показувати купу помилок і те, як вони були виправлені. Про них треба згадати, але демонструвати їх не варто, тому що це забере у вас багато часу і понизить увагу до важливіших завдань.

Якщо команду примушувати проводити огляд, коли у них нічого толком не працює, їм буде некомфортне. Команда запинатиметься і спотикатиметься, показуючи "сиру" функціональність. Це неприємно. Але це діє як гірка пігулка.

Наступного разу команда постарається все доробити до терміну! Команду необхідно привчити до думки, що огляд доведеться проводити не дивлячись ні на що. Якщо огляд виконано добре, то це робить позитивний вплив.

## 7.6 Ретроспектива

Ретроспектива - захід, спрямований на систематичну інспекцію і адаптацію Scrum до умов функціонування конкретної команди. На ретроспективі повинна збиратися уся команда і обговорювати, яким чином зробити наступний спринт продуктивнішим в порівнянні з тим, що пройшов.

Що таке ретроспектива? Це регулярна зустріч, на якій команда обговорює свій робочий процес і щось в нім міняє.

У основі ретроспективи лежить концепція циклу деминга, PDCA(англ. Plan - Do - Check - Act). Мета ретроспективи - до її закінчення отримати практичний план ефективних змін, але це не план остаточних змін в процесі - це ескіз експерименту на найближчий період. Цикл деминга складається з наступних етапів:

* Plan - заплануй.
* Do - виконай.
* Check - перевір.
* Act - прийми якісь подальші рішення, виріши, що далі робити.

Власне, сама ретроспектива - це Plan.

Цілями проведення ретроспективи є:

* інспекція того, наскільки успішно пройшов спринт відносно людей, стосунків між ними, процесів і інструментів;
* визначення і впорядкування того, що пройшло успішно, і того, що потребує поліпшення;
* розробка плану по впровадженню поліпшень в Scrum.

Ретроспектива, як кожне Scrum- захід, повинна мати чітку і однозначну мету. Отримати план процессного експерименту - сама відповідна мета для будь-якої команди, проте багато хто цього не розуміє. На ретроспективі команди іноді намагаються придумати якісь глобальні рішення своїх проблем і стикаються з тим, що не виходить це зробити разом. У результаті вони застряють в проміжному стані повного незадоволення. Якщо ви зіткнулися з такою проблемою, то потрібне розуміння того, що рухатися треба маленькими кроками, пробуючи і перевіряючи різні підходи до рішення проблем, згодом вибираючи кращий з них. В процесі проведення ретроспективи треба виробити не план остаточних змін, а план експерименту на найближчий період, в ході якого будуть перевірені прийняті рішення у вдосконаленні Scrum.

Ретроспектива - цей захід, на якому команда вирішує свої проблеми.

Рішення проблем в Scrum - це не пусте питання. Неможливо зробити так, щоб організаційний керівник або експерт прийшов, подивився і сказав, що команді потрібно робити, а що в робочому процесі варто змінити.

По-перше, приходити до команди з готовими рішеннями відгукнеться відторгненням пропонованих рішень. Цей феномен прийнято називати "not invented here"(не винайдено тут). Тут спрацьовує природна гордість і упертость людської натури. Навіть якщо розробники розуміють, що пропоноване рішення вірне, у них немає упевненості в його працездатності для їх команди, плюс відсутній "власницьке" відношення до нього. Подібні, не "вистраждані" командою рішення мають низькі шанси на реалізацію. Члени команди повинні набити власні шишки проблем і визначитися з рішенням, яке буде оптимальним для усіх.

По-друге, розробка програмного забезпечення - складна дисципліна. Більше того, в кожній компанії, яка займається цією активністю, процеси розробки різні або, принаймні, мають унікальні для їх оточення деталі. Навряд чи знайдеться консультант, який, не знаючи умов і чинників організації конкретного процесу, зможе розписати, як повинна працювати Scrum- команда при рішенні певної задачі в конкретній компанії. Тільки провівши декілька робочих експериментів і зрозумівши суть операційних проблем, можна зрозуміти, хороша або не дуже та або інша практика для цієї команди.

Існує таке поняття, як "best practice". Це тип практик, які підтвердили свою ефективність на ряду прикладів при рішенні конкретних завдань. Незважаючи на уявний універсалізм, кожна подібна практика допоможе не у будь-якому випадку. Це відбувається тому, що дану практику можна оцінити тільки в контексті певної ситуації*.*

Той, що веде ретроспективу(спочатку цю роль повинен виконувати Scrum- майстер) повинен привести команду до конкретних робітників принципам і домовленостям, яких треба дотримуватися і дотримуватися їх при роботі в спринтах.

Ідеї про розвиток команди і вдосконалення Scrum народжуються в процесі обговорення недоліків минулого спринту. Ведучий повинен стежити за тим, щоб хід обговорення мінусів не переростав в пошук конкретних винуватців. Для досягнення цілей Scrum більше важливо розуміти, як не допустити подібну ситуацію в майбутньому і що з нею потрібно зробити, щоб мінімізувати можливі втрати. Спори з приводу ідей, незважаючи на усталену думку про їх необхідність, на цьому етапі даремні. У план будуть включені тільки ті ідеї, з якими згодні усі учасники мітингу.

В результаті обговорення достоїнств, недоліків і пропонованих ідей команда повинна приступити до складання плану, куди мають бути включені не просто підсумки обговорення, а конкретні дії, які повинні виконати команда, або її окремі ролі:

* написати .;
* прийняти .;
* Завдання Х виконувати з використанням підходу N.

Проте не варто намагатися на конкретній зустрічі вирішити усі проблеми відразу - для ефективної роботи в наступному спринті досить плану з 3-6 пунктів. Занадто об'ємний план може у результаті виявитися нездійсненним, демотивирует команду і точно зміниться через короткий інтервал часу. В процесі проведення ретроспективи можуть виникнути проблеми, пов'язані як з Scrum, так і з його учасниками:

* Команда вважає, що у неї немає проблем, Scrum- процес хороший, і вона не бачить сенсу в його поліпшенні. Помилка. Але команді цього так просто не пояснити. Щоб зрушити її з мертвої точки, корисно запросити на ретроспективу когось з менеджерів компанії - замовника або користувачів, які мають конкретні претензії до команди або продукту. Користувачі дуже рідко повністю задоволені. Вони можуть бути задоволені до певної міри, але у них все одно є якісь думки на тему того, що можна зробити краще. Якщо такий замовник приходить на ретроспективу і розповідає це команді, то вона вимушена обговорювати напрями для подальшого розвитку.
* На ретроспективі говорить в основному один або 2-3 людини. Помилка. Людям завжди є що сказати. Якщо увагу забирає негласний лідер, то своїм домінуванням він пригнічує інших членів команди. Якщо кожен висловлюватиме свою думку, то вірогідність знайти кращі рішення набагато зросте.

Групова дискусія спонукає усіх учасників висловлюватися. Це допомагає подивитися на проблему з різних точок зору і придумати краще рішення. Часто впоратися з цією проблемою допомагає той, що веде ретроспективи, який стежитиме за тим, щоб кожен з присутніх обов'язково висловив свою точку зору.

Варто ще раз відмітити, що формат проведення ретроспективи може бути різним. Ретроспективи - цей не одиничний захід. Вони проводяться регулярно, і за результатами кожних таких зборів виконується основна мета - створюється план на найближчу ітерацію. Якщо віднестися до цієї процедури професійно, заздалегідь проаналізувати найбільш типові проблеми, що виникають в ході ретроспективи, можна створити сприятливі умови розвитку справжньої команди, що самоорганизующейся.

## 7.7 Короткі висновки

На цій лекції ми обговорили поняття спринту, яке являється центральним для методології Scrum.

Спринт - це часовий інтервал, у рамках якого перед командою стоїть завдання реалізувати погоджену кількість вимог, об'єднаних спільною метою. Спринт, на відміну від звичайного робітника графіку, підпорядкований особливим принципам і у своєму складі має низку заходів, кожне з яких спрямоване на те, щоб ефективним чином скоординировать процес роботи над завданнями, що розробляються командою. Склад робіт, тривалість спринту фіксовані. Він повторюється до тих пір, поки є необхідність в розробці програмного забезпечення для конкретної організації.

Крім того, в спринті специфічні активності, які спрямовані на підвищення якості процесу і розвиток членів команди як активних учасників Scrum.

У результаті необхідно відмітити, що Scrum вимагає не просто формального дотримання наказаних принципів, але постійної участі в процесі менеджерів, які своїми рішеннями можуть вплинути на напрям розвитку команди.

У лекції "Атрибути Scrum" ми приступимо до розгляду ключових артефактів, використовуваних при роботі в Scrum.

**Питання до теми:**

1. Що таке "спринт" з точки зору методології Scrum?
2. Що є результатом спринту?
3. Що таке "беклог спринту"?
4. Які питання ставить Scrum- майстер кожному членові команди в ході "дэйли"?
5. У який момент відбувається демонстрація функціонала і подальший аналіз спринту?
6. Що таке бізнес груминг і груминг технічних завдань?
7. Що таке ретроспектива? Яка її мета?

# Лекція 8: АТРИБУТИ SCRUM

План

8.1 Story mapping

8.2. Призначені для користувача історії(User story)

8.3. Визначення пріоритетів користувачів

8.4. Дошка завдань

8.5 Беклог продукту

8.6 Беклог спринту

8.7 Інкремент продукту

8.8. Принцип прототипирования

8.9. Короткі висновки

## 8.1 Story mapping

Story mapping(англ. "карта історій") - техніка візуального і фізичного представлення послідовності дій, які мають бути реалізовані в тому, що розробляється командою програмному продукті.

Story mapping - інструмент, що допомагає в осмисленні функціональності ПО і "правильному" проектуванні способів його використання.

Карта історій - це джерело інформації, використовуване для візуалізації вимог до продукту в контексті використання його функціональності і їх пріоритетів. Story mapping допомагає виконати декомпозицію завдання з її високорівневого представлення до рівня конкретних завдань, а виконана декомпозиція забезпечує еволюційне розуміння цілісного продукту, починаючи з повного охоплення усіх потреб і завершуючи зануренням до детальних вимог користувачів. Але зазвичай із зануренням в детальні вимоги користувачів виникають найбільші проблеми.

