МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Кафедра прикладної математики

04-01-06

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни
„ Командна розробка програмних проектів”
для студентів напряму підготовки
6.040302 „Інформатика”
денної форми навчання

Рекомендовано
методичною комісією
зі спеціальності
„Інформатика”
Протокол № 2
від 03.03.2015

Рівне 2015

Конспект лекцій з дисципліни „ Командна розробка програмних проектів” для студентів напряму підготовки 6.040302 „Інформатика”/ Тулашвілі Ю.Й. - Рівне: НУВГП, 2015. – 102 с.

Упорядник:

Тулашвілі Ю.Й., доктор педагогічних наук, професор кафедри прикладної математики;

Відповідальний за випуск:

Турбал Ю.В., кандидат фіз-мат. наук, професор, завідувач кафедри прикладної математики.

Зміст

Змістовий модуль 1. Сучасні підходи до проектування IT-систем

Тема 1. Проект та специфіка проектної діяльності в IT індустрії

План:

1. Проект та специфіка проектної діяльності.
2. Стандарти із розробки IT-систем SWEBOK .
3. Структура і зміст SWEBOK.
4. Управління проектною діяльністю.
5. Організація процесу управління розробкою ПП.

Проект та специфіка проектної діяльності

Поняття проекту (від латинського "кинутий вперед") є досить широковживаним у сучасному житті людини. Під проектом розуміють комплекс науково-дослідних, проектно-конструкторських, соціально-економічних, організаційно-господарських та інших заходів, пов'язаних ресурсами, виконавцями та термінами, відповідно оформлених і направлених на зміну об'єкта управління, що забезпечує ефективність розв'язання основних завдань та досягнення відповідних цілей за певний період. Загальними кінцевими цілями проектів є створення та освоєння дещо нового: матеріалів, технологій, техніки, інформаційних систем тощо. Наведемо деякі дефініції проекту, що знайшли широке застосування.

Англійською асоціацією проект-менеджерів подається така дефініція проекту – “це окреме підприємство з конкретними цілями, які часто включають вимоги до часу, вартості та якості результатів, що досягаються”.

У Тлумачному словнику з управління проектами термін "проект" подається у такому трактуванні : “проект – це певне завдання з визначеними вихідними даними й встановленими результатами (цілями), що обумовлюють спосіб його вирішення”.

Ми, при вивченні даної навчальної дисципліни, будемо вживати більш універсальне визначення проекту. Проект - це задум (завдання, проблема) та необхідні засоби його реалізації з метою досягнення бажаного економічного, технічного, технологічного, інформаційного чи організаційно-управлінського результату.

Проектування інформаційних систем (ІС) завжди починається з визначення мети проекту. Основна задача будь-якого успішного проекту полягає в тому, щоб на момент запуску ІС та протягом усього часу її експлуатації можна було забезпечити:

- необхідну функціональність системи і ступінь адаптації до умов, які змінюються з часом, її функціонування;
- необхідну пропускну здатність системи;

- необхідний час реакції системи на запит;
- безвідмовну роботу системи в необхідному режимі, іншими словами - готовність і приступність системи для обробки запитів користувачів;
- простоту експлуатації і підтримки системи;
- необхідну безпеку.

Проектування ІС охоплює три основні області:

1. проектування об'єктів даних, що будуть реалізовані в базі даних;
2. проектування програм, екранних форм, звітів, що будуть забезпечувати виконання запитів до даних;
3. облік конкретного середовища чи технології, а саме: топології мережі, конфігурації апаратних засобів, використовуваної архітектури (сервер чи клієнт-сервер), паралельної обробки, розподіленої обробки даних тощо.

Продуктивність є головним чинником, що визначає ефективність ІС. Гарне проектне рішення є основою високопродуктивної ІС. В реальних умовах проектування - це пошук способу, що задовольняє вимогам функціональності програмного продукту за умови застосування наявних системних ресурсів засобами сучасних інформаційно-комунікаційних технологій.

Важливе місце при цьому належить проектному управлінню, а саме, необхідності розв'язання таких питань:

1) як спланувати та скоординувати реалізацію проекту яз використанням наявного кадрового потенціалу та матеріальних ресурсів;
2) як залучити кошти із зовнішніх джерел фінансування для реалізації проекту;
3) як краще розпорядитись власними коштами;
4) як досягти максимальних прибутків за мінімальних витрат;
5) як створити команду працівників для реалізації проекту;
6) як мотивувати персонал до ефективної діяльності;
7) як уникати конфлікту в команді проекту.

Успішна реалізація проекту окрім фінансового забезпечення значно залежить від якісного складу команди проектувальників ІС.

Стандарти із розробки ІТ-систем SWEBOK

SWEBOK – керівництво до зводу знань з технології проектуванні ІТ-систем.

Історія виникнення SWEBOK

З 1993 р. IEEE (Computer Society (Комп'ютерне Суспільство) of the Institute for Electrical and Electronic Engineers) і ACM (Association of Computer Machinery) координують свої роботи в рамках спеціального спільного комітету - Software Engineering Coordinating Committee (SWECC). Проект SWEBOK був ініційований цим комітетом у 1998 р. Оцінений можливий обсяг змісту SWEBOK і інші фактори привели до того, що було рекомендовано проводити

роботи з реалізації проекту не тільки силами добровольців з рядів експертів індустрії і представників найбільших споживачів і виробників програмного забезпечення, але і на основі принципу "повної зайнятості". Базовий комплекс робіт, у відповідності зі спеціальним контрактом, був переданий у Software Engineering Management Research Laboratory Університету Квебек у Монреалі (Universite du Quebec a Montreal).

Серед компаній, що підтримали цей унікальний проект були Boeing, MITRE, Raytheon, SAP. У результаті проекту, здійсненого за фінансової підтримки цих і інших компаній і організацій, а також з урахуванням його значимості для індустрії, SWEBOK Advisory Committee (SWAC) прийняв рішення зробити SWEBOK загальнодоступним (http://www.swebok.org).

Сьогоднішня "публічність" (загальнодоступність) результатів проекту стала можлива, у першу чергу, саме завдяки підтримці SWEBOK Industrial Advisory Board (IAB) – структури, що поєднує представників компаній, що підтримали проект.

Проект SWEBOK планувався у вигляді трьох фаз: Strawman ("солом'яна людина"), Stoneman ("кам'яна людина") і Ironman ("залізна людина").

До 2004 р. була випущена версія Посібника зі Зводу Знань 3-їй фази - Ironman, тобто максимально наближена до остаточного варіанту і схвалена IEEE у лютому 2005 р. до публікації в якості Trial-версії.

Після 6 років безпосередніх робіт над документом, SWEBOK включає "лише " 10 галузей знань (knowledge areas, KA). При цьому, додавання нових галузей знань у SWEBOK досить прозоре. Усе, що для цього потрібне, зрілість (чи, принаймні, явний і швидкий процес досягнення зрілості) і загальноприйнятності відповідної галузі знань, якщо це не призведе до серйозного ускладнення SWEBOK (концепція "загальноприйнятності" - generally accepted – визначена в IEEE Std 1490--1998,Adoption of PMI Standard — A Guide to the Project Management Body of Knowledge).

Керівництво до зводу знань, а таким є SWEBOK, включає базове визначення й опис галузей знань (наприклад, конфігураційне управління – configuration management) і, безумовно, є недостатнім для охоплення всіх питань, що відносяться до питань створення програмного забезпечення, але, у той же час потрібним для їхнього розуміння.

Необхідно відзначити, що однією з найважливіших цілей SWEBOK є саме визначення тих аспектів діяльності, що складають суть професії інженера-програміста.

Структура і зміст SWEBOK

Опис галузі знань SWEBOK побудована за ієрархічним принципом і являє собою результат структурної декомпозиції.

Така ієрархічна побудова зазвичай нараховує 2-3 рівня деталізації і прийнятих для ідентифікації тих чи інших загальновизнаних аспектів технології проектуванні IT-систем .

SWEBOK описує 10 галузей знань:

1. Software requirements – програмні вимоги
2. Software design – дизайн, архітектура
3. Software constructions – конструювання ПП
4. Software testing – тестування
5. Software maintenance – експлуатація (підтримка) ПП
6. Software configuration management - конфігураційне управління
7. Software engineering – управління в програмній інженерії
8. Software engineering process – процеси технології проектуванні IT-систем
9. Software engineering tools and methods – інструменти і методи технології проектуванні IT-систем
10. Software quality – якість технології проектуванні IT-систем .

До SWEBOK також входять суміжні дисципліни, зв'язок з якими представлений як фундаментально важливий та обґрунтований, це:

1. Інженерія комп'ютерів
2. Комп'ютерна наука
3. Менеджмент
4. Математичні та обчислювані методи
5. Управління проектами
6. Програмна ергономіка
7. Системи інженерії

Сьогодні існує спрощена версія SWEBOK, яка отримала назву PMBOK. На відміну на PMBOK, галузі SWEBOK не включають входи і виходи. Це пов'язано з тим, що PMBOK є більше прив'язана та деталізована до певних моделей, наприклад: життєвого циклу ПП.

Аналіз і характеристика областей знань SWEBOK

Ядро знань SWEBOK є основоположним науково-технічним документом, що відображає думки багатьох закордонних і вітчизняних фахівців в області технології проектуванні IT-систем та узгоджується із сучасними регламентованими процесами ЖЦ ПП стандарту ISO/IEC 12207. У цьому ядрі знань міститься опис 10 областей, кожна з яких представлена згідно з

прийнятим усіма учасниками створення цього ядра загальної схеми опису, що включає визначення понятійного апарата, методів і засобів, а також інструментів підтримки інженерної діяльності. У кожній області описується визначений запас знань, що повинен бути практично використаний у відповідних процесах ЖЦ.

Для наочного подання понятійного апарата областей знань SWEBOK проведемо умовне розбиття областей на основні (п'ять для проектування ПС і додаткові організаційні методи й підходи, які відображають інженерію керування проектуванням ПС (конфігурацією, проектами, якістю - мал.).

Основні області знань SWEBOK

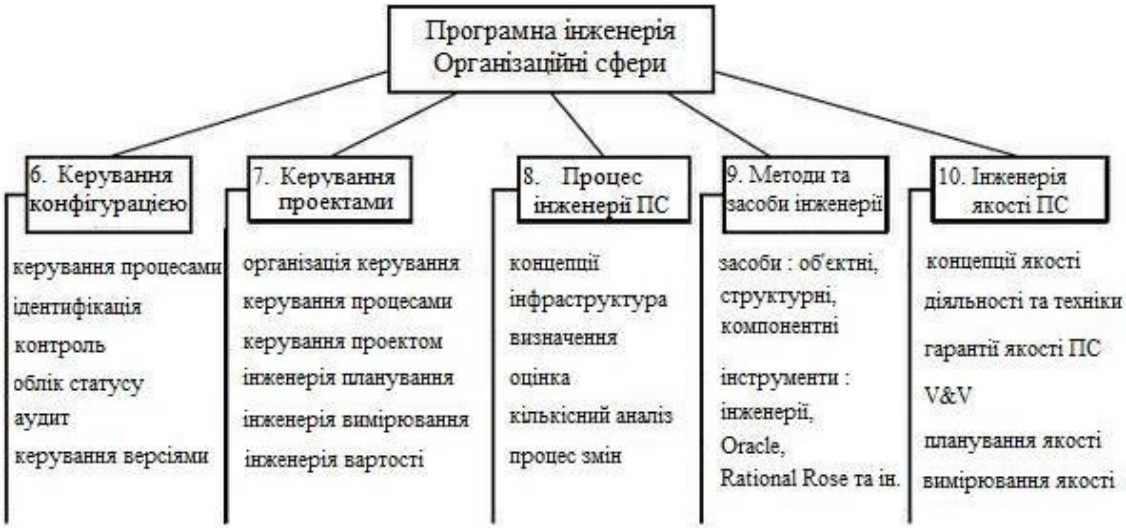


Рис. 1.1. Перелік областей знань SWEBOK

У кожній області наведені ключові поняття, підходи й методи проектування різних типів ПС. Дане розбиття областей на основні та допоміжні відповідає структурі розбиття процесів стандарту ISO/IEC 12207, виконання яких визначається знаннями, що містяться в ядрі SWEBOK. Далі приводиться огляд кожної області ядра знань SWEBOK, визначається її роль у проектуванні та реалізації програмних продуктів (ПП). У деяких підрозділах вказаний зв'язок з положеннями відповідних стандартів, які регламентують і регулюють виконання процесів проектування програмних систем.

Управління проектною діяльністю

Управління конструюванням - це керування процесом конструювання ПП, що базується на моделях конструювання, планування й внесення змін.

Моделі конструювання включають набір операцій, послідовність дій і результатів. Види моделей визначаються стандартом ЖЦ, методологіями й практиками. Деякі стандарти ЖЦ за своєю природою орієнтовані на конструювання, наприклад, екстремальне програмування (XP - eXtreme Programming). Конструювання за допомогою моделювання здійснюється в раціональному уніфікованому процесі - RUP (Rational Unified Process).

Планування складається у визначенні порядку операцій і рівня виконання заданих умов у процесі конструкторської діяльності. Визначається модель ЖЦ, що включає завдання й дії по створенню компонентів, а також їхньою перевіркою, включаючи досягнення показників якості на етапах ЖЦ. Виконавці розподіляються по процесах ЖЦ і виконують відповідні завдання по реалізації проміжного продукту. Потім проводиться вимір різних аспектів конструювання, наприклад, кількісна оцінка обсягу коду, оцінка ступеня використання reuse, обчислення ймовірності появи дефектів й оцінка кількісних показників якості ПП.

Внесення змін пов'язане з помилками, виявленими шляхом різного роду перевірок і тестувань, воно проводиться з метою збереження функціональної цілісності системи. У випадку виявлення помилок на етапі супроводу приймається рішення про зміну коду шляхом виправлення виявлених помилок або внесення змін у вимоги до ПП.

Проектування ПП (Software design) - це процес визначення архітектури, компонентів, інтерфейсів, інших характеристик системи й кінцевого складу

програмного продукту. Область знань "Проектування ПП (Software Design)" складається з наступних розділів:

- базові концепції проектування ПП (Software Design Basic Concepts),
- ключові питання проектування ПП (Key Issue in Software Design),
- структура й архітектура ПП (Software Structure and Architecture),
- аналіз й оцінка якості проектування ПП (Software Design Quality Analysis and Evaluation),
- нотації проектування ПП (Software Design Notations),
- стратегія й методи проектування ПП (Software Design Strategies and Methods).

Базова концепція проектування ПП - це методологія проектування архітектури за допомогою різних методів (об'єктного, компонентного й ін.), процеси ЖЦ (стандарт ISO/IEC 12207) і техніки - декомпозиція, абстракція, інкапсуляція й ін. На початкових стадіях проектування предметна область декомпозується на окремі об'єкти (при об'єктно-орієнтованому проектуванні) або на компоненти (при компонентному проектуванні). Для подання архітектури програмного забезпечення вибираються відповідні артефакти (нотації, діаграми, блок-схеми й методи).

До ключових питань проектування ПП відносяться: декомпозиція програм на функціональні компоненти для незалежного й паралельного їхнього виконання, принципи розподілу компонентів у середовищі виконання і їхньої взаємодії між собою, механізми забезпечення якості й живучості системи й ін.

При проектуванні архітектури ПП використається архітектурний стиль проектування, заснований на визначенні основних елементів структури - підсистем, компонентів і зв'язків між ними.

Архітектура проекту - багаторівневе подання структури системи й специфікація її компонентів. Архітектура визначає логіку окремих компонентів системи настільки детально, наскільки це необхідно задля написання коду, а також визначає зв'язки між компонентами. Існують й інші види подання структур, засновані на проектуванні зразків, шаблонів, сімейств програм і каркасів програмних середовищ.

Одним з найважливіших інструментів проектування архітектури є паттерн - типовий конструктивний елемент ПП, що задає взаємодію об'єктів (компонентів) проектованої системи, а також ролі й відповідальності виконавців. Основною мовою опису цього елемента є UML. Він може бути структурним, що включає типові композиції структур з об'єктів і класів,

об'єктів, зв'язків тощо; поведінковим, що визначає схеми взаємодії класів об'єктів й їхню поведінку діаграмами активності, взаємодії, потоків керування тощо; що породжує та відображає типові схеми розподілу ролей екземплярів об'єктів і способи динамічної генерації структур об'єктів і класів.

Аналіз й оцінка якості проектування ПП включає заходи щодо аналізу сформульованих у вимогах атрибутів якості, оцінки різних аспектів ПП - кількості функцій, структура ПП, якості проектування за допомогою формальних метрик (функціонально-орієнтованому, структурних й об'єктно-орієнтованих), а також проведення якісного аналізу результатів проектування шляхом статичного аналізу, моделювання й прототипування.

Нотації проектування дозволяють представити опис об'єкта (елемента) ПП і його структуру, а також поведінку системи. Існує два типи нотацій: структурна, поведінкові й безліч різних їхніх подань.

