**Лекція 4. Поняття життєвого циклу програмного забезпечення**

**Життєвий цикл ПЗ** - це період часу, який починається з моменту ухвалення рішення про необхідність створення програмного забезпечення й закінчується в момент його повного вилучення з експлуатації (ІEEE Std. 610.12 - 19990 Standard Glossary of Software Engіneerіng Termіnology).

Програми з малою тривалістю життєвого циклу створюються для разового розв'язку наукових і інших завдань. Їхній життєвий цикл - від декількох днів до декількох місяців. Раніше такі програми не мали зручного інтерфейсу, тому що витрати на його розробку ще недавно в кілька раз перевищували витрати на розробку обчислювальної частини. Схематично ЖЦ програмного продукту (далі – ПП) показаний на рис.1:

Кожна програма починається з якої-небудь незадовільненої потреби й, усвідомивши її, необхідно провести **системний аналіз** для виявлення цілей майбутнього програмного виробу й вимог до нього. Наступним етапом буде зовнішнє **специфікування**, призначене для створення "ідеології" програми - загальної спрямованості в наступному проектуванні, аж до зовнішнього вигляду програми й інструкції користування програмою. На етапі **проектування** програмний виріб специфікується в повному обсязі від постановки завдання до робочого проекту з описом внутрішньої структури програми й плану розробки частин програми. Потім відбувається **кодування й тестування**, у результаті чого виходить готова версія програми. Програма випускається в тираж і супроводжується виробником. **Супровід** полягає як в усуненні помилок, що виявляються в процесі експлуатації, і випуску виправлених версій, так і в удосконаленні базової версії програми, що найчастіше призводить до перепроектування програми й випуску радикально оновлених версій.

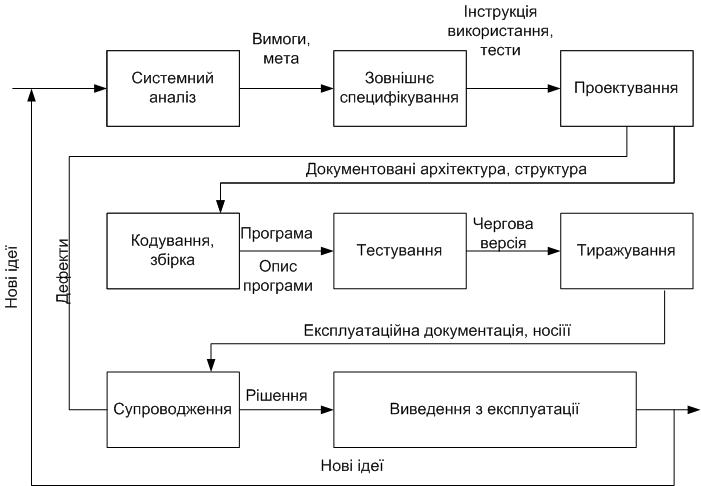
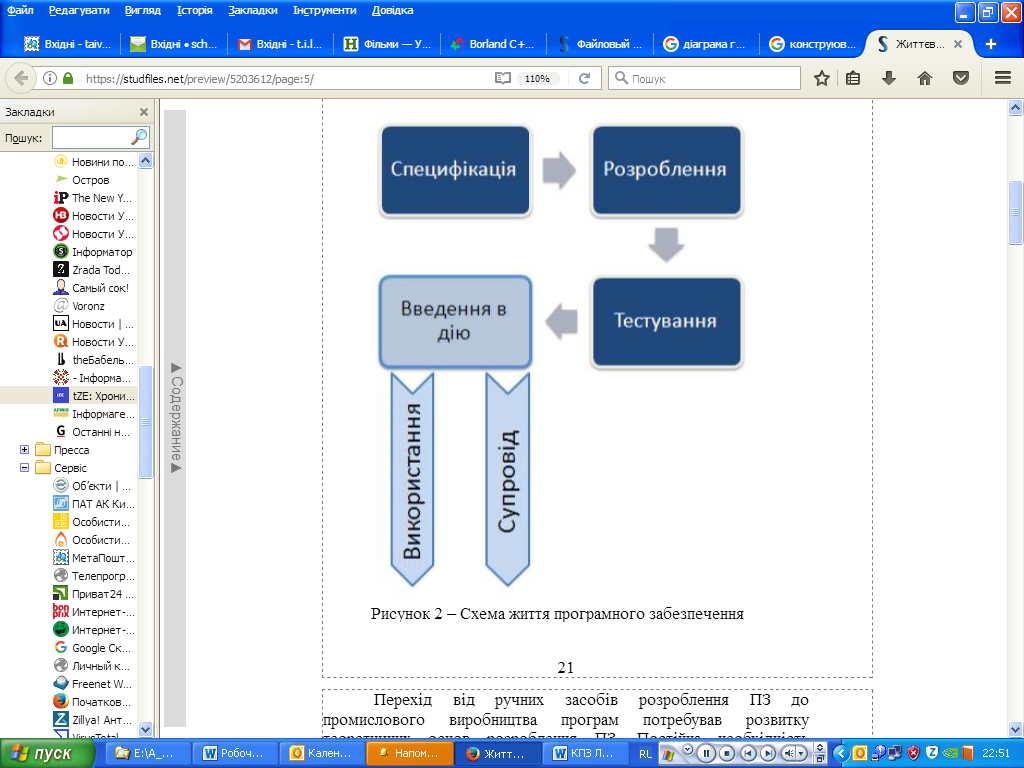


