**Лекція 2. Стандарт і моделі життєвого циклу**.

Методологічну основу будь інженерії складає **поняття життєвого циклу** (ЖЦ) виробу (продукту) як сукупності всіх дій, які треба виконати протягом всього «життя» виробу. Сенс життєвого циклу складається під взаємопов'язаності всіх цих дій.

Життєвий цикл промислового виробу визначається як послідовність етапів (фаз, стадій), які свою чергу складаються з технологічних процесів, дій та операцій.

Зазвичай до таких етапам відносять:

* проектування,
* виготовлення зразка,
* організацію виробництва,
* серійне виробництво,
* експлуатацію,
* ремонт,
* виведення з експлуатації.

Організація промислового виробництва з позиції життєвого циклу дозволяє розглядати всі його етапи у взаємозв'язку, що веде до скороченню термінів, вартості та трудовитрат.

Вперше про необхідність розглядати розробку ПЗ з позицій його життєвого циклу заговорили в 1968 р Історично основними стандартами на ЖЦ ПЗ були:

1985 (уточнено в 1988 г.) **DOD-STD-2167 А** - Розробка програмних засобів для систем військового призначення. Перший формалізований і затверджений стандарт життєвого циклу для проектування ПС систем військового призначення за замовленнями Міністерства оборони США. Цим документом регламентовані 8 фаз (етапів) при створенні складних критичних ПС і близько 250 типових обов'язкових вимог до процесам і об'єктам проектування на цих етапах.

1994р. **MIL-STD-498**. Розробка та документування програмного забезпечення. Прийнятий Міністерством оборони США для заміни DOD-STD-2167 A і ряду інших стандартів. Він призначений для застосування всіма організаціями та підприємствами, отримувати замовлення Міністерства оборони США. У 1996 р затверджено дуже докладний (407 стор.) керівництво "Застосування і рекомендації до стандарту MIL-STD- 498 ". Основну частину складають 75 підрозділів - рекомендацій щодо забезпечення та реалізації процесів ЖЦ складних критичних ПС високої якості і надійності, функціонуючих в реальному часі.

1995р. **IEEE 1074**. Процеси життєвого циклу для розвитку програмного забезпечення. Охоплює повний життєвий цикл ПС, в якому виділяються шість крупних базових процесів. Ці процеси деталізуються 16 приватними процесами. В 24 останніх є ще більш дрібна деталізація в сукупності на 65 процесів-робіт. Зміст кожного приватного процесу починається з опису загальних його функцій і задач і переліку дій-робіт при подальшій деталізації. Для кожного процесу в стандарті представлена вхідна і результуюча інформація про його виконання і короткий опис сутності процесу. У стандарті увагу зосереджено переважно на безпосередньому створенні ПС і на процесах попереднього проектування. В додатку подано чотири варіанти адаптації максимального складу компонентів ЖЦ ПС до конкретних особливостей типових проектів.

Розробка стандартів ЖЦ та їх практичне застосування стикалися з рядом проблем:

* Впровадження стандартів вимагало вкладення значних коштів, що не завжди окупалося.
* Було неясно, чи всі необхідні процеси треба виконувати і якою мірою.
* Різні типи ПО (ІС, реального часу, бізнес системи), різні вимоги.
* Висока динаміка галузі та наявність стандартів, що застаріли.
* Термінологічна неоднозначність різних державних і корпоративних стандартів.

У багатьох випадках застосування стандартів було викликано лише вимогами замовників, хоча на практиці часто гальмувало виконання проектів.

**Стандарт ISO / IEC 12207 - процеси життєвого циклу ПЗ**

У 1995 році ISO спільно з IEC (International Electrotechnical Commission - Міжнародна електротехнічна комісія) був прийнятий міжнародний стандарт ISO / IEC 12207 - Information Technology - Software Life Cycle Processes. У 2000 р він був прийнятий як ГОСТ (ISO / IEC) 12207 - Процеси життєвого циклу програмних засобів.

Стандарт ISO / IEC 12207 розроблявся з урахуванням кращого світового досвіду на основі вище перелічених стандартів. Основними результатами стандарту ISO 12207 є:

* Запровадження єдиної термінології щодо розробки та застосування ПЗ (призначений не тільки для розробників, а й для замовників, користувачів, всіх зацікавлених осіб).
* Поділ понять ЖЦ ПЗ і моделі ЖЦ ПЗ. **ЖЦ ПО** в стандарті вводиться як **повна сукупність всіх процесів і дій щодо створення та застосування ПЗ**, а **модель ЖЦ - конкретний варіант організації ЖЦ, обґрунтовано (розумно) вибраний для кожного конкретного випадку**.
* Опис організації ЖЦ і його структури (процесів).
* Виділення процесу адаптації стандарту для побудови конкретних моделей ЖЦ.

Стандарт ISO / IEC 12207 визначає організацію ЖЦ програмного продукту як сукупність процесів, кожний з яких розбитий на дії, що складаються з окремих задач; встановлює структуру (архітектуру) ЖЦ програмного продукту у вигляді переліку процесів, дій і завдань.

У відповідності зі стандартом ISO / IEC 12207 процеси ЖЦ діляться на три групи:

* Основні
* Допоміжні
* Організаційні

Окремо описаний процес адаптації стандарту, який містить основні роботи, які повинні бути виконані при адаптації даного стандарту до умов конкретного програмного проекту.

До числа **основних** відносяться процеси:

* Замовлення. Визначає роботи замовника, тобто організації, яка набуває систему, програмний продукт або програмну послугу.
* Поставки. Визначає роботи постачальника, тобто організації, яка поставляє систему, програмний продукт або програмну послугу замовнику.
* Розробки Визначає роботи розробника, тобто організації, яка проектує і розробляє програмний продукт.
* Експлуатації. Визначає роботи оператора, тобто організації, яка забезпечує експлуатаційне обслуговування обчислювальної системи в заданих умовах в інтересах користувачів.
* Супроводу. Визначає роботи персоналу супроводу, тобто організації, яка надає послуги з супроводу програмного продукту, що складаються в контрольованому зміні програмного продукту з метою збереження його початкового стану і функціональних можливостей. Даний процес охоплює перенос і зняття з експлуатації програмного продукту.

