**WBS** – **work** **breakdown** **structure** – **ієрархічна** **структура** **робіт** – декомпозиція цілей проекта на менші підцілі\блоки\work package (записуються результати робіт, а не задачі які треба виконати для досягнення цих результатів). Як правило в верху цьої ієрархії знаходиться сам проект, а під ним рівень з основними результатами, які в свою чергу розбиваються на рівень з легшими підрезультатами. В різних вітках буває різна кількість рівнів в залежності від потрібною деталізації.

Для чого:

* добре показує весь обєм проекта,
* фокусує увагу не на процесі, а на очікуваних результатах,
* в створенні АБС бере участь клієнт або його представник,
* можна наглядно продемонструвати, що потрібно більше фінансування чи ресурси команди,
* Допомагає зменшити ризики чи кількість змін в подальшій розробці

Розбивають ієрархію на блоки зазвичай так шоб було зрозуміло команді, але є і стандарті підходи до розбивання:

* По стадіям життєвого циклу проекта (фаза планірування, аналізу, розробки чи здачі).
* По високорівневим результатам проекту (готова система, обучені юзери, улагодження документів з законодавством, написання документації)
* По структурі організації, якшо різні частини команди відповідальні за різні речі.
* По срокам

Якщо потрібно ВБС застосувати для методології з ітераціями, то можна робити ВБС для кожної ітерації або розбивати кожен результат для кожної ітерації, або кожен спрінт буде як окремий блок в ВБС якшо відомо очікуваний результат спрінта на перед.

Також створюється словник ВБС який пояснює розгорнуто що ми мали на увазі в своїй ієрархії під тим чи іншим терміном\виразом.

**PERT** – оцінка по трьом точкам – прийом який дозволяє вирахувати час на виконання роботи за допомогою «оптимістиного», «песимістичного» і «найбільш ймовірного» результату.

Согласно PERT предполагаемая длительность (или EAD) составит:

**EAD = (P + 4M +O) / 6**

где P – «пессимистичная оценка» , O – «оптимистичная оценка», M – «наиболее вероятная оценка».

**Software development process** – структура по якій буде будуватися процес розробки продукту. Відповідно є різні моделі по яких відбувається розробка ПО і кожна з них має свій підхід.

**Стейкхолдер** — физическое лицо или организация, имеющая права, долю, требования или интересы относительно [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) или её свойств, удовлетворяющих их потребностям и ожиданиям

**Software development activities**:

* **Аналіз вимог.** Тобто збір вимог до продукту, їх систематизація, виявдення взаємозвязків між вимогами, документування вимог, первірка вимог. Реквайрменти можуть бути **функціональними** (визначають дії які система повинна могти виконувати і як вона має взаємодіяти з юзером чи іншими системами) і **не** **функціональними** (показують які обмеження може мати система чи які характеристики має система не залежно від поведінки системи. Наприклад, перформанс системи, можливість до розширення, відказостійкість тощо). Також є поняття **специфікація** **реквайрментів** – це повний опис поведінки системи яку мають імплементувати. Включає всі юзер флови в системі і взагалі функціональні і не функціональні реквайрменти.
* **Проектування програмного забезпечення**. На даному етапі створюється архітектура продукту з точки зору ПО. Визначаються внутрішні властивості і представлення даних системи і зовнішні властивості системи та вхідні представлення даних, розробляються потрібні алгоритми, підбираються тести і відбувається документація. Використовуються всякі блок-схеми чи UML-діаграми.
* **Імплементація**. Безпосередньо написання коду, імплементація алгоритмів, створення інфраструктури бази даних, побудова комунікації між компонентами і т.д.
* **Тестування ПО**. Перевіряється чи відповідає реальна поведінка функціоналу – очікуваній поведінці.
* **Системна інтеграція системи**. Процес, коли всі компоненти системи збираються в одне ціле і очікується, що всі компоненти системи взаємодіють між собою без проблем. **Методи** інтеграції: **Вертикальна** інтеграція: інтеграція підсистем відповідного до схожості їх функціональності і створення так званих функціональних ентітей. Відповідно інтеграція проходить швидко і потребує участі тільки потрібних вендорів. **Горизонтальна** інтеграція: ще її називають ESB (enterprise service bus), метод який інтегрує підсистеми між собою, шляхом спеціалізованої підсистеми через яку відбувається комунікація між всіма підсистемами системи. Це дозволяє зменшити кількість конекшенів між підсистемами і так як комунікація відбувається на рівні інтерфейсів, то легко замінити якусь підсистему на підсистему з іншим функіоналом (на відміну від вертикальної інтеграції). **Зірко**-**подібна** або **спагетті** інтеграція: вид інтеграції, коли кожна підсистема має зв'язок з кожною з інших підсистем. Цей метод флексібл для перевикористання функіоналу, але затратний в плані введення нових підсистем.
* **Впровадження** **ПО**. Процес налаштування продукту під умови використання клієнтом і навчання користувачів як працювати з системою. Тобто продукт передається в користування.
* **Супроводження** **продукту**. Процес усунення дефектів, покращення та оптимізація продукту після передання продукту в експлуатацію.