Як правило, чим більше компанії, тим більше розгалуженим і кросс-функциональным являється процес, що автоматизується. Відповідно і вимога буде більше високорівневою і комплексною. Для невеликої компанії реалізація невеликої вимоги займає не більше тижня, а ось у великих організаціях розробка подібної вимоги займе не менше декількох місяців.

Причин, як у кожної проблемної ситуації в області розробки програмного забезпечення, багато. З них слід виділити:

"Неподільність" вимог. Міф про те, що тільки частину високорівневих вимог можна поділити на декілька супідрядних завдань, народився багато в чому завдяки низькій довірі до фахівців в області розробки програмного забезпечення. Особливо це стосується процесів впровадження програмних продуктів і "зовнішніх" аутсорсинговых і консалтингових компаній, яким вигідно "забивати" цей міф в голови замовників і тим самим експлуатувати це розуміння для отримання додаткових контрактів на продовження співпраці. На те, щоб здолати подібний міф, спрямована технологія инкрементальной розробки. У її ході власник процесу і Scrum- команда домагаються коректного розбиття високорівневої ідеї на поэтапность виконуваних завдань, кожна з яких приносить певну цінність бізнес-замовникам.

Небажання розуміти значимі деталі процесу розробки програмного забезпечення. Замовник, особливо якщо це топ-менеджер компанії, як правило, не схильний занурюватися в деталі процесу, що автоматизується. Це зрозуміло. У топ-менеджерів інші цілі і завдання. Тому бажано, щоб власником продукту був користувач, який не лише розуміє бізнес, що автоматизується, але має навички роботи з інформаційними системами і розуміння загальних принципів їх пристрою і функціонування.

Недовіра до області комерційної розробки інформаційних систем. Область комерційної розробки програмного забезпечення - не нова область, і вже багато організацій з нею стикалися не одного разу. Враховуючи стан в цій області, описане в лекції "Введення в Agile", у багатьох користувачів накопичений визначений, як правило негативний, досвід взаємодії з компаніями розробниками програмного забезпечення. Тому замовники вимушені говорити "зробіть все і відразу" просто тому, що це єдиний спосіб отримати необхідні доопрацювання в умовах, що склалися. Тому що якщо не попросять максимум і зараз, то навряд чи отримають це потім.

Виходів з подібної ситуації декілька. Основне рішення, яке пропонує Scrum, - розробляти результат максимально швидко. Для цього і організована "спринтовая" модель постачання функціональності маленькими порціями, відповідно до найбільш актуальних вимог. А це можливо тільки у тому випадку, коли спроектований загальний каркас функціональності через декомпозицію вимог.



Рисунок 8.1 –Представлення карти історій

Карта історій представляється у вигляді двох послідовностей залежних один від одного активностей. Таким чином, ми отримуємо матричний вид карти історій.

Вона є послідовними стадіями, що виконуються в процесі/, що автоматизується, продукті. На самому верхньому рівні розташовується найбільш верхнеуровневое представлення стадій або компонентів продукту, що розробляється. Далі, у міру зниження, кожен компонент розкладається на складові, і так до самого низькорівневого представлення продукту, що автоматизується.

Процес побудови карти історій:

1. Спочатку виділяють ключові види діяльності. Кожен вид фіксується на окремій картці.
2. Вони розташовуються у функціональному порядку використання зліва направо.
3. Після цього визначаються окремі завдання, що визначають кожну активність, і також фіксуються на картках.
4. Усі завдання розташовуються в логічному порядку.

Карта історій містить каркас із завдань, що є "цеглою", що становить кінцевий продукт. Цей каркас покриває великий набір функцій, що реалізовуються відповідно до пріоритетів користувачів.

Story mapping - це архітектурне представлення завдань користувача.

Під каркасом знаходяться детальні вимоги, які описують конкретні функціональні частини для виконання завдань. У Scrum, коли мова заходить про роботу з вимогами, застосовують певну техніку роботи з ними, яка називається User Story(призначені для користувача історії).

## 8.2. Користувацькі історії (User story)

У практиці підготовки вимог, які надалі складатимуть каркас програмного продукту, що розробляється, в Scrum прийнято використати один із стандартизованих підходів для їх опису - Use Cases("призначені для користувача історії").

Use Case("призначена для користувача історія", "юскейс" і т.і.) - це сценарна покрокова техніка опису взаємодії двох або більше за учасників, задіяних в автоматизації. За допомогою Use Case може бути описана і призначена для користувача вимога, і вимога до взаємодії систем, і опис взаємодії людей і компаній в реальному житті. У загальному випадку за допомогою Use Case може описуватися взаємодія двох або більшої кількості учасників, що має конкретну мету.

У розробці програмного забезпечення цей спосіб розробки і управління вимогами часто застосовують для проектування і опису взаємодії користувача і системи, тому назву Use Case часто сприймають як синонім вимоги людини-користувача до рішення певної задачі в системі.

Користувацькі історії формулюються як одне або більше за пропозиції на "повсякденній" мові користувача. Вони виходять відносно невеликі за об'ємом, що зручно як для складання, так і для їх обговорення, приоритизации, планування, оцінки і подальшої роботи з ними. Призначені для користувача історії виходять у вигляді алгоритму дій користувача з програмним продуктом, що реалізовується. Таким чином, кожен квадратик в Story Mapping можна представити у вигляді Use Case. Якщо власник продукту віднесеться до приоритизации призначених для користувача вимог з усією необхідною серйозністю і значущістю, то Scrum- команда може сконцентруватися на найбільш значимих і важливих Use Case, що впливатиме на инкрементальность створюваної інформаційної системи. Адекватна оцінка трудомісткості історій дозволяє планувати терміни її реалізації, тим самим управляючи вмістом спринтів. Розробити оптимальну призначену для користувача історію не так просто, потрібна певна навичка, яка дозволить створити якісний опис вимог користувачів до програмного продукту, що реалізовується, але в нагороду за це виходять наступні переваги:

* Стислість. Призначена для користувача історія описує невелику частину бізнес-цінності, яку можливо реалізувати за період спринту;
* "Незатратность" створення і супроводу. За рахунок своєї "компактності" призначені для користувача вимоги досить просто створити і супроводжувати їх зміни на всьому протязі життєвого циклу продукту;
* Залучення ключових користувачів до процесу створення продукту. За рахунок своєї доступності бізнес-вимоги зможуть стати реальним "містком" між користувачами і Scrum- командою. Це дозволить адекватніше управляти очікуваннями користувачів і залучити їх на потрібну міру занурення в процес розробки продукту;
* Полегшують оцінку завдань. Формат призначених для користувача історій сприяє точнішій оцінці необхідних системних розробок/доопрацювань.

Чим компактніший об'єм має призначена для користувача історія, тим простіше виконувати її оцінку. Це призводить до вірніших оцінок термінів реалізації програмного продукту і до зрозумілішого планування виконуваних робіт. При цьому важливо знати міру в зменшенні і дробленні Use cases. Якщо зробити їх надто багато, то у вас вийде величезний список завдань у беклоге продукту, і процес управління ними і фіксації на карті історій ускладниться.

Для користувачів вимоги, виражені у вигляді призначених для користувача історій, є основним інструментом впливу на програмний продукт, що розробляється. Призначені для користувача історії визначають формат, в якому у користувачів є можливість відбити усі ті важливі чинники, які, на їх думку, мають бути враховані в процесі автоматизації.

Важливо усвідомлювати те, що призначені для користувача історії не забезпечують повноту усіх функціональних вимог і мають ряд недоліків:

* Не масштабуються для великих програмних продуктів. Призначені для користувача історії добре себе зарекомендували, коли йдеться про створення невеликих або середніх по об'ємах і складності програмним продуктам. За рахунок того, що цей вид роботи з вимогами орієнтований на безпосередню роботу з користувачами і підтримується "бізнес-лексиконом", він непридатний в роботі над великими інформаційними системами, коли на перший план виходить організаційна структура проекту/процесу, і важливі формальні ознаки здачі/приймання робіт;
* Вимогливі до кваліфікації розробників. Розробники, працюючі з призначеними для користувача історіями, повинні мати високу кваліфікацію і непогані комунікативні навички, які дозволять їм отримати необхідні уточнення вже викладених вимог. Як правило, таких розробників трохи. І це ще один безперечний не достаток;
* Не є засобом документування. Призначені для користувача історії - це невелике і зручне представлення інформації. Вони сформульовані на щоденній мові користувача і містять невеликі деталі, залишаючись відкритими для інтерпретації. Вони допомагають розуміти, що повинна робити система, але при цьому призначених для користувача вимог недостатньо, щоб зрозуміти, як буде організована логіка системи. Призначені для користувача вимоги є необхідною "верхівкою" для розуміння призначення інформаційної системи, але для реалізації системи розробникові доводиться додумувати безліч значимих деталей.

Інженерія вимог - важливий етап в створенні кожного програмного продукту. Scrum - гнучка методологія, в якій роботі з вимогами відводиться не центральне, але при цьому значиме місце. Функціональні обов'язки по витяганню, розробці і управлінню вимогами лягають на усіх членів команди, саме тому в процесі роботи з ними задіяні такі артефакти, як карта історій і призначені для користувача історії, призначення яких - полегшити отримання інформації, необхідної для якісної реалізації програмного продукту.

## 8.3. Визначення пріоритетів користувачів

Після того, як процес роботи з вимогами в Agile- методологіях став очевидніший, спрощеніший і зрозуміліший порівняно із стандартними, класичними підходами до створення інформаційних систем, доцільно обговорити техніку, яка допоможе полегшити процес визначення пріоритетів окремих завдань для їх подальшого включення в спринти Scrum- команди.

На сьогодні існує безліч різної техніки розставляння пріоритетів. Лише невелика їх частина може бути застосована для приоритизации призначених для користувача історій як одиниці інкремента для продукту, що розробляється, в Scrum.

