Лекції для розділу 1

Етичні та правові засади програмної інженерії

Професійна практика програмної інженерії

У 1958 всесвітньо відомий статистик Джон Тьюки (John Tukey) вперше ввів термін **Software** - програмне забезпечення.

У 1972 році IEEE * випустив перший номер журналу «Transactions on **Software Engineering**» - “Праці з Програмної Інженерії”.

Асоціація обчислювальної техніки

Association for Computing Machinery

Рік створення 1947

Розташування США, Нью-Йорк

Сайт www.acm.org
www.utacm.org

- Асоціація обчислювальної техніки (англ. Association for Computing Machinery, ACM) - найстаріша і найбільш велика міжнародна організація в комп'ютерній області. Об'єднує близько 83 000 фахівців. Штаб-квартира знаходиться в Нью-Йорку.
  Щорічно асоціація присуджує Премію Тьюринга і премію імені Грейс Мюррей Хоппер. У число нагород також входять Премія Геделя, Премія Дейкстри, Премія Кнута, Премія Гордона Белла і Премія Паріса Канеллакіса

Діяльність

ACM складається з більш ніж 170 регіональних відділень і 35 спеціальних тематичних груп (англ. Special interest group, SIG), в рамках яких і ведеться основна діяльність.

Деякі тематичні групи:

- SIGACT [en] - теорія алгоритмів і обчислень
- SIGAI [en] - штучний інтелект
- SIGCHI [en] - людино-машинне взаємодія
- SIGCOMM [en] - передача даних
- SIGGRAPH - комп'ютерна графіка
- SIGKDD [en] - Data Science
- SIGMM [en] - мультимедіа
- SIGMOD [en] - управління даними
- SIGOPS - операційні системи
- SIGPLAN [en] - мови програмування
- SIGSOFT [en] - розробка програмного забезпечення
- SIGWEB [en] - веб-технології

Додатково є більше 500 вузівських відділень. Перший студентський філал організації був створений в 1961 році в Університеті Луїзіани в Лафайетті.

Інститут інженерів з електротехніки і електроніки — **IEEE** (англ. *Institute of Electrical and Electronics Engineers*)

(I triple E — «Ай тріпл і») — міжнародна некомерційна асоціація спеціалистів в області техніки, світовий лідер в області разробки стандартів з радіоелектроніки і електротехніки.

Software Engineering 2014

Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering

A Volume of the Computing Curricula Series

23 February 2015

Joint Task Force on Computing Curricula
IEEE Computer Society
Association for Computing Machinery

Асоціація «Інформаційні технології України».

- У квітні 2004 року українські IT-компанії - «Miratech», «SoftLine», «Mirasoft», «ProFIX», «Ukrsoft», «SoftServe» прийняли рішення про створення Асоціації **«Інформаційні технології України».** Компанії-засновники Асоціації є провідними українськими розробниками програмного забезпечення, які накопичили успішний досвід в області продажу та ліцензування послуг на розвинених ринках Північної Америки та Європи.

  Метою створення Асоціації є консолідація зусиль по просуванню на зовнішніх ринках конкурентоспроможної продукції українських компаній, злиття наукового і виробничого потенціалу України для створення програмного забезпечення, яке відповідає міжнародним стандартам, а також умов для подальшого розвитку галузі.

- http://itukraine.org.ua/
  http://www.epravda.com.ua/cdn/cd1/2016/04/softserve/

ВГО
"Співтовариство
ІТ-директорів
України"

Співтовариство ІТ-директорів України

- http://itdirector.org.ua/

Рейтинг IT-работодавців України

від 800 спеціалістів

SoftServe	91	1058 анкет
DataArt	91	111
Infopulse	90	352
ELEKS	87	278
GlobalLogic	87	586
EPAM	85	1017
Ciklum	84	315
Luxoft	79	329

До 800 спеціалістів

Terrasoft	97	126 анкет
Genesis	97	108
AMC Bridge	96	97
3Shape Ukraine	96	51
Netpeak	96	125
Intellias	95	74
Provectus	93	135
Astound Commerce	92	204
CoreValue	92	78
Sigma Software	91	103

https://jobs.dou.ua/ratings/

https://dou.ua/lenta/articles/top-50-july-2019/?from=hover-news-widget

Місце	Компанія	Спеціалісти в Україні	Технічні спеціалісти	Вакансії в Україні
1■	EPAM Київ, Харків, Львів, Дніпро, Вінниця	7500+900	6700 89%	600
2■	SoftServe Київ, Харків, Львів, Дніпро, Рівне, Чернівці, Івано-Франківськ	7082+749	5780 82%	500
3■	GlobalLogic Київ, Харків, Львів, Миколаїв	4363+214	4071 93%	507
4■	Luxoft Київ, Дніпро, Одеса	3670-250	3598 98%	254
5■	Ciklum Київ, Харків, Львів, Дніпро, Одеса, Вінниця	2867+4	2506 87%	291
6▲	NIX Харків	2004+301 лип. 19	1794 90%	143
7▼	Infopulse Київ, Харків, Львів, Одеса, Вінниця, Житомир, Чернігів	1900+49 лип. 19	1681 88%	170
8▲	Intellias Київ, Харків, Львів, Одеса, Івано-Франківськ	1521+301 лип. 19	1233 81%	165
9▲	DataArt Київ, Харків, Львів, Дніпро, Одеса, Херсон	1474+173	1278 87%	124
10▲	ZONE3000 Харків, Львів, Дніпро	1450+150 лип. 19	392 27%	41
11▼	EVOPLAY Київ	1406+82 лип. 19	1103 78%	261
12▼	ELEKS Київ, Львів, Тернопіль, Івано-Франківськ	1373+46 лип. 19	1048 76%	65
13■	Netcracker Київ, Одеса, Суми	1153+89 лип. 19	954 82%	99

Наиболее влиятельная личность в ИТ-индустрии?

Билл Гейтс

Стив Джобс

Илон Маск

Сатья Наделла

Ларри Пейдж

Лэй Цзюнь

Лей Цзюнь — китайський бізнесмен, засновник і генеральний директор компанії Xiaomi Tech.

Має репутацію «китайського Стіва Джобса». Мільярдер, у рейтингу журналу Forbes у 2015 році посів шосте місце серед китайських мільярдерів зі статком $13,2 млрд.

Народження: 16 грудня 1969 р.(46 років)

Власний капітал: 13,3 мільярда USD (2015) Forbes

Професійна практика з програмної інженерії

Програмна інженерія (Software Engineering) є галуззю інформатики, яка вивчає питання побудови комп'ютерних програм, відображає закономірності розвитку програмування, узагальнює досвід програмування у вигляді комплексу знань і правил регламентації інженерної діяльності розробників Software.

У цьому визначенні виділимо два основних аспекти:

Програмну інженерію можна розглядати як інженерну дисципліну, в якій інженери застосовують **теоретичні ідеї, методи і засоби розробки ПЗ**, створюють продукти відповідно до стандартів, що регламентують процеси їх проектування та розробки.

Програмна інженерія описує **методи управління програмним проектом, якістю та ризиками.** Застосування таких методів дозволяє досягти високої якості програмних продуктів.

На відміну від науки, мета якої - отримання знань, для Software Engineering знання - це спосіб отримання деякої користі.

Крім **програмістів**, що займаються безпосередньо розробкою ПЗ, в програмної інженерії використовуються:

-**менеджери**, які планують і керують проектом, відстежують терміни і витрати; інженери служби ведення бібліотек і репозитаріїв компонентів;

-**технологи**, які визначають інженерні методи і стандарти, створюють для проекту модель ЖЦ, що задовольняє його цілям і завданням;

-**тестувальники**, які перевіряють правильність виконання процесу проектування шляхом тестування і на основі зібраних даних проводять вимірювання різних характеристик якості, включаючи оцінку надійності ПЗ;

-**верифікатори**, які перевіряють правильність реалізації функцій в проекті;

-**валідатори**, які перевіряють ПЗ на відповідність заданим вимогам.

Software Engineering

Розробку програмних систем можна вважати інженерною діяльністю, але вона має деякі відмінності від традиційної інженерії:

-гілки інженерії мають високу ступінь спеціалізації, а в програмній інженерії **спеціалізація торкнулася тільки окремих областей** (наприклад, операційні системи, транслятори, редактори і т.п.);

-програмування об'єктів **ґрунтується на стандартах**, за допомогою яких відображаються типові вимоги замовників, тобто типізація об'єктів і артефактів в сфері програмування;

-технічні рішення класифіковані і каталогізовані, а в програмній інженерії кожна нова розробка - **це нова проблема**, для реалізації якої встановлюють аналогію з раніше розробленими системами.

-**робота в команді - одна з найважливіших навичок**, яку прагнуть роботодавці, вона яка занесена Світовим економічним форумом у 10 навичок, необхідних для досягнення успіху в 2020, 2022, 2025 роках.

Кодекс етики і професіональної діяльності в області програмної інженерії (версія 5.2)

1. СУСПІЛЬСТВО - Програмні інженери повинні діяти неухильно в інтересах суспільства.
2. КЛІЄНТ І РОБОТОДАВЕЦЬ - Програмні інженери повинні діяти відповідно до інтересів клієнта і роботодавця, якщо вони не суперечать інтересам суспільства.
3. ПРОДУКТ - Програмні інженери повинні забезпечувати відповідність якості своїх продуктів і їх модифікацій найвищим можливим професійним стандартам.
4. ОЦІНКИ - Програмні інженери повинні підтримувати цілісність і незалежність своїх професійних оцінок.
5. МЕНЕДЖМЕНТ - Програмні інженери-менеджери та провідні співробітники повинні дотримуватися етичних підходів до управління розробкою і підтримкою програмного забезпечення і просувати ці підходи.
6. ПРОФЕСІЯ - Програмні інженери повинні піднімати престиж і репутацію своєї професії в інтересах суспільства.
7. КОЛЕГИ - Програмні інженери повинні бути справедливі по відношенню до своїх колег, допомагати їм і підтримувати.
8. ОСОБИСТА ВІДПОВІДАЛЬНІСТЬ - Програмні інженери повинні безперервно навчатися навичкам своєї професії і сприяти просуванню етичного підходу до своєї діяльності.

Три закони робототехніки

Айзек Азімов в оповіданні «Хоровод» (1942)

- Робот не може заподіяти шкоду людині, або своєю бездіяльністю дозволити, щоб людині була заподіяна шкода (роботи його вже порушили);
- Робот повинен підкорятися наказам людини, за винятком тих, котрі суперечать першому пункту;
- Робот повинен захищати самого себе, якщо тільки його дії не суперечать першому і другому пунктам.

На думку вчених з Google, треба буде вирішити розробникам штучного інтелекту в першу чергу:

- уникнути негативних побічних ефектів, або ж «як ми можемо бути впевнені, що робот-прибиральник не зіб’є з постаменту вазу, якщо вирішить, що так роботу можна зробити швидше?»;
- уникнути проблем з винагородою, або ж «як ми можемо бути впевнені, що робот-прибиральник просто не замете сміття під диван замість прибирання?»;
- прийнятний нагляд, або «як ми можемо бути впевнені, що робот буде швидко всьому навчатися, а не питати постійно, де стоїть швабра?»;
- безпечне дослідження, або «як ми можемо бути впевнені, що прибиральник вибиратиме найкращі стратегії чищення, але в процесі не вирішить як експеримент сунути швабру в розетку і не спалить весь будинок?»;
- розсудливість при зміні навколишнього середовища, або «як нам навчити робота розуміти, що його навички в нових змінених умовах не приносять користі?».

ПРОФЕСІЙНИЙ СТАНДАРТ

Фахівець з розробки програмного забезпечення

На ринку праці затребувані програмісти, що вміють працювати в команді, володіють інструментами колективної розробки програмного забезпечення.

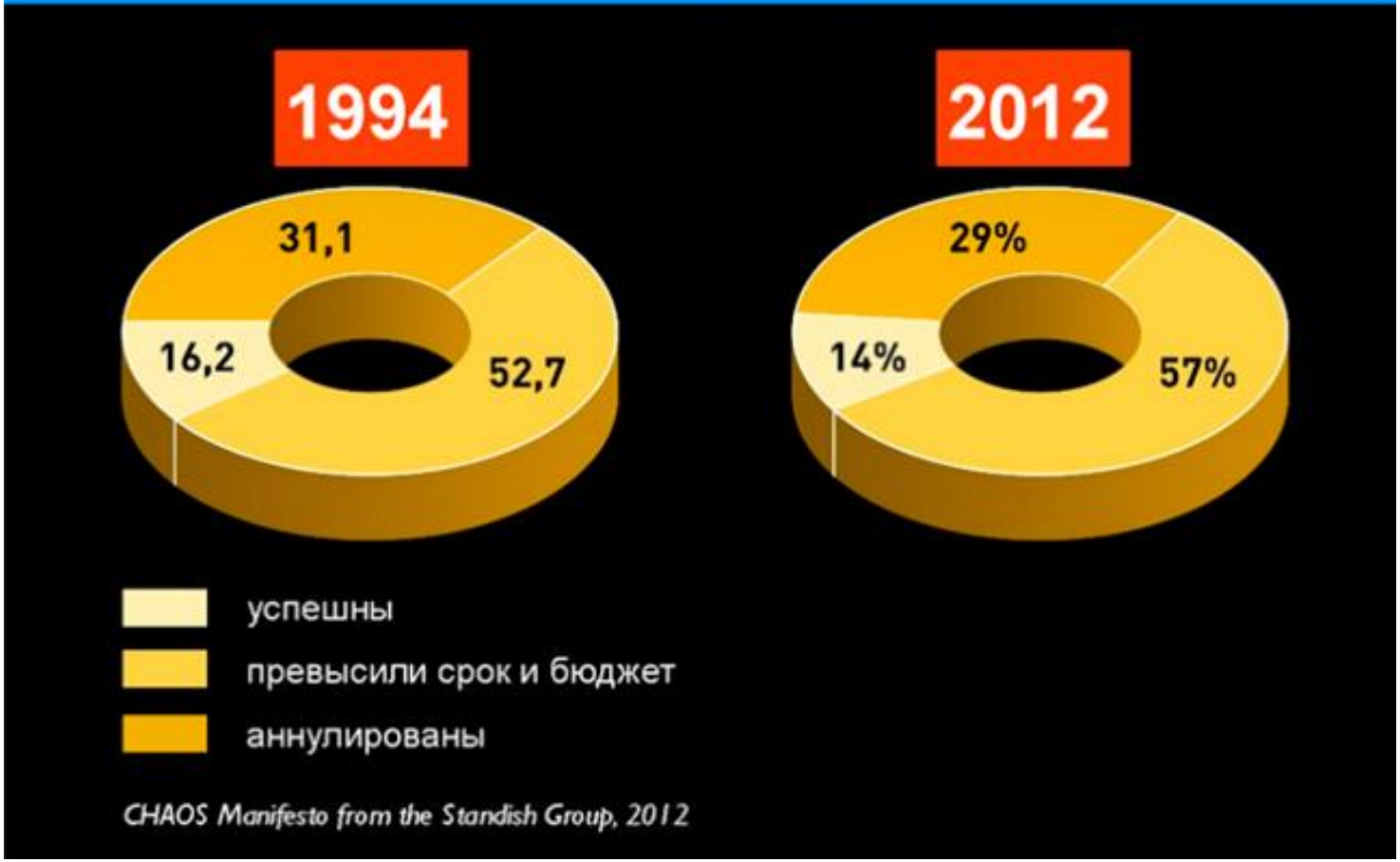
У зв'язку з цим зростає значення професійних компетенцій колективної розробки програмного забезпечення, знання сучасних напрямів, методів і технологій розробки програмного забезпечення:

- розуміння обов'язків різних учасників команди по розробці програмного забезпечення: керівник розробки програмного забезпечення, керівник технічної групи (team leader), архітектор, програміст, тестувальник, дизайнер, верстальник, аналітик;
- володіння сучасними стратегіями і технологіями організації колективної розробки програмного забезпечення, включаючи системи управління версіями, процеси безперервної інтеграції, стандарти оформлення коду і методи інспекції коду;
- розуміння основних напрямів розвитку методів колективної розробки, їх відмінностей і доцільності застосування залежно від типу вирішуваних завдань і вимог організації;
- володіння гнучкими (Agile) методологіями розробки програмних продуктів.