**Допоміжними** процесами є: ·

* Документування. Визначає роботи з опису інформації, що видається в процесі життєвого циклу.
* Управління конфігурацією. Визначає роботи з управління конфігурацією.
* Забезпечення якості. Визначає роботи за об'єктивним забезпеченню того, щоб програмні продукти і процеси відповідали вимогам, встановленим для них, і реалізовувалися в рамках затверджених планів. Спільні аналізи, аудиторські перевірки, верифікація та атестація можуть використовуватися в якості методів забезпечення якості.
  + Верифікації. Визначає роботи (замовника, постачальника або незалежної сторони) по верифікації програмних продуктів у міру реалізації програмного проекту.
  + Атестації. Визначає роботи (замовника, постачальника або незалежної сторони) по атестації програмних продуктів програмного проекту.
  + Спільного аналізу. Визначає роботи з оцінки стану і результатів якої-небудь роботи. Даний процес може використовуватися двома будь-якими сторонами, коли одна зі сторін перевіряє іншу сторону на спільній нараді.
  + Аудиту. Визначає роботи з визначення відповідності вимогам, планам і договором. Даний процес може використовуватися двома сторонами, коли одна зі сторін (що перевіряє) контролює програмні продукти або роботи іншої сторони (перевіреній).
* Рішення проблем. Визначає процес аналізу та усунення проблем (включаючи невідповідності), незалежно від їх характеру і джерела, які були виявлені під час здійснення розробки, експлуатації, супроводу або інших процесів.

**Організаційні** процеси життєвого циклу:

* Управління. Визначає основні роботи з управління, включаючи управління проектом, при реалізації процесів життєвого циклу.
* Створення інфраструктури. Визначає основні роботи по створенню основний структури процесу життєвого циклу.
* Удосконалення. Визначає основні роботи, які організація (замовника, постачальника, розробника, оператора, персоналу супроводу або адміністратора іншого процесу) виконує при створенні, оцінці, контролі і удосконаленні обраних процесів життєвого циклу.
* Навчання. Визначає роботи за відповідним навчання персоналу.

Схематично основні принципи ISO / IEC15504 можна представити у вигляді наступних тез:

Призначення стандарту складається в атестації, вдосконаленні та визначенні зрілості процесів створення ПЗ.

Основу стандарту становить еталонна модель процесів та їх зрілості. Еталонна модель має два виміри: «Процеси» і «Зрілість».

Вимірювання «Процеси» містить класифікацію процесів ЖЦ ПЗ. ця класифікація є розвитком стандарту ISO12207 і включає:

- три групи і п'ять категорій процесів;

- поділ процесів (по відношенню до ISO12207) на базові,

- розширені, нові, складові і розширені складові.

На відміну від CMM, у вимірі «Зрілість» представлено 6 рівнів зрілості процесів, по кожному з яких встановлені атрибути, що відображають досягнення процесом рівня зрілості. Значення атрибутів оцінюються в відсотках від повного досягнення атрибута. Для якісної оцінки вводяться рейтинги атрибутів.

Атестація процесів становить основу для їх оцінки та удосконалення. Атестація процесів:

- полягає у визначенні значень рейтингів атрибутів процесів;

- починається в вибору моделі процесів та їх зрілості організації, що атестується, сумісної з еталонною моделлю стандарту;

- є документованим процесом, представленим в стандарті в вигляді інструкцій і регламентованої процедури атестації;

- проводяться атестуючими, вимоги до компетентності яких також прописані в стандарті.

Оцінка зрілості та удосконалення процесів виконуються за результатами атестації процесів і також є документально оформленими процесами стандарту.

***Стандарт зрілості компанії-розробника ПЗ CMM***

Модель зрілості технологічних процесів організації Capability Maturity Model (CMM) розроблена Інститутом інженерів програмного забезпечення (Software Engineering Institute, SEI) та корпорацією Mitre під керівництвом Уоттса Хамфри (Watts Humphrey).

Методологія CMM розроблялася й розвивалася в США як засіб, що дозволяє вибирати кращих виробників ПЗ для виконання держзамовлень у першу чергу міністерства оборони. Для цього були розроблені критерії оцінки зрілості ключових процесів компанії та визначений набір дій, необхідних для їхнього подальшого вдосконалювання. У підсумку методологія виявилася надзвичайно корисною для більшості компаній, що прагнуть якісно поліпшити існуючі процеси проектування, розроблення, тестування програмних засобів і звести керування ними до зрозумілих і легко реалізованих алгоритмів і технологій, описаних у єдиному стандарті. У подальшому ця модель переросла у методологію підвищення якості процесів підприємства Capability Maturity Model Integration (CMMI). Застосування CMMI дозволяє поставити розроблення ПЗ на промислову основу, підвищити керованість ключових процесів і виробничу культуру в цілому, гарантувати якісну роботу й виконання проектів у строк.

Основою для створення CMM стало базове положення про те, що фундаментальна проблема "кризи" процесу розроблення якісного ПЗ полягає не у відсутності нових методів і засобів розроблення, а в нездатності компанії організувати технологічні процеси й керувати ними.

