Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інформаційні системи

Викладач: к.т.н., доц. Саяпіна Інна Олександрівна

План заняття

- 1. Архітектура ПЗ: від чого залежить та для чого потрібна
- 2. Архітектура ПЗ vs Дизайн ПЗ
- 3. Види моделей життєвого циклу ПЗ
- 4. Послідовні (лінійні) моделі життєвого циклу
- 5. Ітеративні моделі життєвого циклу
- 6. Agile практики та методолгії

Архітектура програмного забезпечення

- **Архітектура програми або обчислювальної системи** – це фундаментальна конструкція її структур, тобто викладення її програмних елементів, їх зовнішніх властивостей та встановлених між ними відношень.
- Це загальна картина або загальна структура всієї системи — як все працює разом.

Software Architecture vs Software Design 1/2

Дизайн програмного забезпечення та архітектура програмного забезпечення пов'язані між собою, але вони представляють різні аспекти процесу розробки програмного продукту. Важливо розуміти відмінності :

Software Design

- Дизайн програмного забезпечення — це специфікація архітектури на рівні окремих компонентів і деталей системи.
- Дизайн фокусується на розробці внутрішньої структури, алгоритмів, деталей реалізації та інтерфейсів між компонентами.
- Він стосується вирішення практичних питань, пов'язаних із тим, як кожен окремий компонент буде реалізований і як вони взаємодіятимуть один з одним.
- Дизайн зазвичай виконується після формування архітектурної основи, коли розробники зосереджуються на роботі над деталями кожного компонента.

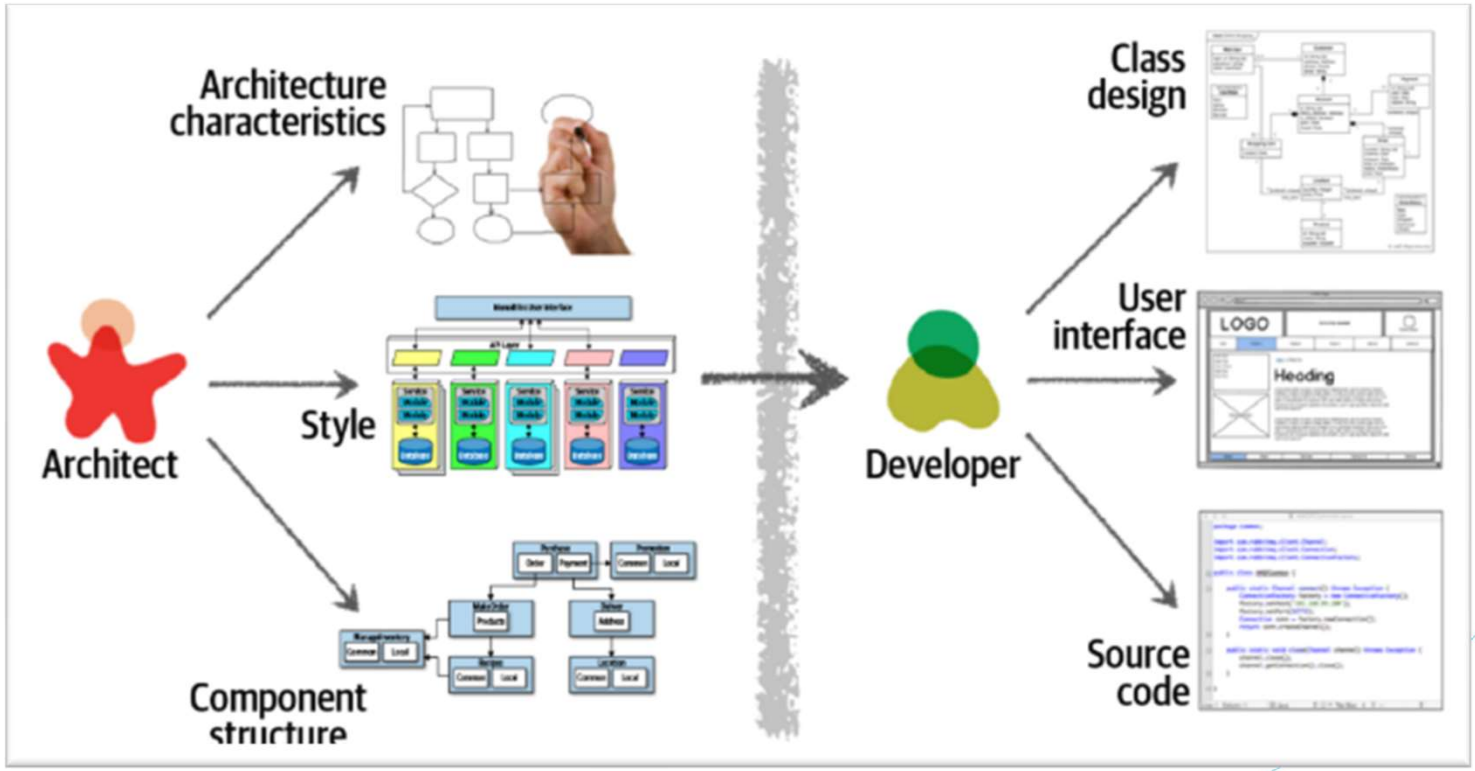
Software Architecture

- Архітектура програмного забезпечення — це високорівневий план або стратегія створення програмного продукту. Він визначає основні компоненти системи, їх взаємодію та структуру програмного продукту в цілому.
- Архітектура зазвичай визначається на ранніх стадіях розробки, до початку детального програмування.
- Вона фокусується на глобальних рішеннях, таких як вибір платформи, основних компонентів і підходів до взаємодії між ними.
- Архітектурний проєкт визначає структуру програмного продукту, яка служить основою для подальшого впровадження та розвитку.