Розглянемо найбільш ефективні з них:

* Принцип Ейзенхауера. Принцип, або, як ще називають в літературі, "матриця Ейзенхауера" - техніка тайм-менеджмента для визначення пріоритетів завдань. Виглядає матриця як чотири квадрати, які виходять при перетині осей "Важливо - не важливо" по вертикалі і "Термінове - не терміново" по горизонталі (рис. 8.2). При використанні цієї матриці по її "частинах" розподіляються завдання відповідно до їх важливості і терміновості.

1. Важливі і термінові завдання. Це ті завдання, які важливі, і виконати їх необхідно терміново. Без них все руйнуватиметься, нічого працювати не буде, і зробити їх завтра - буде вже пізно.
2. Важливі, але не термінові завдання. Це ті, які терміновими стануть незабаром. Успішні команди і співробітники в першу чергу звертають свою увагу саме на цей тип завдань, щоб вони поступово не перейшли в розряд "Важливі і термінові завдання".
3. Не важливі, але термінові. Це тип завдань, які ніяк не наближають до досягнення необхідного результату, які потрібно робити, але виключно для того, щоб їх робити



Рисунок 8.2 –Матриця Ейзенхауера

1. Не важливі і не термінові. Ці завдання не важливі, вони не термінові, але саме їх хочеться робити. Це пожирачі часу, і від них необхідно позбавлятися.

У момент втоми багато розробників починають займатися сторонніми справами, щоб відпочити. Це неправильно. Правильно -запланировать якісний відпочинок відповідно до індивідуальних особливостей кожного розробника. Це завдання з категорії "Важливо і не термінове".

Принцип Ейзенхауера - дуже ефективна техніка розставляння пріоритетів. Але її особливості більше спрямовані на розставляння пріоритетів окремого співробітника, а не команди в цілому.

- Методика "АБВ". Відповідно до цієї методики завдання діляться на три категорії: життєво важливе, важливе, приємне. Ця методика є логічним("глибиннішим") продовженням принципу Ейзенхауера.

Застосування принципу Парето конкретизується, якщо усі завдання проаналізувати відповідно до їх долі в підсумковому результаті і потім розподілити по категоріях важливості. АБВ грунтується на наступних трьох закономірностях, підтверджених досвідом:

* + найкритичніші завдання(категорія "А") складають ~ 20% усіх завдань, якими зайнятий керівник. Значущість цих завдань, вкладу в досягнення кінцевого результату складає ~ 60%.
  + Менш важливі завдання(категорія "Б") складають ~ 20 % від загального числа. Їх значущість - також 20% значущості завдань і справ керівника.
  + Неважливі і несуттєві завдання(категорія "В") складають 60% від загального числа завдань. Вони мають незначну долю - 15% в загальній цінності усіх завдань.

Методику "АБВ" можна застосовувати для оцінки пріоритетів завдань, необхідних для розробки продукту, але для того, щоб грамотно її використати, власник продукту і команда повинні мати певний досвід і навички по розставлянню пріоритетів.

*-* Метод MoSCoW(Oracle). Один з консультантів Oracle(Dai Clegg) запропонував однозначний, логічний і зрозумілий метод приоритизации вимог для аналізу і завдань розробки програмного забезпечення - MoSCoW. Цей метод використовується при фіксованих термінах на реалізацію функціональності, коли уся увага має бути звернена на найпріоритетніші вимоги. Приоритизация завдань методом MoSCoW дозволяє зосередити фокус уваги як власника продукту, так і Scrum- команди на завданнях конкретного спринту.

Усі наявні завдання у беклоге продукту слід згрупувати в чотири пріоритетні кошики за наступними правилами:

* + M(must) - завдання повинно бути реалізовані в першу чергу, без них програмний продукт не має сенсу. Від цих завдань не можна відмовитися. У них знаходиться запорука успіху. Завдання, відмічені як must, мають бути включені в поточний спринт.
  + S(should) - слід було б мати, але можна відкласти на пізніший час. Завдання цього типу також критичні для успіху продукту, що розробляється, але не потрібні найближчим часом. Такі завдання, як правило, мають альтернативні шляхи рішення.
  + C(could) - можна було б мати, але якщо немає можливості їх розробити зараз, то можна і відкласти. Завдання цього типу менш критичні. Вони зазвичай відносяться до типу "хотілося б мати".
  + W(would) - цього разу варто відмовитися від цих завдань, але наступного разу можна їх зробити. Найменш критичні. Завдання цієї категорії не плануються до розробки в найближчі спринти, але при майбутніх постачаннях їх статус може бути переглянутий.

Розглянута нами техніка дозволяє виконувати адекватну оцінку завдань, включення яких потрібне для розробки якісного програмного продукту.

## 8.4. Дошка завдань

Після того, як ми визначилися з "спринтовой" моделлю організації робіт в Scrum і розібралися з тим, як необхідно працювати з вимогами і завданнями, настав час розглянути ще один дуже важливий артефакт кожної Scrum- команди, мета якого - візуалізувати стани завдань в Scrum. Йдеться про Scrum- дошці.

Кожен робочий день усі члени команди збираються на 15-хвилинне daily, на якому вони обмінюються інформацією за станом поточних справ("Що зроблене з моменту попередньої зустрічі"?, "Чим планую займатися сьогодні"?, "Які є перешкоди"?).

На дошці команди розвішені завдання, кожною з яких виділений паперовий стикер, на якому написано, в чому полягає суть завдання ( рис. 8.3).



Рисунок 8.3 –Дошка завдань

Дошка завдань повинна мінімум ділитися на три колонки:

* заплановано(To Do);
* в роботі(In Progress);
* завершено (Done).

На момент планування завдань, які належить узяти в спринт, усі картки поміщаються в колонку "Заплановано". Щодня, коли член команди бере в роботу певне завдання, він говорить: "Я почав працювати над". і переміщає картку в стовпець "In Progress". Після того, як завдання виконане, виконавець говорить про те, що робота над завданням закінчена, і картка переміщається у відповідну колонку "Завершено".

Використання дошки завдань повністю відповідає принципам прозорості робіт, які є однією з головних переваг Agile порівняно з альтернативними підходами до розробки програмного забезпечення. Учасники daily щодня на власні очі бачать, як йде прогрес в роботі над тим або іншим типом завдання. Вони не лише занурюються в значимі деталі робіт, але і отримують цінний досвід. Безперечними вигодами, які отримує команда від використання дошки завдань, є:

* наочність спринтів;
* прозорість стану завдань і проблеми;
* простий контроль за "загрузом" розробників і іншого.

Варто сказати про те, що на початку своєї діяльності кожна команда повинна спробувати саме "матеріальну" дошку, по якій кожен зможе переміщати свої завдання.

По-перше, це привчає усіх членів команди до певної дисципліни. Кожен повинен відзвітувати за свої завдання і як саме над ними ведеться робота.

По-друге, дошка, яка постійно висить на стіні в одному місці, привчає тих, що усіх, що відносяться до команди, що саме на ній можна побачити актуальний стан справ і завдань.

## 8.5 Беклог продукту

У теорії Agile- процесів робота з вимогами і їх трансформація в призначені для користувача історії, карту процесів і набір окремих завдань - це автономний процес, за який відповідальний власник продукту і виділені бізнес користувачі.

На практиці ж цей процес вимагає творчості, навичок і немало ресурсів. Задати високий темп виконання Scrum- процесу за рахунок прозорості вимог, що поставляються для команди, - копітка праця, відома як інженерія вимог. Не обов'язково, щоб цією сферою завідував певний фахівець, більше того, в Scrum- команді цей функціонал має бути розосереджений між усіма членами колективу.

Інженерія вимог є набором техніки, методів і принципів, пов'язаних один з одним.

Інженерія вимог містить величезний потенціал для підняття рівня технічних навичок як розробників, так і власника продукту, з тим щоб створити правильний продукт.

Останнім часом стала поширеною точка зору, що вимоги не грають значимої ролі в гнучких процесах розробки. На це можна заперечити, що знання вимог потрібне, щоб привнести їх суть до створюваного продукту.

Ключові моменти інженерії вимог можна знайти, порівнюючи схожість в роботі по управлінню вимогами в традиційній і Agile- середовищу. Це веде до усвідомлення, яка техніка може підійти конкретному типу процесу.

Основна помилка відносно управління вимогами полягає в тому, що воно торкається тільки належного документування. Кінцева мета процесу збору вимог в тому, щоб полегшити і донести до усіх зацікавлених загальне розуміння вимог до створюваного продукту. На практиці підтверджено, що важливо як мінімум промовити вимоги, щоб переконатися, що усі розуміють їх однаково. Способи інженерії вимог можуть розрізнятися при традиційному і гнучкому підході, але кінцева мета залишається тією ж, і важливо тримати її в думці, щоб не заплутатися, застосовуючи конкретні технології.

Техніка збору вимог схожа при традиційній і гнучкій роботі, як можна побачити на наступних прикладах:

* Завдання --> Рішення.
* Загальне --> Частка.
* Облік впливу різних чинників --> Поліпшення якості продукту.
* Інше.

Якщо усі вимоги будуть вказані разом в усіх необхідних для процесу розробки деталях, то це точно не Agile. Спринт містить обмежений набір вимог, з якими працює Scrum- команда. Кожен спринт синхронізує розуміння власника процесу і команди про кінцевий результат, до якого повинна прагнути команда. У Scrum спринт містить тільки ті завдання, реалізація яких потрібна в короткостроковій перспективі. Їх детально розглядають і реалізують командними зусиллями. Це призводить до того, що власникові продукту швидко показують результат і можуть отримати від нього ефективний зворотний зв'язок, який дозволяє більше дізнатися про вимоги до продукту і оптимальним чином скоректувати робочий процес.

Техніка гнучких процесів розробки відносно вимог заснована на ідеї спільних зусиль відносно набору необхідних вимог, на відміну від класичного підходу, при якому спеціальні співробітники відразу розробляють вичерпний опис. Команди гнучкої розробки вважають за краще збирати вимоги способом, який підтримує взаємодію і рухливість.

Призначені для користувача історії, що створюються на основі вимог, розробляються таким чином, що головним в них виступає користувач. При традиційному підході головним суб'єктом є система. Крім того, традиційний підхід часто ділить функціональні і нефункціональні вимоги, зазвичай розділяючи їх на різні глави специфікації. Це ділення призначене забезпечити повноту збору вимог. У гнучкому підході і функціональні, і нефункціональні аспекти збираються разом.