Для оцінки ступеня готовності підприємства розробляти якісний програмний продукт СММ використовує ключове поняття ***зрілість організації (Maturity)***. *Незрілою* вважається організація, у якій:

• відсутнє довгострокове й проектне планування;

• процес розроблення програмного забезпечення і його ключові моменти не ідентифіковані, реалізація процесу залежить від поточних умов, конкретних менеджерів і виконавців;

• методи і процедури не стандартизовані й не документовані;

• результат не визначений реальними критеріями, які встановлюються за запланованими показниками із застосуванням стандартних технологій і розроблених метрик;

• процес вироблення рішення відбувається стихійно, на грані мистецтва.

У цьому випадку велика ймовірність появи несподіваних проблем, перевищення бюджету або невиконання строків здачі проекту. У такій компанії, як правило, менеджери й розробники не керують процесами - вони змушені займатися поточними та спонтанними проблемами.

Основні ознаки *зрілої організації*:

• у компанії є чітко визначені й документовані процедури керування вимогами, планування проектної діяльності, керування конфігурацією, створення й тестування програмних продуктів, відпрацьовані механізми керування проектами;

• ці процедури постійно уточнюються й удосконалюються;

• оцінки часу, складності й вартості робіт ґрунтуються на накопиченому досвіді, розроблених метриках і кількісних показниках, що робить їх достатньо точними;

• актуалізовано зовнішні й створені внутрішні стандарти на ключові процеси й процедури;

• існують обов'язкові для всіх правила оформлення методологічної програмної й користувальницької документації;

• технології незначно змінюються від проекту до проекту на основі стабільних і перевірених підходів і методик;

• максимально використовуються напрацьовані у попередніх проектах організаційний і виробничий досвід, програмні модулі, бібліотеки програмних засобів;

• активно апробуються і впроваджуються нові технології, виробляється оцінка їхньої ефективності.

СММ визначає п'ять рівнів технологічної зрілості компанії (рис.3), за якими замовники можуть оцінювати потенційних претендентів на підпис контракту, а розробники – удосконалювати процеси створення ПЗ.

Кожний із рівнів, крім першого, складається з декількох ключових областей процесу (Key Process Area), що містять цілі (Goal), зобов'язання щодо виконання (Commitment to Perform), можливість виконання (Ability to Perform), виконувані дії (Activity Performed), їхній вимір і аналіз (Measurement and Analysis) та перевірку впровадження (Verifying Implementation). Таким чином, СММ фактично є комплексом вимог до ключових параметрів ефективного стандартного процесу розроблення ПЗ та засобом його постійного поліпшення. Виконання цих вимог. збільшує ймовірність досягнення підприємством поставлених цілей у сфері якості.

***Початковий рівень*** *(Initial Level – Level 1).*

На даному рівні компанія може одержати замовлення, розробити і передати замовникові програмний продукт. Стабільність розробок відсутня. Лише деякі процеси визначені, результат цілком залежить від зусиль окремих співробітників. Успіх одного проекту не гарантує успішності наступного. До цієї категорії можна віднести будь-яку компанію, що хоч якось виконує взяті на себе зобов'язання.

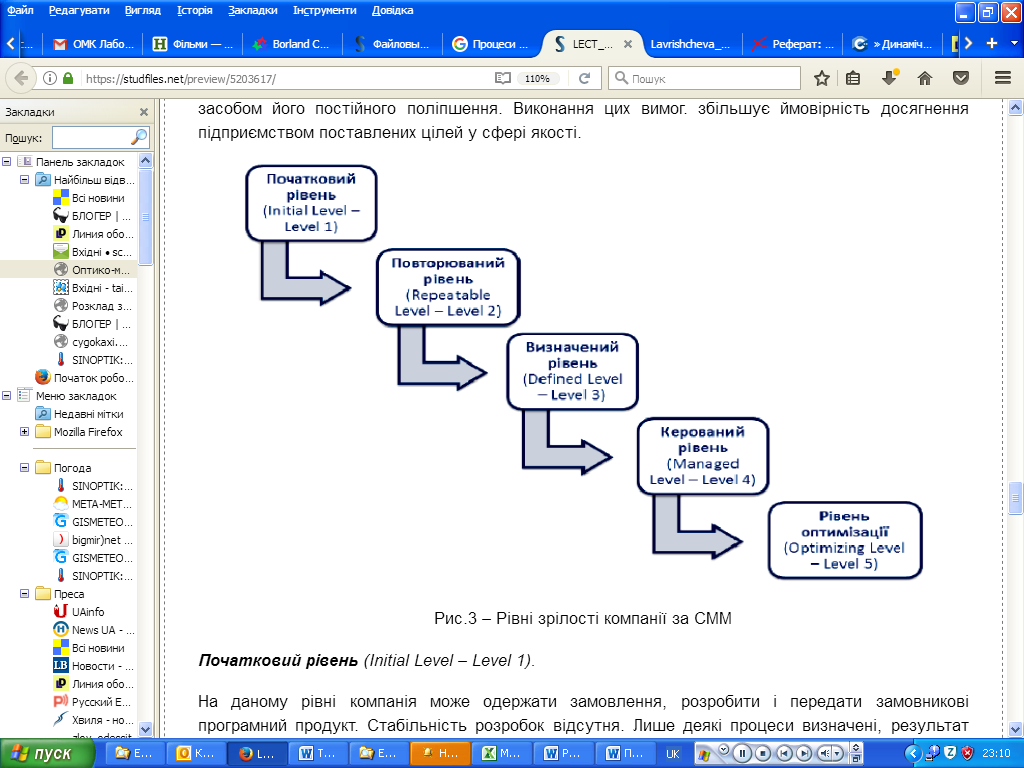
*Ключові області* процесу цього рівня не зафіксовані.

***Повторюваний рівень*** *(Repeatable Level – Level 2).*

Цьому рівню відповідають підприємства, які володіють певними технологіями керування й розроблення. Керування вимогами та планування у більшості випадків ґрунтуються на розробленій документованій політиці й накопиченому досвіді. Установлені й уведені в повсякденну практику базові показники для оцінки параметрів проекту. Менеджери відслідковують виконання робіт і контролюють тимчасові й виробничі витрати.