Динаміка розвитку предметної області «розробка програмного забезпечення» настільки велика, що ринок вимагає постійної зміни кількості і якості знань та умінь від випускника.


Agile
Waterfall
Люди и их взаимодействие
Готовый продукт
Сотрудничество с заказчиком
Реакция на изменения
Процессы и инструменты
Документация
Жесткие контрактные ограничения
Следование плану

Статистика проектов в США



Доля розробника ПЗ

Статистика проектов в США

Agile-маніфест розробки програмного забезпечення

Основні ідеї:

- **Особистості та їхні взаємодії важливіші,** ніж процеси та інструменти**;**
- **Робоче програмне забезпечення важливіше,** ніж повна документація**;**
- **Співпраця із замовником важливіша,** ніж контрактні зобов'язання;
- **Реакція на зміни важливіша,** ніж дотримання плану.

В чому виявляється маніфест?

1. Особи та взаємодії над процесами та інструментами
   - **Взаємодійте, комунікуйте, говоріть!**
2. Робота програмного забезпечення над детальною документацією
   - **Створюйте вирішення, а не описи проблем!**
3. Співпраця з клієнтом над переговорами за контрактом
   - **Зрозуміти, що потрібно вашому клієнтові, не те, що він описує!**
4. Реакція на зміни по проходження плану
   - **Будьте готові до змін, ситуація може змінитися! Плануйте зміни.**

Підписанти

- **Kent Beck**
  Mike Beedle
  Arie van Bennekum
  Alistair Cockburn
  Ward Cunningham
  Martin Fowler
  James Grenning
  Jim Highsmith
  Andrew Hunt
  Ron Jeffries
  Jon Kern
  Brian Marick
  Robert C. Martin
  Steve Mellor
  Ken Schwaber
  Jeff Sutherland
  Dave Thomas

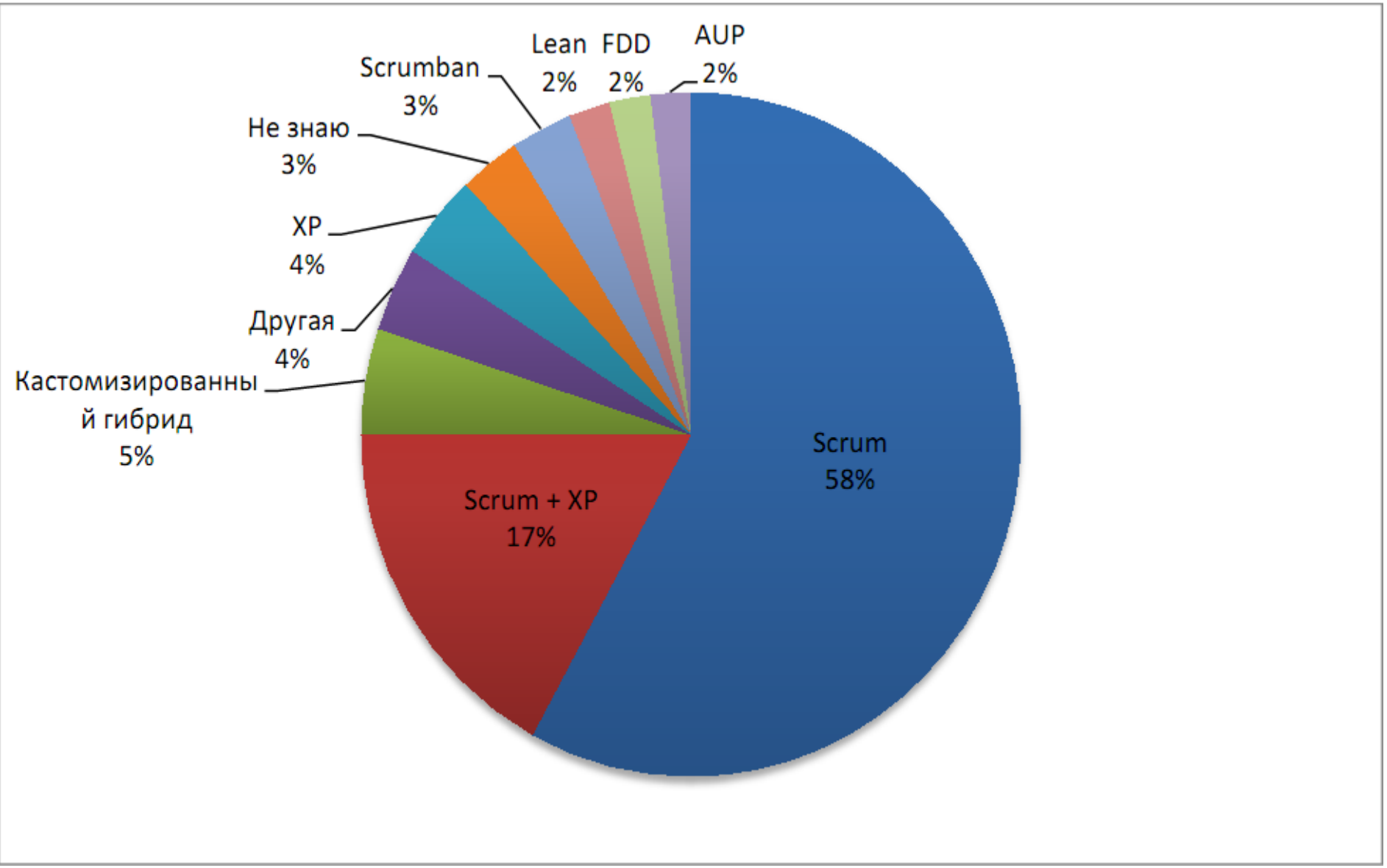
Принципи, які роз'яснюють Agile Manifesto

1. Задоволення клієнта за рахунок **ранньої та безперебійної** поставки цінного програмного забезпечення;
2. Вітання **зміни вимог** навіть наприкінці розробки (це може підвищити конкурентоспроможність отриманого продукту);
3. **Часта поставка** робочого програмного забезпечення (кожен місяць або тиждень або ще частіше);
4. **Тісне, щоденне спілкування замовника з розробниками** впродовж всього проекту;
5. Проектом займаються **умотивовані особистості**, які забезпечені потрібними умовами роботи, підтримкою і довірою;
6. Рекомендований **метод передачі інформації — особиста розмова** (віч-на-віч);
7. **Робоче програмне забезпечення — найкращий вимірювач прогресу;**
8. Спонсори, розробники та користувачі повинні мати можливість підтримувати **постійний темп на невизначений термін**;
9. Постійну увагу **поліпшенню технічної майстерності** та зручному дизайну;
10. **Простота** — мистецтво не робити зайвої роботи;
11. Найкращі технічні вимоги, дизайн та архітектура виходять у **самоорганізованої команди**;
12. Постійна **адаптація до мінливих обставин**.

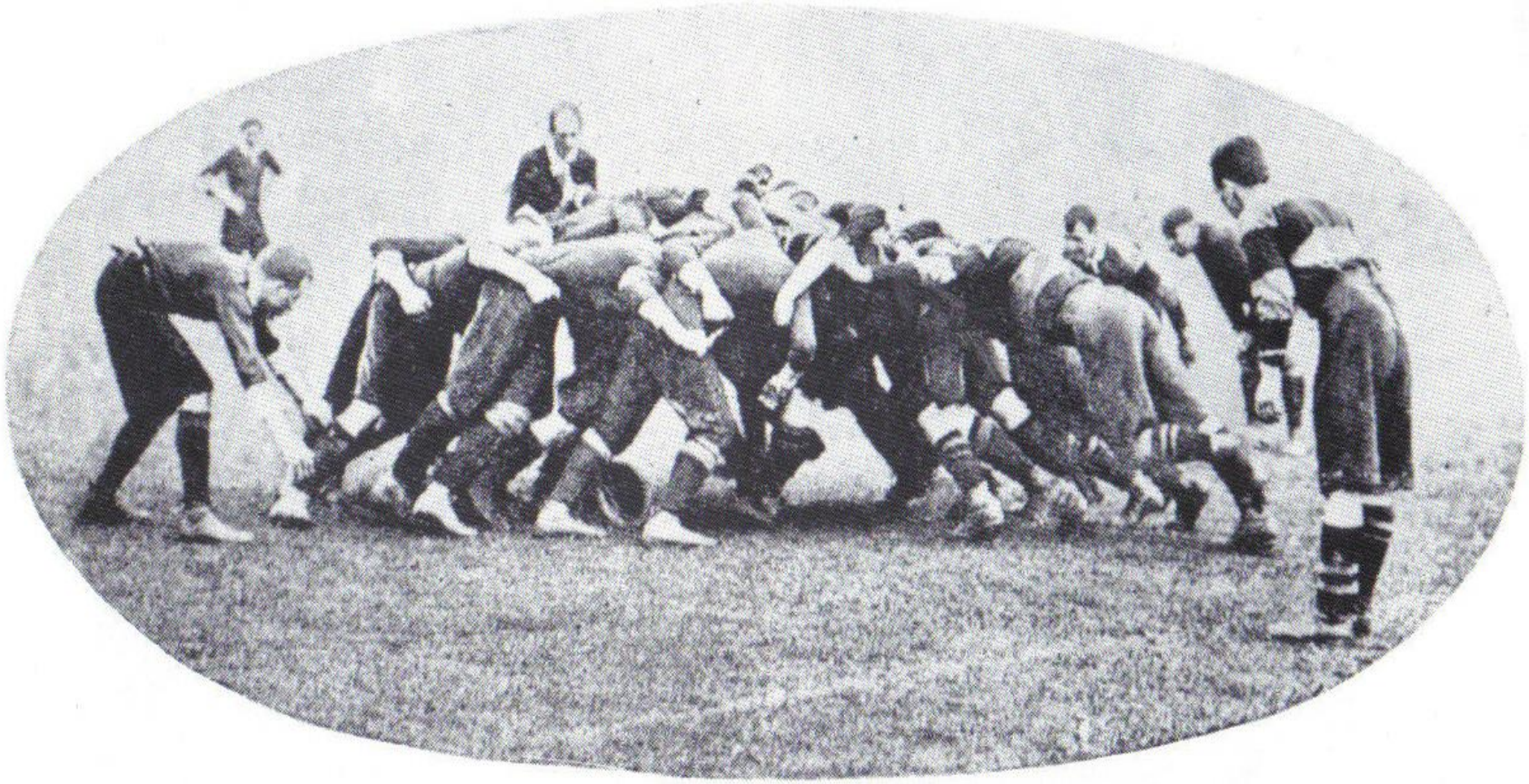
Переваги Agile

- Відповідає на **зміни** швидко
- Концентрується на **цінності** продукту
- Гнучко керує **залізним трикутником**

Використання гнучких технологій



Сутичка (Scrum) в регбі між Newport и London Welsh в 1904



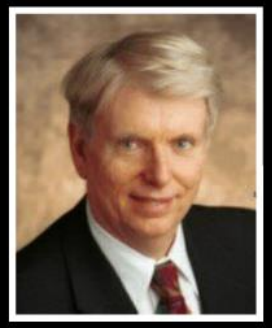
Підхід вперше описали Хіротака Такеуті і Ікудзіро Нонака
в статті The New Product Development Game (Гарвардський Діловий Огляд , січень-лютий 1986).

Вони відзначили, що проекти, над якими працюють невеликі команди з фахівців різного профілю, зазвичай систематично дають кращі результати, і пояснили це як «підхід регбі»