Рисунок 1 – Життєвий цикл програмного продукту

Закінчення життєвого циклу – це **припинення експлуатації** розробки. Однак ідеї, висунуті в процесі експлуатації програми, звичайно використовуються при розробці наступного, більш досконалого й сучасного виробу.

Припинення експлуатації - звичайно не одномоментний акт знищення програми в комп'ютері, а період часу, коли деякі організації або деякі користувачі ще продовжують використовувати стару розробку.

У процесі створення програмного забезпечення (далі –ПЗ) можна виділити 4 базових етапи/стадії (рис.2):

Рисунок 2 – Схема життя ПЗ

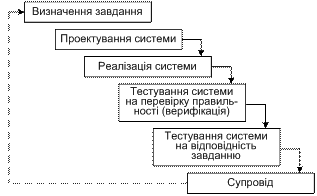
**Специфікація** – визначення основних вимог.

**Розроблення** – створення ПЗ відповідно до специфікацій.

**Тестування** – перевірка ПЗ на відповідність вимогам клієнта.

**Супровід/Модернізація** – розвиток ПЗ відповідно до змін потреб замовника.

**Модель життєвого циклу** - це структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту; охоплює життя системи від визначення вимог до неї до припинення її використання. На сьогодні найбільшого розповсюдження набули дві: каскадна модель та спіральна модель.

[](http://fitm.nusta.edu.ua/mediawiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Pii03.gif)**Каскадна модель**

Однією з перших з’явилась каскадна модель, в якій кожен етап роботи виконується лише раз. На кожному етапі робота виконується настільки ретельно, щоб потреби повертатись до попереднього не виникало. Результат виконання кожного етапу, перед передачею в наступний, піддається верифікації.

Рисунок 3 - Каскадна модель життєвого циклу (далі – ЖЦ) програмних систем

**Ітеративна та інкрементна модель**

Ітеративна, що отримала також від Т. Гілба в 70-і рр. назву еволюційної моделі (IID - ітеративна модель та інкрементна модель). Модель IID передбачає розбиття ЖЦ проекту на послідовність ітерацій, кожна з яких нагадує "міні-проект", включаючи всі процеси розробки в застосуванні до створення менших фрагментів функціональності, порівняно з проектом в цілому. Мета кожної ітерації — отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом всіх попередніх та поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту.

Таким чином, із завершенням кожної ітерації продукт отримує приріст — інкремент — до його можливостей, які, значить, розвиваються еволюційно. З точки зору структури ЖЦ таку модель називають ітеративною. З точки зору розвитку продукту — інкрементальною. Досвід індустрії показує, що неможливо розглядати кожен з цих поглядів ізольовано.

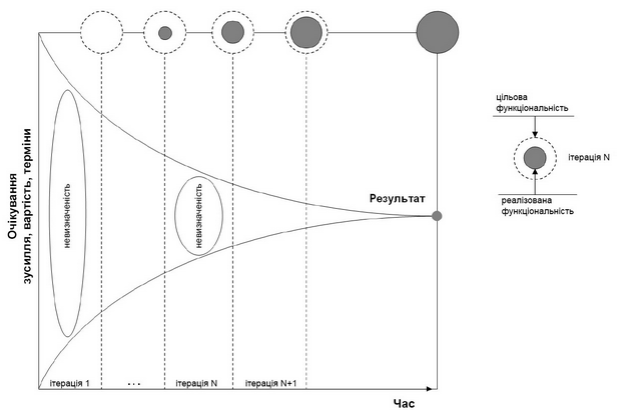


Рисунок 4 - Ітеративна та інкрементна модель

При такому підході кожна версія повинна бути придатною для використання і забезпечувати деяку підмножину необхідних функцій. До недоліків відноситься збільшення обсягу робіт по тестуванню, оскільки з появою кожної нової версії продукту потрібне повторне підтверджуюче тестування, що додані можливості не погіршили якість програмного виробу.