**Моделі SDLC процесу розробки ПО.**

**Водопадна** – **waterfall** – **каскадна** – модель в якій за основу взято строге послідовне виконання всіх етапів розробки при чому при переході на наступний етап – попередній етап повинен бути закінчений. Також при закінчення кожної стадії розробки повинна буде створена повна документація щоб можна було продовжити подальшу роботу навіть іншій команді. Переваги: можна оцінити повністю час на розробку і наявність повної документації кожного з етапів. Недолік: якщо якась із стадій завершується невдачею – потрібно відкочуватися по стадіям назад і проходити все заново. Наприклад якшо буде якась зміна реквайрментів, то це знову ж потрібо задокументувати, спроектувати, імплементувати і т.д, а це прзводить до вибивання з графіку і затрат на розробку. Тому ця модель не підходить для великих проектів, а більш підійде для невеликих систем.

Этапы проекта в соответствии с каскадной моделью:

* Формирование требований;
* Проектирование;
* Реализация;
* Тестирование;
* Внедрение;
* Эксплуатация и сопровождение.

**Agile** – **грнучка розробка ПО** – модель процесів розробки ПО яка базується на ітеративній розробці де відбувається взаємодія між багатофункціональними самоорганізованими командами. Такий підхід дозволяє зменшити ризики, оскільки робота відбувається короткими циклами по тижню-два і за цей час відбувається розробка якоїсь невеликої функціональності від загального скоупу фіч проекту. По суті, кожна ітерація включає в себе аналіз вимог, проектування фічі, імплементація, тестування і її документацію. Дана методологія акцентує увагу на face-to-face спілкуванні на рахунок проекту як в середині команди, так і акрос тімс при чому за участі клієнта або його представника. **Переваги**: такий підхід більш гнучкий за умов зміни реквайрментів, ну і грубо кажучи продукт завжди готовий до релізу в залежності від завершеності функціоналу. **Недостатки**: низький рівень задокументованості в порівнянні з тією ж водопадною моделлю, також є ідея, що при даній методологія можуть поспішно прийматися рішення які не вписуються в еталонну базову модель продукту.

**Методології які дотримуються моделі Agile**:

* **Agile** **Modeling** – націлена на швидкість і простоту документування та моделювання процесу розробки, але не охоплює проектування продукту, імплементацію, тастування і так далі.

**Extreme** **programming** – простими словами це це набір практик розробки ПО, коли, наприклад, один програміст перевіряє код іншого програміста або написання коду представляється як парне програмування, коли один девелопер пише код, а інший девелопер в цей же час моніторить цей код. Так же сюди відносять TDD. Також є якийсь код конвеншн якого всі дотримуються і не сперечаються як правильно має бути написано. Також сюди можна віднести continuous integration, коли часто робиться інтеграція компонентів системи на протязі розробки щоб потім в релізі не мати з цим проблем або хоча б мінізувати ці проблеми. Методология XP похожа на Скрам, но спринты короче, изменения бэклога во время спринта допустимы, а приоритеты XP определяет кастомер.