Software Architecture vs Software Design 2/2

Software Architecture

Software Design



Навіщо потрібна архітектура ПЗ?

- **Деякі переваги архітектури програмного забезпечення:**
- • вища продуктивність команди розробників програмного забезпечення: *чітко визначена структура допомагає координувати роботу, впроваджувати індивідуальні функції або направляти обговорення потенційних проблем*
- • покращена еволюція програмного забезпечення, *оскільки принципи проєктування застосовуються для полегшення внесення змін або легшого пошуку дефектів*
- • підвищення якості програмного забезпечення шляхом ретельного врахування потреб і перспектив усіх зацікавлених сторін
- **Архітектура програмного забезпечення допомагає полегшити:**
- процес обслуговування,
- повторне використання та
- адаптацію системи.

Процес розробки ПЗ

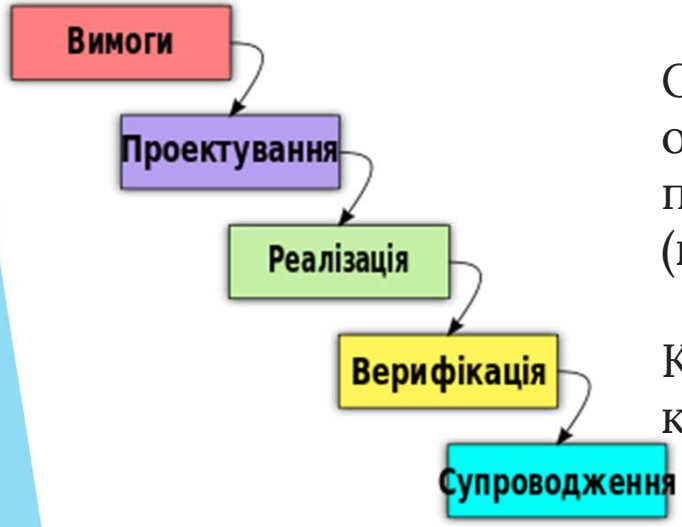
- Спроба закріпити знання про інженерію програмного забезпечення більш ніж 40 років тому отримала результат у вигляді Software Engineering Body of Knowledge” (SWEBOK). У SWEBOK стандартизовані основні формулювання, програмна інженерія представлена як інженерна дисципліна. Мета — створити загальноприйнятий консенсус щодо змісту та концепцій програмної інженерії.
- Досвід, отриманий під час виконання проєктів розробки програмного забезпечення за останні роки, чітко показує, що не може бути єдиного унікального процесу розробки програмного забезпечення: проєкти сильно відрізняються за важливістю, розміром, сферою застосування та середовищем проекту. Натомість докладаються зусилля, щоб зібрати колекцію концепцій, найкращих практик та інструментів, які дозволяють враховувати специфічні вимоги проекту в окремому процесі.

Модель життєвого циклу ПЗ

- *Модель життєвого циклу* – це схема виконання робіт і задач у рамках процесів, що забезпечують розробку, експлуатацію і супровід програмного продукту. Ця схема відображає еволюцію ПЗ, починаючи від формулювання вимог і закінчуючи припиненням користування ним.
- За великим рахунком всі моделі можна розділити на дві великі групи: **послідовні (лінійні)** та **ітераційні** моделі.

Послідовні моделі

- **Модель лінійного процесу** слідує шаблону, у якому фази завершуються одна за одною. У цьому випадку після завершення попередньої фази він переходить до наступної фази, повернутися до попередньої фази неможливо. Модель лінійного процесу підходить для тих проєктів, які мають мало або взагалі не мають поправок чи уточнень.



Waterfall (каскадна) модель

Основна суть моделі Waterfall у тому, що етапи залежать один від одного і наступний починається, коли завершений попередній, утворюючи таким чином поступальний (каскадний) рух уперед.

Команди різних етапів між собою не комунікують, кожна команда відповідає чітко за свій етап.