Кожна лінійна послідовність виробляє працюючий інкремент ПЗ. Перший інкремент приводить до отримання базового продукту, який реалізує базові вимоги (більшість додаткових вимог залишаються нереалізованими). План наступного інкременту передбачає модифікацію базового продукту, яка забезпечить додаткові характеристики і функціональність.

Сучасна реалізація інкрементного підходу – екстремальне програмування XP. Воно орієнтоване на дуже невеликі прирости функціональності.

**Спіральна модель**

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання - щонайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

Виходячи з можливості внесення змін, як в процес, так і в проміжний продукт було створено спіральну модель ЖЦ.

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації не відповідають дійсному стану розробки.

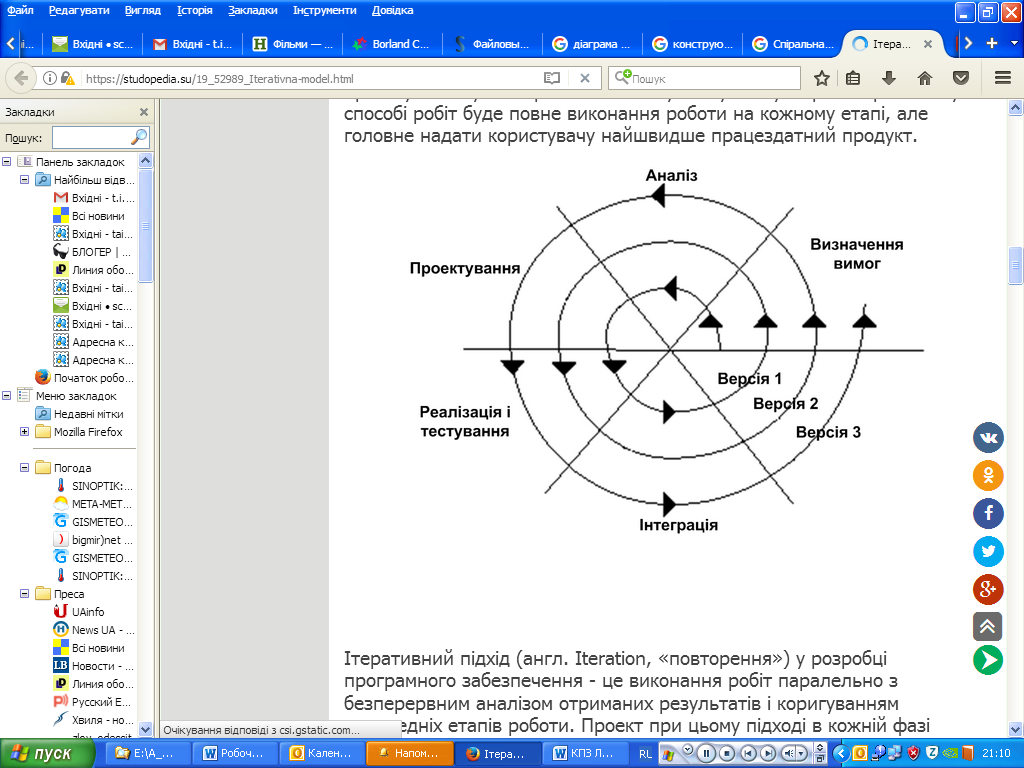


Рисунок 5 - Спіральна модель ЖЦ розробки програмних систем

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку правильності та прийняття рішення про перехід на наступний виток або повернення на попередній виток для доопрацювання на ньому проміжного продукту.

Відмінність цієї моделі від каскадної полягає в можливості багато разів повертатися до процесу формулювання вимог і до повторної розробки версії системи з будь-якого процесу моделі.

Для програмного продукту така модель не дуже підходить з декількох причин. По-перше, висловлення вимог замовником носить суб'єктивний характер, вимоги можуть багаторазово уточнюватися протягом розробки ПС і навіть після завершення та випробовування, і часом може з'ясуватися, що замовник «хотів зовсім інше». По-друге, змінюються обставини та умови використання системи, тому загальновизнаним законом програмної інженерії є закон еволюції, який сформулюємо так: кожна діюча ПС з часом потребує внесення змін або виводиться з експлуатації.

При необхідності внесення змін до системи на кожному витку з метою отримання нової версії системи обов'язково вносяться зміни в заздалегідь зафіксовані вимоги, після чого повертаються на попередній виток спіралі для продовження реалізації нової версії системи з урахуванням усіх змін.