У компанії розроблені деякі внутрішні стандарти й організовані спеціальні групи перевірки якості QA. Зміни версій кінцевого програмного продукту й створених проміжних програмних засобів відслідковуються в системі керування конфігурацією. Є необхідна дисципліна дотримання встановлених правил. Ефективні методики й процеси установлюються, що забезпечує можливість повторення успіху попередніх проектів у тій самій прикладній області.

*Ключові області* процесу розроблення ПЗ цього рівня наведені на рис.4.



***Визначений рівень*** *(Defined Level – Level 3).*

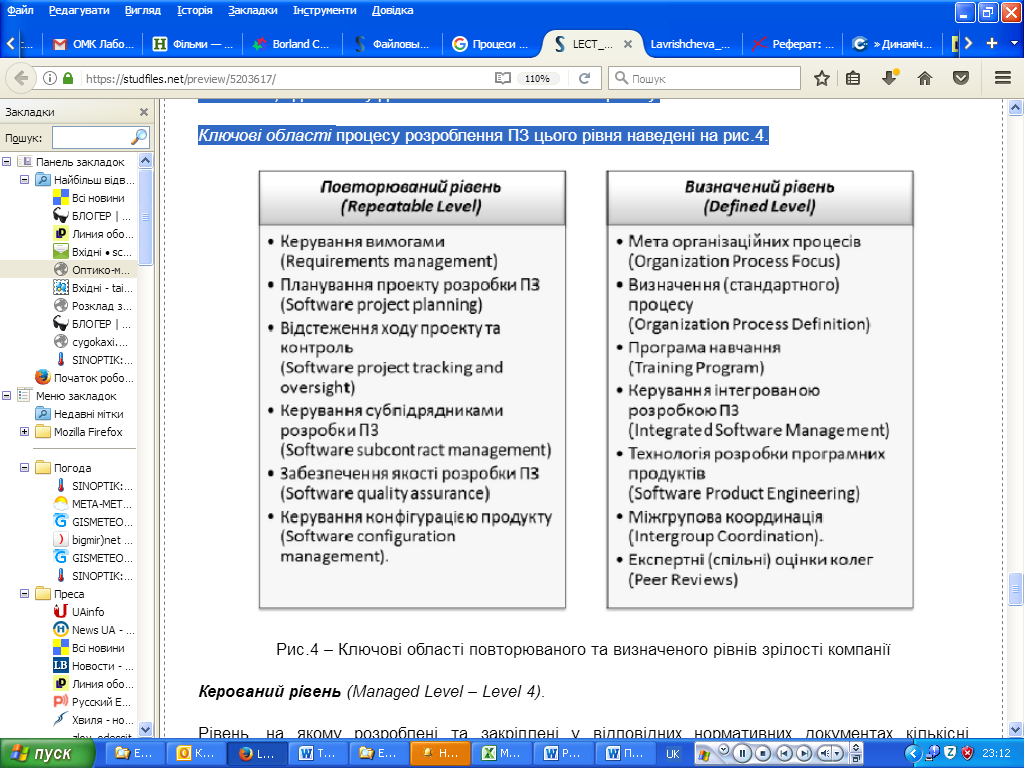
Рівень характеризується деталізованим методологічним підходом до керування (описані й закріплені у документованій політиці типові дії, необхідні для багаторазового повторення: ролі й відповідальність учасників, стандартні процедури й операції, порядок дій, кількісні показники й метрики процесів, формати документів та ін.).

Для створення й підтримки методологій в актуальному стані в організації підготовлена й постійно функціонує спеціальна група. Компанія регулярно проводить тренінги для підвищення професійного рівня своїх співробітників.

Починаючи з цього рівня, організація практично перестає залежати від особистісних якостей конкретних розроблювачів і не має тенденції опускатися на нижчі рівні. Ця незалежність обумовлена продуманим механізмом постановки завдань, планування заходів, виконання операцій і контролю виконання.

Управлінські й інженерні процеси документовані, стандартизовані й інтегровані в уніфіковану для всієї організації технологію створення ПЗ. Кожний проект використовує затверджену версію цієї технології, адаптовану до особливостей поточного проекту.

*Ключові області* процесу розроблення ПЗ цього рівня наведені на рис.4.