Це веде до кращого розуміння усіма залученими сторонами, виявленню нових деталей.

Після того, як необхідні вимоги до продукту зібрані, їх необхідно зафіксувати і "узяти на облік" для подальшої реалізації. Для цього в Scrum є артефакт, який називається беклог продукту(Product Backlog, PB).

Беклог продукту - це впорядкований список завдань, які мають бути реалізовані в кінцевому продукті.

Беклог продукту є єдиним джерелом вимог для будь-яких змін, які може потрібно внести в ПО. Відповідальність за беклог продукту несе власник продукту, включаючи його вміст, доступність, впорядкування і приоритизацию.

Беклог продукту ніколи не є повним. На початку(згори) - тільки спочатку відомі і найбільш зрозумілі вимоги. Беклог постійно оновлюється у міру оновлення самого продукту і середовища, що оточує його, тому беклог продукту "живе" разом з продуктом. Беклог продукту являється динамічним, постійно таким, що змінюються для відповідності вимогам продукту, його конкурентоспроможності і придатності. Беклог продукту існує рівно до тих пір, поки існує і сам продукт:

1. Беклог продукту містить усе "фичи", функції, вимоги, удосконалення і інформацію по виправленню дефектів, тобто ті дані, які і визначають зміни, необхідні в наступних релизах продукту. Кожному елементу беклог продукту привласнюється опис, порядковий номер, оцінка об'єму роботи і цінність.
2. Елементи беклога продукту, розташовані згори, мають бути зрозумілішими і містити більше деталей, ніж ті, які розташовані нижче. Точніші оцінки даються вимогам, які є чіткішими і містять більше додаткової інформації.
3. Чим нижче знаходяться вимоги, тим менше деталей. Беклог спринту є репозиторієм завдань, реалізація яких дозволить инкрементально рухатися до досягнення бажаного кінцевого результату від розробки конкретного програмного продукту.

## 8.6 Беклог спринту

Якщо беклог продукту містить повний список існуючих завдань, пов'язаних з розробкою програмного продукту, то в Scrum є ще один артефакт, який містить список завдань, які необхідно виконати в спринті.

Беклог спринту(Sprint Backlog SB) - це набір елементів Product Backlog, вибраних для виконання в поточному спринті.

SB - це прогноз і зобов'язання Scrum- команди відносно функціональності, яка стане частиною інкремента, що розробляється, за час спринту.

Беклог спринту:

* наповнюється під час планування робіт на спринт;
* візуалізується на дошці завдань;
* визначає той об'єм роботи, яку Scrum- команда повинна виконати, щоб перетворити вимоги і завдання на готовий для використання у бізнес-процесах інкремент.

Беклог спринту має бути досить "глибоко" деталізований в порівнянні із завданнями в Product Backlog, щоб прогрес роботи над ним можна було бачити щодня, не приділяючи додатковий час збору необхідних деталей.

Scrum- команда працює над беклогом спринту на усьому його протязі, і він постійно змінюється разом з прогресом команди.

Зміни відбуваються тому, що в процесі роботи виникають усі нові і нові завдання, які треба виконати для досягнення кінцевого результату.

Якщо виникає необхідність в додатковому об'ємі роботи, то ці завдання додаються у беклог продукту і плануються до виконання в наступних спринтах.

Після того, як вони виконані, оцінки об'єму робіт, що залишився, оновлюються.

Якщо деякі завдання вважаються вже неактуальними, то їх видаляють.

Тільки Scrum- команда може змінювати свій беклог під час спринту. За рішенням команди і її учасників завдання можуть додаватися або віддалятися, але необхідно чітко обгрунтувати, чому це необхідно.

## 8.7 Інкремент продукту

Мета команди Scrum в окремо взятому спринті - надавати працюючий інкремент продукту.

Працюючий "приріст" продукту(інкремент) - обов'язковий результат кожного спринту Scrum.

У складі беклога спринту можуть знаходитися абсолютно різні завдання, але головне для команди - створення інкремента програмного продукту. Окремий інкремент може включати недостатньо функціональності для прийняття власником продукту рішення про його введення в експлуатацію, але команда повинна переконатися, що якість, що поставляється ними, достатня для постачання і запуску його в продуктивну експлуатацію.

Інкремент за своєю суттю - це продукт, який приноситиме бізнес-користь(рис. 8.4) у момент своєї розробки після того, як результати певного спринту реалізовані Scrum- командою і прийняті замовником.

| 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 – Наочне зображення інкремента у будівництві

На рис. 8.4 приведені дві ілюстрації.

Перша(ліва) ілюструє неинкрементальный процес організації спорудження будинку. До тих пір, поки будинок в цілому не буде готовий, в нім не можна буде жити.

На другій(права) продемонстрований инкрементальный процес будівництва. Після того, як перший поверх буде готовий, у будинку можна починати жити і при цьому продовжувати будівництво другого поверху.

Мета спринту - сформувати, спланувати і розробити самодостатній, готовий для застосування продукт або його частину, яка доповнюватиме вже наявні розробки і підвищуватиме сумарну вартість програмного продукту, що розробляється.

Після закінчення спринту інкремент повинен годитися до використання і приносити реальний і відчутний прибуток PO.

Архітектура системи або дорожня карта продукту можуть бути побудовані таким чином, що поставити інкремент продукту в терміни спринту - це дуже складне завдання, а іноді - нездійсненна. Таким чином, ми повинні повернутися до самої ідеї наповнення беклога продукту однозначними і зрозумілими завданнями і переглянути наявні вимоги у бік ітераційної і инкрементальной роботи.

Инкрементальная модель розробки програмного забезпечення є найбільш успішною і перспективною моделлю створення і впровадження програмних продуктів.

## 8.8. Принцип прототипування

Scrum грунтується на "итерационно-инкрементальной" розробці програмних продуктів, які з моменту свого "зачаття" приноситимуть компанії прибуток. Крок за кроком.

А для цього кожен окремий інкремент, що реалізовується, повинен мати певну цінність сам по собі, при цьому додаючи цінності до раніше реалізованих інкрементів. У зв'язку з цим робота з вимогами і як наслідок з архітектурою створюваного програмного продукту трансформується в процес постійного прототипирования рішення за актуальними вимогами. У зв'язку з цим підвищується важливість окремих активностей самих по собі і їх подальшій інтеграції один з одним.

* Робота з вимогами. Їх аналіз, синтез і подальше проектування мають первинну значущість. Необхідно так збудувати процес розробки, щоб кожен подальший інкремент, що розробляється, інтегрувався в існуючий продукт і при цьому оптимізував його з точки зору підвищення цінності вироблюваних інформаційною системою результатів.
* Вимоги мають бути взаємозв'язані, трассируемы по вже реалізованому функціоналу. У кожен момент часу робочий продукт має бути самодостатній з точки зору цінності. Тому важливо правильно проводити аналіз впливу виконуваних розробок з точки зору існуючого функціонала і місця в інформаційному ландшафті замовника створюваного програмного продукту.

З урахуванням позначених чинників підвищується значущість проведення постійного рефакторинга програмного продукту унаслідок його постійної мінливості. При цьому мета і напрям робіт мають бути чітко визначені, і завдання Scrum- команди і власника продукту - постійно відстежувати тренд розвитку продукту.

Подібний принцип організації і управління робіт отримав раніше поширення в області розробки програмного забезпечення і називається принципом прототипирования. Цей принцип якнайкраще підходить до проектування інформаційних систем, що розробляються відповідно до Scrum.

## 8.9. Короткі висновки

У цій главі увага була приділена тим атрибутам, які виділяють Scrum в найбільш формалізований вид гнучких методологій. Саме вони дозволяють говорити про те, що Scrum є найзрозумілішою і ефективнішою методологією розробки програмного забезпечення, впровадження якої є відносно нескладним процесом.

Використання в операційній роботі дошки завдань, беклога продукту і спринту дозволяє зробити прозорим процес роботи над призначеними для користувача історіями і картою завдань, в кінцевому виді що виражає результат у вигляді готового до використання інкремента продукту.

**Питання до теми:**

1. Що таке карта історій?
2. Опишіть процес створення призначених для користувача історій. Охарактеризуйте хто і з якими компетенціями може описувати призначені для користувача історії?
3. Яким чином здійснюється визначення пріоритетів користувачів?
4. Розкрийте поняття беклог продукту і беклог спринту?
5. Що передбачає принцип прототипирования в методології Scrum?

# Лекція 9: ОЦІНКА

План

9.1 PERT - оцінка термінів

9.2 Перехід від оцінки до зобов'язань

9.3 Збалансована система показників Scrum- команди

9.4 Напрацьована статистика результатів - фундамент прогнозування і перемог

9.5 Короткі висновки

## 9.1 PERT - оцінка термінів

Адекватна оцінка трудомісткості виконуваних робіт - майстерність управління, якого досягають тільки фахівці, що постійно стикаються з необхідністю професійно оцінювати ресурси, необхідні для реалізації завдань.

На даний момент накопичені досить багато різних методик, що успішно зарекомендували себе в різних умовах експлуатації. Існуюча техніка розрізняється по наступній, важливій для застосування в операційній діяльності групам характеристикам:

* наукова обгрунтованість застосування;
* необхідність наявності статистичного репозиторія даних по виконанню в альтернативних процесах/проектах;
* можливість оперативного використання і отримання результату;
* наявність певного інструментарію;
* інші.

Після того, як порівнянню було піддано більшість методик, стало очевидно, що для використання в Scrum підходить зовсім не багато хто.

З одного боку, використовувана методика повинна мати науковий базис, на якому було б можливо проводити обгрунтовані обчислення трудомісткості робіт, необхідні для кожного конкретного завдання, але, з іншого боку, вона не має бути громіздкою для застосування в плані часу обчислення і складності використовуваного інструментарію. Таким чином, частина високоточної техніки, заснованої на складному математичному апараті і тривалих обчисленнях, відпала сама собою, так само як і техніка, яка у своїй основі використовує переважно експертні методи оцінки трудомісткості. В лідери вирвався комбінований метод, званий PERT, вживаний в проектному управлінні.

PERT(Program Evaluation and Review Technique) - техніка перевірки і оцінки трудомісткості виконуваних робіт.