### Еволюційна модель

У разі еволюційної моделі система послідовно розробляється з блоків конструкцій. На відміну від інкрементної моделі в еволюційній моделі вимоги встановлюються частково і уточнюються в кожному наступному проміжному блоці структури системи.

Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб її замовника і аналізу можливості застосування цієї моделі для реалізації. Модель використовується для розробки нескладних і некритичних систем, де головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені відразу і повністю. Тому розробка системи здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи–прототипу, на якому перевіряється реалізація вимог. Іншими словами, такий процес за своєю суттю є ітераційним, з етапами розробки, що повторюються, починаючи від змінених вимог і закінчуючи отриманням готового продукту. В деякому розумінні до цього типу моделі можна віднести спіральну модель.

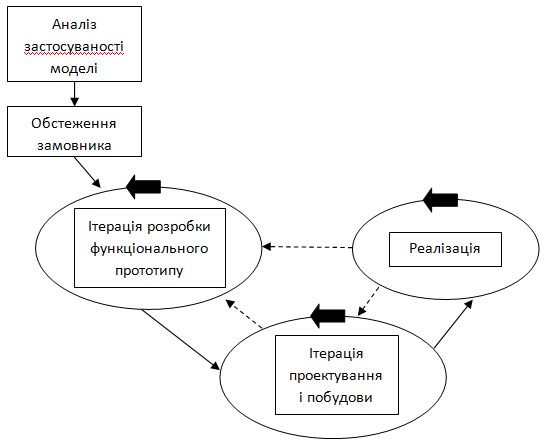


Рисунок 5 - Модель еволюційного прототипування

Розвитком цієї моделі є **модель еволюційного прототипування (макетування)** в рамках усього ЖЦ розробки ПС. У літературі вона часто називається моделлю швидкої розробки програм RAD (Rapid Application Development).

У даній моделі наведені дії, які пов'язані з аналізом її застосовності для конкретного виду системи, а також обстеженням замовника для визначення потреб користувача при розробці плану створення прототипу.

У моделі є дві головні ітерації розробки функціонального прототипу, проектування і реалізації системи з метою перевірки, чи задовольняє вона всі функціональні і нефункціональні вимоги. Основною ідеєю цієї моделі є моделювання окремих функцій системи в прототипі і поступове еволюційне його доопрацювання до виконання всіх заданих функціональних вимог.

Ітерацій з отримання проміжних варіантів прототипу може бути декілька, в кожній з яких додається функція і повторно моделюється робота прототипу. І так до тих пір, поки не будуть промодельовані всі функції, задані у вимогах до системи. Після цього виконується ще одна ітерація – остаточне програмування для отримання готової системи.

Ця модель застосовується для систем, в яких найбільш важливими є функціональні можливості, і які необхідно швидко продемонструвати на CASE-засобах. Оскільки проміжні прототипи системи відповідають реалізації деяких функціональних вимог, їх можна перевіряти і під час супроводу і експлуатації, тобто разом з процесом розробки чергових прототипів системи. При цьому допоміжні і організаційні процеси можуть виконуватися разом з процесом розроблення і накопичувати відомості за даними кількісних і якісних оцінок на процесах розроблення.

При цьому враховуються такі чинники ризику:

* реалізація всіх функцій системи одночасно може призвести до громіздкості;
* обмежені людські ресурси зайняті розробкою протягом тривалого часу;
* або замовник, або розробник може прийняти макет за готовий програмний продукт.

Переваги застосування даної моделі ЖЦ такі:

* швидка реалізація деяких функціональних можливостей системи і їх апробація;
* використання проміжного продукту в наступному прототипі;
* виділення окремих функціональних частин для реалізації їх у вигляді прототипу;
* можливість збільшення фінансування системи;
* зворотний зв'язок із замовником для уточнення функціональних вимог;
* спрощення внесення змін у зв'язку із заміною окремих функції;
* забезпечує визначення повних вимог до ПЗ.

**V-подібна модель**

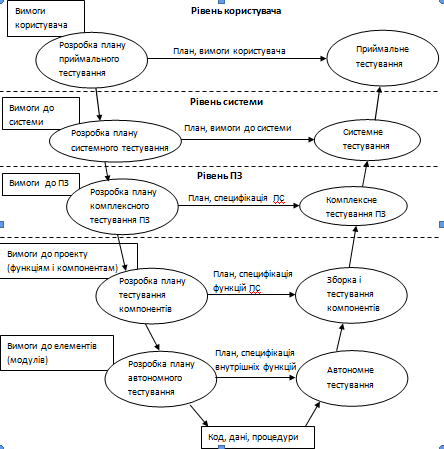


Рисунок 6 - V-подібна модель

В цій моделі тестування розглядається як неперервний процес, інтегрований в процес розробки ПС. Він включає два взаємопов’язаних підпроцеса:

* планування тестування в рамках процесів розробки системи (ліва гілка)
* проведення тестування відповідних об’єктів (права гілка)

**Характеристики V-подібної моделі**:

* Перевірка і оцінка тестопридатності вимог на ранніх стадіях розробки (з допомогою аналізу, який виконується під час тестування)
* Наявність документованих тестових вимог

**Переваги V-подібної моделі**:

* Забезпечує зворотний зв’язок з користувачем на ранніх стадіях ЖЦ
* Покращує планування і розподіл затрат на тестування
* Чіткі документовані цілі тестування

## Адаптивні моделі

* Scrum;
* XP (Extreme Programming);
* Adaptive Software development (ASD);
* Dynamic System Development Model (DSDM);
* Feature Driven Development (FDD)

## Scrum

Scrum – один із основних підходів до гнучкої розробки ПЗ. Був розроблений Джефом Сазерлендом та Кеном Швабером в 1994 році.

Модель Scrum включає три стадії:

* підготовка;
* розробка запланованих функціональних можливостей;
* завершення.

Стадія *підготовки* ітерації включає в себе дві під стадії:

1. планування: розробка нового Product Log/BackLog (список основних вимог до продукту, які мають бути реалізовані протягом виконання проекту), визначення дати кінцевого релізу та функціоналу для однієї або декількох ітерацій; вибір початкової ітерації; визначення команди розробників для ітерації; визначення можливих ризиків та засобів їх контролю; перегляд та можливе коректування Product Log; вибір та валідація інструментів та технології для розробки; визначення бюджету релізу, включаючи розробку, маркетинг, тренінги; узгодження з керівництвом бюджету та функціонального вмісту релізу.
2. високорівневий дизайн: перегляд зазначених в Product Log специфікацій; визначення необхідних змін в специфікаціях для імплементації нових функціональних можливостей програмного продукту; проведення аналізу системи для виділення необхідних дій для створення, розширення чи оновлення існуючого ПЗ; уточнення архітектури системи для підтримки запланованих в Product Log специфікацій; виявлення проблем та питань, що можуть виникнути в процесі розробки; створення плану імплементації для кожної специфікації Product Log, обговорення та затвердження цих планів командою розробників.

*Розробка* – ітеративний процес розробки запланованих функціональних можливостей. Керівництво проекту визначає час, функціональне наповнення, якість кожної ітерації. По завершенню кожної ітерації відбувається аналіз отриманих результатів. Розробка включає наступні процеси:

* періодичне обговорення отриманих результатів та планів наступного релізу;
* перегляд та коригування стандартів, яким буде відповідати розроблений продукт;
* ітеративні спринти, поки програмний продукт буде готовий до випуску.

Спринт – сукупність усіх завдань, що будуть виконувати розробники протягом визначеного періоду (1-4 тижні). Довжина спринту залежить від складності продукту, оцінених ризиків. Кожен спринт включає всі традиційні фази розробки ПЗ: створення специфікації, аналіз, дизайн, розробка, тестування та демонстрація розроблених функціональних можливостей. Довжина спринту має бути фіксованою (30 днів).

*Стадія завершення* в Scrum містить усі необхідні завдання для остаточного випуску системи, після того, як закінчилася розробка усіх запланованих функціональних можливостей. Такі процеси як: системне тестування, написання документації, підготовка тренінгів.