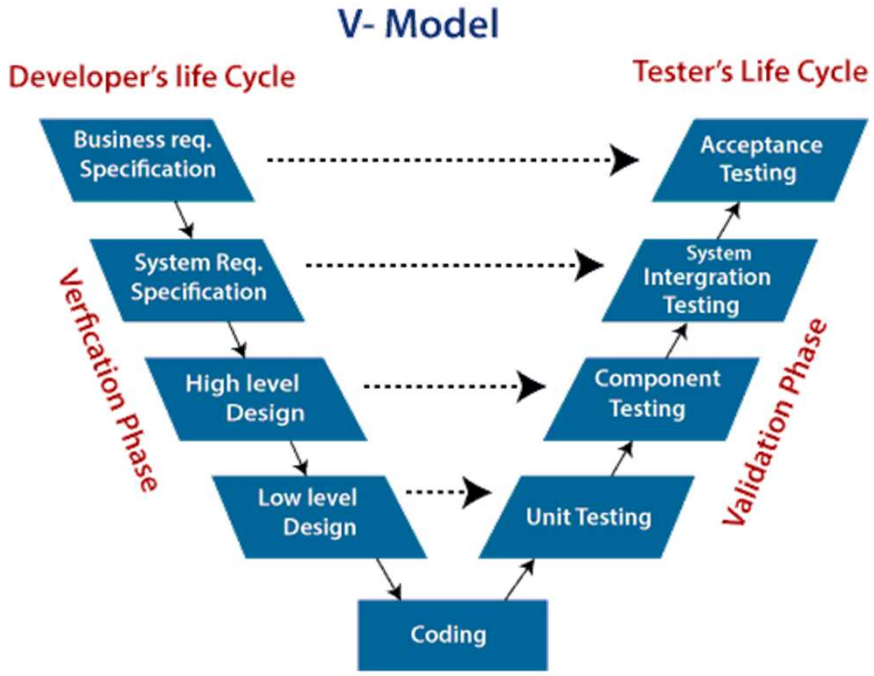
Waterfall (каскадна) модель

- **Недоліки :**
- – процес створення ПЗ не завжди вкладається в таку жорстку форму і послідовність дій;
- – не гнучкий: не враховуються змінювані потреби користувачів, нестабільні умови зовнішнього середовища, коли зафіксували вимоги у ТЗ, їх не можна порушити;
- – значний розрив між часом внесення помилки (наприклад, на процесі проєктування) і часом її виявлення (при супроводі), що призводить до суттєвої переробки ПЗ.

- **Переваги :**
- – прогнозованість (зрозуміло хто в якій послідовності що виконує, скільки на що часу виділено, легко працювати, бо чітко все видно);
- – повністю розроблену систему з документацією на неї легше супроводжувати, тестувати, фіксувати помилки і вносити зміни не хаотично, а цілеспрямовано.

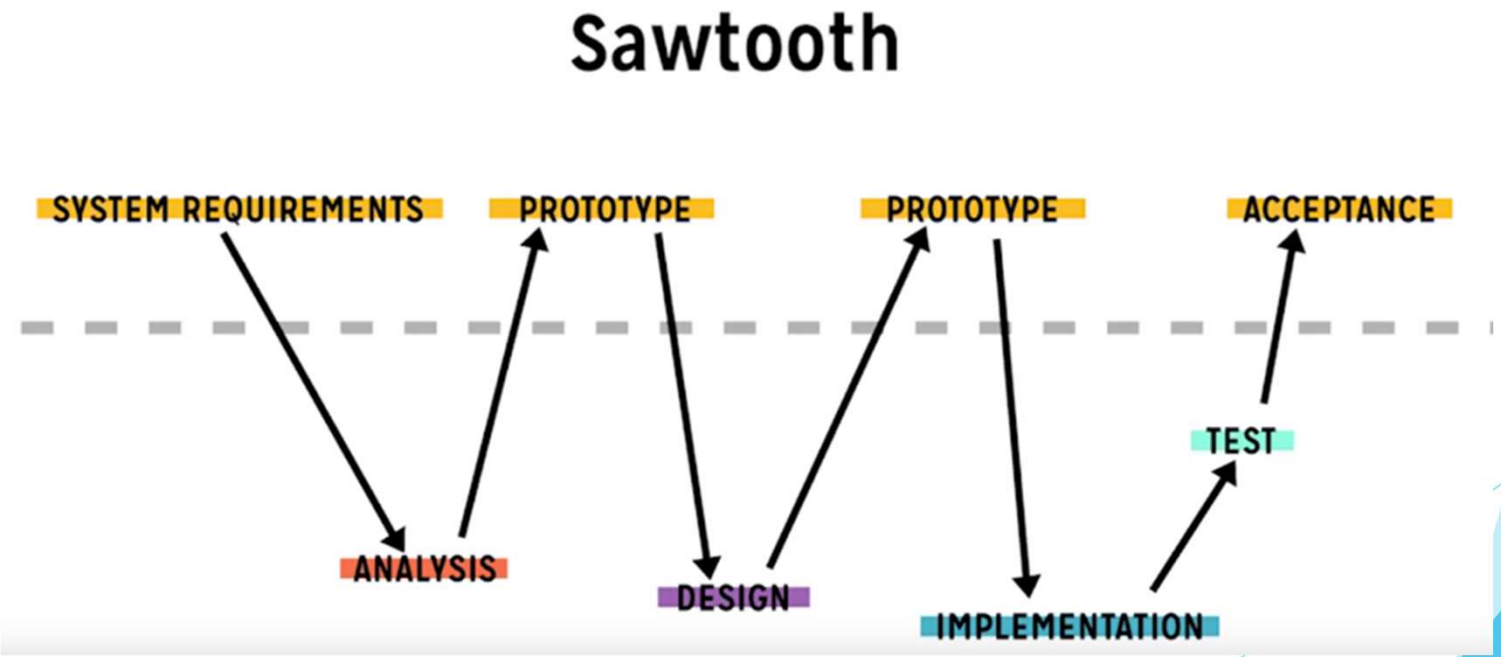
V-модель

- V-модель - це покращена версія класичної каскадної моделі. Тут на кожному етапі відбувається контроль поточного процесу, щоб переконається у можливості переходу до наступного рівня. У цій моделі тестування починається ще зі стадії написання вимог, причому для кожного наступного етапу передбачено свій рівень тестового покриття.



Пилоподібна (Sawtooth) модель

- Це ще одна лінійна модель. У цій моделі розрізняються завдання, які потребують присутності клієнта, і завдання, які потребують лише команди розробників. На відміну від попередніх моделей, тут клієнт задіяний до перевірки проміжних результатів (Prototype), тому зворотній зв’язок можна отримати у відповідний час.