https://www.youtube.com/watch?v=9MRbY8RqQdU

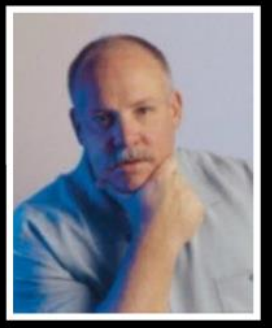
Гуру Скрам

Ken Schwaber
Кен Швабер

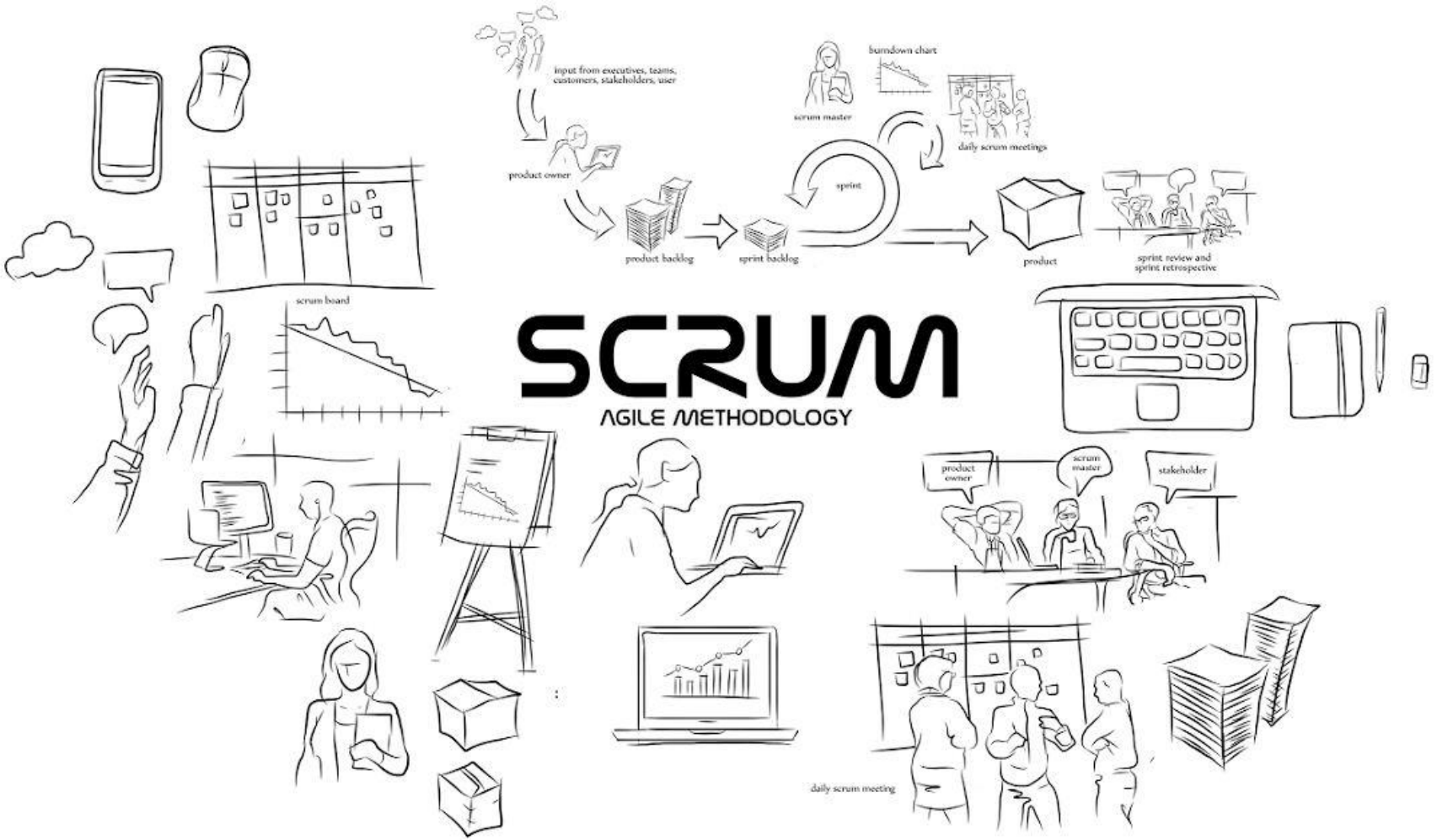


Jeff Sutherland
Джефф Сазерленд

Mike Beedle
Майк Бидл

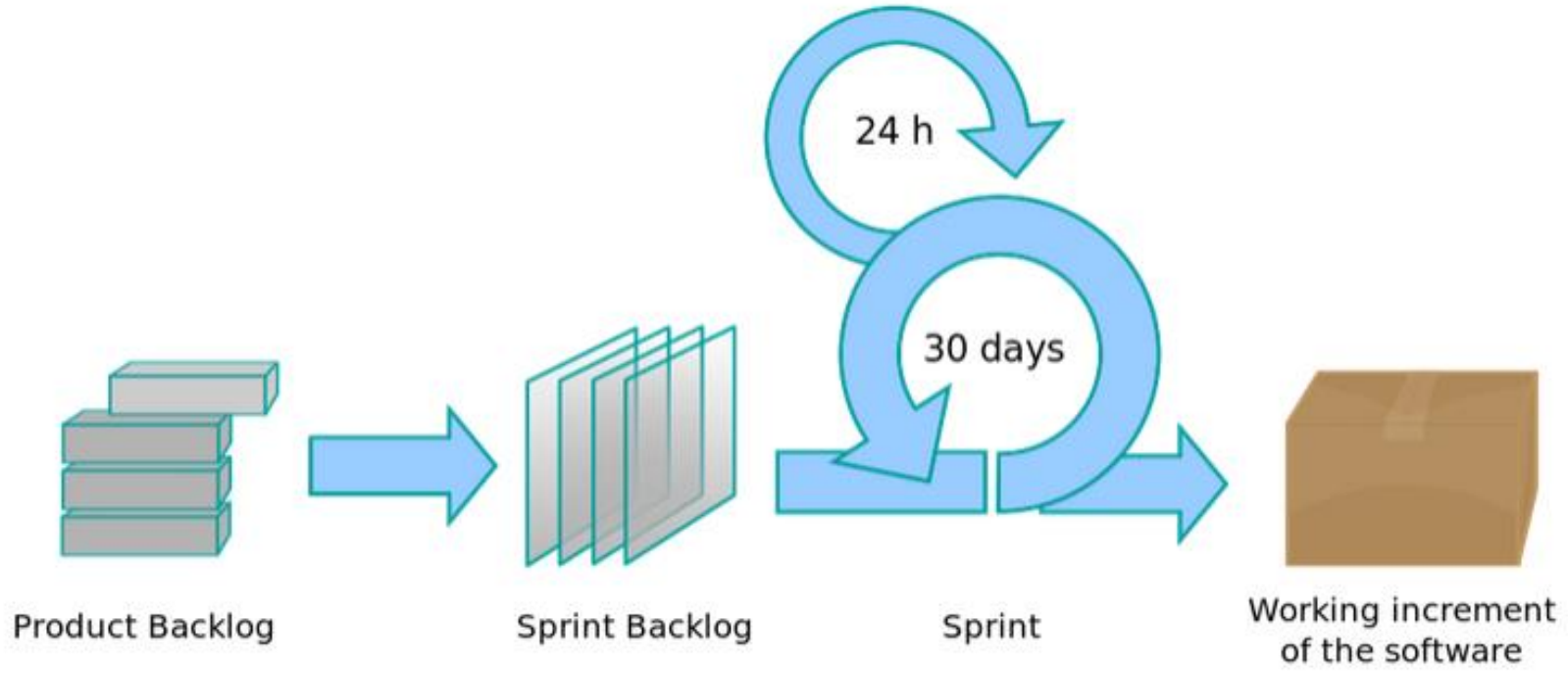


Mike Cohn
Майк Кон


input from executives, teams, customers, stakeholders, user
product owner
product backlog
sprint backlog
sprint
scrum master
burndown chart
daily scrum meetings
product
sprint review and sprint retrospective
scrum board
SCRUM
AGILE METHODOLOGY
product owner
scrum master
stakeholder
daily scrum meeting

scrum

- https://www.youtube.com/watch?v=1LBm0dFFqHA



Ролі в Scrum

У класичному Scrum існує 3 базових ролі:

-Product owner

-Scrum master

-Команда розробки (Development team)

Product owner

Product owner (PO) є сполучною ланкою між командою та замовником. Завдання PO — максимальне збільшення цінності розроблюваного продукту і роботи команди. Одним з основних інструментів PO є Product Backlog. Product Backlog містить необхідні для виконання робочі завдання (такі як Story, Bug, Task та ін), відсортовані в порядку пріоритету (терміновості).

Scrum master

Scrum master (SM) відповідає за організацію Scrum-у

Завдання Scrum Master — допомогти команді максимізувати її ефективність за допомогою усунення перешкод, допомогти у навчанні і мотивації команди.

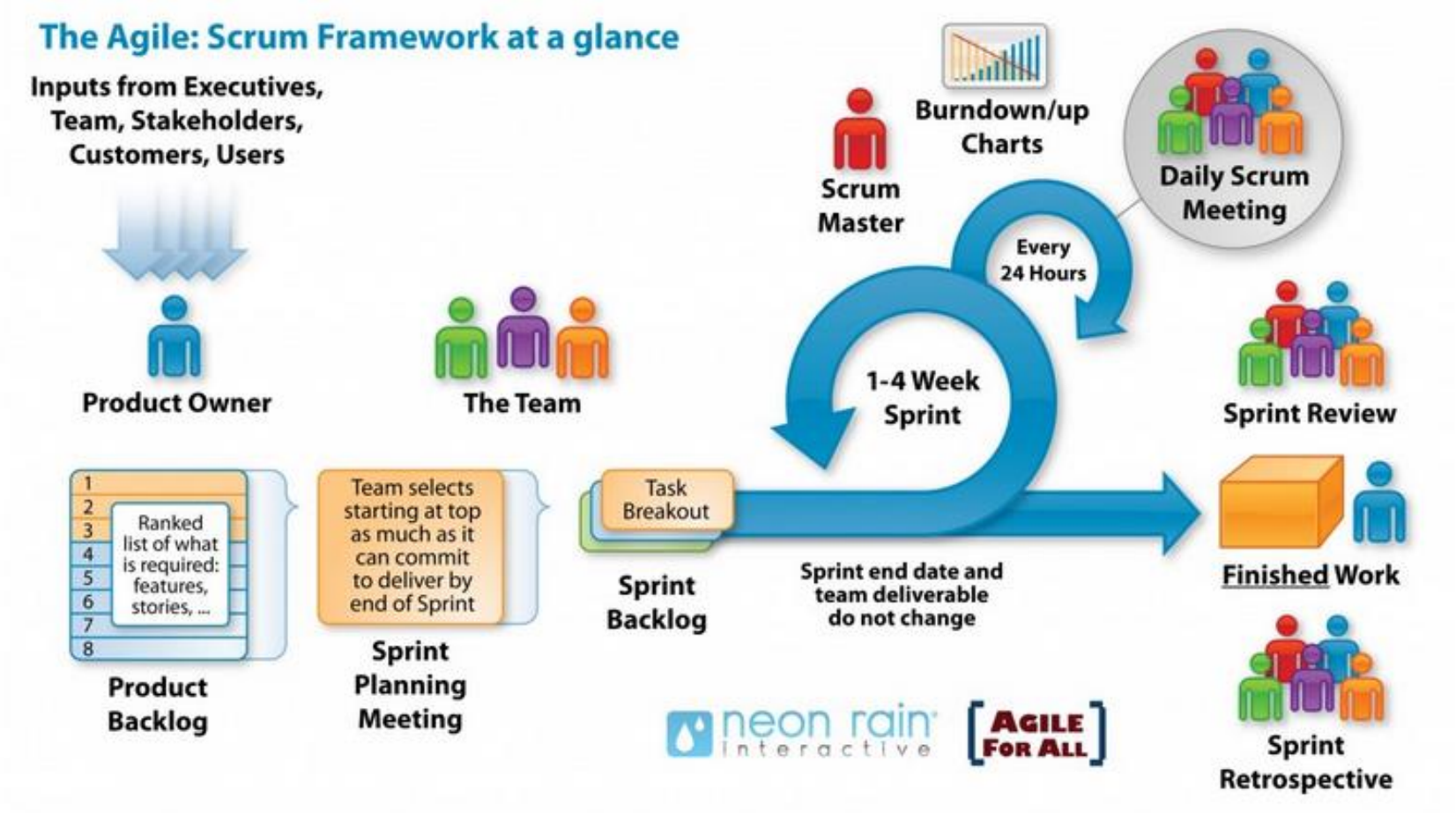
Команда розробки

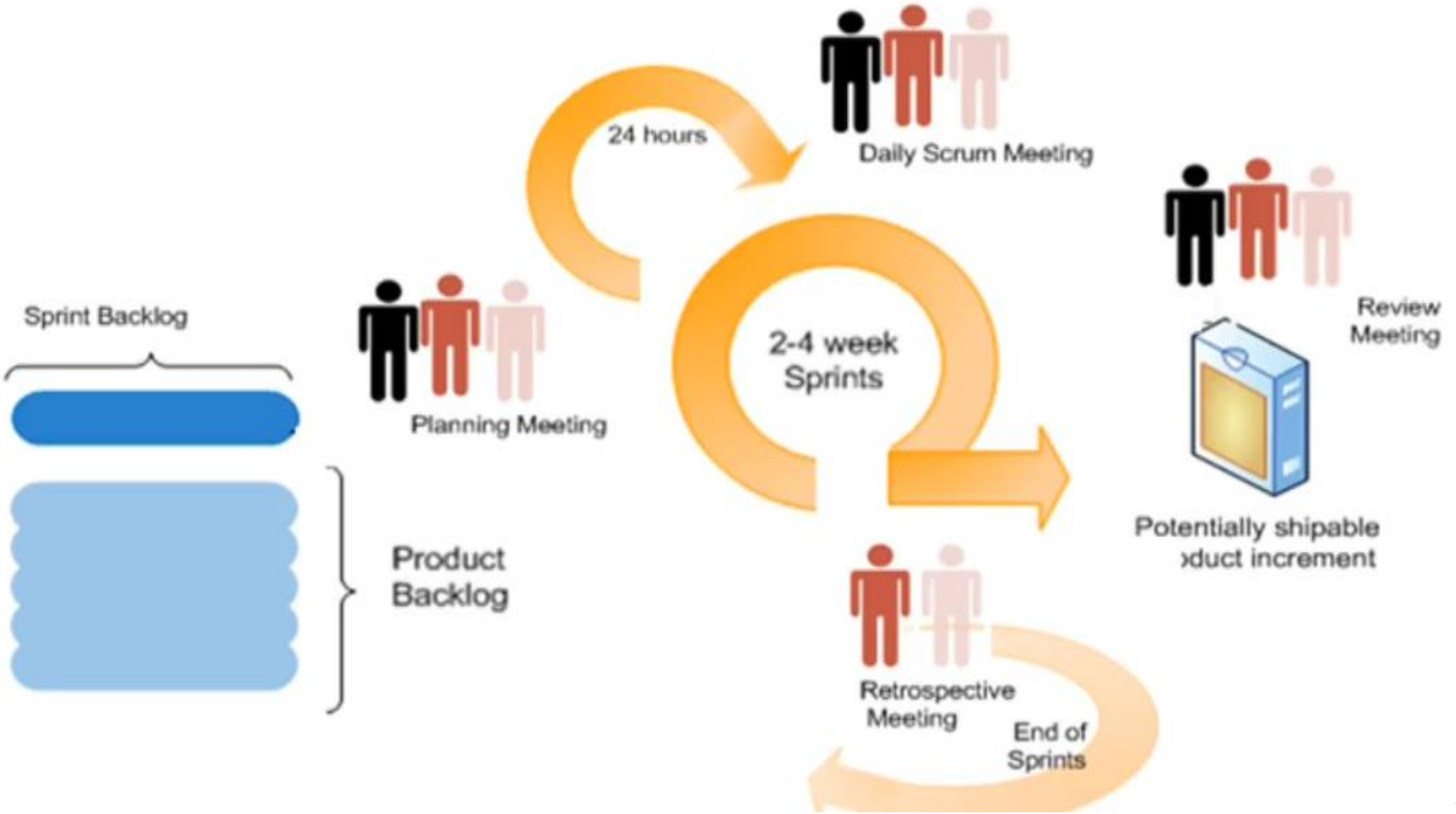
Команда розробки (Development team, DT) складається з фахівців, які виконують безпосередню роботу над продуктом

DT повинні володіти наступними якостями :

- Бути самоорганізованою
- Бути багатофункціональною
- За виконувану роботу відповідає вся команда

Реалізація SCRAMy




Sprint Backlog
Product Backlog
Planning Meeting
24 hours
Daily Scrum Meeting
2-4 week Sprints
Review Meeting
Potentially shipable
duct increment
Retrospective Meeting
End of Sprints

Product backlog

Product backlog - це основа Scrum‘у. З нього все починається. По суті, product backlog є списком вимог, історій, функціональності, які впорядковані за ступенем важливості.

При цьому всі вимоги описані зрозумілою для замовника мовою.
Елементи цього списку ми будемо називати "історіями", **user story**, а іноді елементами backlog'а.
Опис кожної нашої історії включає в себе наступні поля:

Product backlog

1. **ID** - унікальний ідентифікатор - просто порядковий номер.