Після того, як призначені для користувача історії і завдання деталізовані на необхідному рівні, слід мати адекватну оцінку трудомісткості їх виконання. Під трудомісткістю розуміють сукупність наступних значень:

* Термін розробки. Під терміном розробки розуміється дата до якої необхідно підготувати необхідний програмний продукт, його функціональність або окреме завдання. Термін розробок в реаліях сьогоднішнього ринку виробництва програмних продуктів багато в чому зумовлює якість виконуваних робіт і їх об'єм. Коли ж ми говоримо про гнучкі процеси розробки програмного забезпечення, то оцінка термінів робіт відходить на другий план, поступаючись місцем об'єму робіт і їх якості. Термін розробки заміщається поняттям спринтів. Замовник, побачивши на демо "працездатний" результат, готовий до застосування, сам визначає, скільки ще спринтів знадобиться, щоб досягти бажаної їм якості.
* Якість розробки. Здатність підтримати задану якість програмного продукту - якість Scrum- команд, що тільки самоорганізовуються. Якість програмного продукту, задана командою, залежить від безлічі різних чинників, але визначається кваліфікацією членів команди. Проте необхідно пам'ятати, що замовник, підвищуючи вимоги до якості, платитиме за це збільшенням термінів робіт(кількості спринтів);
* Об'єм виконаних робіт. Ця характеристика безпосередньо пов'язана з показниками якості інформаційної системи, що розробляється. Чим вище рівень необхідної якості, тим більший об'єм робіт необхідно реалізувати для досягнення заданих показників, а відповідно, більше часу витратити на те, щоб досягти реалізації кінцевого продукту.

Приведені характеристики, оцінка яких потрібна для розробки програмного продукту або його інкремента у рамках Scrum, не суперечить, а тільки підтверджує вірність твердження з проектного управління про "трикутник ресурсів", використовуючи які, завжди доведеться пожертвувати одним заради досягнення максимальних показників двох що залишилися.

У основі методології Scrum лежить твердження про те, що "кінцеві терміни" розробки програмного продукту - рудимент, від якого необхідно позбавлятися. Scrum має у своїй основі "спринтовую модель".

За результатами кожного спринту замовник повинен отримувати готовий до використання інкремент функціональності програмного продукту. Замовник сам регулює необхідну кількість спринтів для досягнення кінцевого, задовільного для використання результату від розробки програмного продукту.

На практиці підтверджено, що найбільш точна оцінка лежить у рамках відносної погрішності ±5%. Цей допуск є цілком прийнятним для роботи по Scrum, але "потрапити" в нього необхідно як можна точніше. Для цього пропонується до використання методика PERT, адекватність і обгрунтованість якої підтверджені не лише теоретично, але і на масиві конкретних практичних прикладів. Метод PERT часто в літературі називають ще і "методом трьох точок".

У основі методу лежить оцінка трудомісткості робіт певного завдання, отримана від декількох експертів(як мінімум три). Усі отримані оцінки експертів класифікуються по наступних групах:

* оптимістична(a);
* очікувана(m);
* песимістична(b).

І в кожній групі обчислюється середнє значення, що відбиває групову оцінку планованих робіт, виражену в абстрактній або конкретній величині. Після того, як оцінки отримані, виконується їх усереднювання по досить простій формулі (рис. 9.1).

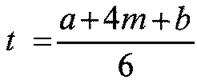


Рисунок 9.1 – Формула усереднювання

У формулі врахована кожна із заявлених "групових" оцінок, але при цьому основний вклад вносить очікувана оцінка. Методика PERT є універсальним інструментом оцінки будь-якої з характеристик "проектного трикутника". У ній поєднуються як експертні думки, так і математична основа, яка дозволяє нівелювати можливі ризики крайніх оцінок.

Після того, як будуть заздалегідь оцінені усі завдання, передбачувані до включення у беклог спринту, стає можливим впливати на тривалість спринту або його об'єм завдань. У тому випадку, коли компанія в змозі достовірно оцінити об'єм необхідних робіт, спринти можуть бути сформовані відповідно до оціненої тривалості робіт. Ще раз зробимо акцент на тому факті, що гнучкі процеси розробки - це спосіб організації процесів, і якщо компанія зацікавлена в його застосуванні, то можливо Scrum "вписати" у будь-яку організаційну структуру і налагодити тісне співіснування різних методик управління процессной діяльністю.

## 9.2 Перехід від оцінки до зобов'язань

Оцінки, що даються окремими членами або командою в цілому, і зобов'язання - це не одне і те ж. Фундаментальна і, мабуть, основна проблема багатьох фахівців і організацій полягає в тому, що оцінки, що даються, і зобов'язання, які вони при цьому переймають на себе, в очах менеджменту вважаються одним і тим же.

Розробники і команди, якій би методології вони не дотримувалася, прикидають, що реалізація бажаної сукупності інкремента - за наявності у команди певних ресурсів - займе певний час. Члени команди повідомляють це керівництву або Scrum- майстрові, після чого вона транслюється замовникові.

В деяких випадках в процесі таких передач критичної для розробки інформації від одних осіб до інших первинна оцінка змінюється до невпізнання, внаслідок чого перед командою ставиться нереальна мета.

Проблема в подібних випадках полягає не в тому, що оцінка команди виявилася правильною або неправильною, а в тому, що оцінка трансформувалася в зобов'язання, які доводиться брати на себе команді. Важливі і оцінка тривалості виконання проекту, і зобов'язання, перейняте на себе командою, проте їх необхідно розглядати як різні види діяльності.

Scrum- команда і сучасні організації повинні уміти відділяти оцінки від зобов'язань. Оцінка завжди первинна, а вже після, грунтуючись на мірі упевненості, досвіді, навичках, ми можемо перетворити її в певне зобов'язання.

Але без надійної первинної оцінки будь-яке зобов'язання команди буде позбавлено всякого сенсу. Щоб отримати надійну оцінку, власник і Scrum- команда повинні мати в розпорядженні усі необхідні для цього даними:

* об'єм роботи, яку належить виконати;
* чинники, що впливають на темп виконання командою роботи, і, як наслідок, динаміка виконання завдань в спринті.

Спочатку при впровадженні Scrum практикується оцінка у вигляді діапазону, потім, у міру зрілості, команда буде в змозі перейти до точкової оцінки, яка використовуватиметься як зобов'язання.

Подібний перехід на більш високий рівень самоорганізації в тактичній перспективі обіцяє команді проблеми з виконанням зобов'язань, але в стратегічній перспективі це допоможе понизити ризик в порушення оцінок і зобов'язань, що даються командою.

Альтернативна ситуація, коли перед командою відразу ставиться відповідальність по реалізації певного об'єму до заданої дати, більше рискованна і у віддаленій перспективі не принесе бажаних результатів.

При впровадженні Scrum в організаціях ефективніше рухатися инкрементально, по кроках, постійно підвищуючи задану планку результатів.

Оцінки і зобов'язання за своєю суттю розрізняються мірою відповідальності за формування кінцевого результату. У разі, якщо команда готова брати на себе зобов'язання по реалізації програмного продукту і його функціональності за певний термін, це свідчить про зрілість її членів і високий рівень якості продукту, який вони створюють.

Якісний перехід від оцінки робіт до зобов'язань свойственен висококваліфікованим командам, які добре знайомі з тим продуктом, який вони розробляють.

Для того, щоб допомогти командам намацати свій оптимальний темп роботи і розкрити їх кращі якості, має сенс замислитися про застосування до Scrum збалансованих системних показників.

## 9.3 Збалансована система показників Scrum- команди

Варіант комплексного погляду на процеси сучасної компанії привів до створення збалансованої системи показників(ССП).

Її ідея полягає в наступному - щоб якнайкраще утямити функціонування будь-якої організації, не можна обмежуватися тільки аналізом показників про прибутки і збитки і фінансових звітів. Вони є лише обмеженим баченням того, як функціонує відповідне підприємство.

Мета ССП полягає в аналізі не стільки фінансових метрик функціонування підприємства, скільки в розгляді чотирьох процессных напрямів(фінанси, клієнти, процеси, навчання і зростання), що в сукупності представляють основу діяльності кожного підприємства.

Якщо спробувати встановити Scrum- команді KPI тільки по їх спеціалізованому напряму розробки програмного забезпечення, то керівництво організації зіткнеться з тим, що діяльність команди буде спрямована не на досягнення цілей діяльності компанії, а на оптимізацію позначеного показника.

Відомо і не ставиться під сумнів, що, якщо ми вводимо який-небудь показник і повідомляємо команду про те, що їх робота оцінюватиметься за цим показником, вони постараються адаптувати свою діяльність так, щоб оптимізувати саме цей показник.

Якщо робота оцінюватиметься, скажімо, по кількості виявлених системою відстежування дефектів, відповідний показник різко почне знижуватися. Можливо, із-за впровадження розумних удосконалень, можливо тому, що команді вдасться неформально інформувати один одного про дефекти або представляти їх в дещо іншій якості. Якби було можливо розробити показник, який неможливо "обійти" будь-яким способом, все одно один показник не відбивав би повну картину діяльності. Потрібно збалансованіший погляд, ніж той, який забезпечується представленням одного показника.

Система збалансованих показників повинна оптимізуватися під умови діяльності конкретної Scrum- команди, але напрями оцінки, як правило, співпадають з класичними метриками ССП, і вони, як правило, загальні для усіх успішних команд:

* Операційна досконалість:
  + Створення продукту з максимальною продуктивністю.
* Орієнтація на користувача:
  + Фокус на розробці пріоритетних функцій.
* Економічна цінність:
  + Забезпечення цінності у формі економії витрат, підвищення доходів і т. д.
* Орієнтація на майбутнє:
  + Створення нових продуктів також сприяє зростанню і підвищенню кваліфікації членів команди.

Вище представлені метрики, які можуть бути узяті за основу при побудові системи збалансованих показників окремої команди, що враховують конкретне оточення Scrum- колективу.