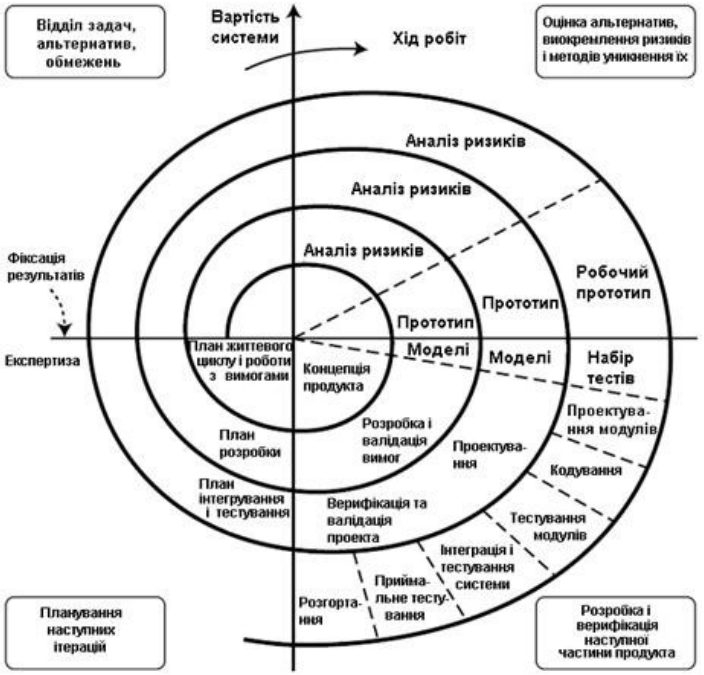


Ітераційні моделі

- **Ітераційна модель** передбачає розбиття проєкту на частини (етапи, ітерації) і проходження етапів життєвого циклу на кожному з них. Кожен етап є закінченим сам по собі, сукупність етапів формує кінцевий результат.
- З кожним етапом розробка наближається до кінцевого бажаного результату або уточнюються вимоги до результату по ходу розробки, і відповідно в будь-який момент поточна ітерація може виявитися останньою або черговою на шляху до завершення.
- Даний підхід дозволяє боротися з невизначеністю, знімаючи її етап за етапом, і перевіряти правильність технічного, маркетингового або будь-якого іншого рішення на ранніх стадіях. Він також дає можливість завершення розробки в кінці будь-якої ітерації (в каскадній моделі ви повинні завершити всі етапи).

Спіральна модель

- Усі етапи життєвого циклу при спіральної моделі йдуть витками, на кожному з яких відбуваються проєктування, кодування, впровадження, тестування і т. д. Такий процес відображає суть назви: піднімаючись, проходиться один виток (цикл) спіралі для досягнення кінцевого результату. Причому не обов'язково, що один і той же набір процесів буде повторяться від витка до витка. Але результати кожного з витків ведуть до головної мети.



Інкрементна модель

- Цю модель (incremental) ще називають моделлю з нарощуванням або з приростом. Її суть полягає в розробці продукту ітераціями, і кожна ітерація закінчується випуском працездатної версії. У кожній новій версії додаються деякі функціональні можливості.
- Кожна ітерація позначається цифрою: 1,2,3 і відповідно продукт після кожної ітерації має версію з відповідним номером: v.1, v.2, v.3. Числами після слова версія позначають масштабні зміни в ядрі продукту.

Agile

- Ітеративні практики, що базуються на основі *Manifesto for Agile Software Development*, називаються **практиками Agile** та відповідають принципам *Agile*. Практики *Agile*, організовані у методології, називаються **методологіями *Agile***. Можна виділити три найпопулярніші методології: *SCRUM, Extreme Programming та Lean*.

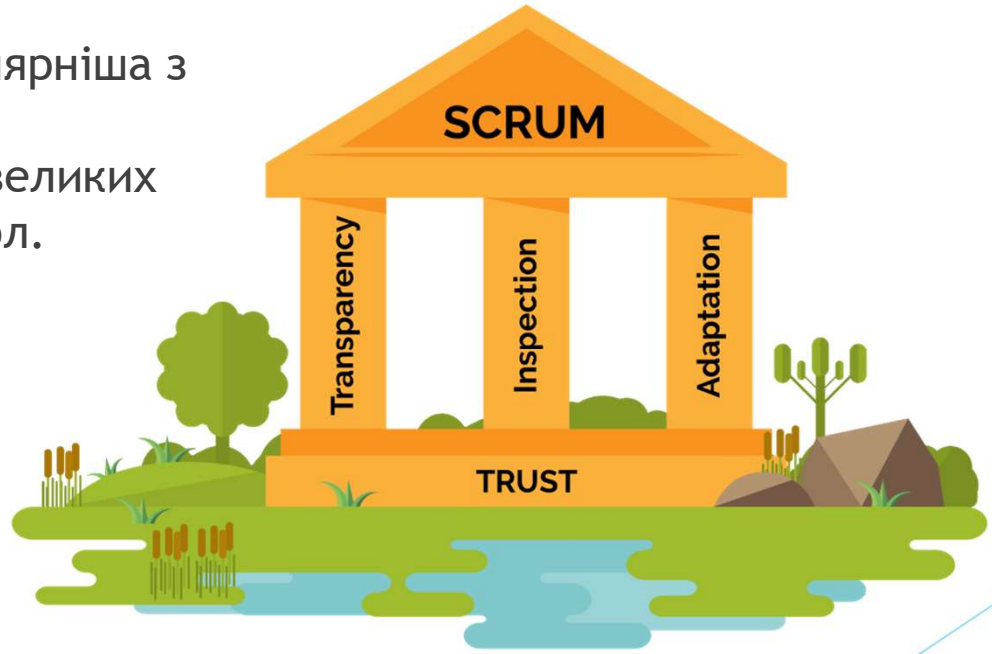
- ***Основні принципи Manifesto for Agile Software Development***
- *«Ми розкриваємо кращі способи розробки програмного забезпечення, роблячи так та допомагаючи так робити іншим. Завдяки цій роботі ми повинні прийти до наступних цінностей:*
- люди і взаємодія важливіші за процеси та інструменти;
- працюючий продукт важливіший за вичерпну документацію;
- співпраця з замовником важливіше узгодження умов контракту;
- готовність до змін важливіше проходження затвердженим планом»