**Керований рівень** (Managed Level – Level 4).

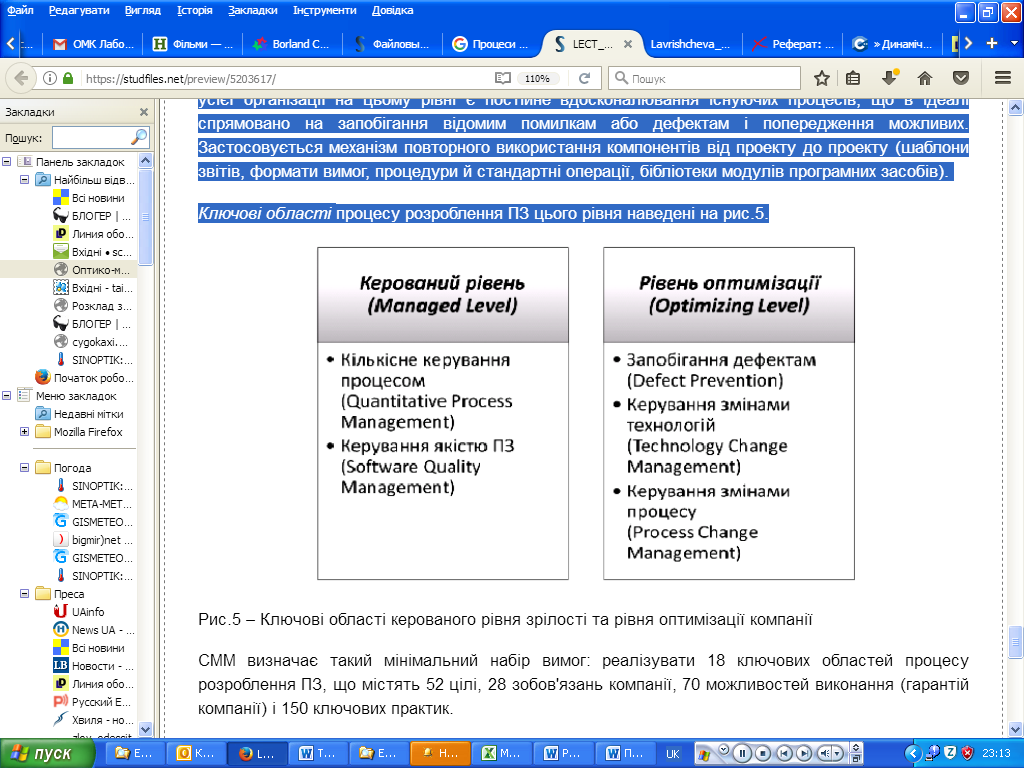
Рівень, на якому розроблені та закріплені у відповідних нормативних документах кількісні показники якості. Більш високий рівень керування проектами досягається за рахунок зменшення відхилень різних показників проекту від запланованих. При цьому тенденції зміни продуктивності процесу можна відділити від випадкових варіацій на підставі статистичної обробки результатів вимірювань у процесах. Ключові області процесу розроблення ПЗ цього рівня наведені на рис.5.

**Рівень оптимізації** (Optimizing Level – Level 5).

Для цього рівня заходи щодо вдосконалювання розраховані не лише на існуючі процеси, але й на запровадження, використання нових технологій і оцінку їхньої ефективності. Основним завданням усієї організації на цьому рівні є постійне вдосконалювання існуючих процесів, що в ідеалі спрямовано на запобігання відомим помилкам або дефектам і попередження можливих. Застосовується механізм повторного використання компонентів від проекту до проекту (шаблони звітів, формати вимог, процедури й стандартні операції, бібліотеки модулів програмних засобів). Ключові області процесу розроблення ПЗ цього рівня наведені на рис.5.

СММ визначає такий мінімальний набір вимог: реалізувати 18 ключових областей процесу розроблення ПЗ, що містять 52 цілі, 28 зобов'язань компанії, 70 можливостей виконання (гарантій компанії) і 150 ключових практик.

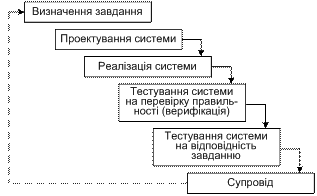
У результаті аудиту та атестації компанії присвоюється певний рівень, що після наступних аудитів може підвищуватися або знижуватися. Кожний наступний рівень в обов'язковому порядку містить у собі всі ключові характеристики попередніх. У зв'язку з цим сертифікація компанії щодо одного з рівнів припускає безумовне виконання всіх вимог більш низьких рівнів.



До *переваг* моделі CMM належить те, що вона орієнтована на організації, які займаються розробленням програмного забезпечення. У даній моделі вдалося більш детально визначити вимоги, специфічні для процесів, пов'язаних з розробленням ПЗ. Із цієї причини в CMM наведені не тільки вимоги до процесів організації, але й приклади реалізації таких вимог.

Основний *недолік* CMM полягає в тому, що модель не авторизована як стандарт ні міжнародними, ні національними органами зі стандартизації. Втім, CMM давно стала промисловим стандартом. До *недоліків* моделі також необхідно віднести більші зовнішні накладні витрати на приведення процесів компанії у відповідність до моделі СММ, ніж до моделей ISO 9000. Це пов'язано з меншою поширеністю моделі в світі, меншою кількістю консалтингових органів і експертів і, в результаті, з набагато більшими зовнішніми витратами на консалтинг і на підтвердження відповідності процесів незалежною третьою стороною. Проте, CMM, поза сумнівом, корисно ISO 9000.

**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**

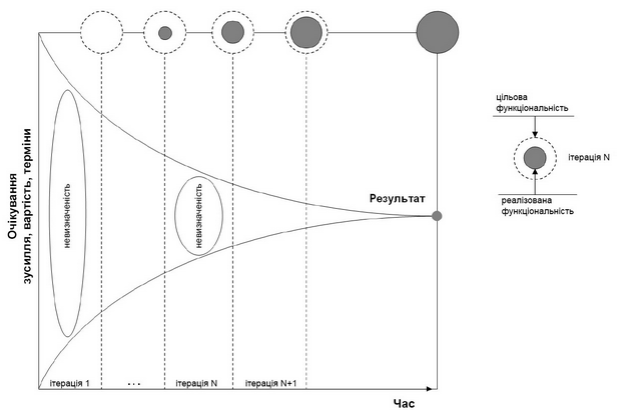
Однією з перших з’явилась каскадна модель, в якій кожен етап роботи виконується лише раз. На кожному етапі робота виконується настільки ретельно, щоб потреби повертатись до попереднього не виникало. Результат виконання кожного етапу, перед передачею в наступний, піддається верифікації.

**Каскадна модель ЖЦ програмних систем**

**Ітеративна та інкрементна модель**

Ітеративна, що отримала також від Т. Гілба в 70-і рр. назву еволюційної моделі (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.