Якщо створення збалансованої системи показників синхронізувати із створенням окремої команди, і команда, відділ або організація потім періодично оцінюватиметься за цими показниками, то прогрес неминучий. Команда, яка діє успішно в умовах використання Scrum, повинна проявити здатність до одночасного поліпшення по кожній з перелічених вище точок зору. Збалансована система показників перемикає увагу команди з повної зосередженості на гнучкій методології розробки на досягнення цілей, які привели конкретну організацію до освоєння гнучкої методології розробки.

Варто звернути увагу на те, що збір метрик, що підтримують впроваджені показники, навіть якщо йдеться про дуже прості показники, пов'язаний з витратами сил і часу і явно не належить досягнень, якими можуть похвалитися компанії. Необхідно розуміти, що на збір метрик витрачається досить багато сил і уваги. Цей процес повинен стати частиною операційної діяльності організації.

Слід вказати три основні переваги, які дає збір показників:

* Подолання можливої організаційної гравітації. Існують певні причини збереження ситуації, що була до початку перетворень. "Організаційна гравітація" прагне повернути нас до того стану, в якому вона знаходилася упродовж тривалого періоду часу. Періодичні оцінки і показники, що демонструють вигоди переходу до Scrum, виявляться найбільш сильним засобом, який допоможе здолати цю силу.
* Пропаганда Scrum. Система операційного оцінювання допомагає не лише долати "силу гравітації", але і явно демонструвати переваги Scrum по усій організації. Щоб прищепити інтерес до переходу на Scrum, дуже важливо розумно рекламувати основні успіхи на цьому шляху.
* Обгрунтований тренд розвитку Scrum. Показники допомагають зрозуміти, на чому слід зосередити зусилля для подальшого вдосконалення. Показники повинні призводити до певних дій. Якщо збирані дані не призводять до якихось дій і рішень, треба припинити збір цих даних і почати збір інших даних.

Зібрані метрики і показники сприятимуть виявленню "відстаючих" процесів і активностей в цілях їх подальшого вдосконалення і розвитку. Також постійний моніторинг показників допоможе тримати команду в тонусі, аналізуючи можливі відхилення в її діяльності і управляючи ними в цілях досягнення кінцевого результату.

Збалансована система показників має бути не "мертвою" системою показників, а живим і динамічним процессным framework, результати якого, накопичені у вигляді масиву інформації, дозволять прогнозувати з високою часткою ймовірності точні результати діяльності команди, залежно від певної кількості ресурсів, що маються в наявності.

## 9.4 Напрацьована статистика результатів - фундамент прогнозування і перемог

Висока швидкість роботи команди повинна скластися відповідно до історичних передумов роботи команди.

Як тільки стає необхідним донести цю інформацію до осіб, що приймають рішення, зазвичай у відповідь ми чуємо безліч обгрунтованих заперечень, таких як нестача ресурсів, а головне, відсутність можливості оперативна напрацювати необхідний масив даних. Але зробити це досить просто, навіть з урахуванням таких поширених проблем, як "нова команда" і чинник постійних змін як команди, так і об'єму необхідної функціональності.

Якщо йдеться про знову сформовану команду, то оптимальним рішенням у такому разі буде надати членам команди можливість виконати декілька спринтів разом, і тільки після цього у них з'явиться можливість перейняти на себе якісь зобов'язання.

Для того, щоб задати швидкість роботи для знову сформованої команди і закласти основу фундаменту оцінки, доцільно зібрати фокус-групу з фахівців, що мають схожий досвід роботи з тими, хто включений в Scrum- команду, і задіяти їх в оцінюванні тих завдань, які треба виконати. Це допоможе направити роботу команди в потрібне русло. При цьому, коли пройде декілька спринтів, результати поступово збільшуватимуться. Таким чином можна здолати проблему "втягування" команди в Scrum.

Але коли чисельний склад постійно міняється, виникають проблеми іншого роду. Цей чинник можна розглядати з багатьох точок зору. З одного боку - стабільність йде на користь. З іншого боку, тривала стабільність призводить до застою і поступової професійної і особової деградації.

Важливо при роботі зі змінами дотримуватися серединної позиції, уникаючи сильних перегинів, при цьому збираючи дані про те, як ті або інші зміни впливають на діяльність колективу в тактичному і стратегічному наближенні. Це дозволить бути готовим до будь-яких змін у складі команди і передбачати і управляти можливими впливами здійснюваних змін. Зміни треба відстежувати постійно, враховуючи різні чинники, після яких порушується стала діяльність команди.

У основі Scrum лежить проста ідея - організація постійного контролю діяльності, що виконується мотивованими професіоналами, адаптованого під конкретні умови роботи, з урахуванням чинника постійних змін як у внутрішньому, так і в зовнішньому середовищі.

## 9.5 Короткі висновки

Уміння складати ефективні плани - дуже важлива якість для будь-якої Scrum- команди. У цій главі ми розглянули способи, за допомогою яких команда, що освоїла основи планування і оцінки трудомісткості спринтів, може поліпшити свої результати. Крім того, ми дійшли висновку про те, що поняття оцінки і зобов'язань, що даються командою і сприймані менеджментом компанії, розрізняються мірою відповідальності. Усі команди, що досягли потрібного рівня зрілості і самоорганізації, повинні прагнути до того, щоб давати обгрунтовані зобов'язання і надалі дотримуватися їх.

Ми поговорили про систему збалансованих показників в додатку до гнучких процесів виробництва програмних продуктів. Якщо оптимально адаптувати техніку побудови показників, використовувану в ССП, під діяльність Scrum- команди, то можна прогнозувати збільшення продуктивності команд і їх поступовий розвиток і вдосконалення.

Фундамент, виражений у вигляді збираної статистики і метрик процесу Scrum, напрацьований за період впровадження і застосування гнучких методологій, не повинен залишатися без уваги. Він має бути покладений в основу системи прогнозування і розвитку Scrum.

**Питання до теми:**

1. Що розуміють під трудомісткістю виконуваних робіт?
2. Оцінка яких характеристик потрібна для розробки програмного продукту або його інкремента у рамках Scrum?
3. Чим відрізняються оцінки і зобов'язання за своєю суттю?
4. Опишіть збалансовану систему показників Scrum- команди.
5. Які основні переваги дає збір показників(метрик) як частини організаційної діяльності підприємства?

# Лекція 10: ПІДСУМКИ І ПЕРСПЕКТИВИ

План

10.1 Співіснування з альтернативними процесами послідовної розробки програмного забезпечення

10.2 Забезпечення відповідності кращим практикам і стандартам

10.3 Використання Lean- методології в Scrum- процесі

10.4 Продуктивність Scrum для цифрової трансформації

10.5 Сучасна критика Agile

10.6 Що далі

## 10.1 Співіснування з альтернативними процесами послідовної розробки програмного забезпечення

У третій лекції ми розглянули і досить детально порівняли класичний, ітераційний і гнучкий підходи до розробки програмного забезпечення, а також постаралися обгрунтовано розвінчати міфи про несумісність спільного використання різних підходів.

Багато в чому такі сторонні точки зору обумовлені протилежним досвідом і ситуацією, в яких здійснювалося і здійснюється експлуатація цих підходів. Але, незважаючи на цей дуалізм, багато сучасних дослідників і фахівці констатують, що можна придбати багато переваг від одночасного використання обох методів в організації.

Одна з найбільш авторитетних організацій в області процесів SEI(організація - автор моделі CMMI) опублікувала технічні замітки, в яких викладається, чому можна використати обидва методи одночасно.

Різні причини ініціювали зростання даних методів. У багатьох організаціях, які прийшли до ідеї впровадження Scrum або будь-якої іншої гнучкої методики, він приходить на зміну класичному підходу. Це відбувається не одномоментно, а впродовж певного інтервалу часу, за який організація повинна здійснювати свою повноцінну діяльність(приносити доход, розвиватися і т. д.), тому виникають перетини цих методів:

* Водоспад на початку. Використовується і ефективний у тому випадку, коли йдеться про запуск проекту або впровадження нового продукту у рамках жорстких термінів. Інакше виникає загроза зриву проекту.
* Водоспад у кінці. Зазвичай відбувається на фазі приймального тестування функціональності розробленого продукту.
* Водоспад в тандемі. Найскладніший спосіб взаємодії. Виникає у разі, якщо команди використовують різні методології розробки ПО. Вирішується у тому випадку, якщо ряд членів команди, що використовує ваговиті технології, бере участь в робочих зустрічах Scrum- команд.

Молоді фахівці, що практикують гнучкі методології, піднімаються по кар'єрних сходах і бачать вплив водоспаду на гармонійний процес гнучкої розробки. Приміром, на рівні проектів впровадження класичний підхід фокусується на тому, щоб продукти розроблялися в повному об'ємі, тоді як гнучка модель концентрується на тому, як його розробляти.

Багато сучасних компаній поставлені перед необхідністю поєднувати основні процеси, побудовані за класичним принципом розробки програмного забезпечення, з наявністю ефективних команд розробників, що використовують гнучкі методології. З іншого боку, є прецеденти, коли в переважно "гнучкому" середовищі використовується обмежений за часом ітеративний підхід.

Немає процесу, який однаково ефективний за різних умов функціонування.

Agile робить акцент на мобільності, змінах і комунікації, а класичний підхід багато хто звинувачує в зайвій вимогливості до середовища використання, що раціоналізувало.

Організації роблять непоправну помилку, якщо є адептами лише одній технології. Вони обмежують себе в розгляді найбільш ефективних віянь і поступово приходять в стан повної зашоренности.

Поточні віяння говорять про те, що найбільш ефективні організації з часом все більше уваги приділятимуть поєднанню гнучких і класичних методологій розробки програмного забезпечення.

## 10.2 Забезпечення відповідності кращим практикам і стандартам

Не кожна Scrum- команда може дозволити собі "матріархально" володіти повним циклом управління використовуваним нею процесом розробки. Як приклад можна привести розробку програмних продуктів за участю субпідрядників, до яких у більшості випадків висуваються вимоги по підтвердженню рівня зрілості їх власних процесів області інформаційних технологій.

По суті це означає, що розробники програмного забезпечення повинні використати схожий набір передових методів і застосовувати в практиці створення програмних продуктів такі стандарти, як ISO 9001(процессные стандарти), ISO 13458(стандарт медичної галузі), вимоги закону Сарбейниса-Оксли(акціонерні компанії на території США) і ін. Таким чином, одному із завдань Scrum- команди, спрямованої на розробку якісного програмного продукту, являється постійна турбота про те, як інтегрувати профільні стандарти в створюваний ними продукт.