Структурні нотації - це структурне, блок-схемне або текстове подання аспектів проектування структури ПП з об'єктів, компонентів, їхніх інтерфейсів і взаємозв'язків. До нотацій відносяться формальні мови специфікацій і проектування: ADL (Architecture Description Language), UML (Unified Modeling Language), ERD (Entity-Relation Diagrams), IDL (Interface Description Language), Use Case Driven й ін. Нотації включають мови опису архітектури й інтерфейсу, діаграми класів й об'єктів, діаграми сутність-зв'язок, конфігурації компонентів, схем розгортання, а також структурні діаграми, що задають у наочному виді оператори циклу, розгалуження, вибору та послідовності. Поведінкові нотації відображають динамічний аспект поведінки систем й їхніх компонентів. Таким нотаціям відповідають діаграми потоків даних (Data Flow), таблиць прийняття рішень (Decision Tables), діяльності (Activity), кооперації (Colloboration), послідовності (Sequence), перед- і пост-умови (Pre-Post Conditions), формальні мови специфікації (Z, VDM, RAISE), мови проектування PDL тощо.

Стратегія й методи проектування ПП. До стратегій відносяться: проектування знизу-вверх, зверху-вниз, абстрагування, використання паттернів й ін. Методи - це функціонально-орієнтовані, структурні, які базуються на структурному аналізі, структурних картах, діаграмах потоків даних (Dataflow) тощо. Вони орієнтовані на ідентифікацію функцій та їхнє уточнення зверху-вниз, після чого уточнюються діаграми потоків даних і проводиться опис процесів.

Коло робіт, пов'язаних із розробкою алгоритмів і програм розв’язку

завдань, а також із підготовкою та проведенням розрахунків, називають *автоматизацією прикладної діяльності.* Відзначимо, що будь-яка конкретна прикладна діяльність, що автоматизується, характеризується двома чинниками.

По-перше, *предметною галуззю,* тобто сукупністю розв'язуваних прикладних задач і методів обчислення, що при цьому використовуються і, по-друге, *дисципліною роботи,* тобто системою правил, угод, технологічних підходів і прийомів, прийнятих під час розробки, налагодженні й експлуатації програм, у тому числі і під час проведення розрахунків.

У структурі ППП, регламентованій предметною галуззю та дисципліною роботи, можна виділити три основні компоненти: *функціональне наповнення*, *мову завдань* і *системне наповнення*. *Функціональне наповнення* відбиває специфіку предметної галуззі пакета, відображає властивості об'єкта автоматизації, його структуру, яка відображається сукупністю модулів. *Мова завдань пакета* є засобом спілкування користувача з пакетом. Вона дозволяє описувати послідовність виконання різноманітних операцій, що забезпечує розв'язок завдання, або постановку задачі, на основі якої ця послідовність будується автоматично. Крім того, мова завдань дає можливість формулювати запити, що стосуються інших видів робіт, які виконуються у межах прикладної діяльності. Можливий набір операцій, лексика і синтаксис мови завдань визначаються предметною галуззю, що обслуговується пакетом, і реалізованою ним дисципліною роботи. Саме через мову завдань користувач сприймає й оцінює, які є послуги обчислення пакета і наскільки зручні вони у використанні.

Системне наповнення являє собою сукупність програм, що забезпечують виконання завдань і взаємодію користувача з пакетом. Системне наповнення організує використання потенціалу знань, закладених у функціональне наповнення, відповідно до можливостей, що передбачені в мові завдань пакета.

Організація процесу управління розробкою ПП

Заходи по організації процесу розробки програмного продукту спрямовані на забезпечення належної якості.

Основними причинами незадовільної якості є:

1. Неповне та нечітке формулювання вимог до програмного засобу.
2. Недостатнє залучення користувачів до робіт над проектом.
3. Відсутність потрібних ресурсів.
4. Незадовільне планування.
5. Часта зміна вимог та специфікацій.

6. Відсутність грамотного управління процесом.
7. Недостатня підтримка збоку керівництва.

В процесі історичного становлення технології проектуванні IT-систем був розроблений SWEBOK який регламентує всі напрямки даної галузі. Процес організації розробки програмних систем на високому рівні потребував визначення певних моделей, щоб визначали характеристики професійного зростання установ, що займаються проектуванням. Була запропонована, у 1987 році, модель оцінки зрілості технологічних процесів в установах програмування, яка отримала назву CMM.

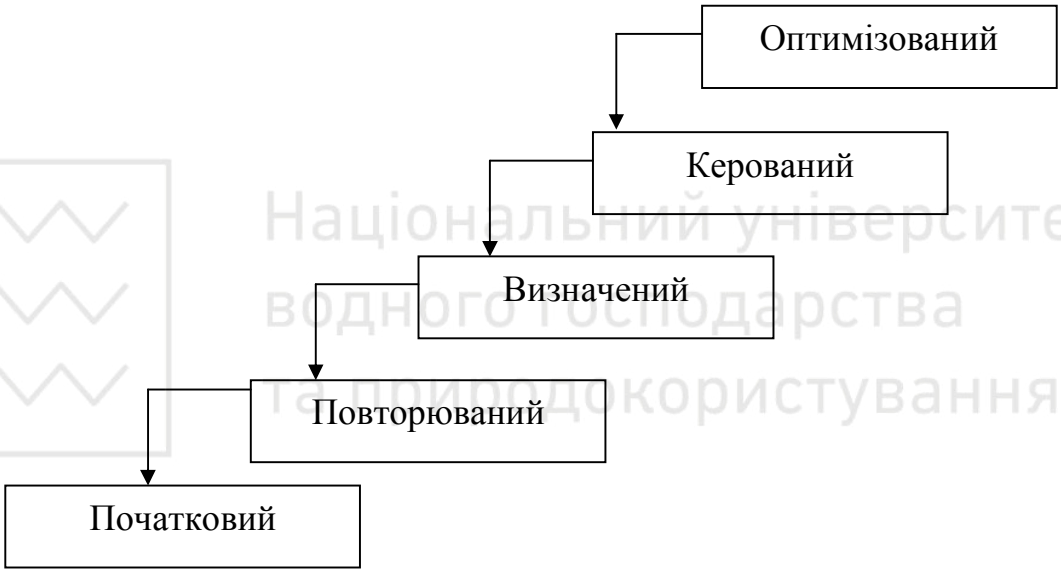


Рис. 1.2. Модель оцінки зрілості технологічних процесів в установах програмування (CMM)

Перший **початковий** рівень характеризується спонтанністю, а іноді і хаотичністю процесів розробки ПП.

Другий рівень, що отримав назву **повторюваний** характерний тим, що в установі здійснюється використання основних процесів управління, що дає можливість визначати та відстежувати функціональні характеристики ПП, розробляти план-графік робіт та на перед визначати їх вартість.

Третій рівень є **визначений** і визначається процесами документування, стандартизації прийнятих рішень. Визначення чітких прийомів управління і врешті решт напрацюванням своїх стандартів процесу розробки ПП.

Керований рівень. Установи цього рівня в наслідок напрацьованого досвіду можуть застосовувати по детальні виміри якості процесу виробництва і за необхідності приймати управлінські рішення для його покращення.

Оптимізований рівень - найвищий рівень зрілості організації. На цьому рівні безперервно здійснюється покращення процесу розробки, яке базується на кількісних показниках виконаних та виконуючих проектів, а також на впровадженні нових ідей та технологій.

Для того, щоб досягнути другий або більш вищий рівень організації установки необхідно визначити певні **ключові процеси**:

1. На рівні оптимізації: керування змінами, процеси застосування в новітніх технологіях.
2. На керованому рівні: управління якістю процесу, вимірювання і аналіз процесу;
3. На визначеному рівні: рецензування та обговорення з кологами результатів роботи, координація та взаємодія між проектами, підвищення кваліфікації робітників, визначення організаційних проектів, індустріалізація підходу до проектування, інтегроване управління.
4. На повторюваному рівні: управління конфігурацією (версіями програми), забезпечення якості, управління роботою субпідрядників, контроль за виконанням проекту, планування проекту, управління вимогами до програмного продукту.

На початковому рівні ключових процесів не виділяють.

Розглянемо приклад (рис 1.3) структури СММ ключового процесу планування програмного проекту для рівня зрілості організації – повторювальний.

Процес управління конфігурацією

1) визначення стану компонентів програмного продукту;
2) управління модифікаціями програмного продукту;
3) опис і підготовка звітів про стан компонентів програмного продукту та запитів на модифікацію;
4) забезпечення повного обсягу сумісності та коректності компонентів програмного продукту;
5) управління зберігання і постачанням програмного продукту.

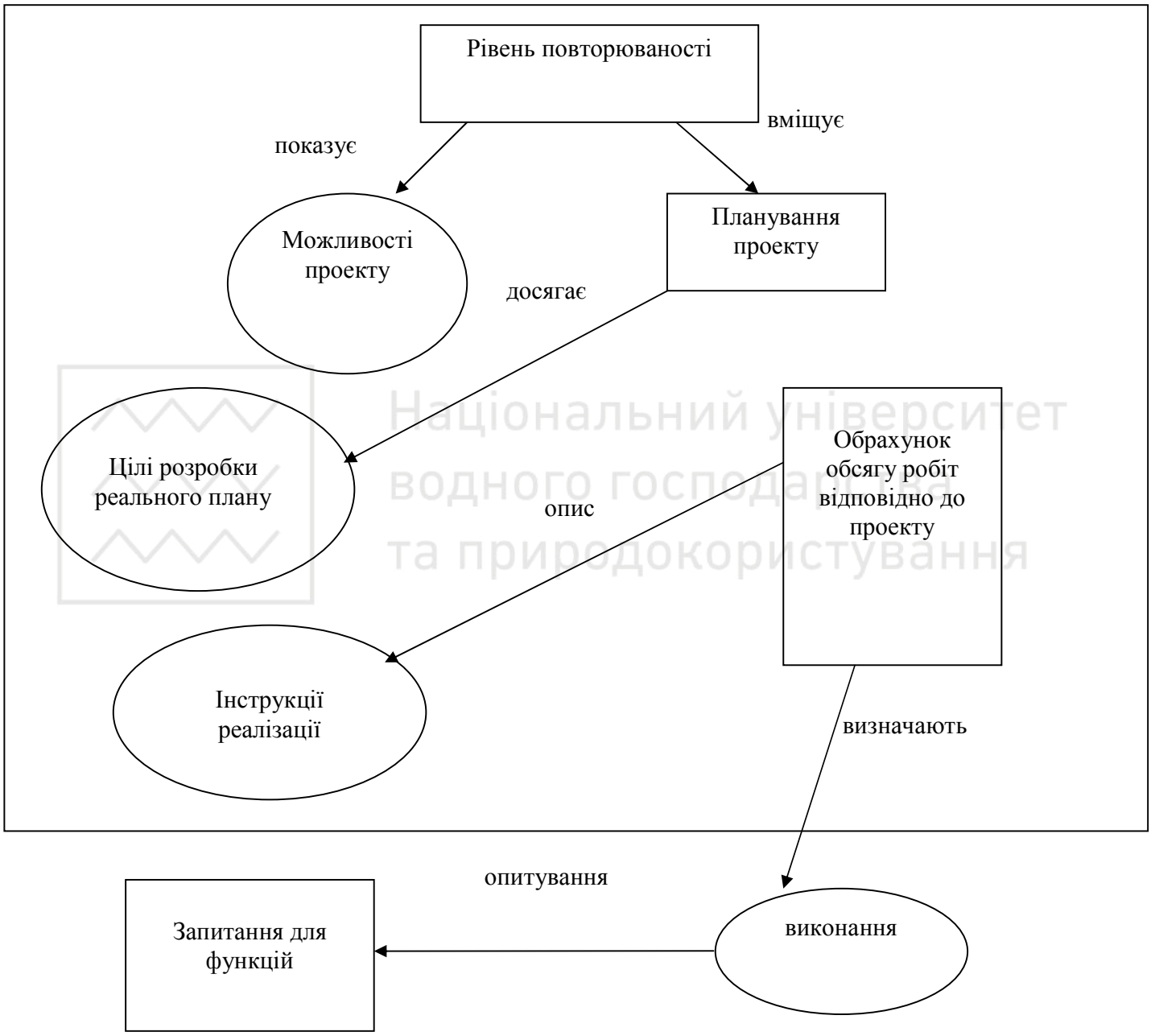


Рис. 1.3. Приклад структури CMM ключового процесу планування програмного проекту для рівня зрілості організації - повторювальний

Тема 2. Життєвий цикл ПП

1. Поняття про життєвий цикл процесу розробки програмного засобу.
2. Каскадна модель життєвого циклу.
3. V-подібна модель життєвого циклу.
4. Модель прототипів.
5. Модель швидкої розробки.
6. Модель життєвого циклу – багатьох проходів.
7. Спіральна модель життєвого циклу.

Поняття про життєвий цикл процесу розробки програмного продукту

Життєвий цикл – це період часу починаючи з моменту прийняття рішення про необхідність створення програмного продукту до моменту його повного вилучення з експлуатації.

Структура ЖЦ складається з процесів дій та завдань, що повинні бути виконані, визначені і регламентуються міжнародним стандартом ISO/IEC 12207.1995 року.

Три базові групи всіх процесів ЖЦ

1. Основні процеси.
2. Допоміжні (підтримуючі) процеси.
3. Організовані процеси.

Основні процеси включають в себе набір певних дій і пов'язаних з ними задач, що повинні бути розв'язані на протязі ЖЦ ПП. До основних відносяться процеси придбання, постановки, розробки, експлуатації та супроводу.

Процес придбання охоплює дії замовника по придбанню програмного продукту. До таких дій відносять:

1. ініціювання придбання;
2. підготовка пропозицій на замовлення;
3. підготовка і корегування договору;
4. нагляду за діяльністю постачальника;
5. приймання і завершення робіт.

Допоміжні процеси в ЖЦ.

Основною метою допоміжних (підтримуючих) процесів є створення програмного продукту, що є надійним і повністю задовольняє вимоги замовника. Допоміжними процесами є:

- процес документування;
- управління конфігурацією;
- процес забезпечення якості;
- процес верифікації;
- процес атестування;

- процес спільного оцінювання;
- процес аудиту;
- процес розв'язання проблем.

Процес управління конфігурацією

1. визначення стану компонентів програмного продукту;
2. управління модифікаціями програмного продукту;
3. опис і підготовка звітів про стан компонентів програмного продукту та запитів на модифікацію;
4. забезпечення повного обсягу сумісності та коректності компонентів програмного продукту;
5. управління зберігання і постачанням програмного продукту.

Процес забезпечення якості

Забезпечує відповідні гарантії того, що програмний продукт і процеси його ЖЦ відповідають заданим вимогам та затвердженим планам.

Для отримання достовірних оцінок програмного продукту та процесів його ЖЦ оцінювання повинно здійснюватись незалежно від суб'єктів безпосередньо задіяних у розробці програмного продукту.

Процес верифікації

Полягає у доведенні того, програмний продукт повністю задовольняє вимогам або умовам, що залежать від попередніх дій. Процес верифікації здійснюється самим виробником або фахівцями сторонніх профільних організацій. Процес верифікації складається з підготовчої роботи і власне верифікації.

Життєвий цикл розробки програмного продукту. Організаційні процеси

Процес управління складається з дій та завдань, що можуть бути виконані як розробником так і замовником в процесі управління своїми процесами.

Процес створення інфраструктури охоплює дії з вибором підтримки супроводу технології стандартів та інструментальних засобів, вибір та встановлення апаратних та програмних засобів, що будуть задіяні в процесі розробки експлуатації та супроводу ПП.

Процес вдосконалення передбачає оцінку вимірювання контроль та вдосконалення процесів життєвого циклу програмного продукту.

Процес навчання охоплює початкове та регулярне навчання по підвищенню кваліфікаційного рівня персоналу.

Взаємозв'язок між процесами життєвого циклу ПП

Процеси життєвого циклу ПП регламентуються стандартом ISO/IEC 12207. Ці процеси можуть бути використані при виконанні конкретних проектів. Стандартом також передбачено деякий набір взаємозв'язків між процесами з різних поглядів.