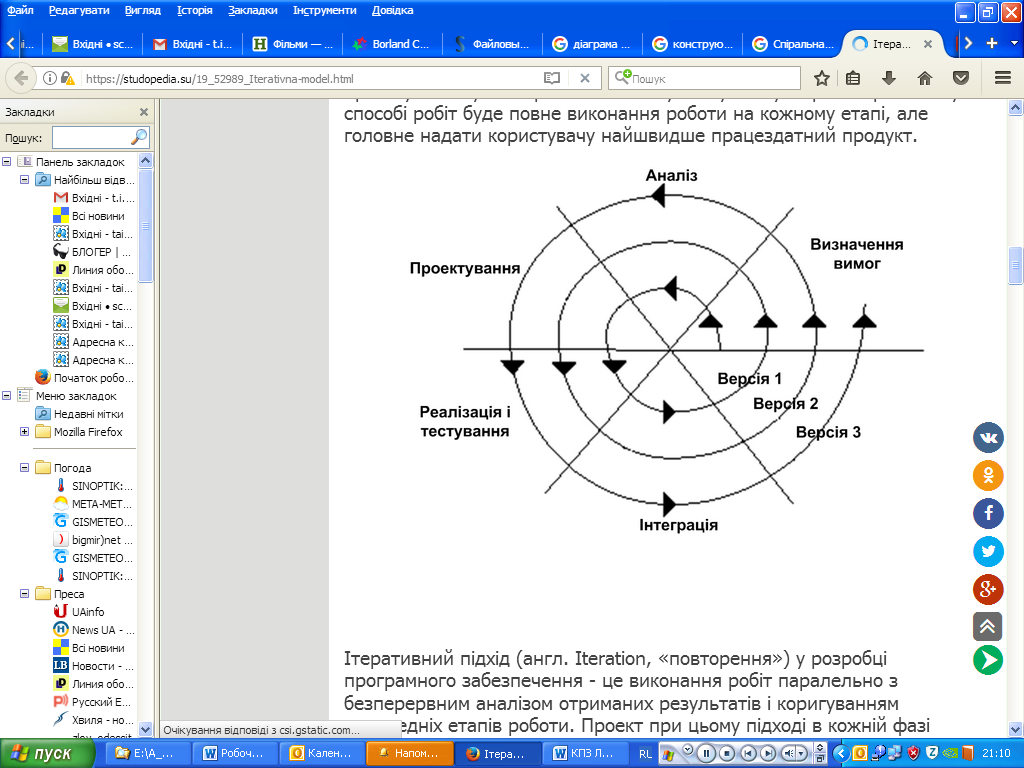
При такому підході кожна версія повинна бути придатною для використання і забезпечувати деяку підмножину необхідних функцій. До недоліків відноситься збільшення обсягу робіт по тестуванню, оскільки з появою кожної нової версії продукту потрібне повторне підтверджуюче тестування, що додані можливості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

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.

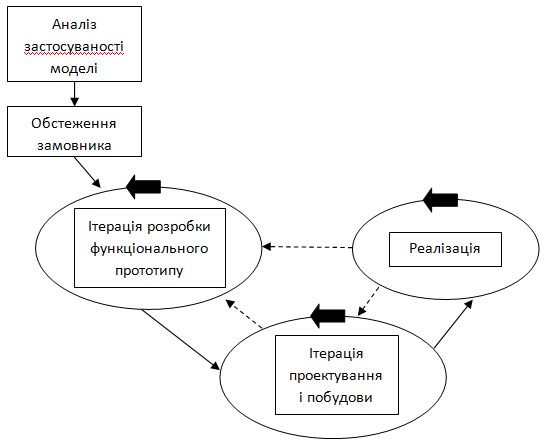
**Спіральна модель ЖЦ розробки програмних систем**

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.

**Модель еволюційного прототипування**

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.

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.

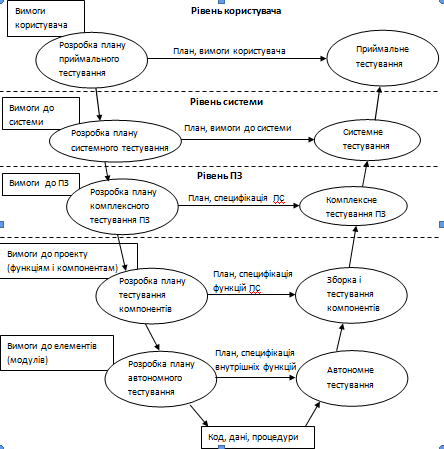
При цьому враховуються такі чинники ризику:

* 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**



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)

**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.

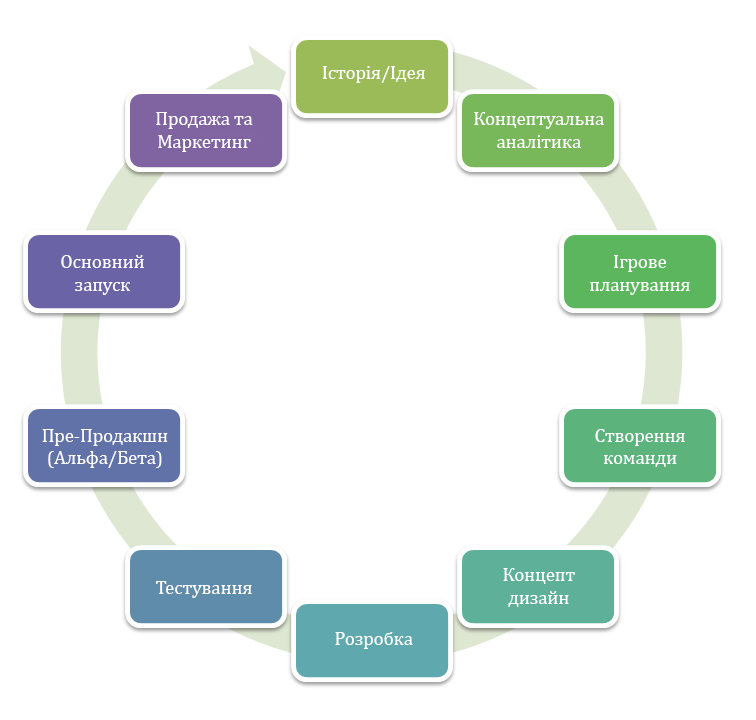
FDD налічує п'ять процесів. Перші три з них відносяться до початку проекту.