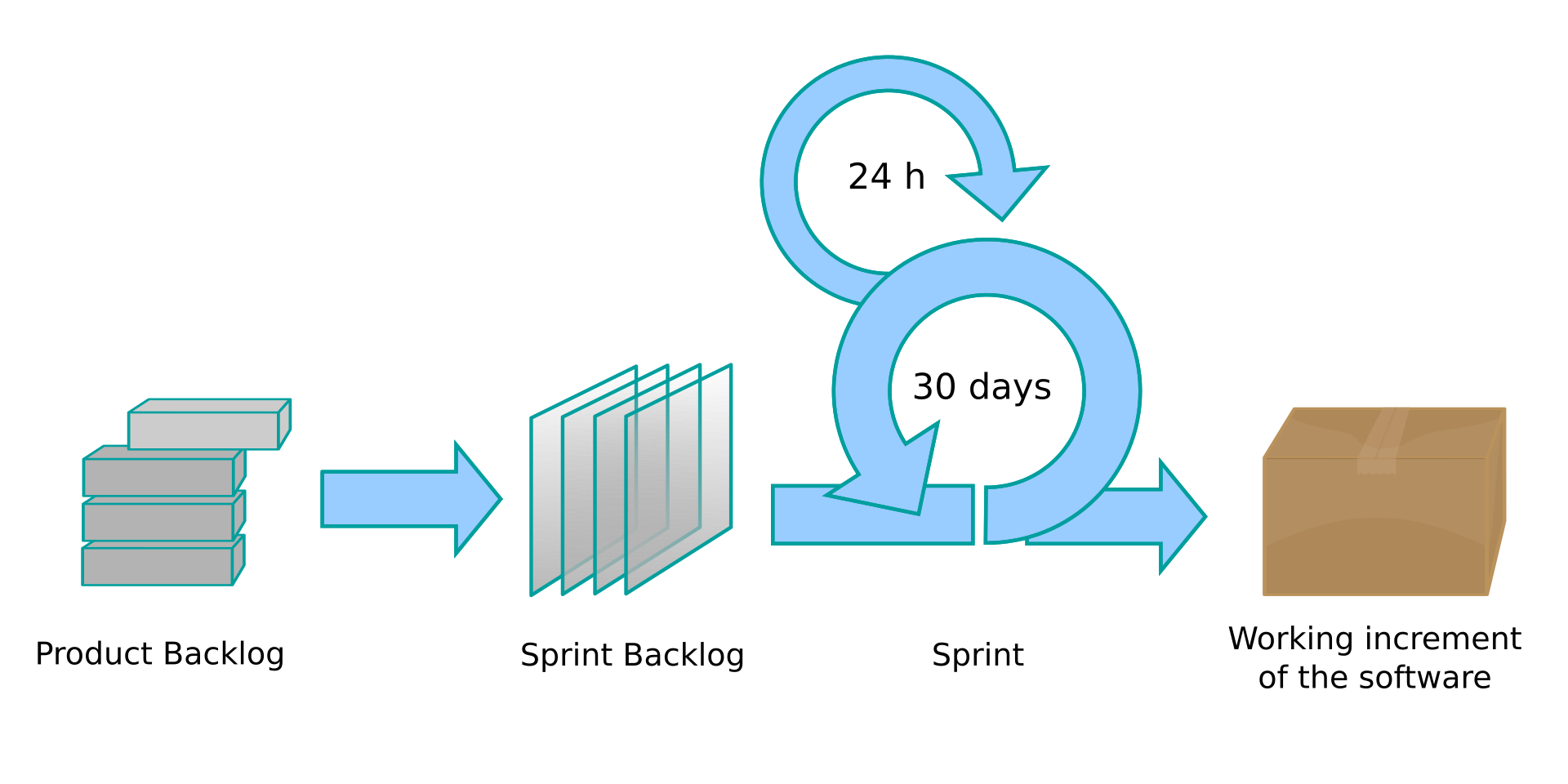


Рисунок 7 – Scrum процеси

Основні переваги моделі Scrum:

1. адаптивність до потреб ринку (додавання нових вимог/ зміна існуючих не призводить до серйозних змін в проекті);
2. короткі ітерації. В кінці кожної ітерації на виході готовий продукт, який може бути доставлений замовникам на перевірку/ нової версії продукту на ринок;
3. висока ефективність команд. Підвищується відповідальність кожного програміста за результат його роботи та проекту в цілому;
4. легкість в управління. Модель є прозорою, зрозумілою та легкою. Не потрібно використовувати важких інструментальних засобів для управління проектом.

Недоліки:

1. обмеження на розмір команди;
2. обмеження на місцезнаходження команди

## XP-програмування

Екстремальне програмування (eXtreme Programming, XP) – це полегшений (рухомий) процес проектування ПЗ (Кент Бек, 1999р.). XP-процес орієнтований на групи малого і середнього розміру, які проектують ПЗ в умовах невизначеності, або швидкої зміни вимог. XP-групу утворюють до 10 працівників, які розміщуються в одному приміщенні.

Основна ідея XP – зняти високу вартість змін, яка характерна для програм з використанням об’єктів, типових рішень і реляційних БД. XP-група має справу зі змінами вимог на протязі всього ітераційного циклу розробки, причому цикл складається з дуже коротких ітерацій. Чотирма базовими діями в XP-циклі є: кодування, тестування, вислуховування замовника і проектування. Динамізм забезпечується за допомогою чотирьох характеристик: безперервного зв’язку з замовником (і в межах групи), простоти (завжди вибирається мінімальне рішення), швидкого зворотного зв’язку (за допомогою модульного і функціонального тестування), сміливості в проведені профілактики можливих проблем.