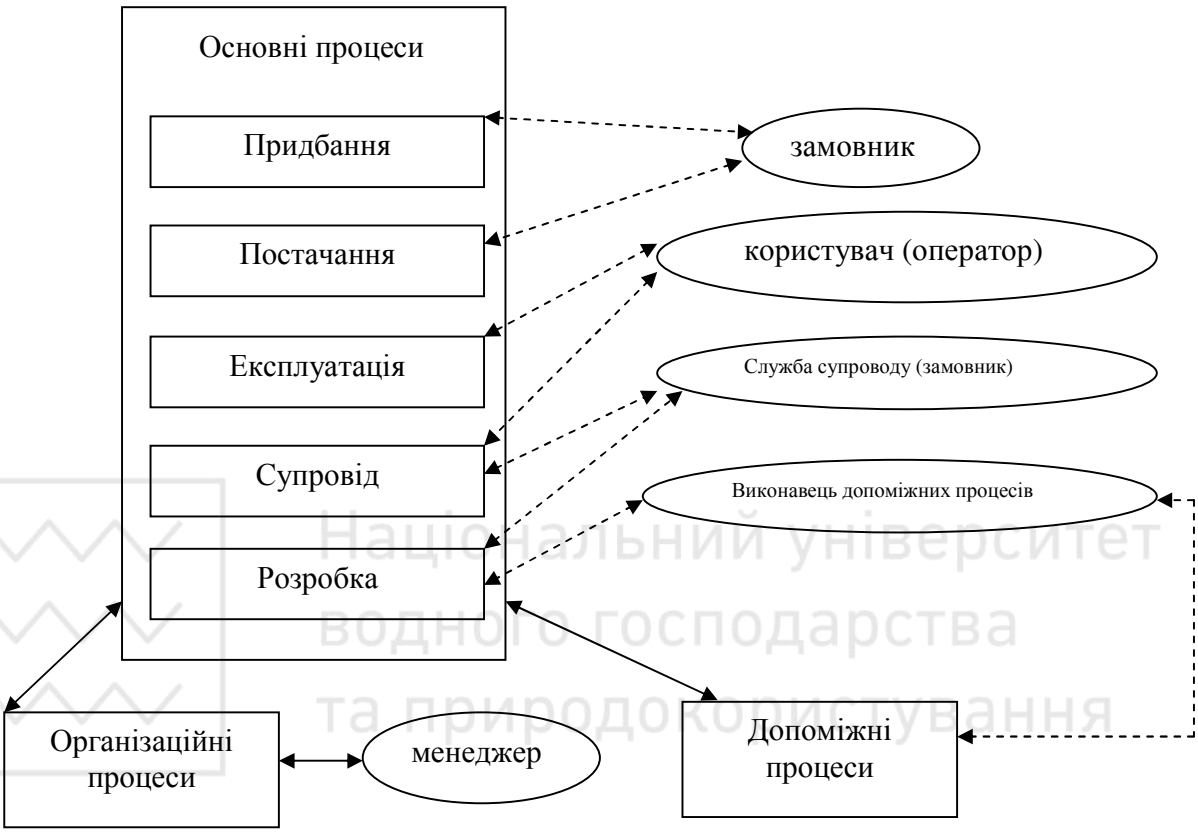


Рис. 2.1.

Штрихові лінії показують зв’язок діючих осіб з конкретними процесами, а суцільні лінії – зв’язок процесів між собою.

У договірному аспекті замовник і постачальник входять у договірні відносини та реалізують відповідні процеси придбання і постачання.

В аспекті управління замовник, постачальник, розробник, оператор, служба супроводу та інші сторони здійснюють управління своїх процесів.

В аспекті експлуатації оператор, що експлуатує систему надає потрібні послуги користувачам.

У інженерному аспекті розробник або служба супроводу розв’язують відповідні технічні завдання в процесі розробки або модифікації програмного продукту.

У аспекті підтримки служби, що реалізують допоміжні процеси, надають потрібні послуги усім іншим учасникам робіт.

Моделі життєвого циклу розробки програмного продукту

Під моделлю ЖЦ розробки програмного продукту розуміється структура, що визначає послідовність виконання та взаємозв'язок між процесами, діями та завданнями, які виконуються під час ЖЦ у програмному продукті.

Вид моделі ЖЦ залежить від специфіки та складності проекту, а також від умов, в яких створюється і буде функціонувати програмний продукт.

Модель ЖЦ конкретного програмного продукту визначає характер процесу його створення і відображає сукупність впорядкованих у часі етапів робіт.

Найбільш розповсюдженими є такі моделі ЖЦ програмного продукту:

1. **Каскадна модель** – це прямолінійна і найбільш проста для застосування модель. В процесі її реалізації потрібно здійснювати жорсткий контроль за ходом робіт. Програмний продукт, що розробляється під час реалізації моделі є недоступним для внесення змін.
2. **V-подібна модель** є найбільш простішою за попередню, особливу увагу в ній приділяють тестуванню та порівнянню результатів фаст-тестування та проектування.
3. **Модель прототипів.** За цією моделлю створюється досить швидко часткова реалізація системи, яка і визначається як прототип. В процесі розробки за цією моделлю зв'язок між замовником забезпечується більш ефективно на стадіях оцінки прототипів.
4. **RAD-модель**. (швидка розробка додатків). Основні ознаки: проектні групи (три-сім чоловік), що складаються з висококваліфікованих фахівців, здійснюють тісний взаємозв'язок з замовником. Це дає можливість значно зменшити цикл розробки програмного продукту.
5. **Модель багатьох проходів.** Це швидка структура для створення працездатних систем. Зменшується можливість внесення змін в процесі розробки. Основним недоліком є неможливість переходу від поточної реалізації до нової версії.
6. **Спіральна модель** охоплює каскадну модель, розділяє фази розробки на більш дрібні частини, що забезпечує гнучкість при проектуванні.

Каскадна модель життєвого циклу

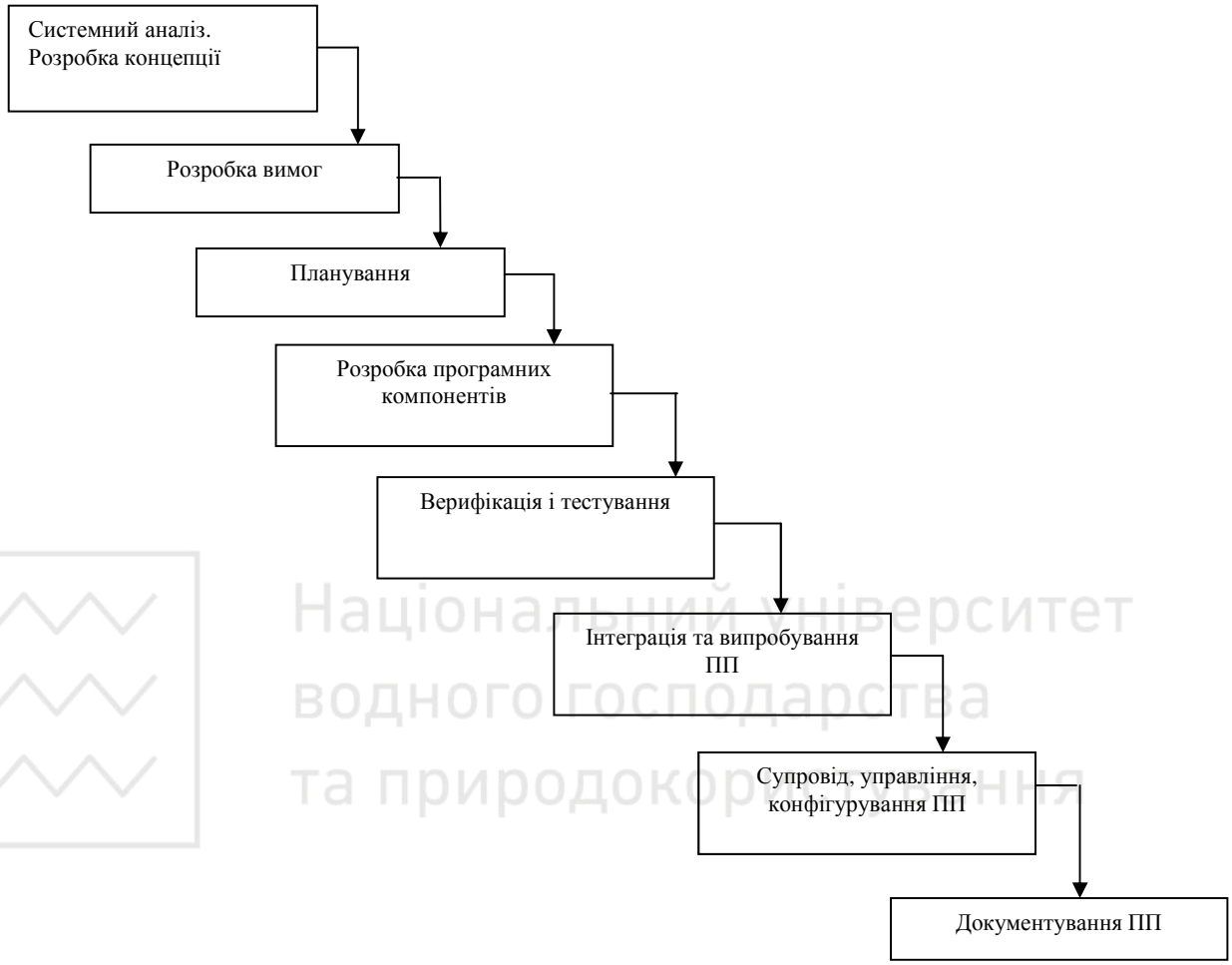


Рис. 2.2. Етапи каскадної моделі

Є найбільш вживаною моделлю, яка реалізовує принцип одноразового виконання кожного з базових процесів та етапів в їх часових межах.

Переваги:

1. На кожному етапі формується завершений набір проектної документації.
2. Стадії, що виконуються в логічній послідовності дозволяють чітко спланувати терміни завершення робіт і відповідні витрати.

Недоліки:

1. Суттєве запізнення з отриманням кінцевого результату, а також, як наслідок, підвищення ризиків створення програмного продукту, який повністю не задовольняє вимоги замовника.
2. Вносити суттєві зміни до проекту можна лише після того, як буде завершено проектування.

V - подібна модель життєвого циклу

Є різновидом каскадної моделі, у якій особливе місце приділяється верифікації та атестації програмного продукту. Дана модель розкриває, що тестування програмного продукту проходить погодження та обговорюється, проектується та планується вже починаючи з перших етапів життєвого циклу.

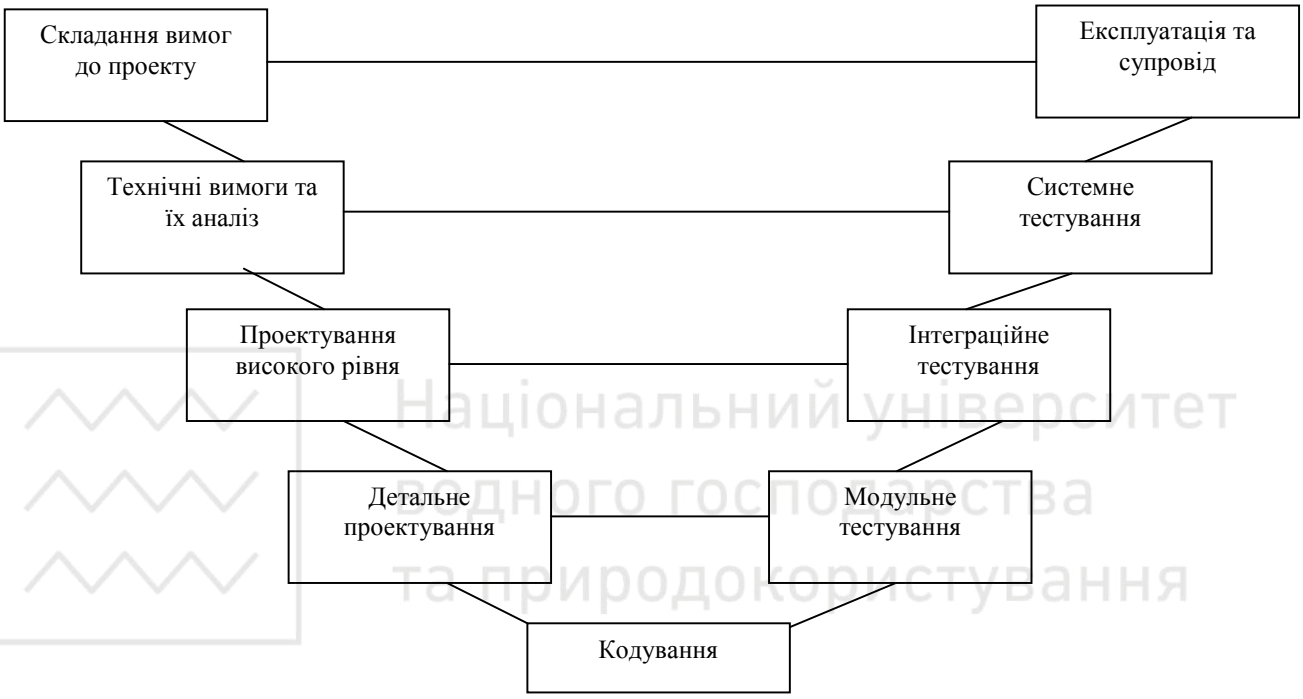


Рис. 2.3. Етапи V - подібної моделі

На моделі дуже добре проглядаються взаємозв'язки між аналітичними фазами та фазами проектування, що передують кодуванню та тестуванню.

Переваги:

1. Значна увага приділяється верифікації та атестації, починаючи з ранніх етапів проектування, що в свою чергу дає можливість передбачити потрібну якість ПП на виході.
2. Хід виконання робіт може легко відстежуватись, так як завершення кожної фази є контрольною точкою.

Недоліки:

1. Не враховуються ітерації між фазами.
2. Не можна вносити зміни на різних етапах життєвого циклу.
3. Тестування вимог здійснюється відносно пізніше, що впливає на виконання графіку робіт.

Модель прототипів

Дозволяє створювати прототип до або під час етапу складання вимог до програмного продукту.

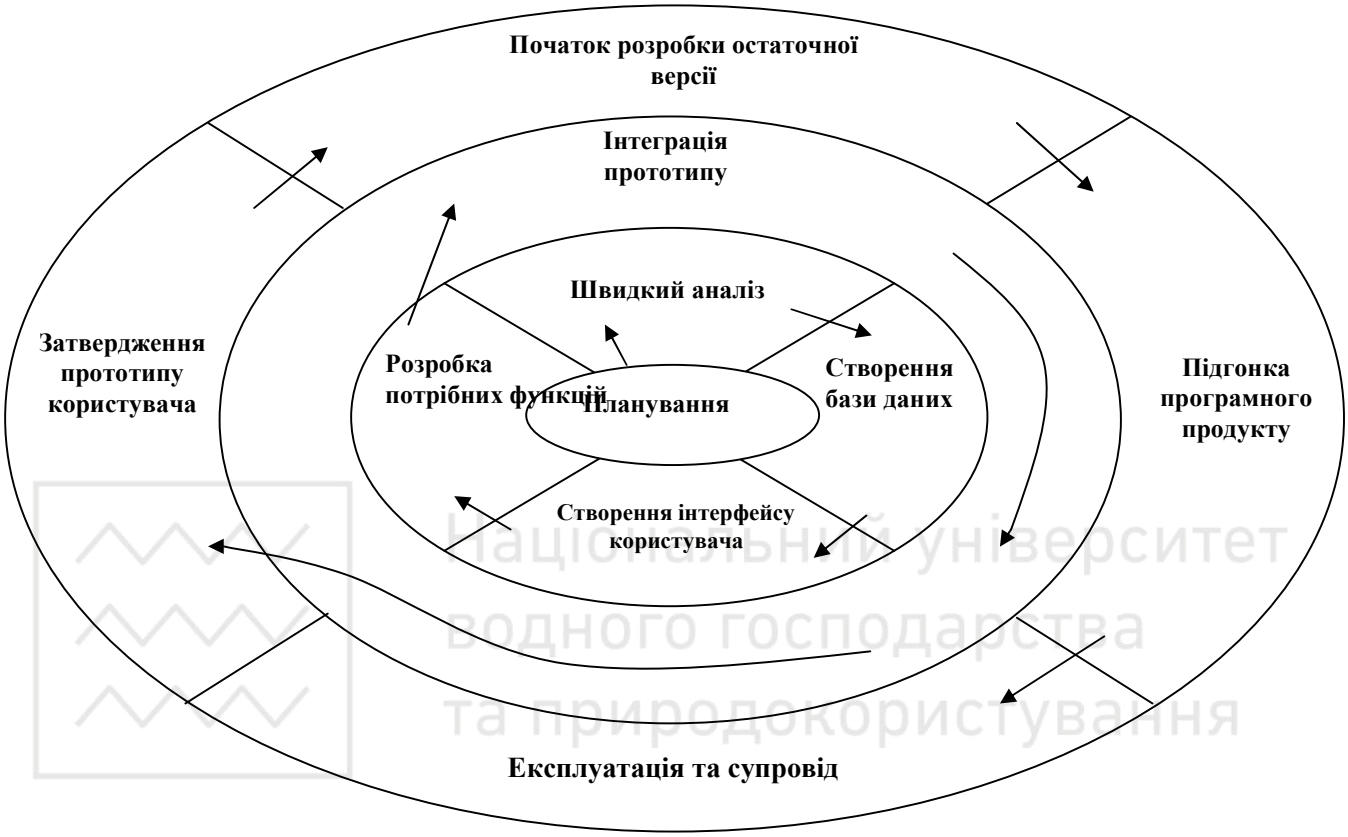


Рис. 2.4. Етапи моделі прототипів

За результатами визначення прототипу розробник надає замовнику готовий прототип, а користувач замовник оцінює рівень повноти виконання заданих функцій і підтверджує застосування прототипу або відхиляє. Цей процес продовжується до тих пір поки користувач замовник не буде задоволений степенем відповідності запропонованого програмного продукту.