SCRUM

- SCRUM – це методологія Agile, що складається з легких управлінських практик, які мають відносно невеликі накладні витрати. Ці практики прості для розуміння, але дуже складні для опанування в повному обсязі. SCRUM застосовує підхід, що є водночас ітеративним та інкрементним. Передбачає часте оцінювання проєкту, що підвищує передбачуваність і зменшує ризики.

- Scrum – найпопулярніша з Agile.
  Для відносно невеликих проєктів до 30 чол.

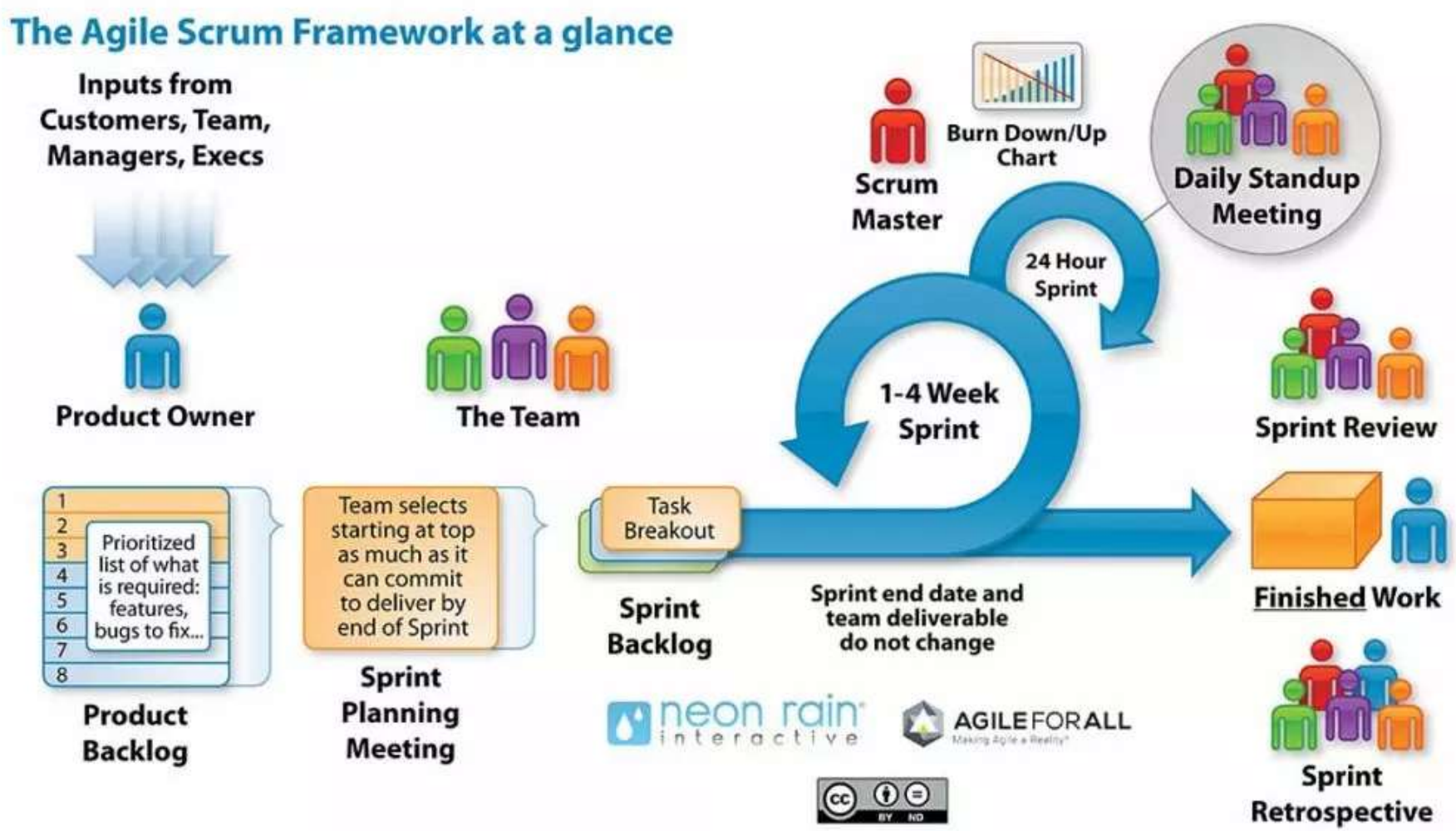
Основні цінності:
- Прозорість
- Інспекція
- Адаптація