Застосовується для ідентифікації історій в разі їх перейменування.

2. **Назва** - короткий опис історії.

Наприклад, "Перегляд журналу своїх транзакцій". Вона має бути однозначною, щоб розробники та product owner могли приблизно зрозуміти, про що йде мова, і відрізнити одну історію від іншої. Зазвичай від 2 до 10 слів.

Product backlog

3. **Важливість** (Importance) - ступінь важливості даного завдання, на думку product owner'а. Наприклад,10 чи 150. Чим більше значення, тим вища важливість.

4. **Попередня оцінка** (initial estimate) - початкова оцінка обсягу робіт, необхідного для реалізації історії в порівнянні з іншими історіями.

Вимірюється командою в story point'ах. Приблизно відповідає числу "ідеальних людино-днів".

TFS

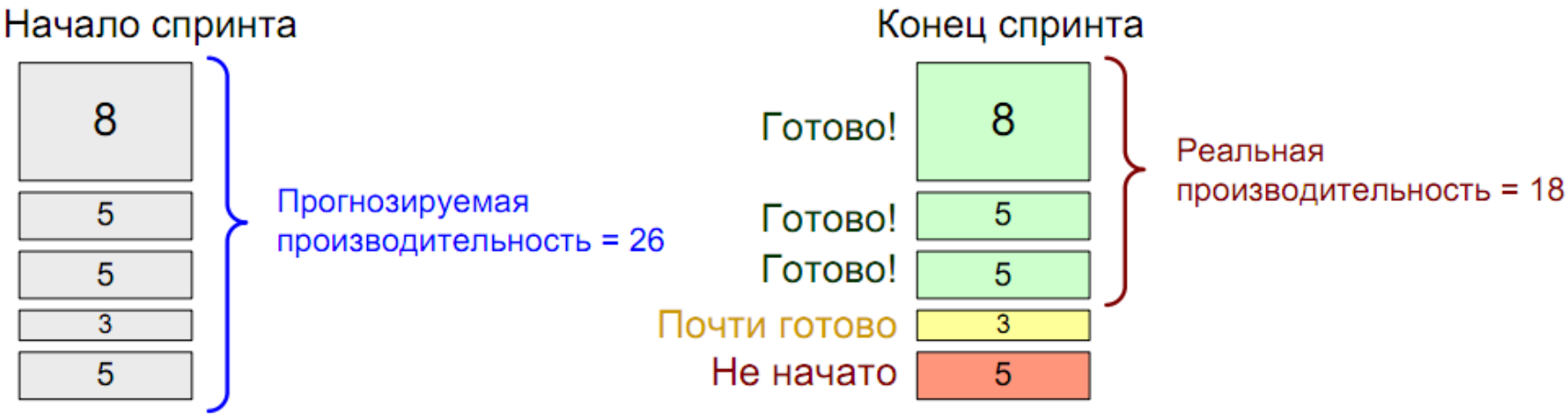
https://www.youtube.com/watch?v=T83qmM4-T88

https://www.youtube.com/watch?v=I6OMeXu3CZc

Планування спринту

- Мета спринту.
- Список учасників команди (і ступінь їх зайнятості, якщо вона не стовідсоткова).
- Sprint backlog (список історій, які увійшли в спринт).
- Розподіл на tasks та закріплення їх за членами команди.
- Дата демонстрації.
- Місце і час проведення щоденного Scrum‘у.

Планування продуктивності



Планування продуктивності

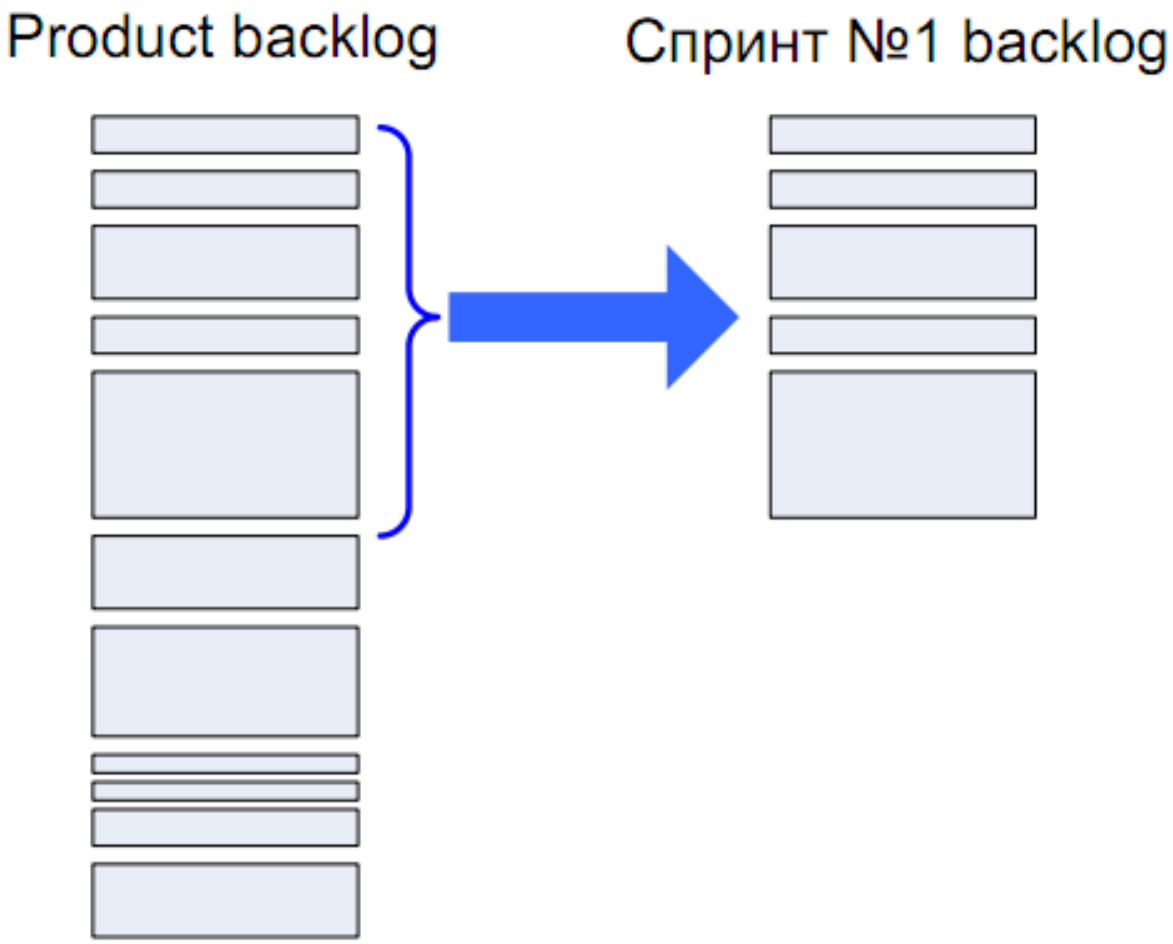
Для відповіді на нього вводимо визначення фокус-фактора.

Фокус-фактор - це коефіцієнт того, наскільки команда сфокусована на своїх основних завданнях.
Прогнозована продуктивність цього спринту:
(Прогнозована продуктивність) =

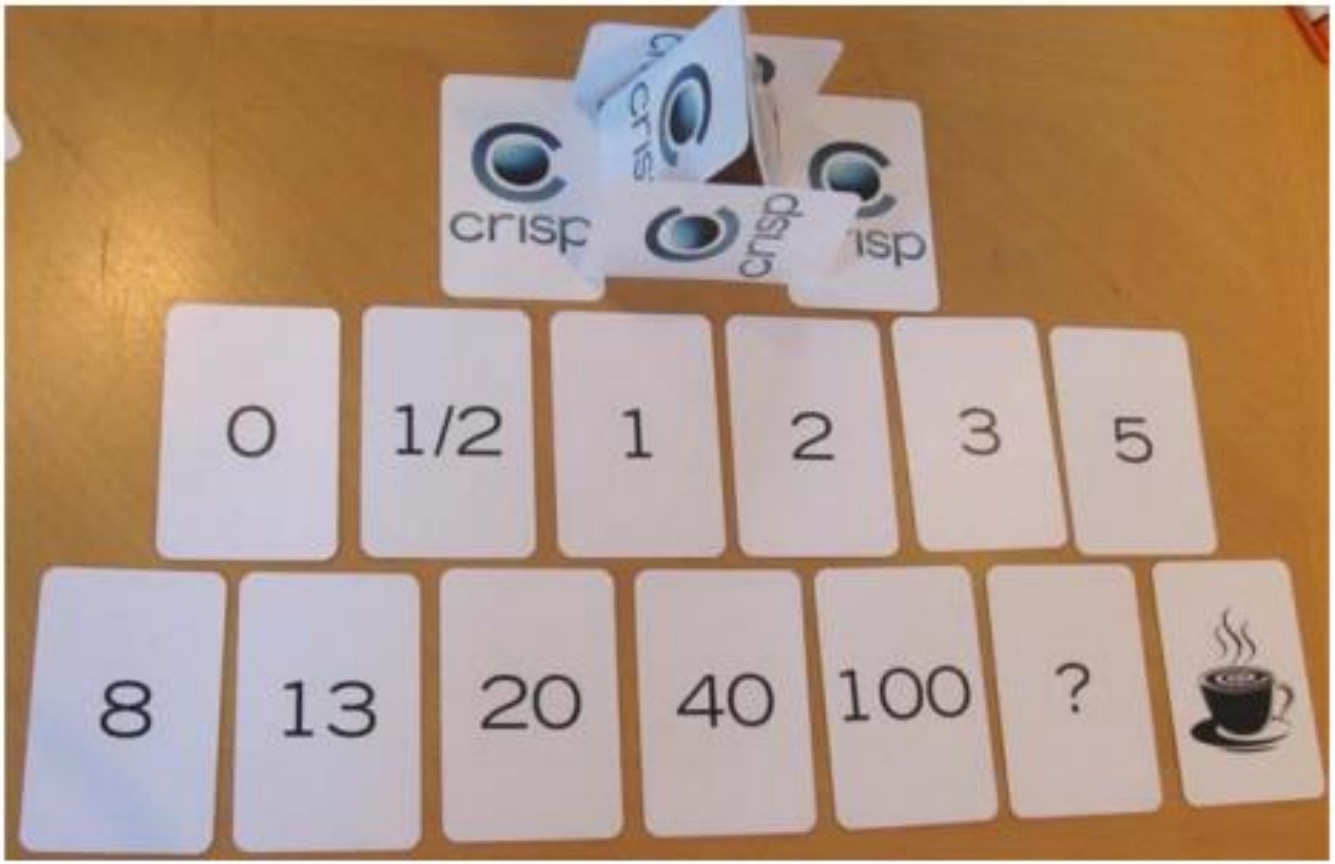
(Доступні людино-дні) * (фокус-фактор)

Низький фокус-фактор може означати, що команда очікує неодноразового втручання в свою роботу або передбачає, що оцінки надто оптимістичні.
Вибрати розумний фокус-фактор найкраще, взявши його з останнього спринту (а ще краще – середнє значення за кілька останніх спринтів).
Фокус-фактор останнього спринту:
(Фокус-фактор) =

(Дійсна продуктивність)/(Доступні людино-дні)