Переваги:

1. Безпосередній зв'язок із замовником на етапі затвердження прототипу дозволяє звести до мінімуму кількість невідповідностей вимогам програмного продукту.
2. У процесі розробки завжди можна врахувати нові вимоги замовника.
3. Замовник бачить прогрес в процесі розробки, що створює позитивний імідж розробника.

4. Ймовірність виникнення непорозумінь між замовником та розробником значно зменшується.

Недоліки:

1. Розв'язок складних задач у процесі проектування на майбутнє.
2. Замовник може зупинитись на певному прототипі, що в свою чергу не дозволить розробнику отримати більш якісний програмний продукт.
3. Знаходження прототипу може безпідставно бути розтягнуто в часі.
4. На початку робіт важко передбачити кількість ітерацій, що потрібно буде використати, що ускладнює процес планування і визначення остаточних термінів завершення проекту.

Модель швидкої розробки (або RAD– модель)

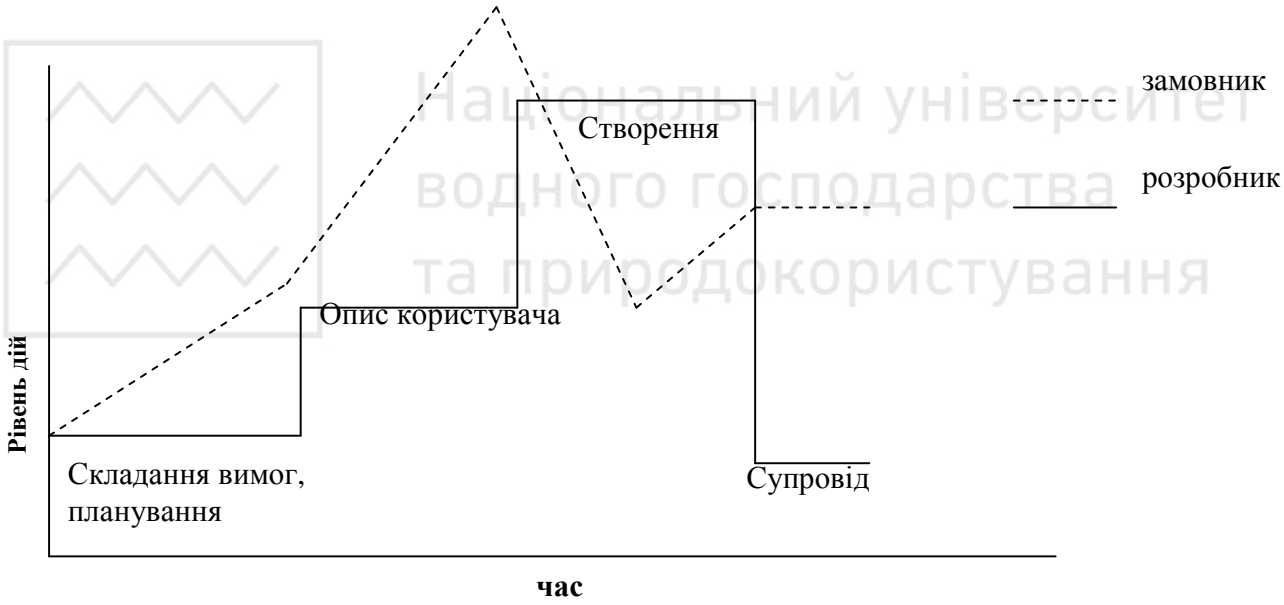


Рис. 2.5.

Переваги:

1. Застосування сучасних інструментальних засобів дозволяє значно скорочувати термін розробки.
2. Залучення до роботи замовника значно знижує ризики того, що він буде незадоволений готовим програмним засобом.
3. За цією моделлю можливе багаторазове застосування компонентів уже існуючих програм.

Недоліки:

1. У разі, якщо замовник не в змозі постійно приймати участь у процесі розробки, то це може негативно відбитись на ПП.
2. Для виконання такого проекту потрібні висококваліфіковані програмісти, які не тільки можуть виконувати своє завдання, а і користуватись сучасними інформаційними засобами.
3. Існує ризик, що робота над програмним продуктом може ніколи не завершитись внаслідок відсутності домовленості про результат між замовником та розробником, в цьому випадку потрібно всім вміти вчасно зупинитись.

Модель життєвого циклу – багатьох проходів

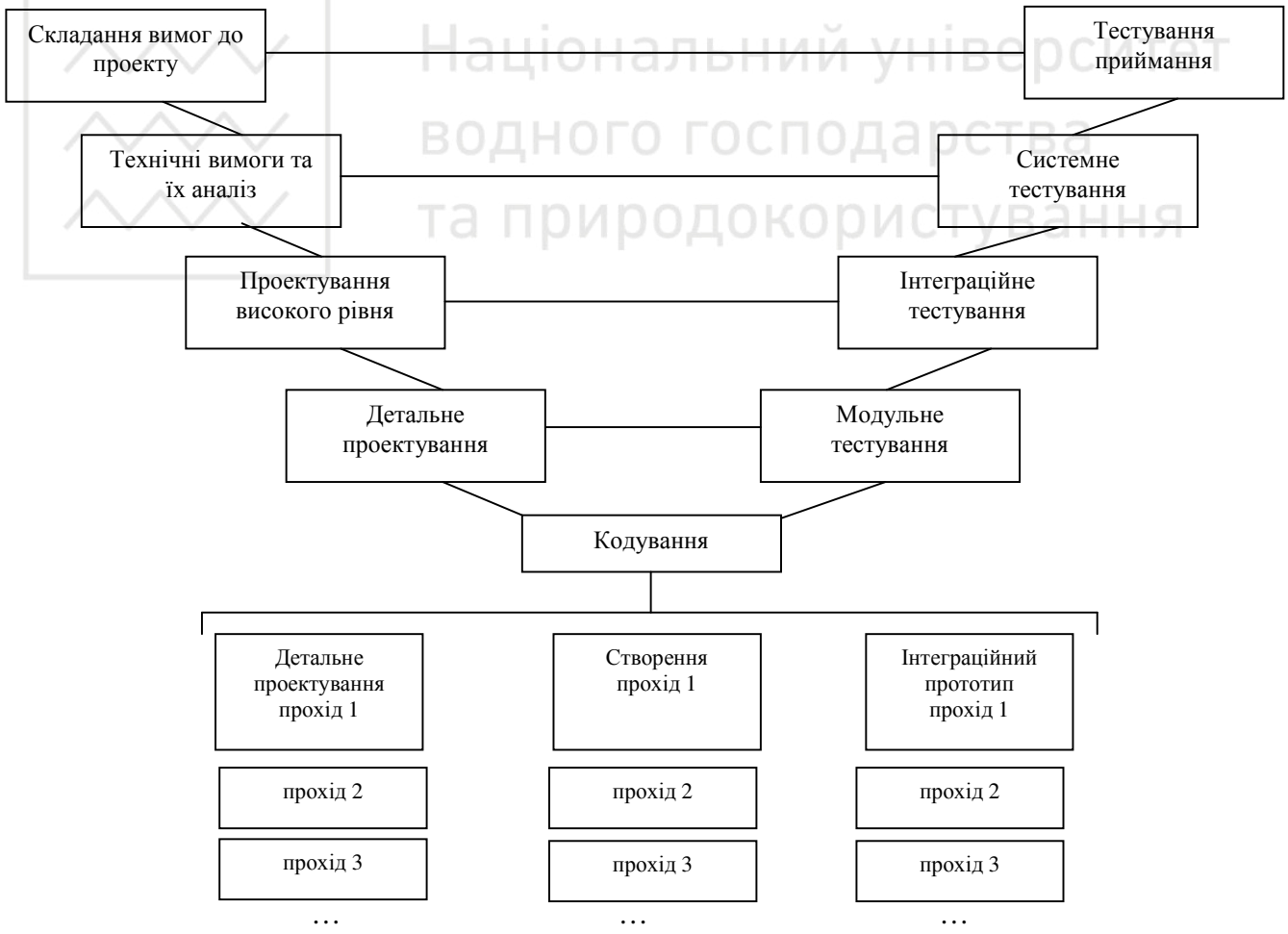


Рис. 2.6.

Модель багатьох проходів являє собою декілька ітерацій процесу побудови прототипу програмного продукту коли на кожній наступній ітерації відбувається добавлення нових функціональних можливостей, або підвищення ефективності програмного продукту. За цією моделлю вважається, що на ранніх етапах життєвого циклу (планування, аналіз вимог, розробка проекту) виконується конструювання програмного продукту в цілому, тоді і визначається кількість потрібних інкрементів і тих функцій, що будуть додані. Кожен з інкрементів проходить через фазу життєвого циклу, що залишились (кодування та тестування). За даною моделлю спочатку виконується конструювання, тестування базових функцій, що складають основу, наступні ітерації направлені на покращення функціональних можливостей даного продукту.

Переваги:

1. На початку розробки потрібні засоби тільки для реалізації базових функцій програмного засобу.
2. Після кожного інкременту ми отримуємо функціональний продукт.
3. Покращується розуміння як розробника так і замовника вимог до наступних ітерацій програмного продукту.

Недоліки:

1. Може виникнути тенденція відтягування рішення складної задачі.

Спіральна модель життєвого циклу

Для подолання проблем з використанням моделі багатьох проходів у середині 80 років була запропонована спіральна модель. Її особливість полягає в тому, що програмний продукт створюється по частинах з використанням методу прототипів.

Тема 3. Командна організація проектування ІТ-систем

План:

1. Основні засади командної організації проектування
2. Етапи формування проектних груп.
3. Склад команди для проектування ІС.

Основні засади командної організації проектування

Склад команди залежить від характеристик проекту, а саме від його масштабу та складності. Проектна команда може існувати тільки у межах кожного окремого підприємства, формувати тільки частину проектної організації. Кожен з виконавців є повноправним членом проектної команди та може залучатись до проектних робіт на певний проміжок часу.

Людина-фахівець – головна фігура проекту. Будь-який проект з будь-яким матеріальним і фінансовим забезпеченням, що виконується не достатньо кваліфікованими виконавцями, стає неефективним. Тому на проект-менеджера покладаються особливі вимоги. Керівник проекту повинен детально знати всі життєві фази проекту. Проте найважливіша сфера його діяльності – це ефективна співпраця з великою кількістю людей: замовником, членами команди, працівниками фірми, навколишнім середовищем прямого та непрямого впливу.

Проект-менеджер має розбиратися в людях, оцінювати й передбачати, чого в тій чи іншій ситуації можна від них очікувати. Такі знання допомагають проект-менеджеру знайти контакт із членами команди, працівниками фірми, замовниками й іншими учасниками проекту. Розуміння психології людей дасть йому змогу зайняти правильну позицію на переговорах, нарадах, у конфлікті.

Поведінською основою спілкування людей і їхніх вчинків у різноманітних ситуаціях є психологічна характеристика особистості. Для її визначення існує багато тестів із практичної психології. Найпоширеніші серед них тести соціоніки, які поділяють людей за проявами їхніх емоцій на екстравертів та інтровертів. В екстравертів усе "написано на обличчі", вони так виражають свої почуття (словами, жестами, мімікою), що співрозмовнику дуже легко визначити їхню реакцію на свої слова та дії. Інтроверти, навпаки, дуже скупо виражають свої емоції, вони ніби звернені всередину себе; їхню реакцію, як правило, неможливо розпізнати одразу. Тільки психолог за певними ознаками може встановити, як сприйнято його інформацію і що можна очікувати у відповідь.

Мета створення проектної команди і завдання проект-менеджера

Як зазначалося, характерною ознакою управління проектом є наявність постійної проектної команди. Проект-менеджер і його команда мають бути єдиним цілим і працювати взаємоузгоджено. Створення кваліфікованої команди для нового проекту – один з основних обов'язків проект-менеджера на першому етапі його роботи. Цей процес потребує навичок керування щодо добору й об’єднання в команду спеціалістів з різних відділів і організацій.

Формуючи команду, проект-менеджер намагається об’єднати її членів загальною метою та завданнями. Створення нової команди для проекту утруднене ще й тим, що добрані спеціалісти раніше не працювали разом, не мають загальних цінностей та норм однак при цьому повинні працювати ефективно. Потрібен час для того, щоб всередині групи виникло відчуття команди, сформувалися загальні норми, стандарти й цінності.

Проект-менеджер повинен сприяти процесу перетворення групи спеціалістів на команду. Насамперед він повинен створити сприятливий психологічний клімат у команді для якнайшвидшої адаптації її учасників у проекті.

Проект-менеджер повинен вирішити основні організаційні проблеми: створити професійно стимулююче оточення; забезпечити групу кваліфікованим технічним персоналом; заручитись підтримкою керівництва; сформувати стабільно сприятливе навколишнє середовище.

Етапи формування проектних груп

Зазвичай проектна команда переживає п'ять стадій. Розглянемо їх.

1. Формування. Результативність команди на цьому етапі низька, тому що її члени ще не знайомі й не впевнені один в одному. Основні труднощі й перші "підводні камені" на етапі формування команди можна сформулювати в такому вигляді : особисті відчуття працівників, пов'язані з визначенням їхнього місця в команді та місця тимчасової команди всередині фірми.

На цьому етапі проект-менеджер має прикласти зусилля для подолання "підводних каменів" і об’єднання команди з орієнтацією на основну мету проекту.

2. «Притирання» учасників. Коли члени команди починають працювати разом, вони розуміють, що застосовують різні підходи й методи в роботі над проектом. Такі розбіжності можуть спричинити суперечки й навіть конфлікти, що не сприяє підвищенню ефективності команди.

Загальні проблеми на цьому етапі можна визначити так: "борсання" без просування вперед; перекладання повноважень і відповідальності; зіткнення характерів (властолюбних учасників, неформальних лідерів, "мильних бульбашок", ледарів); суперечки з будь-якого приводу чи навпаки: прийняття будь-яких думок без заперечень. Проте поступово в разі вмілого керівництва на основі загальних цінностей і норм у команді формуються ділові та дружні відносини.

3. Нормальне функціонування. Отже, вирішено всі суперечки й конфлікти. Кожний член команди зрозумів свою роль і місце в колективі, де він працюватиме протягом життєвого циклу проекту. На двох перших етапах у групі формується командне почуття, що вкрай необхідно для досягнення мети. Це дає основу, на якій команда може продуктивно працювати. Третій етап - найтриваліший і найрезультативніший для проекту, і проект-менеджер повинен використовувати його максимально.

4. Реорганізація. Проект-менеджер змінює кількісний та якісний склад команди з кількох причин: внаслідок зміни обсягів і видів робіт, заміни деяких працівників через їхню непридатність, залучення нових спеціалістів, запрошення тимчасових експертів тощо.

5. Розформування команди. Після завершення проекту команду розформовують. При цьому можливі дві ситуації. У разі ефективності проекту й відповідної мотивації члени команди відчувають задоволення від своєї роботи і сповнені бажання працювати разом й далі. У разі невдалого виконання проекту члени команди залишають її з відчуттям моральної незадоволеності. Завдання проект-менеджера – домагатися першої ситуації, тобто створити ефективну команду.

Склад команди для проектування ІС

Програмісти або розробники ІС мають свої етапи професійного зростання. Зазвичай це Junior Developer (молодший програміст), Middle Dev (чи, як говорять куди частіше Developer – просто розробник) і Senior Developer (провідний розробник). Дамо характеристики основних “гравців” команди розробників ІС.

Junior Developer

Це молодий розробник з малою кількістю досвіду або зовсім без такого, який тільки почав роботу в обраній технологічній області. Навчається, як

правило, по відеокурсах і відеоуроках, причому постійно адже досвіду розробки у нього так мало, що він дуже багато часу витрачає на консультації з досвідченішим розробником і розумінням того як команда працює над проектом. Накопивши досить досвіду і реалізувавши декілька проектів, junior переходить на middle рівень – стаючи повноцінним розробником. Середній термін росту – 1,5-2 року, щоб зайняти позицію повноцінного розробника. Вимоги, що, зазвичай ,виставляються до Junior розробника, приблизно такі, як і у Developer і Senior.

Це наступні знання:

- основи програмування (системи числення, різниця між оператором і операцією, розуміння алгоритмів, методик ООП);
- мови: C++, C#, Java, JavaScript, PHP (синтаксис, ООП можливості, багатопоточність, стандартні бібліотеки, паттерни проектування);
- OOP і OOD (парадигми);
- бази даних (MYSQL, JDBC, мова SQL, MSQL, SQL lite тощо);
- фреймворки (наприклад, YII, веб-сервис або Spring);
- володіти технологіями створення розподілених систем (DCOM);
- тенденції розвитку ІС від суцільних додатків до компонентних систем (рис. 3.1).