Scrum- команда поставлена в умови, коли сама відповідає за кінцевий результат і при цьому мінімальним чином залежить від зовнішнього оточення. У таких умовах зростає необхідність постійного вдосконалення використовуваних процесів в плані відповідності галузевим і міжнародним практикам для цілей найменш витратній синхронізації отримуваних результатів з результатами діяльності можливих партнерів і колег. У разі, якщо цього не відбуватиметься, команда ризикує відстати від реальності ринку як процессно, так і технологічно, і розробляти "відсталий" продукт.

Членам Scrum- команд необхідно піклуватися про те, щоб дотриматися набору правив і принципів, пропонований "best practice" і профільним стандартам. Для цього слідує:

* Організувати процес документування вимог до продукту в погодженому командою форматі. Система документування вимог дозволить описати набір автоматизованих функцій і мати постійний доступ до майданчика оперативного обговорення подальших доопрацювань і розвитку функціонала продукту відповідно до актуальних вимог бізнесу.
* Створити базу знань команди/продукту. База знань команди/продукту - це свого роду інтелектуальний компонент команди, який дозволяє організувати процеси рішення виникаючих проблем шляхом документування в різних форматах(керівництво, описи, статті і т. д.) найбільш важливої інформації. Бази знань дозволяють накопичувати інформацію і перетворювати її відповідно до актуальних потреб команди. Основне призначення - надати допомогу найменш досвідченим співробітникам при рішенні певних проблем.
* Управляти змінами. Про важливість процесу управління змінами було сказано в попередніх главах. Тут же ще раз вкажемо на те, що управляти змінами треба на постійній основі не лише в розрізі розробки продукту, але і в розрізі організації "гнучких" процесів, оптимальним чином адаптуючи їх під потреби організації.
* Документувати актуальний стан процесів в команді. Для того, щоб на постійній основі займатися розвитком і вдосконаленням процесів по передових практиках і стандартах, необхідно мати актуальний стан процесів задокументованим. Тільки тоді буде можливість оперативно розглянути їх і прийняти управлінські рішення.
* Застосовувати передові технології для автоматизації процесів(тестування, збір вимог і т. д.). Члени команди, використовуючи процедури ретроспективи або механізми самоаналізу виконуваної ними роботи, повинні постійно замислюватися над використовуваною ними технікою, інструментами, професійними звичками. Найбільш "відсталі" з них, ті, ефективність яких низька, необхідно модифікувати відповідно до актуальних умов Scrum, підвищуючи їх цінність і значущість, або ж відмовлятися від них на користь нових. Описане так само вірно і для автоматизації процесів тестування, роботи з вимогами і ін.
* Шукати, тестувати і використати інноваційні інструменти, які оптимізують вашу діяльність. Члени команди повинні постійно знаходитися в технологічному пошуку. Якість постійної незгоди з чинниками, вплив яких негативно відбивається як на командній, так і на особистій роботі, - відмінна якість розробника. Розробник може, а іноді і повинен змиритися з недоліками конкретної ситуації, але при цьому йому слід постійно шукати шляхи рішення системних проблем. Він повинен постійно шукати, пробувати, змінюватися, пропонувати вдалі і відмовлятися від незручних робітників інструментів. Це, по суті, і є Agile.
* Запрошувати професійних аудиторів. Роль аудитора в реаліях Agile не треба недооцінювати. У команди, як правило, завжди все добре. Недоліки і недосконалість можуть "замилюватися" за операційною діяльністю колективу. Людська натура так влаштована, що вона поступово звикає до всього. До хорошого швидко, до поганого трохи повільніше. Аудитор - співробітник, який повинен неупереджено провести валидацию процесів. За допомогою його звітів можна об'єктивно зіставити, як у команди виходить рухатися у бажаному напрямі розвитку, виділити найбільш слабкі місця, на роботі яких варто зробити акцент. Аудитор може бути як зовнішній, так і внутрішній, головне в його роботі - бачити процес і ролі, а не ситуацію і окремих осіб.
* Запрошувати кваліфікованих консультантів. "Варитися у власному соку" дуже важливо. "Не варитися" порівнянно з "не розвиватися". В силу того що члени команди не завжди знають, як найбільш оптимально вирішувати виникаючі проблеми і чим прийняті рішення можуть "загрожувати" надалі, важливо притягати консультантів, які мають практичний досвід рішення проблем, з якими зіткнулася або може зіткнутися команда. Перш ніж їх запрошувати, слід переконатися в тому, що консультант зможе допомогти, а не прийде "відсидіти" свою вартість.

Роль аудитора і консультанта можуть виконати співробітники організації. При цьому вони повинні мати потрібний рівень кваліфікації, який дозволив би успішно виконувати покладені на них функції і обов'язки, а також відповідні риси вдачі.

Підсумовуючи, слід зазначити, що Scrum, як і будь-який інший процессное напрям діяльності, повинен розвиватися за рахунок прийняття і адаптації найбільш успішних практик, що довели свою ефективність. Крім усього іншого, важливо відмітити необхідність постійного вдосконалення профільного напряму діяльності. У нашому випадку йдеться про розробку програмного забезпечення, яка є однією з найдинамічніших областей і консерватизм в якій, як правило, обертається втратою швидкості розробки і впровадження нових продуктів.

## 10.3 Використання Lean- методології в Scrum- процесі

Методологія Lean спрямована на вдосконалення і процессное розвиток. Вона не підходить для тих процесів, які знаходяться на "зачатковому" і початковому рівнях зрілості. По суті, про необхідність впровадження методології Lean замислюються ті компанії, які досягли певного якісного рівня.

Lean - це маршрут для тих організацій, які вже упевнено стоять на процессных рейках. Вона має на увазі менше конкретних методик. Її застосовують в контексті власної організації, повністю адаптуючи під свої умови.

У Lean, так само як і в Scrum, робота розбивається на невеликі пакети(порівнянно з поняттям беклога спринту), які повинні реалізовуватися окремо і незалежно. Lean, в цілях розробки передбачуваного результату, містить певний потік операцій(workflow) з етапами. У Scrum існують свої специфічні процеси і процедури, які підтвердили свою ефективність. Lean додає до "гнучких" принципів схему потоку операцій, для того, щоб кожна з ітерацій виконувалася однаково якісно.

Обидва напрями у своїй основі мають системний підхід, заснований на постійному навчанні і вдосконаленні. В даному випадку Scrum є базисом, на основі якого стане можливим "надбудовувати" специфічні методи і методики, які сприятимуть підвищенню ефективності процесів компанії і, як наслідок, збільшуватимуть/покращувати результат.

Lean- методи спрямовані на "розширення вузьких шийок" і мінімізацію надмірної складності для підвищення ефективності Scrum. Це дозволить не лише змінити внутрішні процеси Scrum- команди, але і дасть можливість міняти процеси зовні.

Організація методології Lean і її етапів дозволяють бути упевненими в тому, що кожна частина проекту/продукту/процесу реалізується так, як цього вимагає результат. У Lean, як, втім, і в Agile, немає чітких межі етапів, але, як і в Scrum, прописані обмеження спринтів.

Lean дозволяє паралельно виконувати декілька завдань на різних етапах, що підвищує гнучкість і збільшує швидкість виконання проектів.

Як і гнучкі процеси, Lean більше схожа на концепцію, образ мислення, ніж чітко регламентовану методологію. Використовуючи принципи Lean, можна створити адаптивну систему, що задовольняє конкретним вимогам. Для цього необхідно наслідувати наступні принципи:

* Постійно визначати цінність конкретного продукту. Розробник повинен досліджувати продукт, що реалізовується, і спостерігати за ним. Більше того, у міру функціонування команди і розробки зрілого продукту кожен розробник повинен навчитися оцінювати те, які параметри необхідно мати, щоб стати цінністю.
* Визначити і контролювати потік створення цінності. Потік створення цінності - це послідовність дій, що здійснюється для перетворення початкового матеріалу в кінцевий продукт, що приносить цінність. Якщо конкретний етап не створює цінності, доцільно провести його зміну або видалення з процесу.
* Забезпечити безперервність потоку створення цінності. Замість дискретної організації переміщення продукту по стадіях процесу необхідно розглянути можливість створення потоку, в якому відбувається безперервний рух продукту по стадіях через спеціалізовані точки(віхи), в яких буде можливість організувати операційний контроль як стадії, так і якості продукту шляхом виміру заздалегідь розроблених метрик.
* Дозволити користувачеві/власникові продукту "витягати" продукт. Користувачі, що знаходяться "вище" по процесу, нічого не повинні робити до тих пір, поки кінцевий споживач, розташований "нижче", цього не зажадає. Повинні створюватися усі умови для того, щоб споживач "витягав" продукт, тобто виробництво продукту здійснювалося під конкретне замовлення. Таким чином, саме кінцевий користувач запускатиме "витягування" по усіх етапах виробництва.
* Постійно прагнути до розвитку і досконалості. Розробку продукту в Lean можна удосконалювати нескінченно, максимально наближаючи його до "ідеалів" очікувань замовника. Можна скільки завгодно зменшувати трудовитрати, час на розробку і ін.

Підсумовуючи, варто сказати, що сильною стороною Lean є різноманітний набір інструментів і підходів до роботи для задоволення різноманітних вимог. Lean поєднує гнучкість і структурованість, але трохи відрізняється в цьому від Scrum.

Його недоліком є занадто детальне і дійшле опрацювання кожної стадії роботи. Lean припускає такий підхід до кожного завдання і етапу. Це основний мінус застосування Lean, коли йдеться про середні і дрібні продукти. На відміну від Scrum, Lean не пропонує чіткого робочого процесу для реалізації "шматочків", що сприяє розтяганню термінів проекту. Ця проблема може бути розв'язана за допомогою ефективного керівництва і чітких комунікацій.

Впровадження і подальше використання Lean повинні стати метою для тих організацій, в яких Agile вже завоювала своє місце і на ряду проектів підтвердила свою ефективність.