* **Lean** – клієнтоорієнтованість.
* **Scrum –** методологія керування розробки ПО де є система ролей, мітингів, артифактів і правил притримуючись яких саморганізована команда, або декілька команд розробляють по поетапно, тобо інкрементативно. Тому скрам ще називають фреймворком керування проектом. В одній команді може буди близько 5-9 людей. Вони і відповідають за самостійну побудову процесів. Команди використовують інтерація які називають спрінтами, які зазвичай тривають 1-2 тижня, але не більше 1 місяця. Тобто по скраму продукт розробляється не одразу, а невеликими частинами готовими до релізу.

**Ролі в SCRUM:**

* **Product owner –** людина яка розуміє цінність продукту з точки зору бізнеса і доносить бажання клієнта до команди. Відповідає за юзер сторі і визначення їх пріоритету. Не відповідає за технічну частину.
* **Команда девелоперів –** відповідає за аналіз, дизайн, проектування, програмування, тестування і технічну комунікацію керуючись юзер сторями і відштовхуючись від їх пріоритету.
* **Скрам-мастер –** допомагає продук овнеру і команді виконувати роботу без всяких ішью чи блокерів шоб їх нічого не відволікало. Тобто вся комунікація поза командою відбувається через скрам-мастера. Іноді команди взаємодіють між собою в форматі scrum of scrum – це коли збираються всі скрам мастера команд і обговорюють всякі важні вещі.

**Царемонії в SCRUM:**

* **Release planning –** приблизна оцінка і розбір фіч які потрібно імплементувати в майбутньому релізі. Результат такого пленінга може слугувати орієнтиром того як просувають роботи вже безпосередньо на протязі розробки реліза.
* **Sprint planning –** мітинг в якому бере участь вся команда і де обговорюєть фіча, яку потрібно зробити в спрінті і в які сроки буде виконана робота. Обговорюються деталі і можливі ризики.
* **Daily scrum –** щоденний короткий мітинг де планується робота для кожного учасника команди на цей день. Тут можна прояснити якісь моменти по сторі чи обговорити якісь складнощі які виникли по ходу розробки.
* **Sprint review –** мітинг який проходить в кінці спрінта щоб показати стейкхолдеру виконану за спрінт роботу.
* **Retrospective –** мітинг на якому обговорюють моменти з попередньосго спрінта і розбирають, що було зроблено добре, а які моменти потрібно покращити в майбутньому.

Такожє **Backlog refinement** на якому уточнюють і пріоритизують елементи які лежать в беклозі для підготовки до наступного спрінта.

**Артефакти SCRUM**

* **Беклог продукта –** всі необхідні айтеми які повязані з технічною та не технічною сторонами проекту.
* **Беклог спрінта –** набір задач які потрібно виконати на протязі спрінта. Задачі в цей беклог перетягуються з беклогу продукта.
* **Інкремент -**  представляє собо суму всіх елементів Беклогу продукта які були виконані в спрінті і value всіх попередніх виконаних спрінтів. В кінці спрінта кожен інкремент повинен бути готовий.
* **Sprint goal –** виконати елемент беклога продукта за спрінт. Зазвичай цей елемент розбвають на менші таски.
* **Scrum board –** зазвичай віртуальна панель в якій прослідковується статус робіт на протязі спрінта. Така доска розбивається на колонки з статусами айтемів над якими ведеться робота які підходять для команди. Наприклад, сторі з спрінта можк знаходитися в статусах To do, In progress, In code review, Ready for testing, In Testing, Ready for PO review, Closed. І, відповідно до статусу такска сторі візуально знаходиться в певній колонці і перпеміщається з колонки в колонку по ходу виконання роботи
* **Srum cards** – атрибут пленінг покеру, тобто техніки оцінки сроків виконання задачі. Тобто обговорюється фіча яка описана на карті і коли діло доходить до естімейтів, то кожен з команди дає карту зі своїм естімейтом і як правило береться середнє значення або починається аргументування карт.
* **Epic** – сукупність User story які обєднують в групу для досягнення якоїсь більш глобальної фази продукту.
* **User story** - короткий опис фічі, який описується з точки зору кінечного юзера чи системи. Наприклад, я як пацієнт, хочу мати можливість завантажувати документи в свій онлайн кабінет. Зазвичай за це відповідальний продукт овнер.
* **Task –** це частинка роботи на які розбивають сторі.