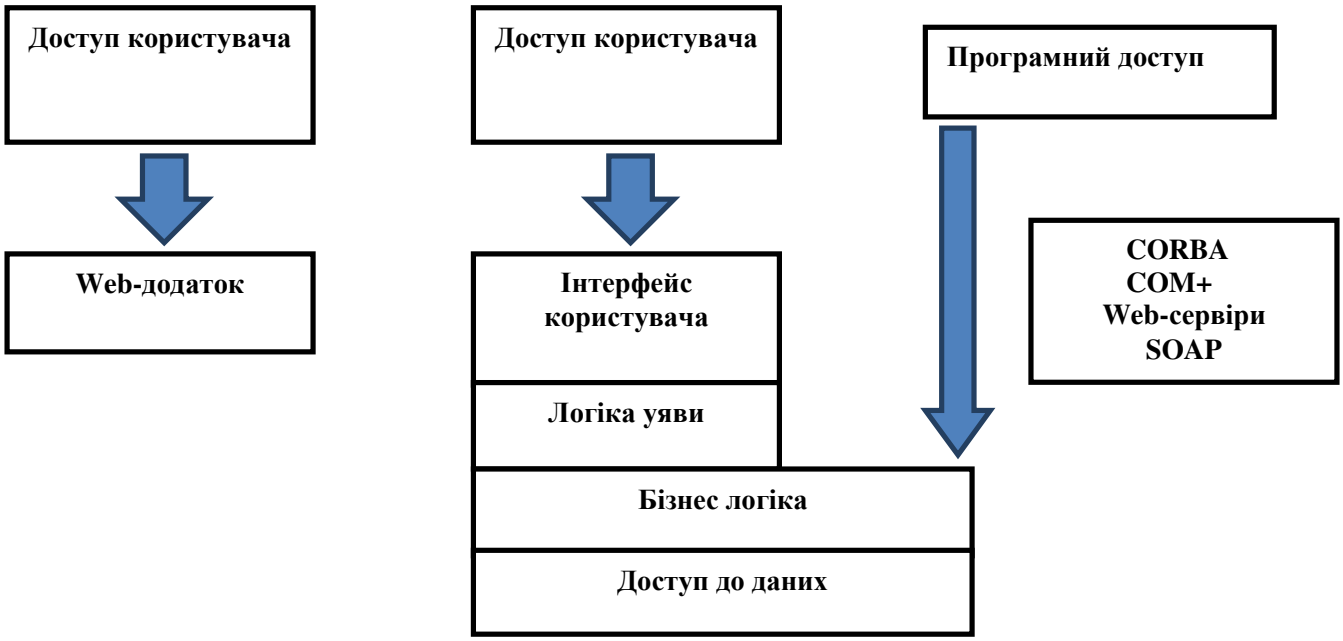


Рис. 3.1. Від суцільних додатків до компонентних систем

Junior швидко пише прототипи, але дуже довго доводить до реальної працездатності через малу кількість досвіду, грішить помилковою ініціативою. Йому ставлять задачу та пояснюють як її потрібно розв’язати.

Для **Junior** розробників важливо показати, що вони уміють вчитися і при цьому вирішувати поставлені завдання. Мислити в межах використовуваних технологій, читати спецлітературу, проходити відеокурси та відеоуроки.

Middle dev (розробник)

Це людина, відповідальна за якісне і своєчасне виконання розробки інформаційно-програмних систем, заснованих на застосуванні сучасних програмних технологій. Програміст виконує завдання по написанню і базовому тестуванню доручених йому компонентів системи, працює **developer** по зовнішніх специфікаціях. Уміє вирішувати поставлені завдання самостійно, розуміє основи побудови архітектури проектів, мислить не категоріями мови, від яких, нарешті, відв'язався, але поняттями предметної області, уміє писати структурно і послідовно, думає про майбутню підтримку продукту. Підтримує Junior розробників, займається як архітектурою проектів, так і модульною реалізацією, забезпечує реалізацію працездатності прототипів, постійно займається самоосвітою, розуміє Software Engineering Process.

Middle dev повинен:

- знати спец мови презентаційного рівня (HTML, PHP, JavaScript);
- знати спец мови рівня логіки предметної галузі (Java, C++, C#);
- знати спец мови рівня даних (SQL);
- розуміти технології web-серверів і серверів додатків;
- володіти технологіями створення розподілених систем (DCOM);
- володіти клієнтськими і серверними технологіями;
- знати роботу браузеру;
- знати СУБД;
- вміти розв’язувати проблеми безпеки та проблеми роботи у гетерогенному середовищі;
- знати операційні системи;
- застосовувати офісні пакети;
- працювати в середовищах розробки IDE;
- спілкуватись технічною англійською мовою.

Основний критерій відбору для **Middle dev** – це наявність знань і досвіду. Він повинен самостійно вирішувати задачі, що йому ставлять. Може

використовувати інформаційні ресурси для developers:

- Github.com - професійний ресурс для розробників;
- ITVDN.com - портал відео IT навчання;
- MSDN.com - продукти і технології Microsoft;
- Codefor.com - форум розробників.

Senior developer (провідний розробник-програміст)

Це член команди, що є відповідальним за якість і своєчасність робіт по розробці інформаційно-програмних систем, заснованих на застосуванні новітніх програмних технологій. Має глибокі, структуровані знання і працює усередині проектної команди, абсолютно не маючи необхідності контактувати з представниками менеджменту замовника.

Виконує такі роботи, як:

- відповідає за проект;
- підготовлює інфраструктуру;
- розробляє архітектурі (логіку) продукту;
- здійснює детальне проектування і створення специфікацій проектів;
- повністю контролює проектування дрібних проектів і внутрішніх під-проектів (модулів), контролює інших розробників і ставить їм завдання;
- програмує і тестує компоненти, забезпечує верифікацію проекту;
- займається рішенням складних завдань.

Як правило, має стаж від 3-х років як developer. На відміну від інших уміє коментувати програми, не удаючись до використання словника, розробляти документацію, вільно спілкуватися англійською мовою, володіє методами та інструментами аналізу і проектування, Software Engineering Process. Має набір конкретних рішень в області тих або інших найбільш часто виникаючих завдань, за рахунок чого демонструє високу продуктивність порівняно з Middle dev. Робить на порядок менше помилок, помилки допущені Middle dev, як правило, він легко усуває. Чудово концентрується на головному, ігноруючи несуттєві деталі. Здатний побачити суть, нерідко навіть не дочитавши до кінця технічне завдання (ТЗ).

Team Lead

Серед розробників **Senior** виділяється **Team Lead** (Тимлид) – IT-фахівець, який управляє своєю командою розробників, володіє технічною стороною, бере участь в роботі над архітектурою проекту, займається ревью кода, а також

розробкою деяких особливо складних завдань на проекті.

В проектах є дві **lead** ролі: менеджерська - **PM**, і технічна - **System Architect**. Тимлид частково виконує обидві ролі, але акцент його обов'язків спрямований на менеджмент (акцент на технічну частину - це tech lead).

До ролі **PM lead** – менеджерської відноситься управління проектами, основним етапом якого є – планування.

У всіх проектах процес планування має чіткі логічні та інформаційні взаємозв'язки. Наприклад, спочатку слід визначити, з яких робіт складається проект, а потім визначати склад команди та розраховувати терміни виконання та вартість проекту. Основними етапами процесу планування є:

1. Визначення цілей – це процес розробки документа (технічного завдання), в якому формулюються цілі проекту (констатація цілей), що є основою для наступних проектних рішень, включаючи визначення критеріїв успішності виконання проекту.

2. Визначення операцій – це процес ідентифікації та документування операцій, які слід виконати для отримання результатів;

3. Призначення персоналу – призначення людських ресурсів для виконання робіт проекту.

4. Підготовка умов проектної діяльності – розробка вимог до постачань та визначення потенційних постачальників.

5. Планування взаємодії – визначення потоків інформації та способів взаємодії, необхідних для учасників проекту.

6. Оцінювання тривалості операцій – це визначення робочого часу, потрібного для їх виконання. Тривалість одних операцій визначається можливостями ресурсів, інших – тільки календарним часом.

7. Розрахунок вартості проекту – включає оцінку вартості ресурсів та вартості операцій. Вартість ресурсів може визначатися по-різному. Для відновлення ресурсів задається вартість часу їх роботи, для матеріалів — вартість одиниці.

8. Ідентифікація ризику – визначення та документування подій ризику, які можуть впливати на проект.

9. Якісне та кількісне оцінювання ризиків – розташування пріоритетів ризиків за ступенем впливу на результати та визначення ймовірності настання подій ризику, їх характеристик і впливу на проект.

10. Розробка реагування – визначення необхідних дій для запобігання ризикам та реакції на загрожуючі події.

Тема 4. Системи контролю версій розробки IT-проектів

План:
1. Класифікація систем контролю версій розробки ІС.
2. Засади командної організації проектування ІС.
3. Розгалуження робочих версій.
4. Система контролю версій Git.

Класифікація систем контролю версій розробки ІС

Система управління версіями (від англ. Version Control System (VCS) або Revision Control System (RCS), перші виникли в 1986р.) - програмне забезпечення для полегшення роботи з інформацією яка змінюється.

VCS дає можливість повертати окремі файли до попереднього вигляду, повертати до попереднього стану весь проект, переглядати виконані зміни у визначений час проектування, встановлювати того, хто останнім вносив зміни за умови коли проектний модуль раптово перестав працювати, тим самим встановити хто і коли припустився помилки та багато.

Стадії розробки ПП (характеризують готовність)

- ✓ **Pre-alpha(pa)** – початкова стадія розробки. Характеризується значними змінами та великою кількістю помилок. **Pa** - релізи не виходять з відділу розробок.
- ✓ **Alpha(a)** - етап завершення розробки нового функціоналу. Можливо додавання нових функцій. Стадія внутрішнього тестування. Застосовується тільки для ознайомлення з майбутніми можливостями.
- ✓ **Beta(b)** - стадія публічного тестування. Перший реліз, що виходить за межі відділу розробки.
- ✓ **ReleaseCandidate(rc) -** пре-реліз —стадія-кандидат на стабільну версію. Реалізовано весь функціонал та виконано комплексне тестування.
- ✓ **Release to marketing** (rtm) – промислова — стабільна версія програми, що підготовлена до тиражування, відповідає всім вимогам з якості та готова для масового розповсюдження.
- ✓ **Generalavailability**(ga) - фінальний реліз - завершення всіх робіт з комерціалізації продукту. Готов до продаж через веб або на фізичних носіях.

VCS для розробників забезпечує:

- підтримку збереження файлів у репозиторії (структура даних - **repository**);

- підтримку історії версій файлів у репозиторії;
- пошук конфліктів у процесі внесення змін до вихідного коду та забезпечення синхронізації під час роботи в багатокористувацькому середовищі;
- відслідковування змін коду різними розробниками під час командної розробки.

VCS можна поділити на **класи** за такими ознаками:

за структурою :

- ***централізовані***, мають центральний репозиторій. Кожен розробник по завершенні роботи над версією закачує її на сервер (**CVS, Subversion (SVN)**);
- ***розподілені*** – кожний розробник має свій репозиторій для роботи (**Mercurial, Git**).

за характером роботи з об'єктами :

- **блокуючі** , коли накладається заборона на зміни у файлі поки один з розробників працює над ним.
- **не блокуючі** – код у одному файлі можуть змінювати декілька розробників.

за типом об'єкта :

- для контролю текстового вмісту коду;

для контролю бінарного вмісту файлів проекту.

Основні терміни :

Репозиторій (repository, depot), сервер—сховище, зберігає всі вихідні коди програми, а також історію їх змін.

Тег – це текстова мітка, що прив'язується до якої-небудь ревізії файлу або репозиторію.

- Одна ревізія може вміщувати декілька тегів.
- Вибірку ревізії файлу/файлів можна здійснювати за тегами.

Ревізія (revision) —версія документу, **нові зміни (changeset)** створюють нову ревізію репозиторія.

Гілка (branch) – напрямок розробки, незалежний від інших. Являє собою копію частини сховища, в яку можна вносити свої зміни, що не впливатимуть на інші гілки.

Централізовані CVS, Subversion

Система CVS (Concurrent Versions System)

Має дві частини : сервер та клієнта.

Стандартний клієнт – консольний, який дозволяє виконувати всі дії.

Має велику кількість графічних клієнтів.

Розробник : Cyclic Software (http://www.cyclic.com).

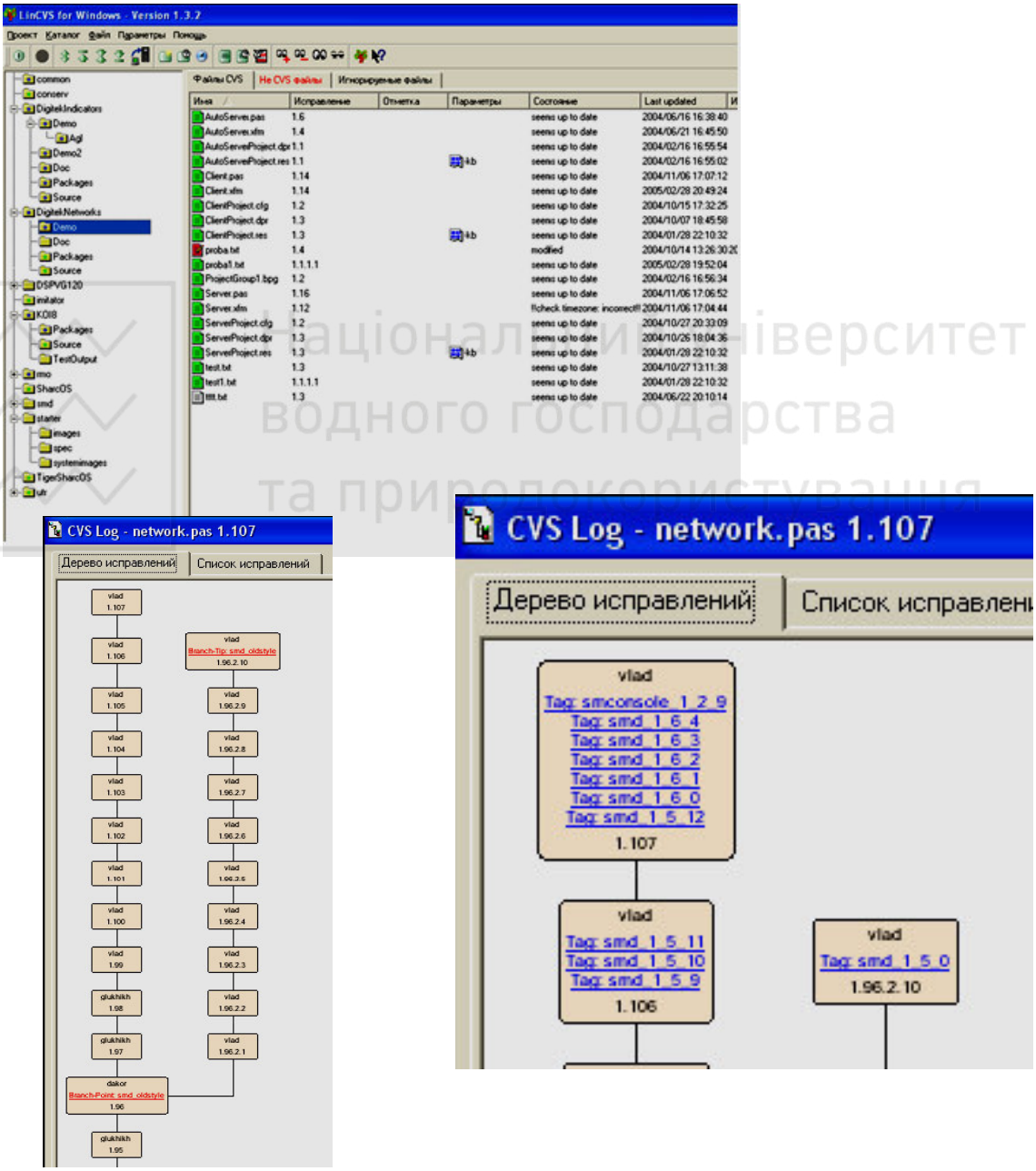


Рис. 4.1. Візуальні вікна Concurrent Versions System

Subversion (SVN) була створена для заміни CVS

Дозволяє :

- Замінювати ***теги*** та ***гілки*** на віртуальні каталоги.
- Версіонувати каталоги.
- Версіонувати метадані.
- Атомарна фіксація змін.
- Повна історія версій (видалення, перейменування тощо).
- Зберігання файлів у стиснутому форматі.
- Вибір способу доступу до репозиторію.
- Інтегрування з web-сервером.

Розробник : CollabNet, Inc (http://subversion.tigris.org).