## 10.4 Продуктивність Scrum для цифрової трансформації

Необхідність цифрової трансформації(ЦТ) на сьогодні усвідомлена вже багатьма компаніями. Під ЦТ розуміється передусім створення гнучкої процессной інфраструктури, здатної підтримати будь-які почини по розвитку бізнесу і сприяти досягненню намічених результатів в як можна більше стислі терміни.

Те, як сьогодні працюють фахівці і як вони взаємодіють з клієнтами, визначає можливість підприємства стати лідером профільного сегменту ринку і добитися успіху.

Не для кого вже не секрет, що роль і вплив інформаційних технологій в цих процесах дуже великі. Для кінцевого споживача вже недостатньо інформаційних систем, які можуть задовольнити тільки мінімальні вимоги до автоматизації.

Користувачі чекають продуктів, в яких успішно поєднувалися б зручність в отриманні необхідної інформації з її подальшим аналізом і обробкою, як на своєму робочому місці, так і поза ним. Інтерфейс має бути зрозумілий без тривалого вивчення різних інструкцій, швидке виправлення проблемних ситуації у разі збою, а в ідеалі передбачення таких збоїв, - усе це невід'ємні характеристики робочого середовища успішної "цифрової" компанії. Метаморфоза, яка дозволить компаніям набути подібних рис, і називається цифровою трансформацією.

Вирішення проблеми метаморфози багато хто бачить в переході до гнучкого стилю роботи не лише в процесах розробки, впровадження і подальшої експлуатації ИТ-продуктов, але і в компанії в цілому. Scrum дозволить організувати стабільні, передбачувані постачання робочого інкремента корпоративних інформаційних систем. Більше того, бізнес-процеси, організовані по гнучких принципах, дозволять досягти максимального завантаження співробітників компанії і знайти ресурси для стратегічного або інноваційного розвитку організації.

Крім того, в якості основних характеристик компанії, що пройшла стадію цифрової трансформації, можна виділити наступні:

* Фокус на "розумінні" клієнтів. Результат від впровадження цифрової трансформації, як правило, розпочинається з плодів виділення основних категорій клієнтів компанії в тих або інших регіонах і ринкових сегментах. Більшість компаній звертаються до соціальних мереж, щоб зрозуміти переваги клієнтів і їх основні споживчі цінності.
* Реклама через соціальні медіа. Компанії вчаться бути ефективнішими в соціальних мережах і інших медіа. Для цих цілей створюються співтовариства, щоб консультувати клієнтів і формувати у них лояльність.
* Акцент на вимірних результатах. Підприємства використовують інструментарій аналітичних систем для детальнішого вивчення своїх явних і потенційних клієнтів, вивчення їх поведінкових моделей і ін. експерименти, що Проводяться, дозволяють динамічно регулювати ціни на продукцію залежно від попиту і інших умов.
* Трансформація точок контактів з клієнтами. Якість що надаються клієнтам продуктів/сервісів/послуг підвищуватиметься за рахунок інновацій по використанню цифрових технологій. Приміром, стали створювати аккаунти в соціальних мережах для прийому скарг. Цей підхід дозволяє оперативно вирішувати питання, притягати різні співтовариства для вирішення різноманітних завдань. Крім того, компанії, що використовують мультиканальне обслуговування, випробовують утруднення при забезпеченні комплексної підтримки каналів. Перехід до омниканальности(єдиному каналу спілкування) допоможе оптимізувати витрати і сконцентрувати компетенції і досвід на найбільш критичному виді послуг.
* Перетворення операційних процесів. Цифрова трансформація - один з найцікавіших і складніших процесів перетворення компанії, але при цьому він, як правило, призводить до необхідності трансформації пов'язаних з ним внутрішніх процесів.
* Зростання виручки. Компанії зацікавлені в цифровій трансформації і технологіях для того, щоб підвищити свою виручку, акумулюють інформацію про клієнтів для забезпечення точніших продажів і обслуговування, що персоналізується. Таким чином, компанія може в режимі реального часу відстежувати реакцію на свої продукти, що допоможе надалі коригувати взаємовідносини з клієнтами.

У цифровому бізнесі продуктивність кожного учасника бізнес-процеса визначається якістю роботи інформаційних систем. Цифрова трансформація компанії вимагає сильного керівництва - тільки воно може бути драйвером серйозних змін. Потрібно чітке розуміння того, які частини компанії і яким чином необхідно перетворити. Цифрова трансформація багато в чому є експериментом, плоди успішності якого дозволять компанії перейти на інший рівень зрілості. При цьому абсолютно не важливий профіль компанії - інформаційні технології в силах надати потрібний інструмент з необхідним рівнем налаштування, а якщо узяти за основу основні постулати і принципи Scrum, то це налаштування здійснюватиметься найбільш відповідним для конкретних умов чином і в максимально стислі терміни.

## 10.5 Сучасна критика Agile

Новачки, що познайомилися з Agile, упевнені в тому, що, застосувавши гнучкі методики до будь-якого типу процесу і організації, можна добитися високих результатів. Це не так. Це неправдива реклама. Agile в цілому і Scrum зокрема вимагають багатьох зусиль і осмисленого застосування для того, щоб добитися результатів.

Якщо вам заявляють, що варто запросити зовнішнього Agile coach або Scrum- майстра і вони зможуть виконати усю необхідну роботу, то це теж неправда. Сьогодні на ринку є безліч компаній, які не програмували і не мають досвіду керівництва людьми. Подібне "навчання" за допомогою таких фахівців не принесе значимих плодів. Люди, що займаються пропагандою і впровадженням гнучких практик, повинні уміти програмувати, аналізувати, керувати, тестувати і мати в цих навичках досить високий рівень компетенції. Якщо подібного досвіду у "всезнаючого" консультанта немає, то як з ним можна обговорювати специфічні ситуації, пов'язані з продуктом або Scrum? Консультант по впровадженню гнучких процесів, який отримав вищу освіту учора і не написав ні рядка коду, приносить менше користі, ніж члени команди. Не можна навчати тому, чого ти жодного разу не робив. І "робити" означає "робити багаторазово і постійно", а не мати в послужному списку один пілотний проект.

При цьому знайти причини невдач дуже просто. Їх і шукати не потрібно - вони, як правило, лежать на поверхні. Принципи Scrum пропонують, як треба діяти, але в них не описаний алгоритм роботи. Ця концептуальність критикується, але, якби гнучкі методології претендували на роль конкретної техніки, тоді їх використання не знайшло б такого широкого поширення. Алгоритм роботи має бути визначений в конкретній організації з урахуванням умов її діяльності і його поетапного коригування в цілях вдосконалення. Гнучкі методології не повинні нав'язуватися. Вони мають бути добровільно прийняті і схвалені колективом розробників, які зацікавлені в його застосуванні. Спроба нав'язати його застосування приведе до пригнічення ролі команди і подальшого відторгнення.

Це лише малих перелік проблем, які призводять до критики культури Agile. Компаніям, які прийшли до необхідності впровадження і використання цієї методології, рано чи пізно доведеться ці проблеми вирішувати. Ігнорування і замовчання приведе до "тихих" або явних революцій. Підсумовуючи, виділимо наступні постулати:

* Догматичность. Agile припускає адаптивний, гнучкий і креативний framework, який народжує фанатиків, "повернених" на його використанні. Без професійного запалу складно постійно удосконалювати будь-яку методологію, але надмірне захоплення нею відлякує потенційних користувачів.
* Сектантство. Не варто замикатися на Scrum, слід звертати увагу і на інші підходи до розробки.
* Домінування консультантів. Переважання колишніх "гуру", які стали теоретиками і відірвані від реальної практики роботи.
* Люди. Розрахунок на жорстку самодисципліну. Психологічне, емоційне, професійне вигорання членів команди. Залежність від злагодженої командної роботи. Залежність від ресурсів команди і окремих "лідерських" персоналій. Необхідність постійної участі в команді компетентного керівництва, зацікавленого в досягненні високих результатів.

Agile - дуже вимоглива культура. Мінятися і гнутися належить усім.

Коли менеджмент приходить до необхідності впровадження гнучких методологій, то необхідно усвідомити, що бути гнучкими чекає усі. Якщо команді у рамках її повноважень доручено виконувати ряд дій, приймати рішення і нести за них відповідальність, то це повинно бути дійсно так. Якщо команда не приймає рішень, то це руйнує її зсередини. Команда повинна прийняти рішення про те, що і як вона розвиватиме і що для цього потрібне. Якщо рішення приймаються не командою, а спускаються "згори", то зрештою ви отримаєте не колектив мотивованих і самостійних розробників, а безініціативну групу слухняних співробітників.

Багато прекрасних команд на перших етапах своєї діяльності терплять невдачі. Багато команд успішні без Scrum. Приймаючи рішення про використання Scrum, треба зважити все за і проти і наслідувати прийняте рішення.

## 10.6 Що далі

Навряд чи можна розраховувати на те, що гнучкі методології застосовуватимуться в ідеальному середовищі, вільному від втручання навколишнього реального світу. Неідеальність реального світу - це реальний факт. Цей факт необхідно враховувати завжди. Таким чином, очевидна необхідність супроводу і розвитку окремо узятого Scrum конкретної компанії. Scrum - гнучка процессная методологія, або, по-іншому, - управлінська дисципліна. І як будь-яку дисципліну, мета якої - приносити передбачуваний результат, її необхідно контролювати. Але трудомісткість менеджерських активностей можна зменшити і із самого початку "виховувати" в співробітниках, що входять в Scrum- команду, лояльність до змін, прихильність до навчання і вдосконалення. Явним результатом цього буде досягнення більш високого рівня процессной зрілості і професійної культури.

Шляхом експериментування Scrum- команди можуть повести організацію до освоєння все більш і більш ефективних способів роботи.

**Питання до теми:**

1. Яким чином здійснюється співіснування з альтернативними процесами послідовної розробки програмного забезпечення?

2. Яких наборів правил і принципів необхідно дотримуватися членам Scrum- команд, пропонований "best practice"?

3. На що спрямована методологія Lean?

4. Приведіть основні характеристики компанії, що пройшла стадію цифрової трансформації.

5. Приведіть перелік проблем, які призводять до критики культури Agile?

# 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