Більшість принципів, які підтримуються XP (мінімальність, простота, еволюційний цикл розробки, мала тривалість ітерації, участь замовника, оптимальні стандарти кодування і т.д.) застосовуються в будь-якому впорядкованому процесі. Проте в XP ці принципи досягають екстремальних значень.

Базис ХР утворюють дванадцять методів.

1. Гра планування – швидке визначення області дії наступної реалізації шляхом об’єднання ділових пріоритетів і технічних оцінок. Замовник формує область дії, пріоритетність і терміни з точки зору бізнесу, а розробник оцінює і слідкує за прогресом проектування. Гра планування і часта зміна версій залежать від замовника, який забезпечує набір „історій” (коротких описів), що характеризують роботи, які будуть виконуватися для кожної версії системи. Версії генеруються кожні два тижні, тому розробники і замовник повинні досягнути погодження про те, які історії будуть реалізовані в межах двох тижнів. Повну функціональність характеризує пул історій; але для наступної двохтижневої ітерації з пула вибирається підмножина історій, що найбільш важлива для замовника. В будь-який час в пул можуть бути добавлені нові історії, таким чином, вимоги можуть швидко змінюватися. Проте процеси двохтижневої генерації базуються на найбільш важливих функціях, які входять в поточний пул, значить, зміни стають керованими. Локальний замовник забезпечує підтримку цього стилю ітераційної розробки.

2. Часта зміна версій – швидкий запуск в виробництво простої системи. Нові версії реалізуються в дуже короткому (двохтижневому) циклі.

3. Метафора – вся розробка проводиться на основі простої, загальнодоступної історії про те, як працює вся система. Метафора забезпечує глобальне „бачення” проекту. Вона могла б розглядатися як високорівнева архітектура, але ХР передбачає проектування при мінімумі проектної документації. Точніше кажучи, ХР пропонує неперервне перепроектування (за допомогою реорганізації), при якому немає потреби в деталізованій проектній документації, а для інженерів супроводу єдиним надійним джерелом інформації є програмний код. Часто після написання коду проектна документація викидається. Вона зберігається тільки в тому випадку, коли замовник тимчасово втрачає здатність придумувати нові історії. Тоді систему поміщають в „нафталін” і пишуть інструкцію сторінок на п’ять-десять за „нафталіновим” варіантом системи.

4. Просте проектування – проектування виконується настільки просто, наскільки це можливо в даний момент.

5. Тестування – неперервне написання тестів для модулів, які повинні виконуватися бездоганно; замовники пишуть тести для демонстрації закінченості функцій. „Тестуй, а потім кодуй” означає, що вхідним критерієм для написання коду є тестовий варіант, який дав відмову. Ця фраза виражає акцентування ХР- процесу на тестуванні. Вона відображає принцип, за яким спочатку планується тестування, а тестові варіанти розробляються паралельно аналізу вимог, хоча традиційний підхід полягає в тестуванні „чорного ящика”.

6. Реорганізація – система реструктурується, але її поведінка не змінюється; мета – усунути дублювання, покращити взаємодію, спростити систему або добавити в неї гнучкість. Використання реорганізації приводить до реалізації найпростішого рішення, яке задовольняє поточні потреби. Зміни в вимогах заставляють відмовитися від всіх „загальних рішень”.

7. Парне програмування – весь код пишеться двома програмістами, які працюють на одному комп’ютері. Це один із найбільш суперечливих методів в ХР, він впливає на ресурси, що важливо для менеджерів, які вирішують, чи буде проект використовувати ХР. Може здатися, що парне програмування подвоює ресурси, але дослідження показали: парне програмування приводить до підвищенню якості і зменшення часу циклу. Для узгодженої групи затрати збільшуються на 15%, а час циклу скорочується на 40-50%. Співробітництво покращує процес вирішення проблем, покращення якості суттєво знижує затрати на супровід, які перевищують вартість додаткових ресурсів на всьому циклі розробки.

8. Колективне володіння кодом означає, що будь-який розробник може змінити будь-який фрагмент коду системи в будь-який час.

9. Неперервна інтеграція – система інтегрується і будується багато разів на день, в міру завершення кожної задачі. Неперервне регресивне тестування, тобто повторення попередніх тестів, гарантує, що зміни вимог не приведуть до регресу функціональності. Неперервна інтеграція, неперервне регресивне тестування і парне програмування ХР забезпечують захист від проблем, які виникають при цьому.

10. 40-годинний тиждень – як правило, працюють не більше 40 годин на тиждень. Не можна збільшувати робочий тиждень за рахунок позаурочних робіт.

11. Локальний замовник – в групі весь час повинен знаходитися представник замовника, який дійсно готовий відповідати на питання розробників.