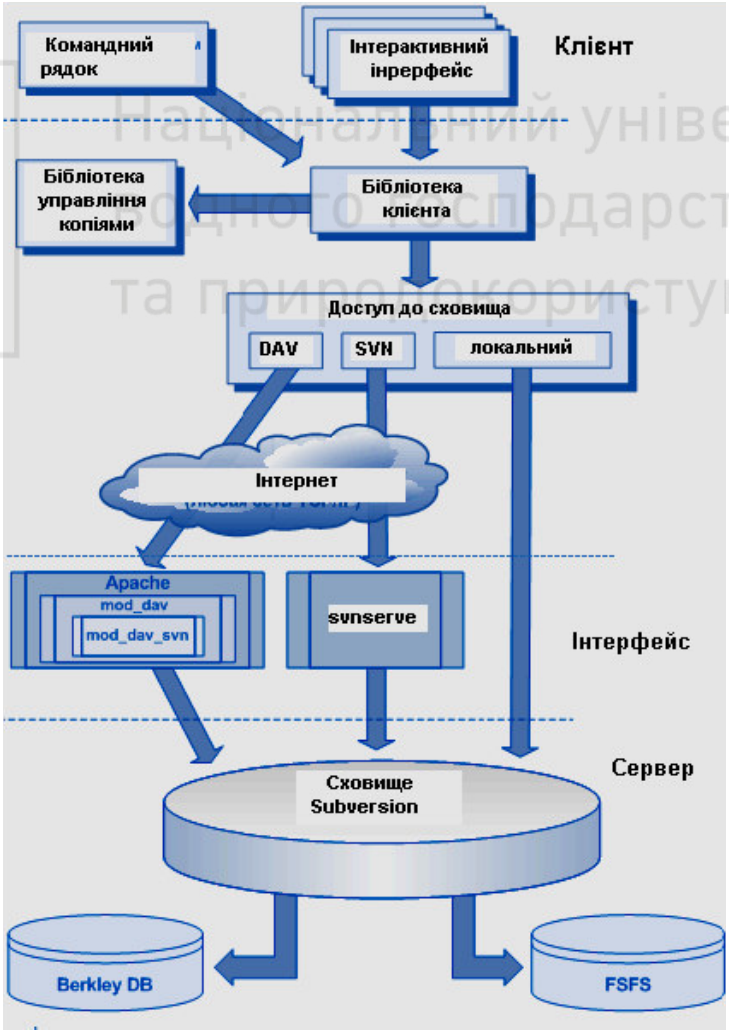


Рис. 4.2. Ілюстрація принципу роботи Subversion

Централізовані CVS, Subversion

Дозволяють співпрацювати **декільком розробникам** з **центральним сервером**, на якому зберігаються всі файли версій під контролем, та інформація щодо клієнтів,що взаємодіють з цим сервером.

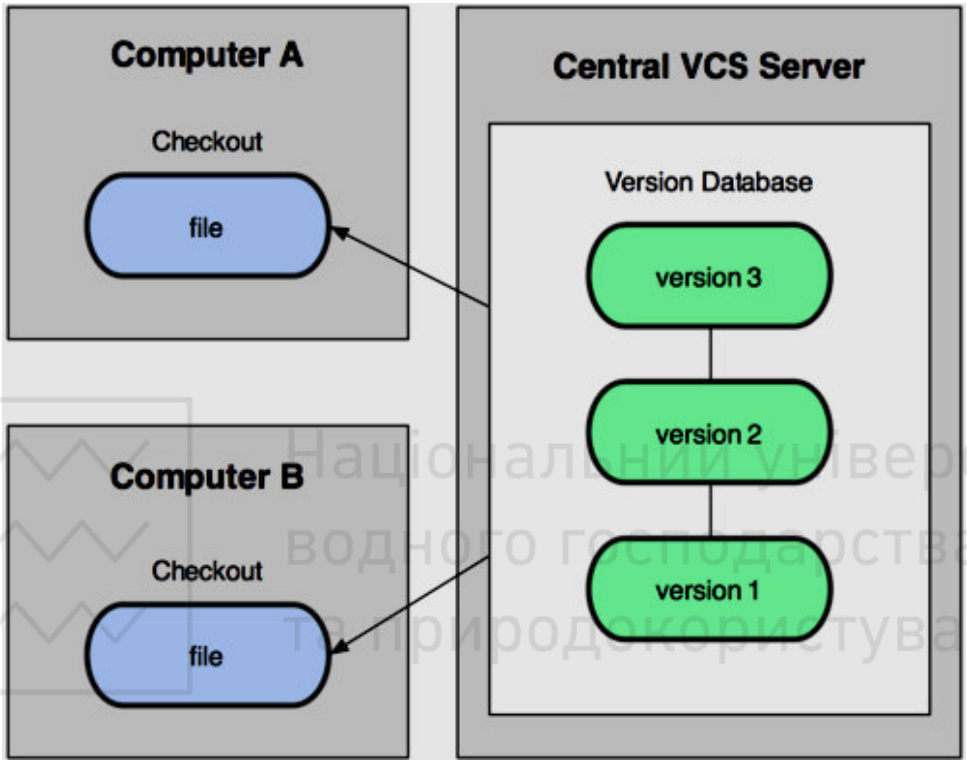


Рис. 4.3. Схема взаємозв'язку сервера з клієнтами в Subversion

Переваги централізованих систем :

- всі знають, хто чім займається в проекті;
- адміністратор може здійснювати чіткий контроль над тим, хто та що може робити.

Недоліки:

- централізований сервер являє найбільш вразливе місце всієї системи;
- якщо сервер стає недоступним, то розробники не можуть взаємодіяти, та ніхто не може зберігати нову версію проекту;
- якщо пошкоджується диск з центральною базою даних та немає резервної копії, розробники втрачають усі дані - всю історію проекту, за винятком декількох робочих поточних версій, що збереглись на комп'ютерах окремих розробників.

Розподілені: Git, Mercurial

На відміну від централізованих систем, клієнти в розподілених системах можуть не тільки отримувати останні версії файлів, а також повністю копіювати весь репозиторій.

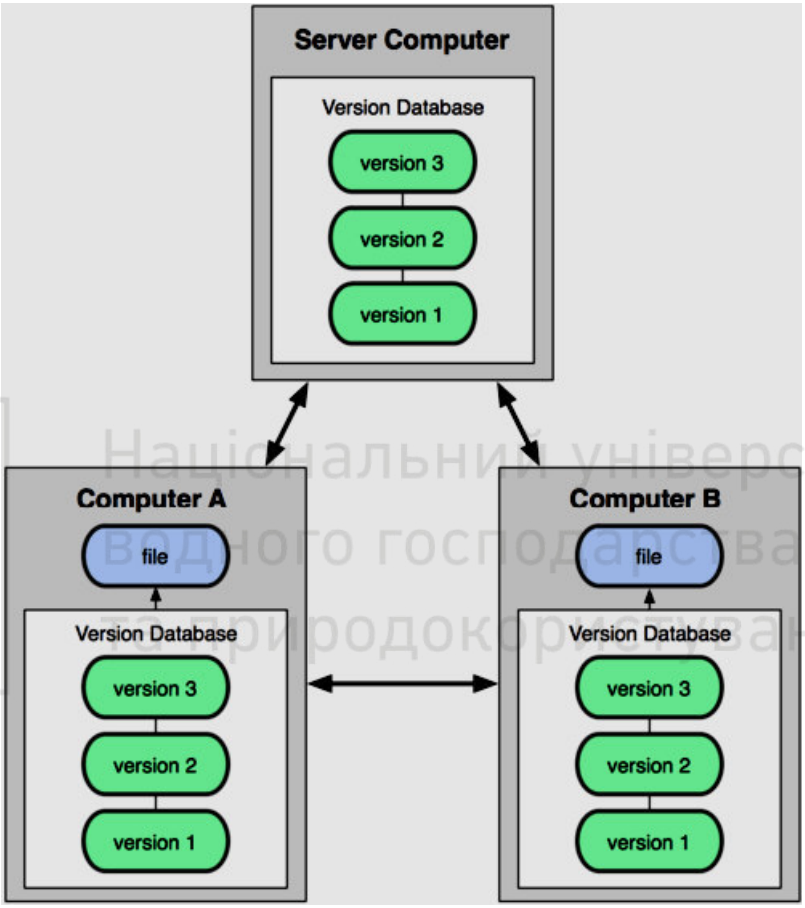


Рис. 4.4. Схема взаємозв'язку сервера з клієнтами в розподілених системах

Переваги розподілених систем :

- Так як кожен раз клієнтом **отримується свіжа версія** файлів то він створює в себе повну копію всіх даних, а це у випадку виникнення збою на сервері дає **можливість відновити репозиторій** застосувавши клієнтський та відновити всю втрачену базу.
- Створена **можливість працювати** з декількома віддаленими репозиторіями **одночасно**, таким чином, стало можливим одночасно працювати по-іншому **з різними групами розробників** в межах одного

проекту (важливо для керівників груп). Так, в одному проекті можна одночасно вести декілька типів робочих процесів, що було неможливо в централізованих системах.

Засади командної організації проектування ІС

VCS реалізує такі завдання командної роботи :

- Підвищує надійність зберігання даних проекту.
- Забезпечує спільний доступ до файлів.
- Зберігає історію модифікування файла кожним розробником.
- Виділяє та зберігає різні версії файла.
- Підтримка та розвиток декількох паралельних історій файла.

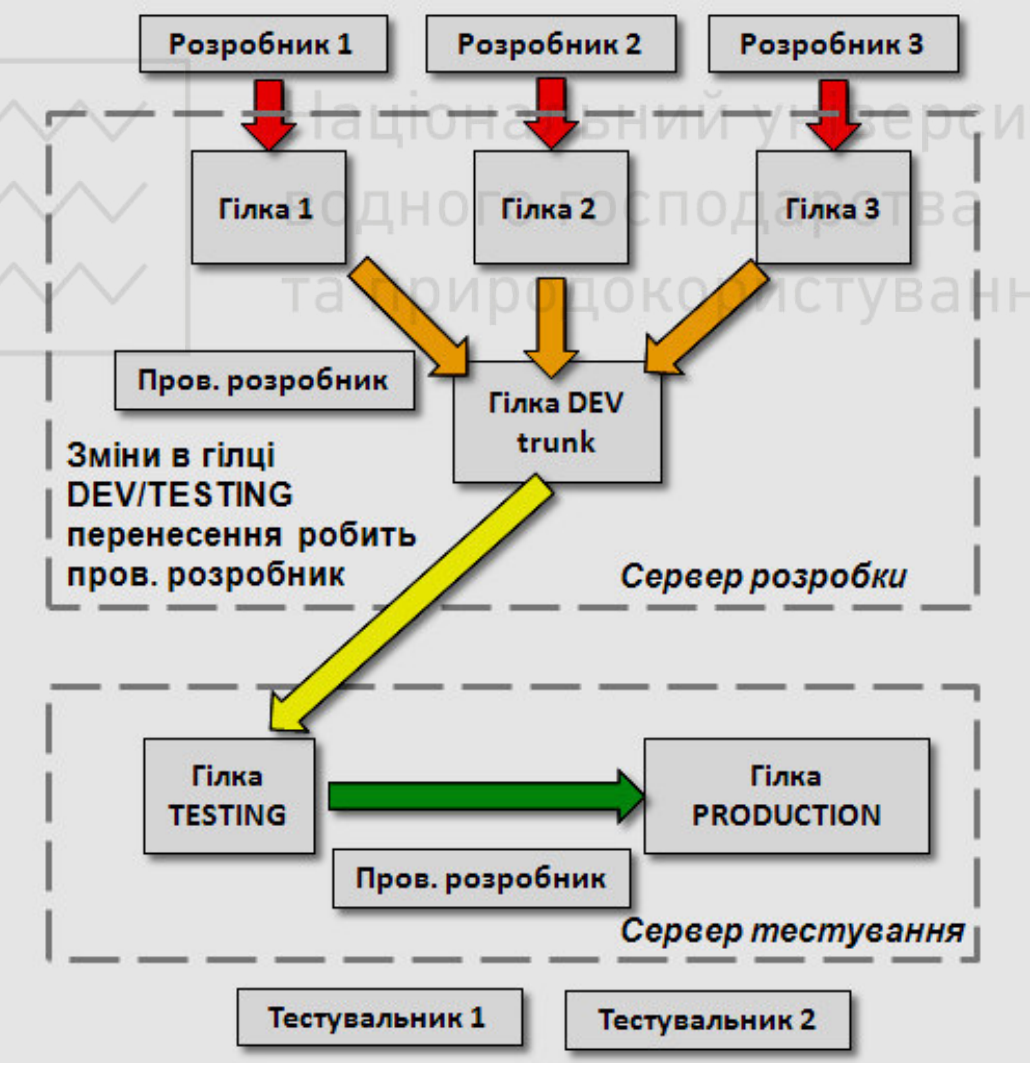


Рис. 4.5 Схема командного проектування в системі VCS

Основні терміни:

Клієнт – має свою **локальну** копію (**workingcopy**) вихідних кодів, з якими працює**.**

Локальна копія (workingcopy)(check-out, clone) – отриманий документ із сховища та створення робочої копії**.**

Зміни (changeset, activity) – набір змін, помічений (поіменований) набір правок, зроблених у локальній копії.

Мітка (tag, label) – позначка початку відліку змін в дереві, групує декілька файлів в придатний для застосування блок. Частіше всього застосовується для позначення кінцевої версії файлів для збирання.

Кожна версія файлів проекту повинна мати свої позначення

Причини позначень :

- Якісна версія.
- Версія, що має певні властивості та особливу історію.
- Версія, що є частиною релізу проекту певної версії.

Етапи роботи розробника IT-проектів:

1. **Оновлення робочої копії** версії з репозіторія (**update, clone, pull**).
2. **Модифікація проекту**, змінюючи локально код у файлах робочої копії.
3. **Фіксація змін** після завершення поточного завдання. Розробник фіксує (**commit**) свої зміни та передає (**push**) їх на сервер (здійснює злиття).

Гілка (branch)

Застосовуються :

- для роботи з групою розробників;
- для зменшення ризику втрати змін;
- для забезпечення можливості аналізу та повернення до попереднього варіанту коду.

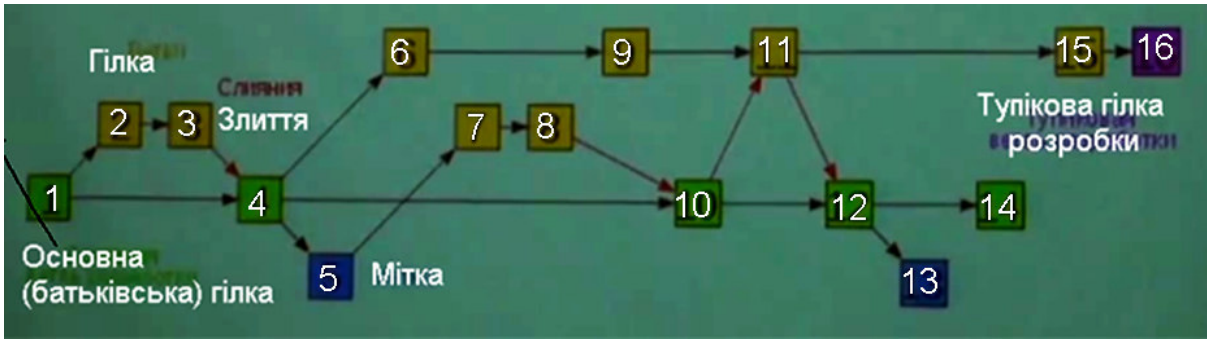


Рис. 4.6 Приклад еволюції гілок проекту

Розгалуження робочих версій

Гілка є копією частини (як правило одного каталогу) проекту, коли до коду вносяться зміни, які не впливають на інші гілки, та скидаються на репозиторій.

Документи на **різних гілках мають однакову історію до точки** коли вони розгалузились **від батьківської** та різні історії – після.

Зміни з однієї гілки можна переносити в другу **через операцію злиття**.

Ствол (**trunk**, mainline, **master**) – *основна* (**батьківська**) **гілка** розробки проекту. В більшості випадках створюється папка під назвою **Master** або **Trunk**.

Причини розгалуження версій :

1. Розвиток декількох версій проекту :
   - Переданих замовнику.
   - Таких, що далі розробляються.
2. Наявність декількох конфігурацій проекту:
   - Для різної апаратури
   - Для різних операційних систем

Ревізія файлу – унікальний ідентифікатор версії файлу в системі контролю версій:

- ✓ CVS: 1.2
- ✓ SVN: 238
- ✓ Git, Mercurial: кеш SHA-1

При змінені файлу номер ревізії збільшується за певними правилами:

- ✓ іноді номер ревізії є атрибутом всього репозиторію.

Атрибути ревізії:

- ✓ Ідентифікатор.
- ✓ Автор змін.
- ✓ Дата.

✓Текстовий опис змін (тег).