Ролі у SCRUM

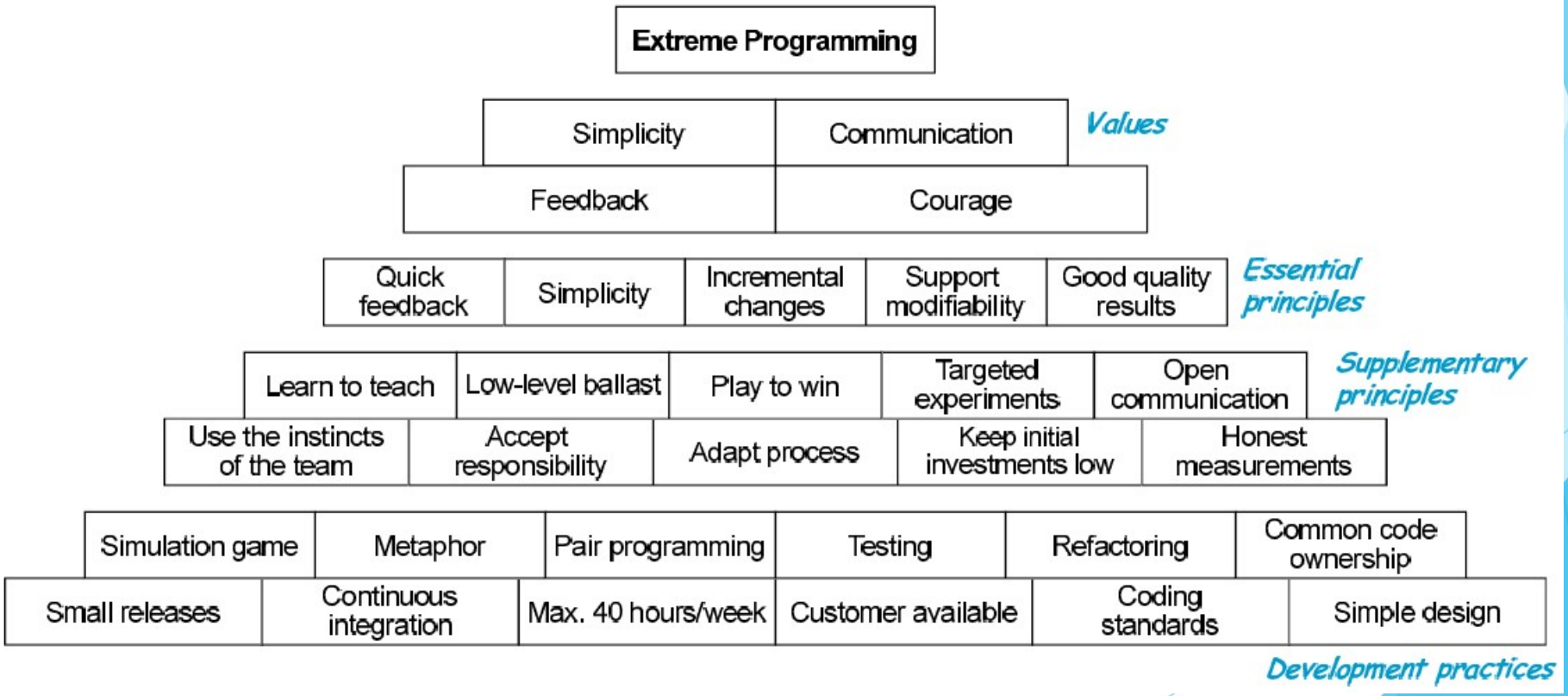
- **Власник продукту (Product owner)** – це єдина особа, яка відповідає за продукт. Усі вимоги до продукту для розробників мають проходити через цю роль. Список вимог до продукту зібрано в беклозі (backlog – список завдань) продукту, за який відповідає власник продукту. Власник продукту також визначає пріоритети цих вимог і забезпечує чітке визначення вимог.
- **Скрам-майстер (Scrum master)** повинен забезпечити, щоб команда scrum дотримувалася практик scrum, допомагаючи власнику продукту та команді розробників. Для власника продукту ця допомога включає в себе пропозицію методів управління беклогом продукту, досягнення чітких вимог і визначення пріоритетів для досягнення максимальної цінності. Для членів команди розробників ця допомога включає навчання команди самоорганізації та усуненню перешкод.
- Розробники в **команді scrum** повинні бути самоорганізованими. Команди Scrum-розробників невеликі, в ідеалі від трьох до дев’яти осіб. Вони самодостатні, складаються з усіх спеціалістів, хто потрібен для завершення продукту. Відповідальність лежить на всій команді.