Product backlog
Спринт №1 backlog

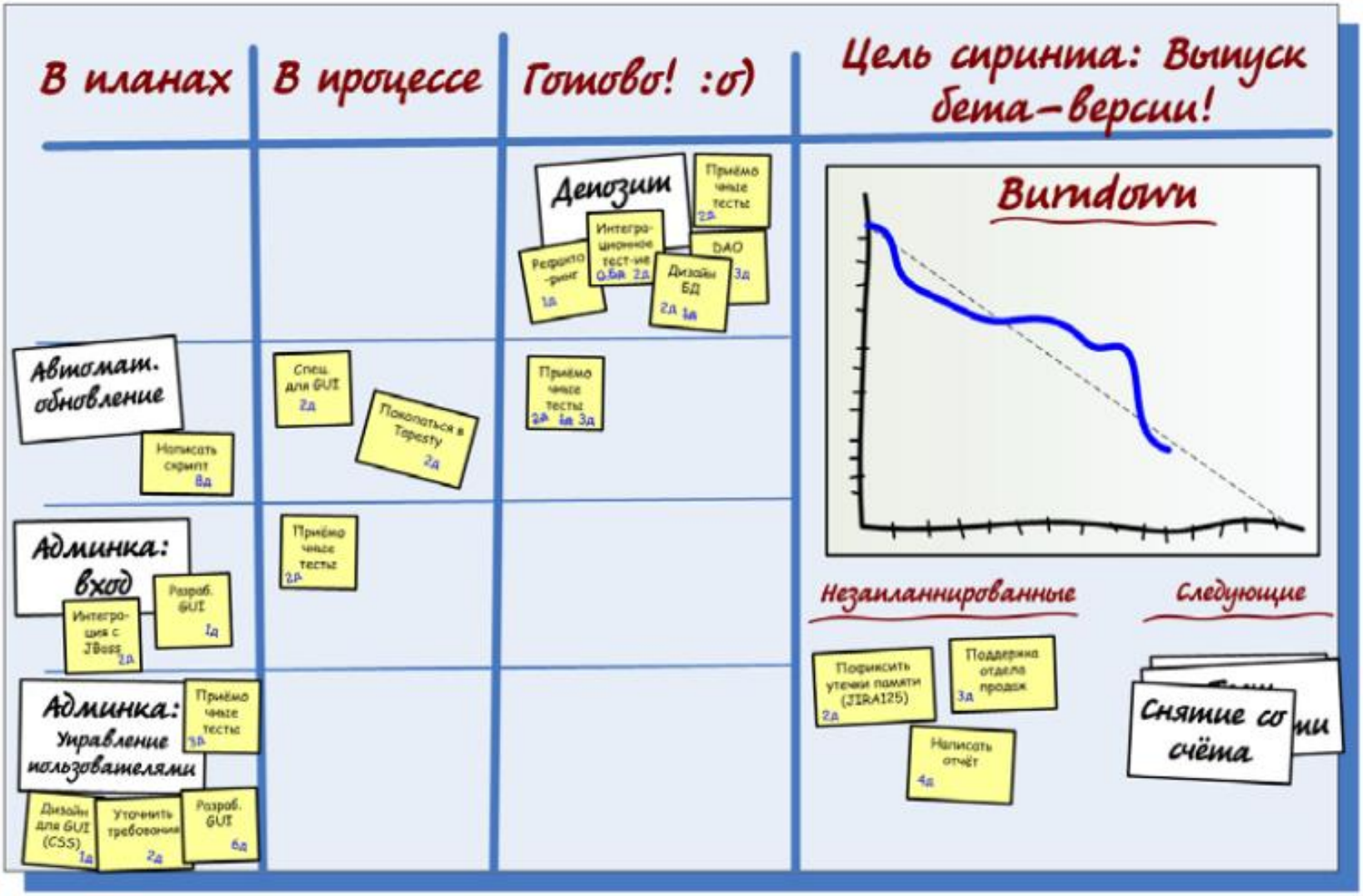
Planning Poker

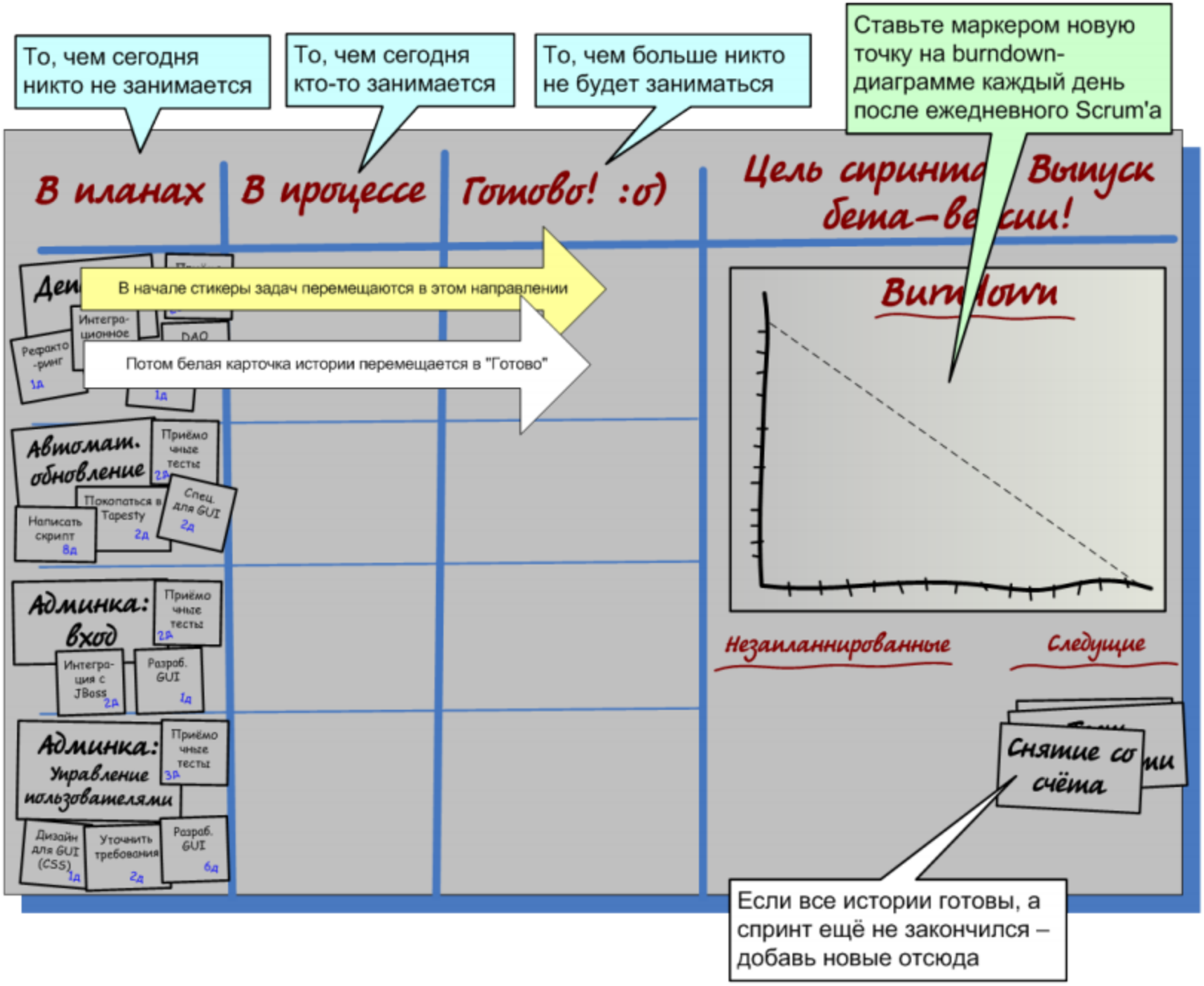


Інформація про спринт

Реліз, готовий до бета-тестування!
Команда «Покер", спринт №5
мета спринту
Sprint backlog (в дужках - оцінки)
• Депозит (3)
• Автоматичне оновлення (8)
• Адмінка: вхід (5)
• Адмінка : управління користувачами (5)
Прогнозована продуктивність: 21
Розклад
• Спринт: з 2016-03-03 по 2016-03-24
• Щоденний scrum: 9:30 - 9:45, в головній кімнаті
• Демонстрація: 24/03/2016, 13.35, в ОЦ-1
команда
• Джим
• Еріка (ScrumMaster)
• Том (75%)
• Єва
• Джон

Дошка спринта




То, чем сегодня никто не занимается
То, чем сегодня кто-то занимается
То, чем больше никто не будет заниматься
Ставьте маркером новую точку на burndown-диаграмме каждый день после ежедневного Scrum'a
В планах
В процессе
Готово! :о)
Цель спринта
Выпуск бета-версии!
В начале стикеры задач перемещаются в этом направлении
Потом белая карточка истории перемещается в "Готово"
Интегра-ционное
DAO
Рефакто-ринг
1д
1д
Автомат. обновление
Приёмо чные тесты
2д
Покопаться в Tapesty
2д
Спец. для GUI
2д
Написать скрипт
8д
Админка: вход
Приёмо чные тесты
2д
Интегра-ция с JBoss
2д
Разраб. GUI
1д
Админка: Управление пользователями
Приёмо чные тесты
3д
Дизайн для GUI (CSS)
1д
Уточнить требования
2д
Разраб. GUI
6д
Burndown
Незапланированные
Следущие
Снятие со счёта
Если все истории готовы, а спринт ещё не закончился – добавь новые отсюда


Команда не обращает внимание на приоритеты задач в backlog'e и делает всё в произвольном порядке!
В и
ль спринта: Выпуск бета-версии!
Burndown
Депозит
Приёмочные тесты 2д
Рефакторинг 1д
Интеграционное тест-ие 2д
Дизайн БД 1д
DAO 3д
Автомат. обновление
Приёмочные тесты 2д
Написать скрипт 8д
Покопаться в Tapestry 2д
Спец. для GUI 2д
Админка: вход
Разраб. GUI 1д
Интеграция с JBoss 2д
Приёмочные тесты 2д
Админка: Управление пользователями
Приёмочные тесты 3д
Дизайн для GUI (CSS) 1д
Уточнить требования 3д
Разраб. GUI 6д
Незапланированные
Следующие
Подписать утечки памяти (JIRA125) 2д
Поддержка отдела продаж 3д
Написать отчёт 4д
Снятие со счёта

Як ми оновлюємо дошку задач

Зазвичай ми оновлюємо дошку задач під час щоденного Scrum‘у. У міру того, як кожен член команди розповідає про те, що він зробив за вчорашній день і чим буде займатися сьогодні, він переміщує стікери на дошці задач. Як тільки розповідь стосується якогось незапланованого завдання, то для нього клеїться новий стікер.

При оновленні тимчасових оцінок, на стікері пишеться нова оцінка, а стара закреслюється.

Іноді стікерами займається Scrum Master, поки учасники говорять. У деяких командах прийнято, що всі члени команди оновлюють дошку задач перед кожною зустріччю. Це теж добре працює. Просто вирішите, що вам ближче, і дотримуйтеся цього.

Як бути з запізненнями?

Деякі команди вводять спеціальну скарбничку. Якщо ви запізнилися, навіть на хвилину, ви кидаєте в скарбничку певну суму. Без варіантів. Навіть якщо ви подзвонили перед початком щоденного Scrum‘у і попередили, заплатити все одно доведеться: о)

Відкрутитися можна лише у виняткових випадках. Наприклад, візит до лікаря, власне весілля або щось не менш важливе. Гроші з скарбнички використовуються на суспільні потреби. Наприклад, на них можна замовити піцу, коли ми вирішуємо пограти ввечері: о)

Цей підхід працює непогано. Але користуватися ним потрібно лише в тому випадку, коли люди часто спізнюються. Деяким командам це просто не потрібно.

Що чинити з тими, хто не знає, чим себе зайняти?

Присоромити: "Гаразд, якщо не знаєш, як принести користь команді, йди додому, почитай книгу і т.д. Або просто сиди тут, поки комусь не потрібно твоя допомога

По-старому: Просто призначити їм завдання.

Моральний тиск: Скажіть їм: Діма і Ліза! Не смію вас більше затримувати. А ми всі просто постоїмо тут, поки у вас не з'являться ідеї, як допомогти нам в досягненні мети .

Закабалити: Скажіть їм: "Ви зможете допомогти команді, виконуючи роль прислуги сьогодні. Готуйте каву, робіть масаж, винесіть сміття, приготуйте обід: робіть все, про що вас можна попросити команда ".

Ви будете здивовані, наскільки швидко Діма і Ліза знайдуть для себе корисні технічні завдання: о)

Якщо вони не дуже потрібні , постарайтеся виключити їх зі своєї команди.

Чому потрібно, щоб кожен спринт закінчувався демонстрацією

- Добре виконане демо має великий вплив, навіть якщо воно не здалося захоплюючим.
- Позитивна оцінка роботи надихає команду.
  Усі інші дізнаються, чим займається ваша команда.
- На демо зацікавлені сторони обмінюються життєво важливими відгуками.
- Демо проходить в дружній атмосфері, тому різні команди можуть вільно спілкуватися між собою і обговорювати нагальні питання. Це цінний досвід.
- Проведення демо змушує команду дійсно доробляти завдання і випускати їх. Без демо ми постійно виявлялися з купою на 99% зробленої роботи.
- Проводячи демо, ми можемо отримати менше зроблених завдань, але вони будуть дійсно закінчені.
- Якщо команду змушувати проводити демо, коли у них нічого толком не працює, їм буде не по собі.

- Постарайтеся якомога чіткіше озвучити мета даного спринту. Якщо на демо присутні люди, які нічого не знають про ваш продукт, то не полінуйтеся приділити пару хвилин, щоб
ввести їх в курс справи.
• Не витрачайте багато часу на підготовку демо, особливо на створення ефектної презентації.
Викиньте все непотрібне і сконцентруйтеся на демонстрації тільки реально працюючого коду.
• Слідкуйте, щоб демо проходило в швидкому темпі. Сконцентруйтеся на створенні не тільки
красивого, скільки динамічного демо.
• Нехай ваше демо буде бізнес-орієнтованим, забудьте про технічні деталі.Зосередьтеся на тому "що ми зробили", а не на тому "як ми це робили".
• Якщо це можливо, дайте аудиторії самій спробувати пограти з продуктом.
• Не потрібно показувати купу виправлень дрібних багів і елементарних фич.

Як проводити ретроспективи

Хоча основний формат трохи варіюється, але в основному ми робимо так:

• Виділяємо 1-3 години, в залежності від того наскільки довга очікується дискусія.

• Беруть участь: product owner, вся команда.

• Розташовуємося або в окремій кімнаті з затишним м'яким куточком, або на терасі, або в якомусь іншому схожому місці, оскільки нам подобається вести дискусію в спокійній і невимушеній атмосфері.

• Найчастіше ми намагаємося не проводити ретроспективи в робочій кімнаті, так як це розсіює увагу учасників.

• Вибираємо когось в якості секретаря.

- ScrumMaster показує sprint backlog і за участю команди підводить підсумки спринту. Важливі події, висновки і т.д.
  • Починаємо "серію" обговорень. У цей момент кожен має шанс висловитися про те, що, за його думку, було хорошого, що можна було б поліпшити і що б він зробив по-іншому в наступному спринті. При цьому його ніхто не перебиває.
  • Ми порівнюємо прогнозовану і реальну продуктивність. Якщо є істотні розбіжності, то намагаємося проаналізувати і зрозуміти, чому так вийшло.
  • Коли час добігає кінця, ScrumMaster намагається узагальнити всі конкретні пропозиції щодо того, що ми можемо поліпшити в наступному спринті.

У нас є три колонки:

- Добре: Якщо потрібно було б повторити цей спринт ще раз, то ми б зробили це точно так же.
- Чи могло б бути і краще: Якщо потрібно було б повторити цей спринт ще раз, то ми б зробили це по-іншому.
- Покращення: Конкретні ідеї про те, як в майбутньому можна щось поліпшити.

13 українських IT-компаній потрапили в ТОП-100 світових аутсорсерів - інфографіка

Міжнародна асоціація IAOP опублікувала черговий щорічний рейтинг найкращих у світі постачальників послуг аутсорсингу The Global Outsourcing 100.

У цьому році в рейтинг потрапили відразу 13 компаній з офісами в Україні - на три більше, ніж торік.

Як пише ain.ua, серед 13 компаній - як міжнародні бізнес з великими представництвами в Україні, так і місцеві IT-проекти.

У топ-100 кращих аутсорсерів увійшли: **EPAM, Ciklum, ELEKS, Luxoft, N-iX, Miratech, Intetics, SoftServe, Softjourn, Sigma Software TEAM International Services Program-Ace і Softengi.**

Повністю рейтинг топ-100 кращих аутсорсерів світу виглядає так (компанії наведені в алфавітному порядку):

The 2017 Global Outsourcing 100

IAOP is pleased to announce that the following companies have been selected as 2017's best outsourcing service providers – The Global Outsourcing 100®.

Companies included on the list will have demonstrated their global excellence. Companies were judged on five critical characteristics: size and growth; customer references, awards & certifications, programs for innovation, and Corporate Social Responsibility.

Accelya
Accenture
Acquire BPO
Aegis Limited
AGS Health
Alorica
Aon Hewitt
Arthur Lawrence
Arvato
Auriga, Inc.
Auxis
Bell Integrator
Canon Business Process Services
CBRE
CGI
CGS
Chinasoft International
Cienet
Ciklum
Colliers International
Concentrix
Cushman & Wakefield
DHC
Datrose
Donlen
ELEKS
Ellucian
Emerio GlobeSoft Pte Ltd
Endava
EPAM Systems
EXL
First Line Software
Firstsource Solutions Ltd
Fischer
FPT Software
gA - Grupo ASSA
GeBBS Healthcare Solutions, Inc.
GEP
Happiest Minds Technologies
HCL Technologies Limited
Hexacta
HGS
IBA Group
ICL Services
IMS People
Indecomm Global Services
Integreon
Intetics Inc
ISS
ITC Infotech
Itransition
JLL (Jones Lang LaSalle)
KellyOCG
Knoah Solutions
LeasePlan USA
LegalBase
LiquidHub
Long View Systems
Luxoft
MAYKOR
MERA
Mindtree
Miratech
Newmark Grubb Knight Frank
Nexient
N-iX
OneLink BPO
Pactera
Program-Ace
PromonLogicalis
Pythian
Quatrro
QuEST Global
QuintilesIMS
QuisLex, Inc.
QX Limited
RR Donnelley
Sigma Software
Sitel Operating Corporation
Softengi
Softjourn, Inc.
SoftServe
Sonata Software
SPi CRM
Stefanini
Sutherland Global Services
Swiss Post Solutions
Syntel
Tata Communications Transformation Services (TCTS)
TEAM International Services Inc.
Teleperformance
TeleTech
TELUS International
TIVIT
transcosmos inc.
Trigent Software Inc.
Vee Technologies
VirtusaPolaris
Willis Towers Watson
WNS Global Services

План релізу у Scrum

Іноді потрібно планувати далі, ніж на один спринт вперед. Це типова ситуація для контрактів з фіксованою вартістю, коли нам доводиться планувати наперед, або ж є ризик підписатися під нереальною датою поставки.
Як правило, планування релізу для нас - це спроба відповісти на питання: "коли, в найгіршому випадку, ми зможемо поставити версію 1.0 ".
1. Визначаємо свою приймальних шкалу
У доповненні до звичайного product backlog'у, product owner визначає приймальних шкалу, яка є ні що інше, як просте розбиття всіх історій product backlog'а на групи.