✓Зовнішні атрибути:

- Теги
- Ідентифікатори гілок

Система контролю версій Git

Ядро **Git** являє собою набір утиліт. Всі налаштування зберігаються в текстових файлах конфігурації. Така реалізація робить **Git** легким для інтегрування з будь-якою платформою.

GitHub (github.com)

Один з найбільших веб-сервісів для хостінгу проектів та командної розробки. Його називають «соціальною мережею для розробників».

Сервіс використовує систему контролю версій Git, та розроблений на мовах функційнального програмування **Ruby** (Рубі) та **Erlang** (Ерланґ).

Сервіс безкоштовний для open source проектів та надає їм всі можливості, в том числі и SSL (Secure Sockets Layer — **рівень захищених сокетів**) — криптографічний протокол.

Девіз сервісу –«SocialCoding» (Пишемо код разом)

Для приватних проектів передбачено різні **тарифні** плани.

Перший приватний репозиторій був створений **12 січня 2008 року**. **Свої офіційні репозиторії** розміщують : **Facebook, Twitter, Yahoo, Ruby on Rails** та інші.

Основні терміни :

head — сама свіжа (остання оновлена) версія проекту в репозиторії.

commit, check-in, submit – фіксація змін;

clone, check-out, – отримання розробником робочої копії проекту;

update, sync – синхронізація робочої копії проекту до деякого заданого стану в репозиторії;

marge, integration – злиття незалежних змін проекту;

conflict – ситуація коли декілька розробників зробили зміни в одній і тій же ділянці коду файла;

patch – файл, що описує різницю меж файлами.

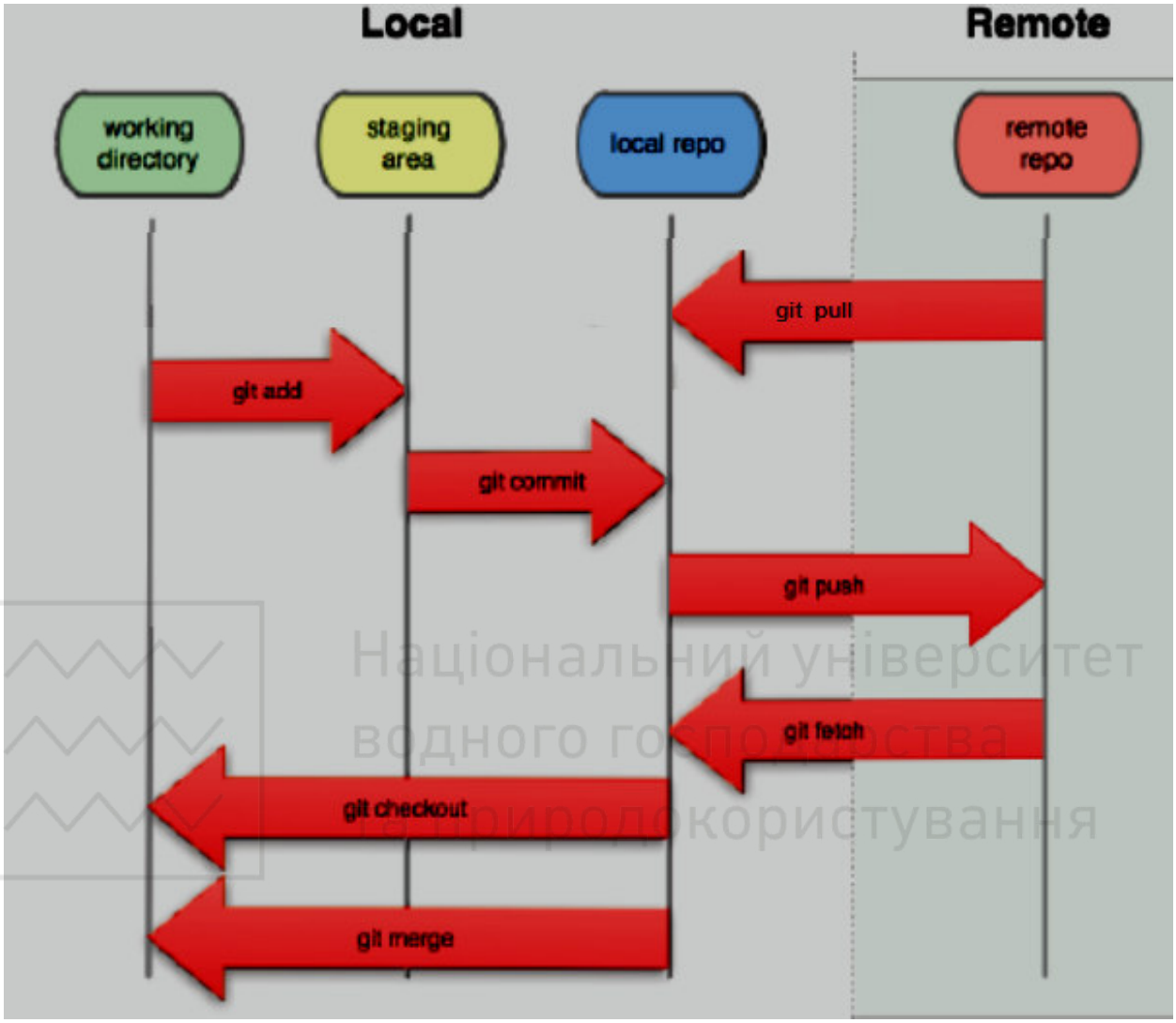


Рис. 4.7 Схема роботи з Git

Робота з Git (на кожен день)

1. Оновлення репозиторія та робочої копії
   git pull
2. Додавання файлу до проекту
   git add newFile.cpp
3. Комміт **git commit –m “опис того, що зроблено”**
4. Передавання змін до зовнішнього репозиторію
   Git push

Основні способи розмічування версій :

- Теги файлів;
- Віртуальні каталоги.

Git зберігає дані як **зліпки (стани) невеликої файлової системи**. В процесі **фіксації** поточної **версії** проекту, **зберігає зліпок** того, як виглядають **всі файли** проекту **на той момент часу** (рис. 4.8).

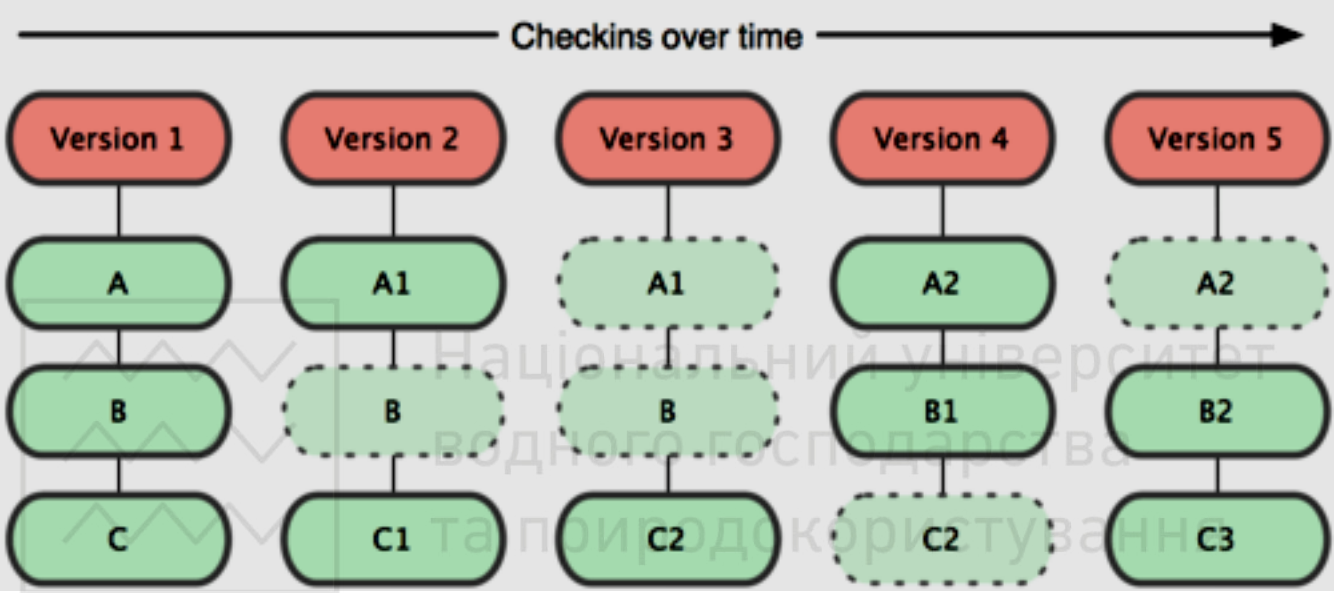


Рис. 4.8. Стани проектних дій в часі (пунктиром позначені не змінені частини проекту)

Основний принцип функціонування Git

Перед збереженням будь-якого файлу **Git обчислює контрольну суму,** і вона стає **індексом** цього **файлу.** Тому неможливо змінити вміст файлу або каталогу так, щоб **Git** не знав про це.

Ця функціональність вбудована в сам фундамент **Git** та є важливою складовою його філософії. Якщо інформація загубиться в процесі передавання або пошкодиться на диску, **Git** завжди буде знати про це.

В Git файли можуть знаходитись в одному з трьох станів:

1. **Зафіксованому**
2. **Зміненому**
3. **Підготовленому**

"**Зафіксований**" означає, що файл вже збережено в вашій локальній базі.

До **змінених** відносяться файли, які змінилися, але ще не були зафіксовані.

Підготовлені файли — це змінені файли, що відзначені для включення до наступного комміту.

У проектах, що застосовують **Git**, є три частини:

- **каталог Git** (Git directory),
- **робочий каталог** (working directory) та
- **область підготовлених файлів** (staging area).

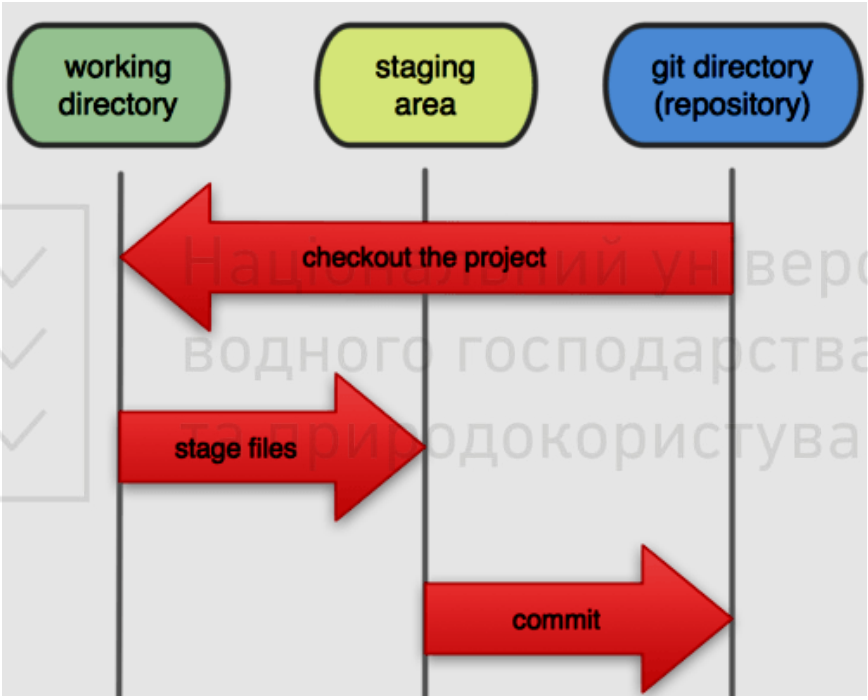


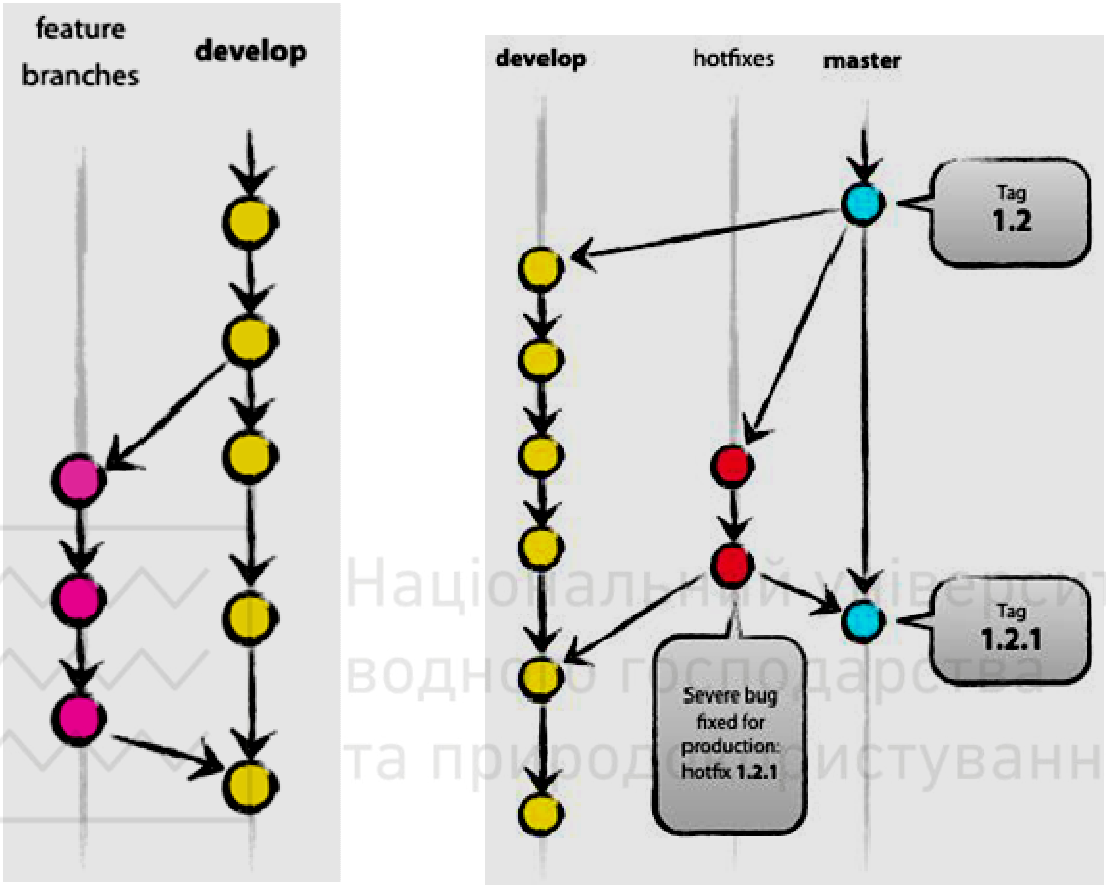
Рис. 4.9. Три частини проектів, що застосовують Git

Каталог Git — це місце, де Git зберігає метадані та базу даних об'єктів вашого проекту. Це найбільш важлива частина Git, тому, що вона копіюється, коли ви клонуєте репозиторій..

Робочий каталог — це зроблена копія з бази певної версії проекту. Ці файли розміщуються в стиснутій базі даних у каталозі Git та розміщуються на диску для того, щоб ви їх переглядали та редагували.

Область підготовлених файлів — це звичайний файл, зазвичай який зберігається в каталозі **Git**, який вміщує інформацію про те, що повинно увійти

до наступного комміту.. Іноді його називають індексом (**index**), але за останній період визначається стандартом як область підготовлених файлів (**staging area**).



а) б)

Рис. 4.10. Види гілок Git

Види гілок Git

Гілки функціональностей (**Feature branches**) (рис. 4.10.а):

1.Створюються від develop.

2.Повинні інтегруватися в develop.

3.Застосовуються для розробки нових функцій.

4.Існують стільки , скільки розробляється нова функція (feature).

5.Как правило існують в репозиторіях розробників, а не в центральному.

Гілки релізів (**Release branches**):

1. Можуть породжуватися від develop.
2. Повинні інтегруватися в develop, master.
3. Назва : release-*.
4. Застосовуються для підготовки версій до релізу.

Гілки виправлень (**Hotfix branches**) (рис. 4.10.б):

1. Породжуються від master.
2. Інтегруються в develop, master.
3. Назва : hotfix-*.

***Похожі на гілки релізу, відрізняються тим, що вони не плануються. Створюються за потребою коли потрібно терміново виправити помилки в Production-Ready коді.**

Коли *код* у гілці *develop* стає *придатним для наступного релізу*, то його *інтегрують* (зливають) до основної гілки *master* та помічають тегом версії.

1. **Оновлення** робочої копії (**update, sync**) – зміни, що зроблені в основній версії, зливаються з локальними, тобто синхронізується до деякого заданого стану сховища.

Можливі причини змін версій:

- **модифікування** файлу;
- **створення** нового файлу або каталогу;
- **видалення** нового файлу або каталогу;
- **перейменування** раніше існуючих файлів або каталогу в проекті.

2. **Фіксація** змін (комміт (**check-in, commit, submit**)) – локальні зміни зливаються з змінами, що вже зафіксовані в основній версії.