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.

**GDLC модель (Game Dev Life Cycle)**



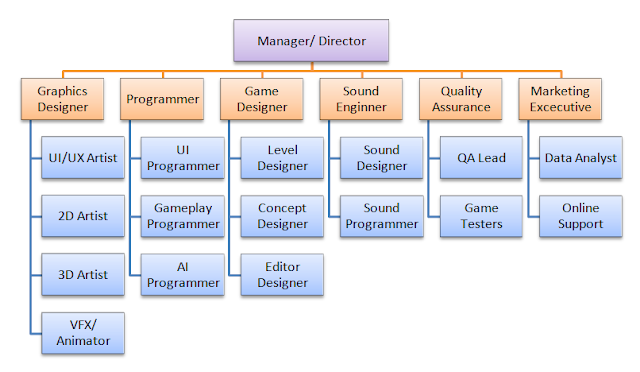
Гра являє собою єдиний тип програмного забезпечення, що має за мету забезпечення розваги. GDLC модель представляє процес розробки гри з нуля до розгортання..

**Фаза 1 - Історія / Ідея:** Історія / Ідея - це фактична ігрова концепція та вимога проекту, які. повинні бути зазначені у вимозі / проектному документі. Ідеї представляють лише фрагменти гри, це фактичний прототип гри, який охопив усі ідеї. Група ідей називається Game Concept / Game Story.

**Фаза 2 - Концептуальна аналітика**: Відповідно до історії чи концепцій потрібно глибоко проаналізувати вимогу до початку розробки гри. Потрібно провести техніко-економічне обґрунтування, проаналізувати окремі сфери, визначити фактичні вимоги, ціноутворення,технічні можливості, організаційні, культурні чи юридичні питання та рішення, а також визначити обсяг проекту.

**Фаза 3 – Ігрове планування**: Після визначення всіх вимог та дослідження даних, необхідних для планування розвитку гри, потрібно створити план проекту. Це можна зробити у вигляді переліку усіх функцій, завдань (графіка, анімація, звуки тощо) та ідеї у документі або через надання діаграм потоку завдань. Для кожного завдання потрібно провести планування часу та оцінку, хід роботи подання завдання. Потрібно визначити робочий процес, кейси та плани випробувань.

**Фаза 4 - Створення команди**: Для початку проектування та розробки потрібна команда, яка втілить концепції в життя. До складу команди відповідно до вимог гри входять такі члени як дизайнери, програмісти, тестувальники тощо. Загалом, у розробці ігор так багато категоризованих фахівців. В наданій нижче таблиці представлені загальні категорії фахівців та занять для будь-яких ігор.



**Етап 5 - Концепт-дизайн**: Дизайн концепції означає дизайн-прототип фактичної потреби / ідеї / історії, він також називається ігровим дизайном. Ігровий дизайн - це ядро будь-якої гри / продукту. Це найбільш інноваційний, креативний та складний процес будь-якої гри, який є демонстрацією майстерності ремесла для втілення ідеї в реальність у вигляді якісних ігор. Це вимагає критичного інтерактивного мислення, розуміння, реалізації, виконання, поведінки, інтерфейсу користувача. Перед початком розробки створюють документ під назвою "Документ дизайну ігор (GDD)", в якому описують усі віртуальні ідеї та дизайн як план гри / продукту, зокрема, елементи ігрового дизайну: Інтерфейс користувача; Дані гри; Дані гравця та характеристики; Дизайн рівня Гри та механізм 3D / 2D ігрова арена; Об'єкти гри / Повноваження / Властивості; Штучний інтелект VFX (Visual Effects) / Анімація; Звукова музика; Підтримка та майбутні вдосконалення.

**Фаза 6 - Розробка**: Після завершення GDD розпочинається розробка фактичної ігрової концепції / ідеї, як описано в Ігровому дизайні. Щоб вибрати Game Engine (реалізація базової функціональності) та допоміжні модулі / плагіни / фреймворки / платформи потрібно запустити процес розробки / програмування. Провідний програміст несе основні обов'язки щодо розвитку та якості гри, він / вона повинен скласти контрольний список очікуваного / робочого / виконаного списку завдань відповідно до завдання розробника. Кожен програміст / розробник повинен подати свою роботу ведучому програмісту. Провідний розробник повинен дотримуватися системи перегляду коду відповідно до обраної ним архітектури гри. Для програміста вхідні та вихідні дані прості, однак все значно ускладнює обробка цих даних із зображеннями, візуальними елементами, анімацією, VFX, зіткненнями, фізикою та звуками. управління сценою / рівнем, AI (Artificial intelligence), динамічне створення та знищення об’єктів, завантаження та вивантаження ресурсів, шейдери, візуалізація, обробка збирачів сміття, використання процесора та графічного процесора, завантаження гри, підтримка мультиплатформ, інтеграція реклами, реалізація віртуальних товарів. Провідний розробник / розробник повинен відповідати за системи кодування та її термінологію. Абстракція Модульність Шаблон дизайну Архітектура програмного забезпечення / ігор Структура / стиль кодування. Потрібні навички ООП для забезпечення надійного програмування та реалізації тезису " Використовуйте менше ресурсів і генеруйте більше результатів". Окрім того потрібно відчути досвід кінцевого користувача.