12. Стандарти кодування – повинні витримуватися правила, які забезпечують однакове представлення програмного коду в усіх частинах програмної системи.

## Adaptive Software development (ASD)

**Адаптивна розробка ПЗ**  (АРПЗ) — це процес розробки ПЗ, запропонований Джимом Хайсмітом та Семом Байером під час швидкої розробки програмних продуктів. Дана модель втілює в собі принцип безперервної адаптації процесу розробки, близького до нормального плину справ.

Адаптивна розробка замінює водоспадну модель повторюваними серіями обдумування, співробітництва та навчання. Цей динамічний цикл передбачає постійне навчання та адаптацію до виникаючих станів проекту. Він пов'язаний із постійними змінами, повторними оцінками, намаганнями передбачити невідоме на поточний момент майбутнє проекту і потребує тісного зв’язку між розробниками, тестувальниками і замовниками. Характеристиками ЖЦ АРПЗ є зосередженість уваги, ітеративність, обмеженість за часом, управління ризиком, терпимість до змін.

АРПЗ побудована на концептуальній базі теорії складних адаптивних систем. Вона розрахована на використання в екстремальних проектах, в яких превалюють швидкий темп розробок, непередбачуваність і часті зміни.

При АРПЗ звичайний статичний ЖЦ «**Планування-Проектування-Конструювання**» змінюється на динамічний «**Обдумування-Взаємодія-навчання**).

***Dynamic System Development Model (DSDM)***

Методологія DSDM з'явилася в результаті роботи консорціуму з 17 англійських компаній. Ціла організація займається розробкою посібників з цієї методології, організацією навчальних курсів, програм акредитації і т. ін.

За цією методологією все починається з вивчення здійсненності програми і області її застосування. На першому етапі потрібно зрозуміти, чи підходить DSDM для даного проекту. На наступному - вивчення області застосування програми; передбачається на короткій серії семінарів, де програмісти дізнаються про ту сферу бізнесу, для якої їм потрібно працювати. Тут же обговорюються основні положення, що стосуються архітектури майбутньої системи і план проекту.

Далі процес ділиться на три взаємопов'язаних цикли:

1. цикл функціональної моделі (відповідає за створення аналітичної документації та прототипів);
2. цикл проектування і конструювання (приведення системи в робочий стан);
3. цикл реалізації (забезпечує розгортання програмної системи).

***Feature Driven Development (FDD)***

Ця методологія була розроблена Джеффом Де Люка і Пітером Коад. Вона робить основний акцент на короткі ітерації, кожна з яких служить для опрацювання певної частини функціональності системи. Згідно FDD, одна ітерація триває два тижні.

FDD налічує п'ять процесів. Перші три з них відносяться до початку проекту.

1. Розробка загальної моделі
2. Складання списку необхідних властивостей системи
3. Планування роботи над кожним властивістю
4. Проектування кожної властивості
5. Конструювання кожної властивості

Останні два кроки необхідно робити під час кожної ітерації. При цьому кожен процес розбивається на завдання і має критерії верифікації.

Всі розробники діляться на два види: "Власники класу" (власники класів) і «головні програмісти" (старші програмісти). Старші програмісти - це найбільш досвідчені розробники. Саме їм доручається розробка конкретних властивостей системи. Однак вони не займаються цим самостійно: старший програміст визначає, які класи зайняті в реалізації даної конкретного властивості, після чого збирає команду з власників необхідних класів, яка і займатиметься розробкою. Сам він діє як координатор, головний проектувальник і керівник, а на частку власників класів залишається, здебільшого, безпосереднє кодування.

***Контрольні запитання для самоперевірки***.

1. Що таке життєвий цикл програмного забезпечення?
2. Які різновиди моделей ЖЦ ПЗ Вам відомі?
3. В чому сутність каскадної моделі? Назвіть основні характеристики і фази каскадної моделі
4. В чому полягає інкрементна стратегія?
5. В чому полягає еволюційна стратегія?
6. Назвіть основні переваги каскадної моделі. Де краще використовувати каскадну модель?
7. Назвіть основні недоліки каскадної моделі?
8. В чому сутність ітераційної спіральної модель ЖЦ ПЗ? Які характеристики спіральної моделі життєвого циклу є її перевагою?
9. В чому сутність інкрементної ітераційної модель ЖЦ ПЗ.?
10. В чому сутність V-подібної моделі?

**Рекомендована література**

1. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
2. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
3. Філдінг Пол Джон. Як керувати проєктами. Основні навички проєктного менеджменту: вчасні результати в межах бюджету. Фабула.2020. 240 с

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.