**Estimation:**

* **Sprint Burdown chart –** графік який показує скільки роботи залишилося до кінця виконання всіх задач спрінта.
* **Release burn down chart –** графік яки показує скільки спрінтів виконано і скільки роботи в сторіпойнтах залишилося ще зробити на даний момент по закінченню спрінта. Тобто по осі х йдуть спрінта, а по осі у йдуть сторіпойнти.
* **Story point –** оцінка тривалості роботи для одного девелопера. Виглядають ці пойнти в представленні чисел Фібоначчі. В Фібоначчі тому, що ідея полягає в тому, що чим більше сторіпойнтів займає естімейт, ти більше в ньому є незрозумілостей і ризиків, тому відповідно такі речі закладаються в цей інтервал між числами Фібоначчі.
* **Velocity –** обєм роботи яку виконує команда на протязі спрінта. Виглядає як сумма сторіпойнтів всіх задач, які були виконані в спрінті.

**Переваги:**

- **відгук на потреби клієнта**. Завдяки роботі спрінтами клієнт завжди може отримати реліз продукта, коли йому треба і є можливість вносити зміни в продукт з невеликими ризиками.

- **зниження вартості розробки** за рахунок того, що в команді люди є багатофункціональними і можуть робити багато речей самостійно, наприклад девелопати і тестувати фічу самостійно.

- **більше швидких доходів.** Тобто з точки зору менеджмента кошти від клієнта за виконану роботу поступають часто, а не декілька раз в рік як в лінійних підходах розробки ПО.

**Недоліки**:

* Можуть виникнути нюанси при роботі команд в яких люди з різних країн та часових поясів. Тобі комунікація на якій базується сркам може бути недостатньо ефективною.
* Інколи мітинги забирають дуже багато часу

**Kanban –** методологія розробки ПО яка націлена на баланс в команді. Її основна задача це щоб робота була збалансованою для всіх учасників команди і не було випадків, коли, наприклад, бекенд завалений роботою, а юай девелопери сидять пів спрінта без роботи бо такої на разі нема. Тут немає поняття продукт овнера чи скрам мастера, а вся команда вважається єдиною. При чому задачею поетапно можуть займатися різні спеціалізовані команда, наприклад аналітики, потім дизайнери, потім девелопери, хоча багатофункціональні команди в Канбан не заборонені.

**Різниця між Scrum i Kanban:**

* основна різниця заключається в тривалості ітерацій. Якшо в скрамі це допустим 2 тижня, то в Канбані можна підсовувати задачі девелоперу хоч кожного дня.
* В Скрам задачі оцінюють в сторіпойнтах і в кінці спрінта визначається велосіті команди, щоб знати чи вписується фіча в спрінт. В Канбан же не обовязково робити естімейти і нема поняття велосіті, а визначається середній час на задачу. Тобто скільки часу пройшло з моменту коли сторі попала в In progress до моменту, коли вона одобрена РО і задеплоєна (це ще називають cycle time).
* В скрам головна ціль – закінчити спрінт, в канбан головна ціль – закінчити задачу.

Рекомендації по вибору: якщо це новий продукт чи реліз, то краще використовувати скрам бо він дає контроль над процесом розробки і прозорість сроків виконання роботи. Також завдяки скрам царемоніям команда часто і детально обговорює продукт і завдяки чому краще погружається в розуміння продукту. А от коли реліз відбувся і пішля стадія підтримки, то від клієнта може прилітати багато дрібних задач чи зауважень на які потрібно швидко реагувати при чому вони можуть міняти послідовність чи пріоритети і в такому випадку добре підійде канбан.

**Scope**

- Project scope - обєм роботи який потрібно завершити щоб заделіверити продукт чи фічу.