**Фаза 7 - Тестування**: Тестування - дзеркало кінцевого продукту, є найважливішою частиною GDLC. Тестування та ігровий дизайн мають однакову вагу в будь-якій архітектурі розробки ігор / концепцій. Тестування - це не просто гра в гру на вашому робочому місці / арені, а фактичний досвід кінцевого користувача стосовно продукту. Це повторюваний та інтерактивний процес того самого введення потоку екрану та очікування результату від користувача щодо якості ігор. Процес QA (Quality Assurance) відповідно до звіту про відстеження помилок викликає потребу команді тестування працювати над двома документами, I) Документи для тестів II) Документи планів випробувань. Команда контролю якості повинна забезпечити участь деяких документів та файлової системи у кожному процесі тестування, де відображуються. дефект / помилка, їх поширення, модуль, частота, номер журналу помилок, статус / поява, знімки екрана, платформи, дата, час. Застосовується декілька типів методології тестування, які повинна застосовувати команда з контролю якості, такі як функціональне тестування, тестування переривань, модульне тестування, тестування продуктивності, тестування навантаження, тестування пам’яті, тестування сумісності, тестування відповідності, тестування поведінки, тестування досвіду кінцевого користувача, тестування мережі тощо. .

**Фаза 8 – Пре-Продакшн** або попереднє виробництво (випуск альфа / бета-версії): Передвиробнича техніка - це мистецтво або хороший знак успішного менеджера / режисера / продюсера. Перед реальним виробництвом / випуском будь-якого продукту обов’язково потрібне виконання попереднього виробництва. Потрібно випустити ігровий продукт для альфа / бета-версії, щоб з’ясувати або відстежувати помилки в реальному часі та отримати досвід користувача. Альфа / бета-версія - це хороший спосіб дізнатись про досвід користувача та його сприйняття продукту. Якщо зроблено якісь помилки після завершення випробувальної частини, то їх можна ідентифікувати з використанням техніки попереднього виробництва перед основним виробництвом. Потрібно сформувати звіт для випуску альфа / бета-версії. Якщо необхідна переробка, то потрібно зробити це для продукту та підготувати його до основного виробництва / основного релізу.

**Фаза 9 - Основне виробництво**: На етапі виробництва готується коротке відео, подібне до трейлера гри, робиться привабливий знімок екрана. Після цього розгортається гра належним чином (натискається «Опублікувати» та публікується у відповідних магазинах). Щоб підготувати примітку про випуск та політику конфіденційності, а також умови використання продукту, готується в основний документ. Ігровий кредит дуже важливий для гри, і команда повинна віддавати належну йому копітку роботу.

**Фаза 10 - Продажі та маркетинг**: Після випуску кінцевого продукту нове завдання виконується командою з продажу та маркетингу. Команда маркетингу повинна продовжувати зосереджуватись та аналізувати звіт про дані живої гри, а також створювати звіт про досвід користувачів. Люди, що займаються маркетингом, також перевіряють рейтинги / відгуки, надані користувачами, що стосуються продуктів, а також дають відповідну відповідь цим користувачам, надають підтримку для запитів користувачів. Надсилається звіт про запити команді з контролю якості, а потім команда з контролю якості вживає заходи та інформує про це відповідні відділи. Люди, що займаються маркетингом, також вдосконалюють продукт, також використовуючи методології маркетингу в Інтернеті та в режимі офлайн. Щоб залучити більше користувачів та активізувати гру, потрібно застосувати маркетингову методологію для залучення більшої аудиторії. Команда маркетингу повинна відстежувати всі дані поточної гри, поточних одночасних користувачів, поточну гру, а також аналізувати дані, і, якщо потрібні зміни, повідомляти про це у відповідний відділ, ця команда працюватиме з поточною аудиторією та намагатиметься збільшити кількість людей. Команда маркетингу також планує збільшення доходу та виробляє нові ідеї та термінологію для сприяння отриманню доходу. Команда маркетингу також стежить за діяльністю нашого конкурента та новими випусками. Щоб утримувати товар на вищій позиції, потрібно знати, хто є конкурентом і що він пропонує, як можна вдосконалити продукт, зробити кращим ніж у інших і тримати його завжди на вищій позиції.

***Для самостійного вивчення*** *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Література***

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
3. Проектування інформаційних систем: Посібник // За редакцією Пономаренка В.С. – К.: Видавничий центр "Академія". 2002. ­ 488 с. URL: <http://www.dut.edu.ua/uploads/l_874_10304054.pdf>.
4. Соммервиль И. Инженерия программного обеспечения, 6 изд. – И.д. "Вильямс", 2002. URL: https://www.studmed.ru/download/sommervill-ian-inzheneriya-programmnogo-obespecheniya\_4935164f089.html
5. ДСТУ ISO/IEC/IEEE 15289:2019 (ISO/IEC/IEEE 15289:2017, IDT) Інженерія систем і програмних засобів. Уміст інформаційних об’єктів життєвого циклу. - К.: Держстандарт України, 2019.
6. ДСТУ ISO/IEC TR 12182:2004. Інформаційні технології. Класифікація програмних засобів (ISO/IEC TR 12182:1998, IDT) - К.: Держстандарт України, 2004.
7. Unity3d Tech Guru. GDLC [Game Development Life Cycle] URL: http://www.unity3dtechguru.com/2018/01/gdlc-game-development-life-cycle.html (дата звернення: 18.01.2021).

***Контрольні запитання.***

1. Дайте визначення життєвого циклу програмного продукту.
2. Які основні результати створення стандарту ISO / IEC 12207?
3. На які групи діляться процеси ЖЦ ПЗ за стандартом ISO / IEC 12207?
4. Які основні принципи стандарту ISO / IEC15504?
5. Як визначається незріла компанія за стандартом СММ?
6. Як визначається зріла компанія за стандартом СММ?
7. В чому полягають принципи каскадної моделі?
8. В чому різниця між ітеративною та інкрементною моделлю?
9. Що спільного в спіральній, еволюційній моделях та моделі еволюційного прототипування?
10. Охарактеризуйте V-подібну модель.
11. Які основні риси адаптивних моделей?