Практики SCRUM



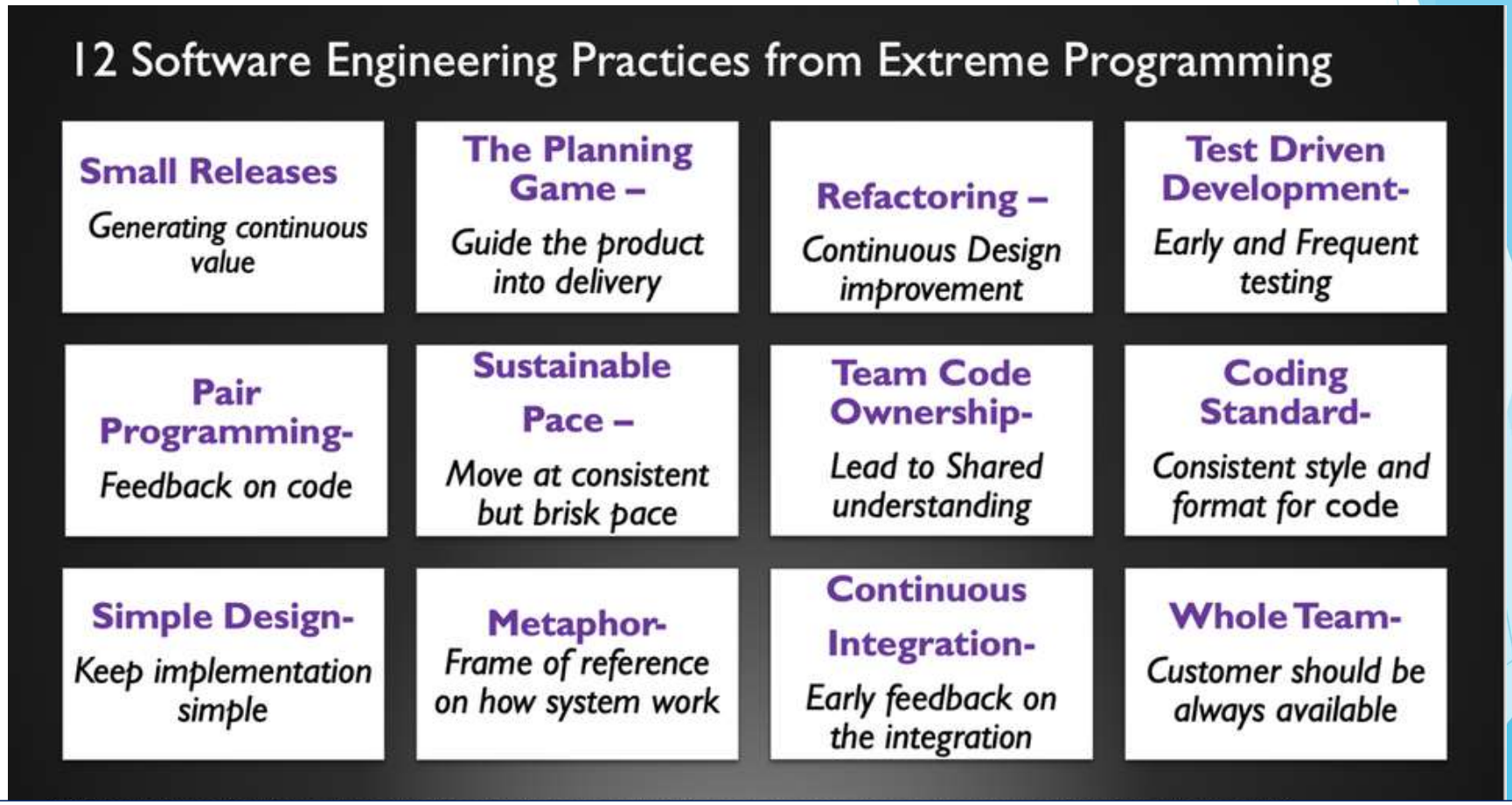
Extreme Programming (XP)

- Це методологія, орієнтована на поліпшення якості програмного забезпечення та чутливість до змін у вимогах замовників. Пропонує 12 підходів (Development practices) для досягнення ефективних результатів у подібних умовах.



B. Rumpe. Agile Modeling with UML. Springer, 2017, 388 p. DOI 10.1007/978-3-319-58862-9

Практики Extreme Programming (XP)