Ось приклад діапазонів з нашої приймальної шкали:

• Всі елементи з важливістю> = 100 мають стояти в версії 1.0, інакше нас оштрафують по повній програмі.

• Всі елементи з важливість 50-99 повинні бути включені в версію 1.0, але в разі чого ми можемо викотити цю функціональність в наступному додатковому релізі.

• Елементи з важливістю 25-49 необхідні, але можуть бути зроблені в подальшому релізі версії 1.1.

• Важливість елементів <25 вельми спірна, оскільки можливо, що вони взагалі ніколи не знадобляться.

важливість	назва
130	Банан
120	Яблуко
115	Апельсин
110	Гуава
100	Груша
95	Ізюм
80	Арахіс
70	Пончик
60	Лук
40	Грейпфрут
35	Папайя
10	Чорниця
10	Персик

Червоні = обов'язково повинні бути додані у версію 1.0 (банан - груша)
Жовті = бажано включити в версію 1.0 (родзинки - цибуля)
Зелені = можуть бути додані пізніше (грейпфрут - персик)
Отже, якщо до крайнього терміну ми закінчимо все: від банана до цибулі, то нам боятися нічого.

Якщо час буде нас підтискати, то ми ще встигнемо викрутитися, прибравши родзинки, арахіс, пончик і цибулю.

Все, що нижче цибулі - бонус.

Щоб спланувати реліз, product owner'у потрібні оцінки, як мінімум оцінки всіх включених в контракт історій.

Як і в разі планування спринту, це - колективна праця команди і product owner'а.

Команда планує, а product owner пояснює і відповідає на питання.

- Нехай команда проведе оцінку.
- Не давайте їм витрачати на це багато часу.
- Переконайтеся, що команда розуміє, що потрібно отримати приблизні оцінки, а не контракт, під яким треба ставити підпис.

https://www.youtube.com/watch?v=gj_UUqFZA0U

Масштабування Scram’у

Коли над одним продуктом працюють відразу декілька Scrum- команд, все набагато складніше.

Це загальна проблема і вона характерна не лише для Scrum 'а.

Ключовими є наступні питання:

- - Скільки сформувати команд?
- - Як розподілити людей по командах?

Досвід показує, що набагато краще мати декілька великих команд, чим багато маленьких, які постійно заважатимуть одна одній.

Створюйте маленькі команди тільки тоді, коли вони не потребують взаємодії одна з одною.

важливість	назва	оцінка
130	Банан	12
120	Яблуко	9
115	Апельсин	20
110	Гуава	8
100	Груша	20
95	Ізюм	12
80	Арахіс	10
70	Пончик	8
60	Лук	10
40	Грейпфрут	14
35	Папайя	4
10	Чорниця	
10	Персик	

Наступний крок - прогноз середньої продуктивності команди

Це означає, що для початку ми повинні визначити наш фокус-фактор. Це значення ніколи не досягне 100%, так як команда завжди витрачає час на незаплановані завдання.
Припустимо, що фокус-фактор нашої команди дорівнює 50%.
Припустимо також, що довжина нашого спринту буде 3 тижні (15 днів), а розмір команди - 6 чоловік.
Таким чином, кожен спринт - це 90 людино-днів, однак, в кращому випадку ми можемо сподіватися
тільки на 45 людино-днів (так як наш фокус-фактор становить всього 50%).
Отже, прогнозована продуктивність складе 45 story point'ов.

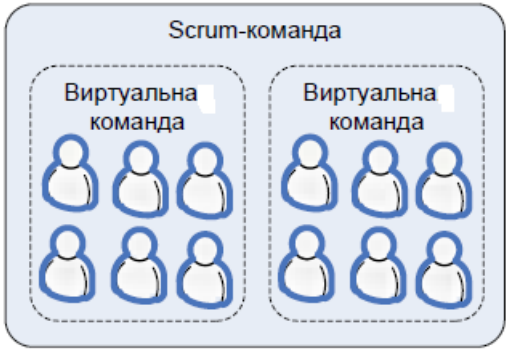
Зараз, коли у нас є оцінки і прогнозована продуктивність, ми можемо легко розбити product backlog на спринти:

Важливість	Назва	Оцінка
Спринт 1		
130	Банан	12
120	Яблуко	9
115	Апельсин	20
Спринт 2		
110	Гуава	8
100	Груша	20
95	Ізюм	12
Спринт 3		
80	Арахіс	10
70	Пончик	8
60	Цибуля	10
40	Грейпфрут	14
Спринт 4		
35	Папайя	4
10	Чорниця	
10	Персик	

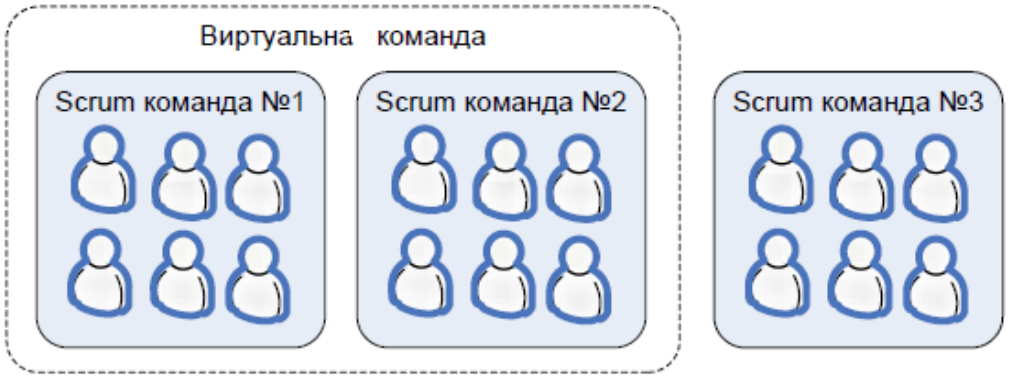
Кожен спринт складається з набору історій, кількість яких не перевищує прогнозовану продуктивність 45. Тепер видно, що, швидше за все, нам буде потрібно 3 спринти для завершення всієї обов'язкової і бажаною функціональності. 3 спринти = 9 календарних тижнів = 2 календарні місяці.

Віртуальні команди

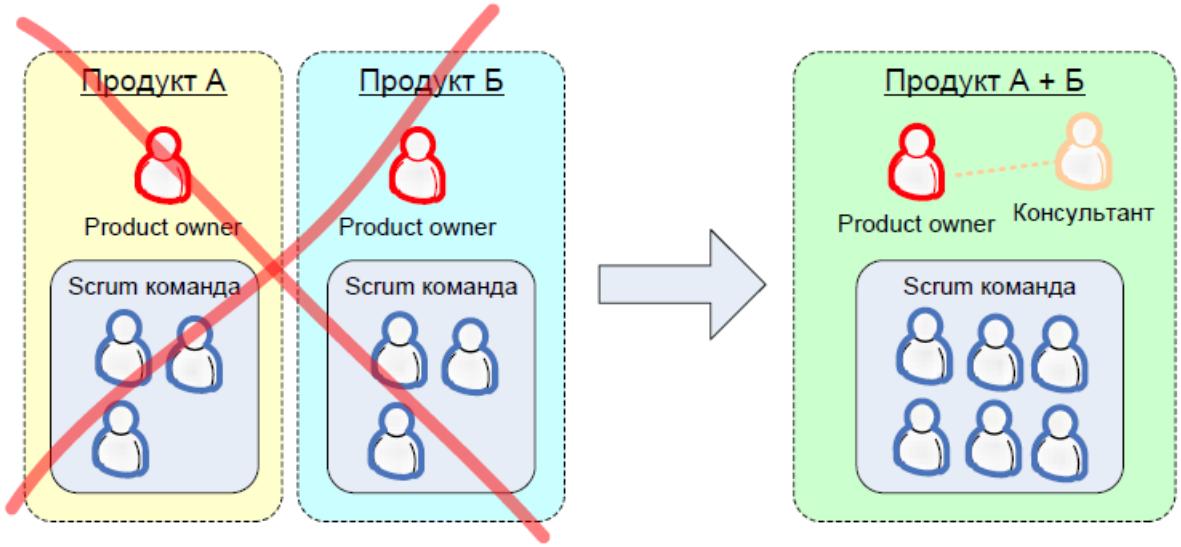
- Приклад 1



- Приклад 2



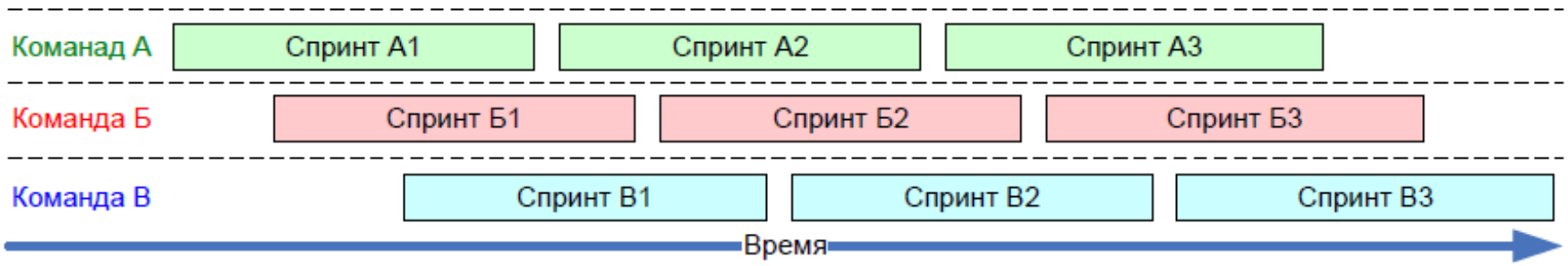
Приклад 3

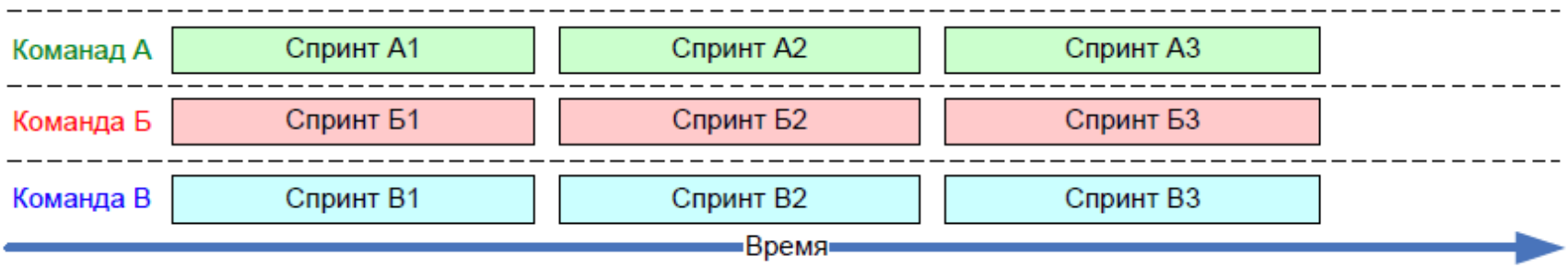

Продукт А
Product owner
Scrum команда
Продукт Б
Product owner
Scrum команда
Продукт А + Б
Product owner
Консультант
Scrum команда

Синхронізація спринтів

Припустимо, є три Scrum команди, які працюють над одним проектом.

Чи повинні їх спринти бути синхронізованими, тобто починатися і закінчуватися одночасно?





Переваги синхронізованих спринтів в наступному:

- З'являється природна можливість перетасовувати команди між спринтами! При пересічних спринтах немає можливості реорганізовувати команди так, щоб не потурбувати жодної команди у розпалі спринту.

- Усі команди можуть працювати на одну мету впродовж спринту і проводити планування спринту разом, що призводить до кращої співпраці між командами.

- Менше адміністративної мороки, наприклад менша кількість зустрічей для планування спринту, демонстрацій і релизів.

Як розподіляти людей по командах?

На випадок, коли у вас декілька команд працюють над одним і тим же продуктом, існує дві стратегії розподілу людей по командах:

- Дозволити спеціально призначеній людині провести розподіл, наприклад product owner 'у або будь-якому іншому менеджерові.
- Дозволити командам якимсь способом самоорганізовуватися.

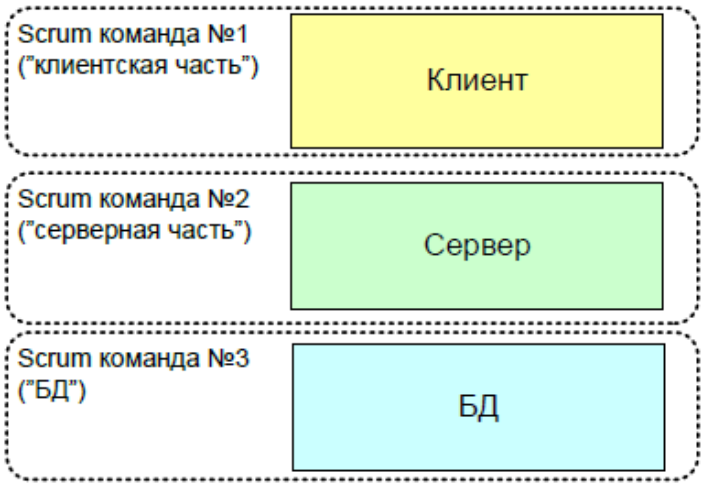
І виявилось, що комбінація двох стратегій працює краще всього.

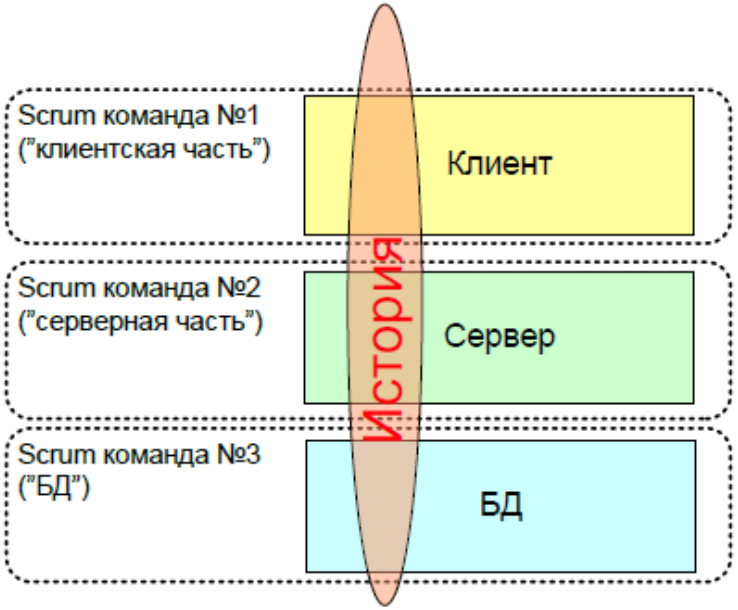
Чи потрібні вузькоспеціалізовані команди?