- Product scope – функціонал який охоплює продукт в якийсь певній області.

**Estimation techniques** - counting, expert judgment (individual and group), decomposition, analogy, proxy estimation, planning poker etc.

**Goal** – цілі конкретного етапу, тобто якась точка в графіку розробки. Наприклад, ціль написати фічу за 2 місяці і викатити реліз за 3 місяці.

**Commitment** – це згода закінчити розробку певного функціоналу в умовлений срок.

**overestimate or underestimate –** з точки зору девелопера це заестімейтити замало або забагато часу на девелопмент фічі, з точки зору менеджмента потрібно зберігати баланс між цими речами оскільки оверестімейт ніби добре з точки зору притоку грошей від клієнта, але можлива втрата цього самого клієнта, якому не сподобаються сроки виконання робіт. Так само і андерестімейт принесе менше коштів, створює ризик не завершити роботу в сроки, але є комфортним для клієнта, оскільки потребує менше фінансових затрат.

**Decomposition** – розбивання задачі на дрібніші підзадачі.

**Recomposition** – процес обернений до декомпозиції.

**Analogy-based estimations –** естімейт фічі и продуктку оснований на попередньому досвіді або по аналогу до схожого проекту, який вже пройшов стадію розробки, яку розглядається на даний момент в проекті який естімується.

**Story based estimation –** коли командою беруться сторі з беклогу і відбувається їх естімейт. Відповідно на базі цих естімейтів можна побачити яку роботу і в які сроки може виконати команда.

**Software requirements** – опис вимог та очікуваної поведінки фіч чи певного функціоналу. Вони описують на яку поведінку розраховує юзер від продукту.

**Functional requirements** – реквайрменти які мають безпосереднє відношення до функціоналу продукту. Наприклад, є вимога, що юзер може написати репорт в випадку, якшо якась його дія зафейлиться.

**Non-functional requrements** – реквайрменти які не мають прямого відношення до функціоналу яким користується юзер, але мають відношення до самого продукту. Наприклад, є вимога, щоб продукт був засекюрений, шоб дії користувачів логувалися, шоб з перформансом продукту було все ок і так далі. Сюди відносять system requiremets.

**Business requirements** – вимоги до функціонлу продукту проаналазовані з точки зору вимог кінечного юзера системи. Створюються БіЕями або продукт овнерами.

**User requirements** – вимоги до функціоналу який важливий і видимий для клієнта які мають задовільняти бізнес реквайрменти продукту.

### **Most Common Requirement Risks**

* Недостатнє задієння людей
* Мінливі чи двозначні реквайроменти
* Додавання девелопером власних незапланованих функцій
* Мінімальна специфікація
* Необережне планування

### **Characteristics of Excellent Requirements**

* Statement
  + Закінчені
  + Коректні - все що записане є обґрунтованим і має значення
  + Пріоритизовні
  + Однозначні
  + Можуть бути перевірені
* Specification
  + Завершена
  + Консистентна
  + Може мінятись
  + Може відслідковуватись

### **Benefits from High Quality Requirement Process**

* Менше дефектів в ріквайроментах
* Зменшено час розробки
* Швидка розробка
* Менше непотрібних фіч
* Менше витрат на покращення та розвиток продукту
* Менше міскомунікації
* Зменшено можливі похибки сковпу
* Більш точні естімейти на тестування системи

## **Root Causes of Project Success and Failure**

* **Проект може зайфелитися якщо:**
  + Нема зворотнього відклику від клієнта
  + Недостатня погруженість людей в проект
  + Незавершені реквайрменти
  + Зміна реквайрментів по ходу розробки
  + Помилка в реквайрментах призводить до:
    - Чим більший проект – тим більше помилок
    - Може переробляються дизайн, а це куча уточнень
    - Переписується бекенд
    - І це все треба перетестувати
    - Може змінюватися порядок виконання завдань, що порушує плани і комітменти.
* **Успішний продукт, коли**
  + Команда погружена в продукт
  + Є зворотній відклик з кінечним юзером
  + Чіткі та зрозумілі вимоги до продукту