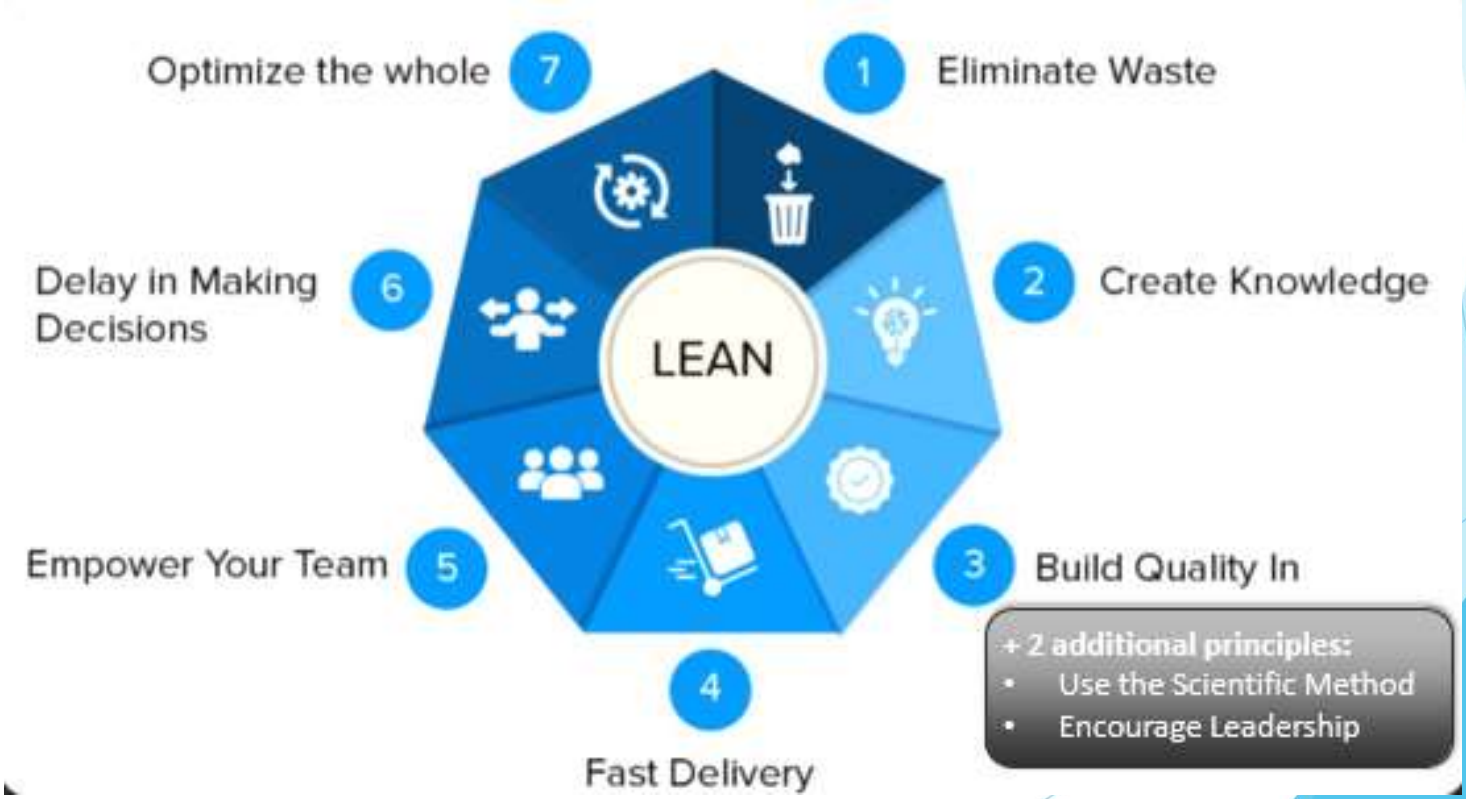
https://www.facebook.com/103027077852925/posts/extreme-programming-xp-is-an-agile-software-development-methodology-used-to-impl/162797361875896/

Kanban

- Концепція менеджменту, що була створена на японському підприємстві Toyota і заснована на неухильному прагненні до усунення всіх видів втрат та підвищення якості.
- - Підходить для проєктів з постійно виникаючими терміновими задачами, які потрібно брати у роботу.
  Добре підходить для проєктів по підтримці (maintenance) продукту, коли знаходять баги і основний scope задач не дуже визначений.

Agile-практики (Lean)

- Концепція менеджменту, що була створена на японському підприємстві Toyota і заснована на неухильному прагненні до усунення всіх видів втрат та підвищення якості.



Agile-практики (Lean)

- Усуньте зайве. Подумайте про потенційну втрату часу та зусиль, що можуть виникнути через нечіткі вимоги, вузькі місця процесів, дефекти продуктів та непотрібні зустрічі. Зайве також виникає через те, що «бути зайнятим» не означає бути продуктивним. Кодування може бути надлишковим, якщо воно не передбачає релізу. Все, що не додає цінності продукту, визначається як зайве та має бути ідентифіковано та видалено.
- Створюйте знання. Визначте усі ідеї перед тим, як діяти. Не фокусуйтесь лише на одній ідеї, поки повністю не познайомитися з іншими можливостями. Пошукайте найкращий варіант. Різнобічна оцінка генерує альтернативні підходи і чим їх більше, тим вище якість рішень.
- Створюйте з якістю. Ставте за мету створювати якісний продукт. Способи побудувати якісне ПЗ включають відслідковування відгуків про роботу продукту, підготовка та проведенні автоматизованих тестів, рефакторинг коду для спрощення його побудови, забезпечення корисною документацією та змістовні коментарі до коду. Розробка якісного рішення зменшує час розробникам на виправлення дефектів. Активне використання цього принципу дозволяє команді розробників економити час та зусилля.

Agile-практики (Lean)

- Швидкий випуск. Є зв’язок між принципами Lean. Усунення зайвого вимагає обдумування що ти робиш. Тому ти маєш посилювати навчання. Посилення навчання вимагає часу на оцінку альтернатив, тому ти повинен прийняти рішення якомога пізніше. Швидкий випуск призначений для розвитку продукту через серію швидких ітерацій. Кожен реліз може сконцентруватися на ключових характеристиках продукту, тому час та зусилля не тратяться на неважливі характеристики. Часті релізи дають можливість клієнту дати зворотній зв’язок для майбутнього удосконалення.
- Підсилюйте свою команду. Lean спонукає менеджерів дослухатись до розробників замість того, щоб говорити як їм треба робити їх роботу. Ефективний менеджер надає розробникам вирішувати як створювати ПЗ
- Затримка у прийнятті рішень. Приймайте рішення тільки після наявності усієї інформації. Прийняття рішення якомога пізніше дозволяє дослідити багато альтернатив і обрати найкраще рішення, засноване на наявній інформації. Передчасні рішення жертвують потенціалом для кращого рішення та «правильного продукту»
- Оптимізуйте цілісно. Основна увага на досвід кінцевого користувача. Якісний продукт зв’язаний та добре спроєктований у цілому. Індивідуальні складові повинні доповнювати весь досвід користувача.