Припустимо, ваша система складається з трьох основних компонентів: КлиентСерверБД

Припустимо, що над вашим продуктом працюють 15 чоловік, і вам не дуже хочеться збирати їх в одну Scrum- команду. Як же розділити людей на команди?

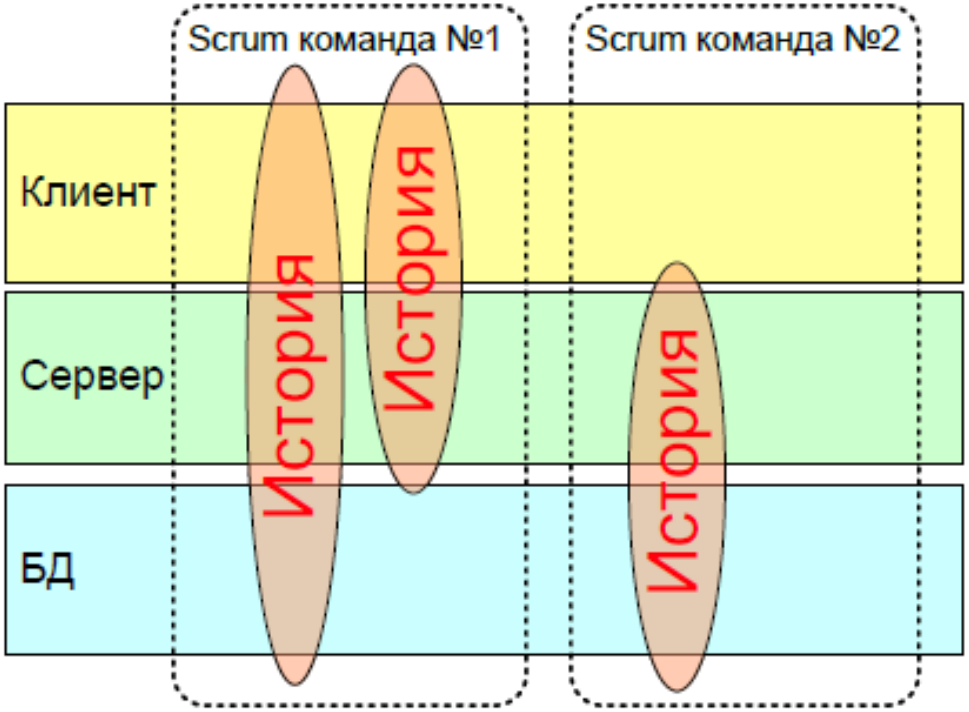
Варіант 1
Спеціалізація на компонентах





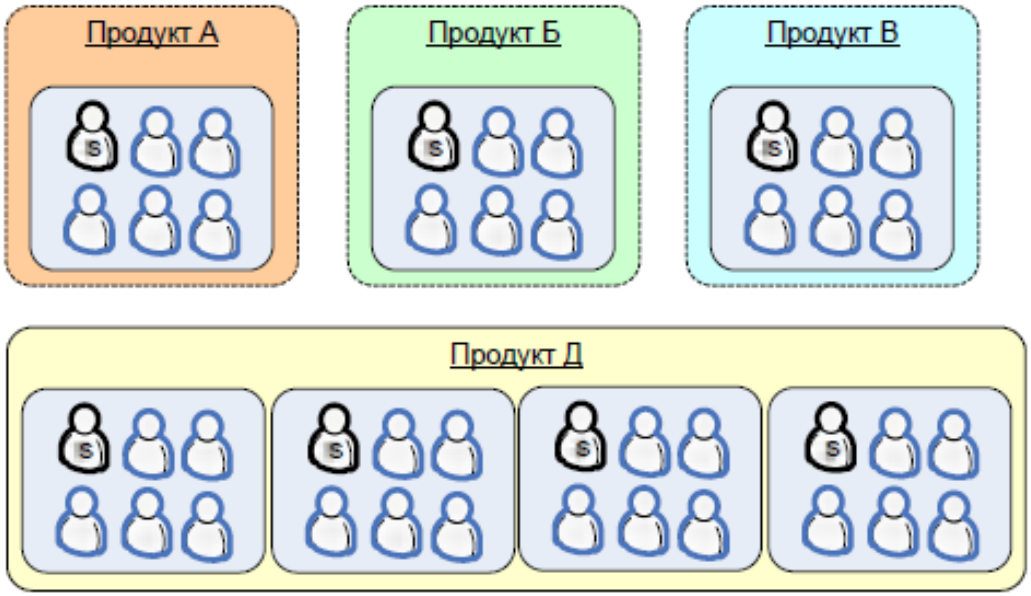
Варіант 2

Універсальна команда



Scrum - of - scrums

Scrum - of - scrums - це регулярні зустрічі, мета яких - обговорення різних питань між Scrum- майстрами та командами.



Scrum - of - Scrums рівня продукту

Ця зустріч була дуже важливою. Ми проводили її один раз в тиждень. Ми обговорювали проблеми інтеграції, балансування команд, підготовку до наступного планування спринту і так далі. Ми виділяли на це 30 хвилин, але часто нам їх бракувало. Наш порядок денний мав наступний вигляд:

1. Кожен по черзі розповідав, що його команда зробила минулого тижня, що планує закінчити на цього тижня, і з якими труднощами вони зіткнулися.
2. Будь-які інші проблеми, що відносяться до компетенції декількох команд одночасно, які треба обговорити. Наприклад, питання інтеграції.

Scrum - of - Scrums рівня компанії

Ми назвали цю зустріч "Пульсом". ЦЕ щотижневІ збори тривалістю 15 хвилин, в якому бере участь увесь колектив (взагалі-то все ті, хто беруть участь в процесі розробки).

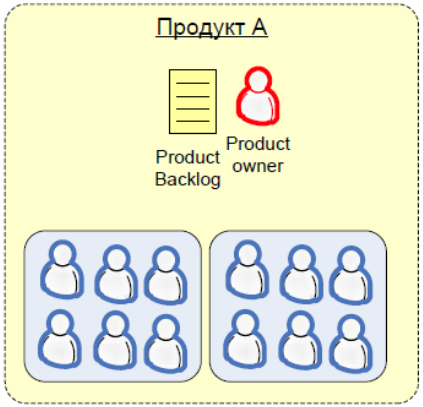
Це працює? Так - працює, якщо ви дуже строгі відносно того, щоб збори були стислими.

Формат зборів :

1. Новини і уточнення з боку керівника розробки. Наприклад, інформація про майбутні заходи, події.
2. "Карусель". Одна людина з кожної продуктової групи звітує в тому, що було зроблено за минулий тиждень, що планується зробити цього тижня і про проблеми.
3. Усі, хто хоче, можуть вільно висловитися і поставити будь-які питання.

Розбивати product backlog чи ні?

Припустимо, у вас є один продукт і дві Scrum- команди. Скільки вам треба product backlog ‘ів? Скільки product owner 'оів? Вибір досить сильно вплине на те, як проходитимуть зустрічі по плануванню спринту. Ми оцінили три можливі підходи.



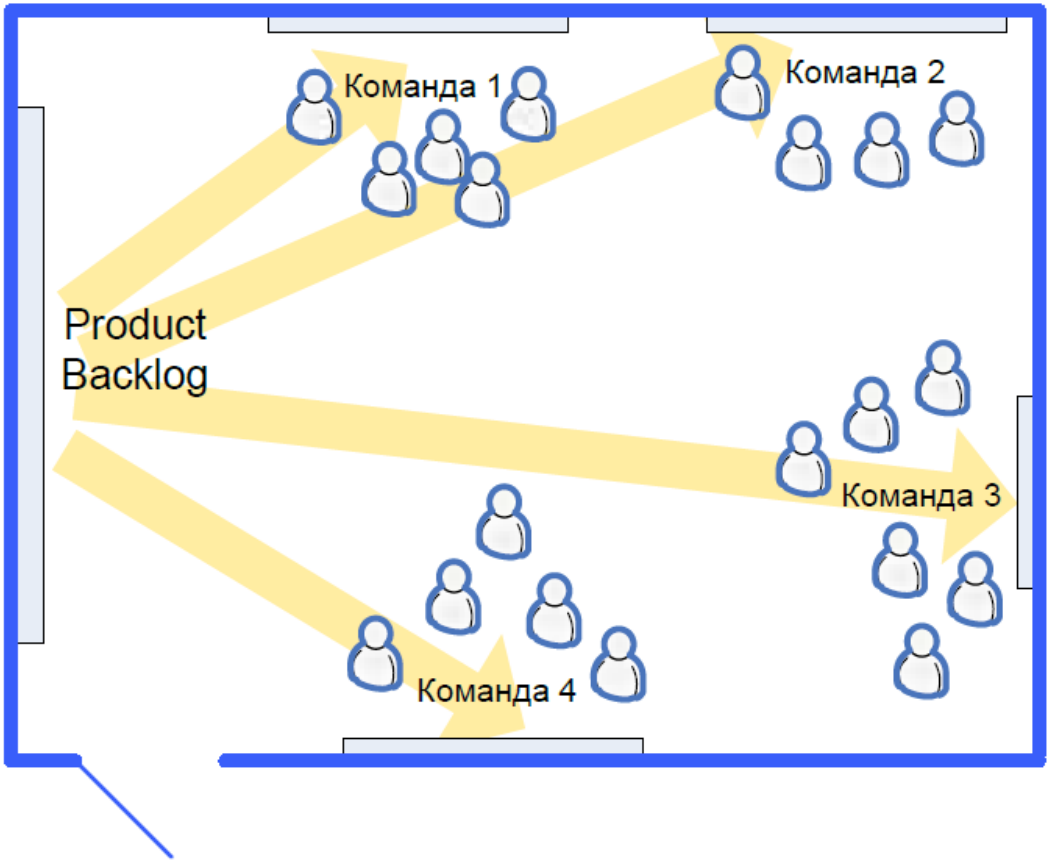
Підхід перший :

Один product owner - один backlog

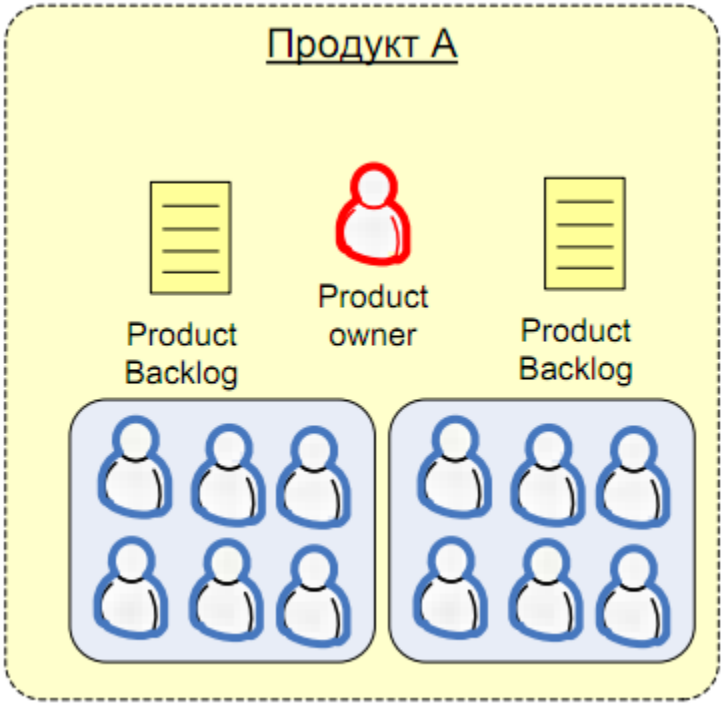
"Повинен залишитися тільки один".

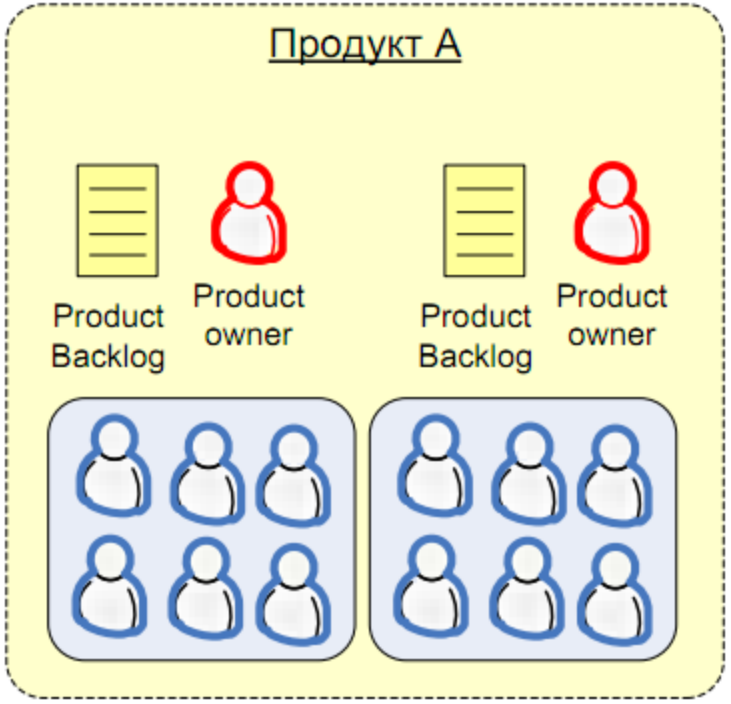
Перевага цього підходу в тому, що можна доз самій планувати роботу на основі пріоритетів, розставлених product owner 'ом.

Product owner може зосередитися на тому, що йому треба, і надати командам самим, розбивати історії на завдання.


Команда 1
Команда 2
Product
Backlog
Команда 3
Команда 4

Можливі варіанти





Scrum для географічно розподілених команд

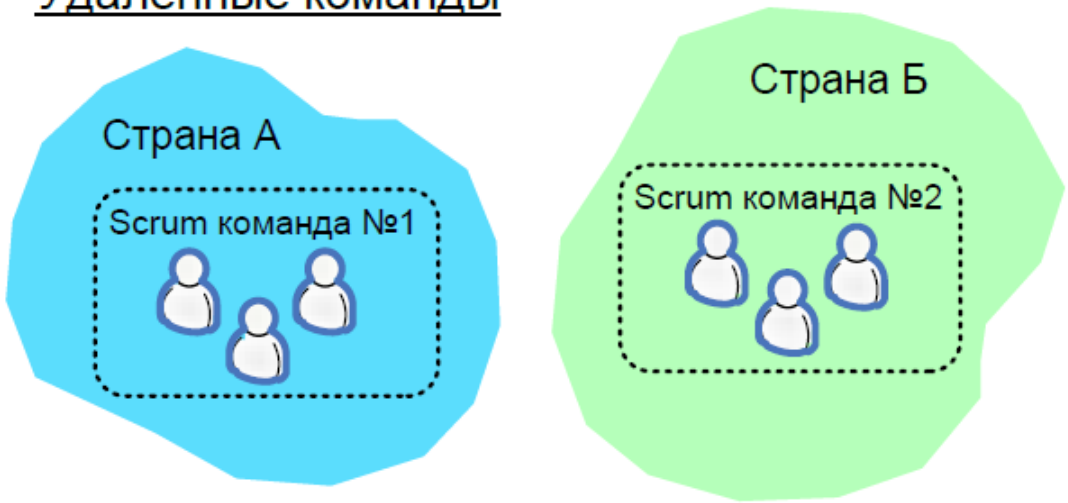
Засоби комунікації повинні надавати :

- Можливість практикувати парне програмування.
- Можливість зустрічатися лицем до лиця в ході щоденного Scrum 'а.
- Можливість побачити один одного у будь-який момент.
- Можливість особистих зустрічей і живого спілкування.
- Можливість проводити незаплановані наради усією командою.
- Можливість бачити одні і ті ж версії sprint backlog 'а, sprint burndown 'а, product backlog 'а і інших джерел інформації за проектом.

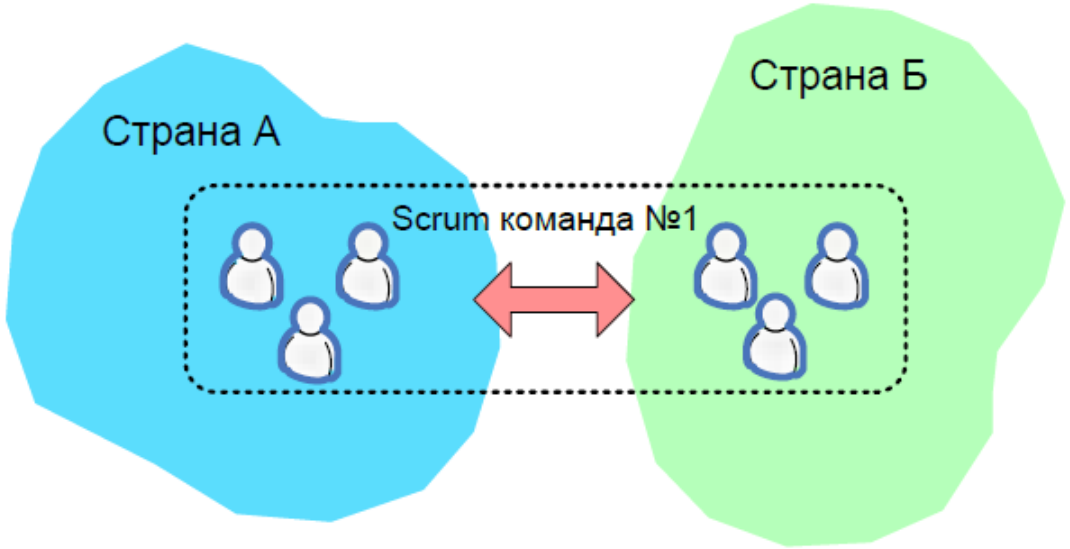
Ось деякі заходи для досягнення мети :

- Web- камера і навушники з мікрофоном на кожній робочій станції.
- Кімната для проведення телеконференцій, обладнана web-камерами, мікрофонами, комп'ютерами з усім необхідним ПЗ - для загального доступу до робочого столу, проведення телеконференцій і так далі
- "Віддалені вікна". Великі монітори в кожному офісі, на яких постійно можна бачити, що відбувається в інших офісах. Щось типу віртуального вікна між двома відділами. Можна стояти перед ним і спостерігати.

Удалённые команды



Удалённые участники команд



Парне програмування

Переваги парного програмування:

• Парне програмування дійсно покращує якість коду.

• Парне програмування дійсно збільшує зосередженість команди (наприклад, коли напарник каже: "Слухай, а ця штуковина точно потрібна для цього спринту?")

• Дивно, але багато розробників, які виступають проти парного програмування, насправді не практикували його, проте раз спробувавши - швидко розуміють всі переваги.

• Парне програмування вимотує, так що не варто займатися ним цілий день.

• Часта зміна пар дає хороший результат.

Парне програмування

• Парне програмування дійсно сприяє поширенню знань всередині команди, помітно прискорюючи цей процес.
• Деякі люди відчувають себе некомфортно, працюючи в парах. Не варто позбавлятися від хорошого програміста, тільки тому, що йому не подобається парне програмування.
• Рев'ю коду - хороша альтернатива парному програмування.
• У "штурмана" (людини, який не пише код) повинен також бути свій комп'ютер, але не для розробки, а для виконання дрібних завдань, коли це необхідно - перегляду документації, якщо "водій" (людина, яка пише код) запнувся і так далі.
• Не нав'язуйте парне програмування людям. Надихнете їх, дайте необхідні інструменти та дозвольте самим дійти до цього.

Пам'ятка Scrum Master‘у

На початку спринту:

- Після планування створити "сторінку з інформацією про спринт".

-На стартовій сторінці wiki-порталу помістити посилання на створену сторінку.

-Роздрукувати цю сторінку і повісити її на стіні, яка у всіх на очах.

-Розіслати e-mail'и з повідомленням про початок нового спринту. Не забути вказати мету спринту і дати посилання на "сторінку з інформацією про спринті".

-Оновити статистику спринтів. Додати оцінку попередньої продуктивності, розміру команди, довжини спринту і т.д.
https://www.youtube.com/watch?v=sL_ZtTxkbk8

-

Кожен день:

- Стежити за тим, щоб щоденний Scrum починався і закінчувався вчасно.
- Стежити за тим, щоб у разі додавання або видалення історії з sprint backlog'a все було зроблено, як годиться, щоб ці зміни не зірвали графік робіт.
- Стежити за тим, щоб product owner знав про ці зміни.
- Стежити за тим, щоб команда постійно оновлювала burndown-діаграму.
- Стежити за тим, щоб всі проблеми вирішувалися. Як варіант поінформувати про них product owner'a і / або начальника відділу розробки.

В кінці спринту:

- Провести відкриту демонстрацію результатів спринту.
- За кілька днів до демонстрації нагадати всім про її проведення.
- Провести ретроспективу за участю всієї команди і product owner'а. Запросити начальника відділу розробки, щоб він допоміг знайти оптимальне рішення проблем.
- Оновити статистику спринтів. Внести значення реальної продуктивності і основні тези ретроспективи.

Certifications

- **Scrum Alliance**

- **ICAgile**

- **PMI Agile Certified Practitioner (PMI-ACP)®**

IT-спеціальності

1. **Frontend Developer,**
2. **.NET Developer,**
3. **ASP.NET MVC Developer,**
4. **Java Developer,**
5. **PHP Developer,**
6. **Python Developer,**
7. **Unity / Game Developer,**
8. **Quality Assurance,**
9. **iOS Developer,**
10. **Ruby Develope,**
11. **JavaScript Developer,**
12. **ASP.NET Core Developer,**
13. **C++ Developer,**
14. **Android Developer,**
15. **Angular Developer,**
16. **.NET Desktop Developer,**
17. **React Developer,**

http://gamedev.lviv.ua/?utm_source=Infopartners&utm_campaign=IPGD

https://www.youtube.com/watch?v=MoKkYb3h6Qo

Kanban

Kanban (яп. かんばん(看板)) (буквально білборд, рекламний щит) - це система розпорядку для ощадливого та якраз вчасного (JIT) виробництва.

Kanban - це система для контролю логістичного ланцюга з точки зору виробництва, і не система інвентаризації.

Канбан було розроблено Таїті Оно, в Toyota, з метою досягнення та підтримки високого рівня виробництва. Kanban - це один з методів досягнення принципу *Якраз вчасно*.

Kanban

В основі Kanban а лежить проста ідея:

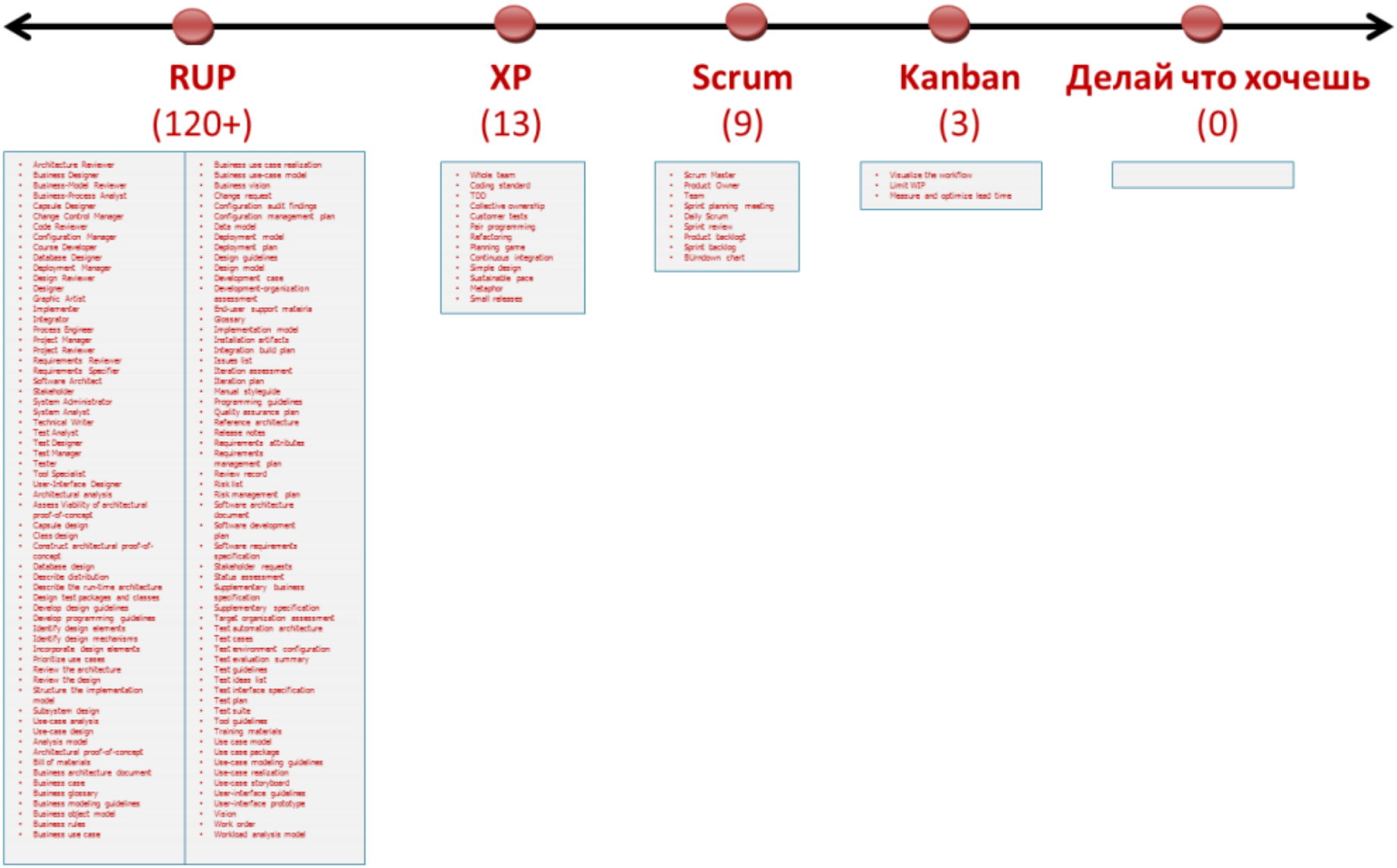
Кількість незавершеної роботи (НЗР) має бути обмежена, і щось нове може починатися тільки тоді, коли якийсь існуючий шматок роботи поставляється
чи в термінах lean *витягується* наступним елементом конвеєра.

Kanban

- **Візуалізуйте потік робіт**
  -Розбийте роботу на частини, впишіть кожен з пунктів на картку і прикріпіть на стіну.
  -Підпишіть стовпці, щоб бачити на якій стадії знаходиться кожне завдання.
- **Обмежте НЗР (WIP)** (прим. work-in-progress - незавершена робота) -визначте можливу кількість незавершених пунктів на кожній стадії робочого процесу.
- **Виміряйте час виконання завдання** (lead time) (середню тривалість часу) для завершення одного пункту, щоб звести час виконання завдання до мінімуму і зробити його настільки прогнозованим, наскільки це можливо.

https://www.youtube.com/watch?v=R8dYLbJiTUE

Кількість директив



Kanban має такі переваги:

- Гнучкість планування. Команда концентрується тільки на поточну роботу, пріоритет завдання виставляється менеджером.
- Висока залучення команди в процес розробки. Завдяки постійним зборам, прозорості процесів і можливостям самоорганізації працівники гуртуються і проявляють щирий інтерес.
- Менша тривалість циклу. Якщо кілька людей має схожі навичками, тривалість скорочується, якщо ж тільки один - з'являється вузьке місце. Тому співробітники повинні ділитися знаннями і тим самим оптимізувати тривалість циклу. Тоді вся команда зможе взятися за роботу, яку забуксувала, і відновити плавний потік.
- Менше вузьких місць. Ліміти **НЗР** дозволяють швидко знаходити вузькі і проблемні місця, які з'явилися через дефіцит уваги, людей або навичок.

Kanban має такі переваги:

- Наочність. Коли всі виконавці мають доступ до даних, то вузькі місця легше помітити. Kanban-команди, крім самих карток, зазвичай використовують два загальних звіти: графіки управління і сукупного потоку.
- На практиці система відмінно себе показує в сферах неосновного виробництва:
- групи підтримки програмного забезпечення або служби підтримки.
- Канбан добре працює при управлінні стартапами без чіткого плану, але де активно просувається розробка.

Канбан має і недоліки:

- система погано працює з командами чисельністю понад 5 осіб
- він не призначений для довгострокового планування.

Відмінності від скраму

Скрам, як і agile kanban, є гнучкою методикою і теж часто застосовується в IT-сфері. Відмінності між ними не очевидні на перший погляд. Існує багато схожості, наприклад, наявність беклога в обох підходах.

	Скрам	**Канбан**
Темп	Повторювані спринти фіксованою тривалості	Безперервний процес
випуск релізу	В кінці кожного спринту після схвалення Product Owner-ом	Потік триває без перерв або на розсуд команди
Ролі	Product Owner, Scrum-майстер, команда розробників	Команда під керівництвом ПМ, в деяких випадках залучаються тренери по agile kanban
Головні показники	Швидкість команди	Провідний час
Прийнятність змін	В ході спринту зміни небажані, так як можуть привести до невірної оцінки завдань	Зміна можуть трапитися в будь-який момент