3. **Злиття гілок** (**merge, integration**) – об'єднання незалежних змін в єдину версію документа - зміни, що зроблені в одній гілці розробки, зливаються з змінами, що зроблені в іншій гілці.

Конфлікт в системі VCS

Конфлікт (conflict) – ситуація, коли в процесі злиття декількох версій зроблені в них зміни перетинаються меж собою.

Конфлікт може виникати в разі:

- **Видалення** та **змін одного й того ж файлу** або каталогу.
- Видалення та **перейменування одного й того ж файлу** або каталогу (у випадку, якщо система підтримує операцію перейменування).
- **Створення в різних версіях** файлу **з одним** і тим же **іменем** та різним вмістом.
- **Зміни в межах одного текстового файлу**, що зроблені в різних версіях, якщо вони перетинаються.
- **Зміни** в межах одного файлу, **якщо** він **не є текстовим**, завжди є конфліктними й не можуть бути об'єднані автоматично.

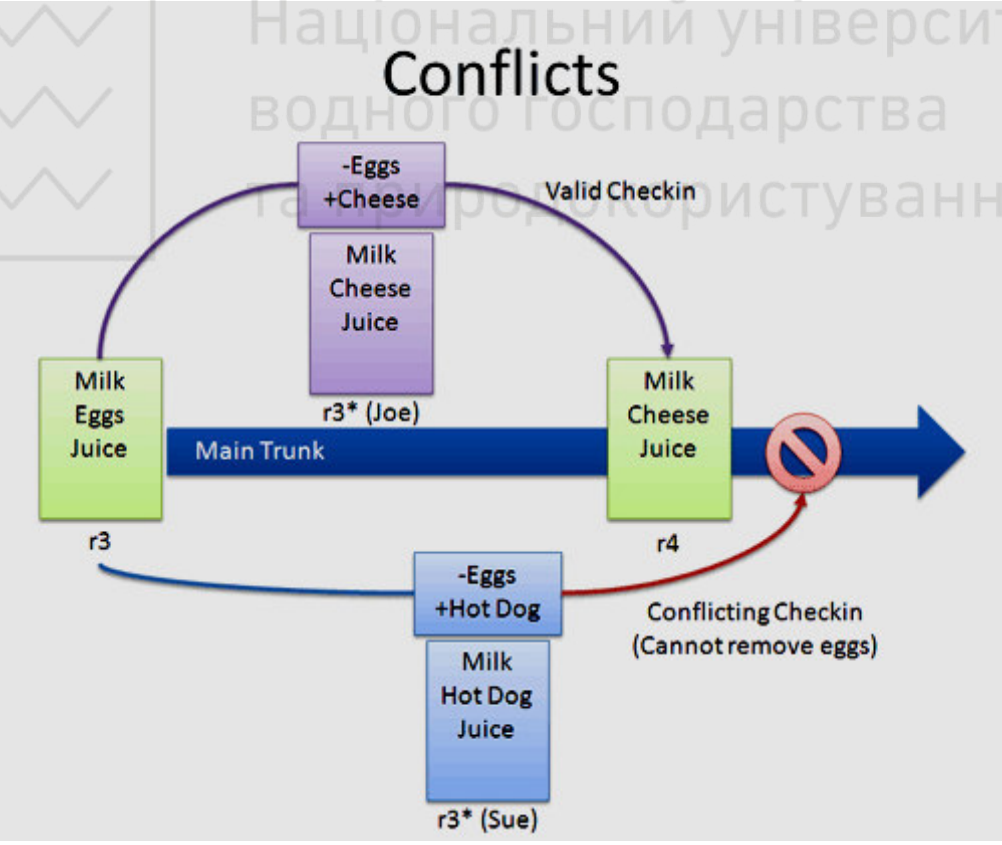


Рис. 4.11. Ілюстрація конфлікту

Розв'язування конфліктів

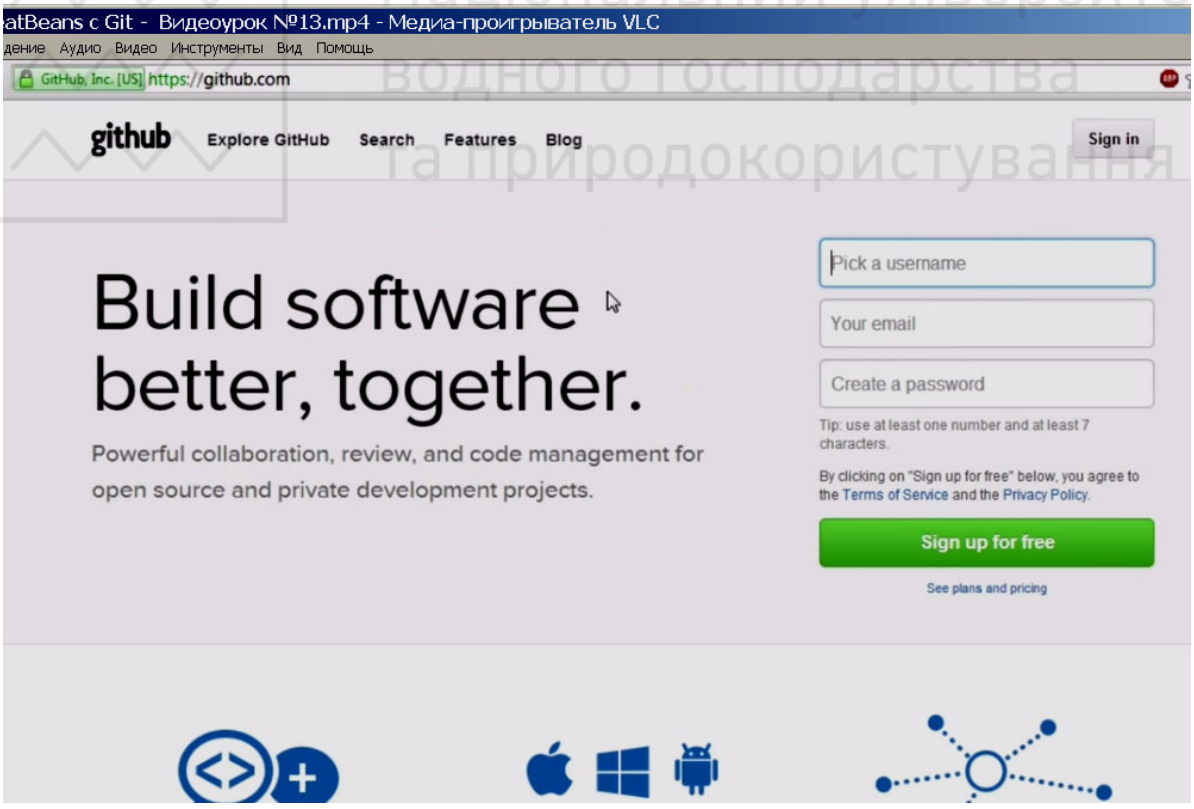
Розв'язування конфліктів можливе тільки за участю розробника. Для цього застосовують операцію блокування.

Блокування – механізм розв'язування конфліктів. Механізм блокування **дозволяє одному з розробників захопити в монопольне застосування файл** або групу файлів для внесення до них змін. На той час, доки файл заблокований, він **стає доступним решті розробникам** тільки для читання, та будь-як спроба внести зміни відкидається сервером.

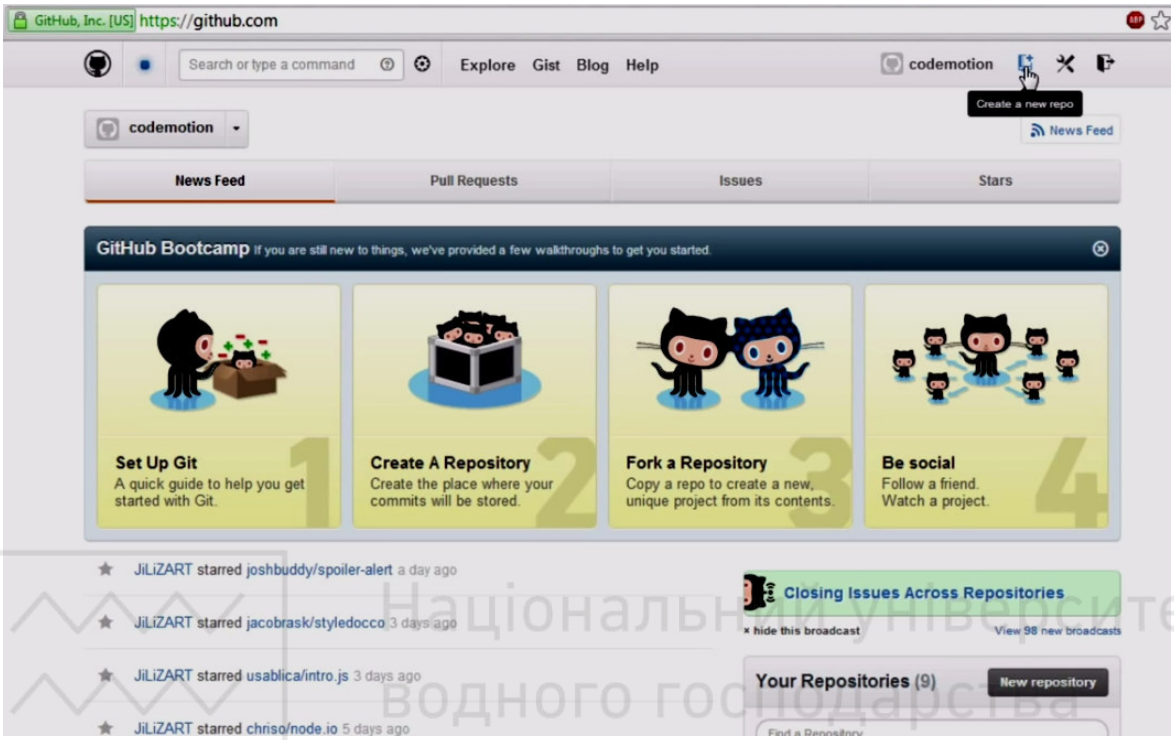
Приклад застосування блокування - робота з **бінарними файлами**, для яких немає інструментальних засобів злиття змін и таке злиття принципово неможливе (наприклад, для файлів зображень).

Етапи встановлення зв'язку на github.com

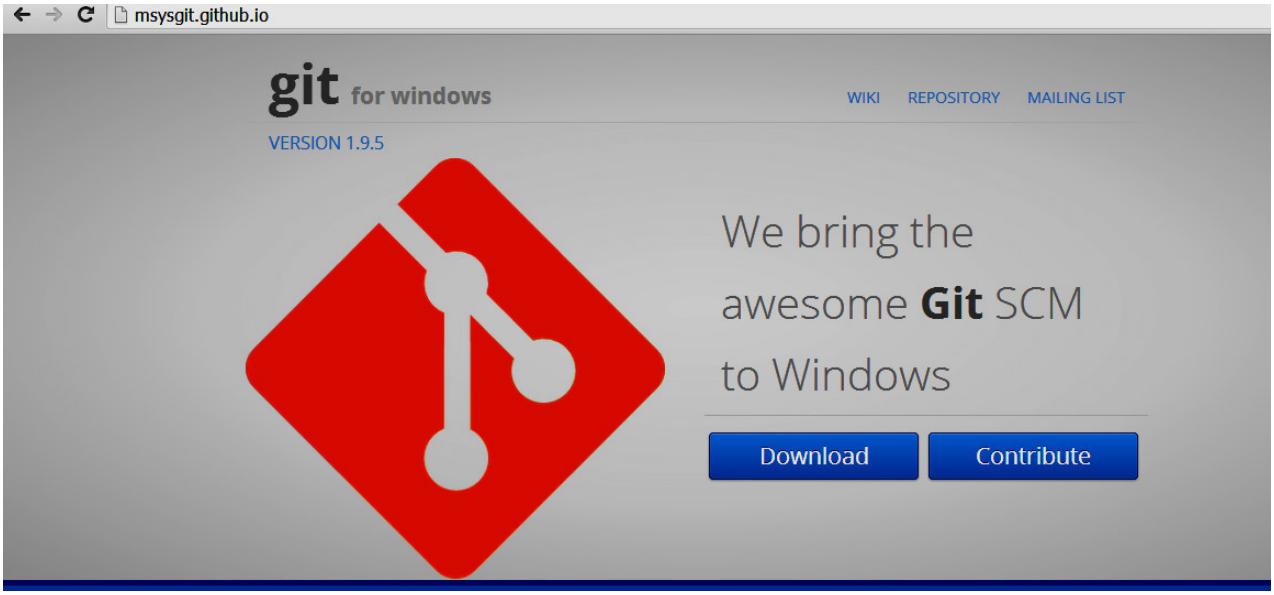
1) Реєстрація (за посиланням github.com/signup/free) - вводимо свої дані.



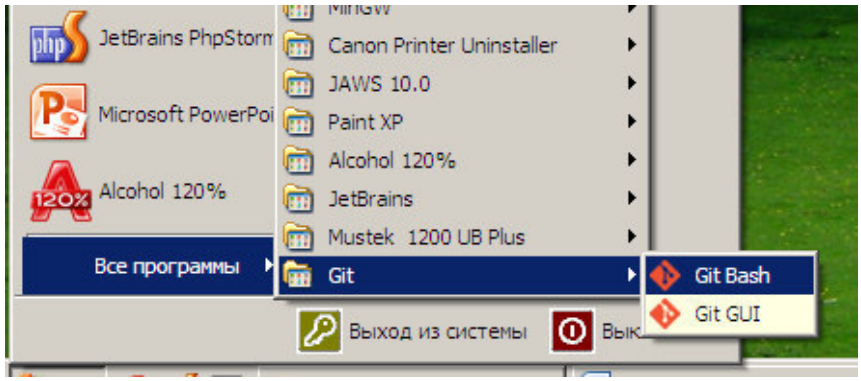
2) Після реєстрації ми потрапляємо на Dashboard нашого аккаунта.



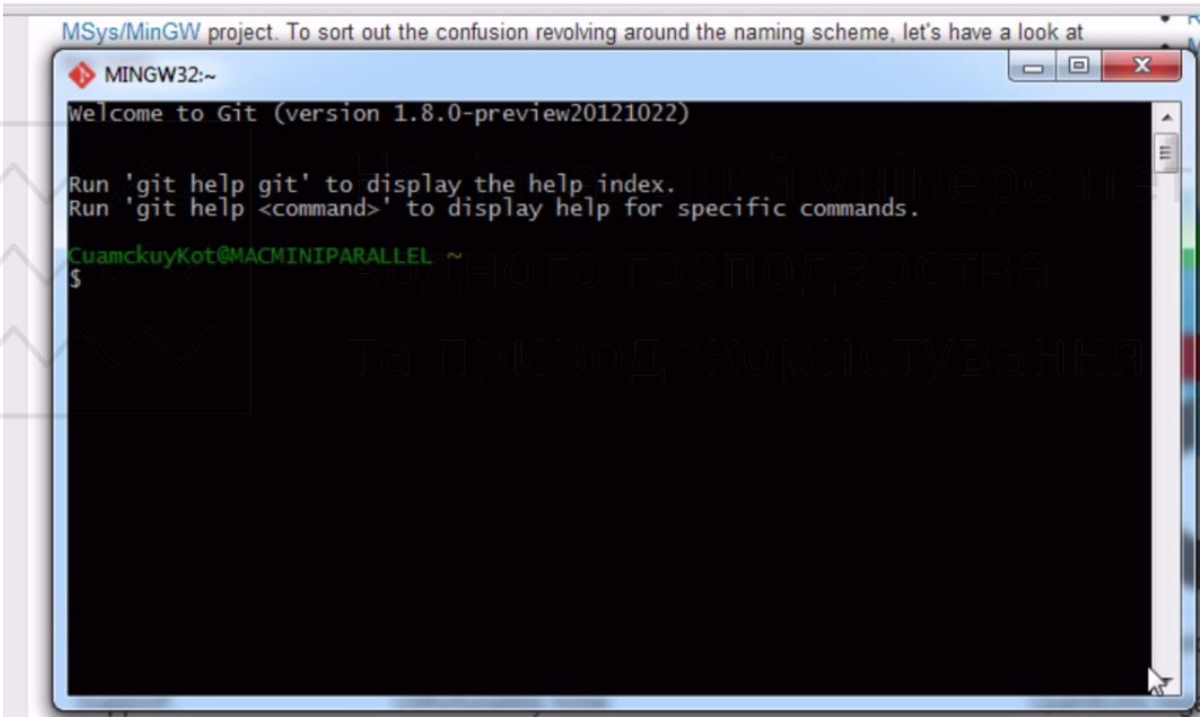
3) Для роботи в Windows, качаємо і встановлюємо **msysgit** (http://msysgit.github.io). Це консольна версія git для Windows.



4) Запускаємо панель MSysGit MINGWIN32 (Git Bash)



5) Вигляд панелі MINGWIN32 (Git Bash)

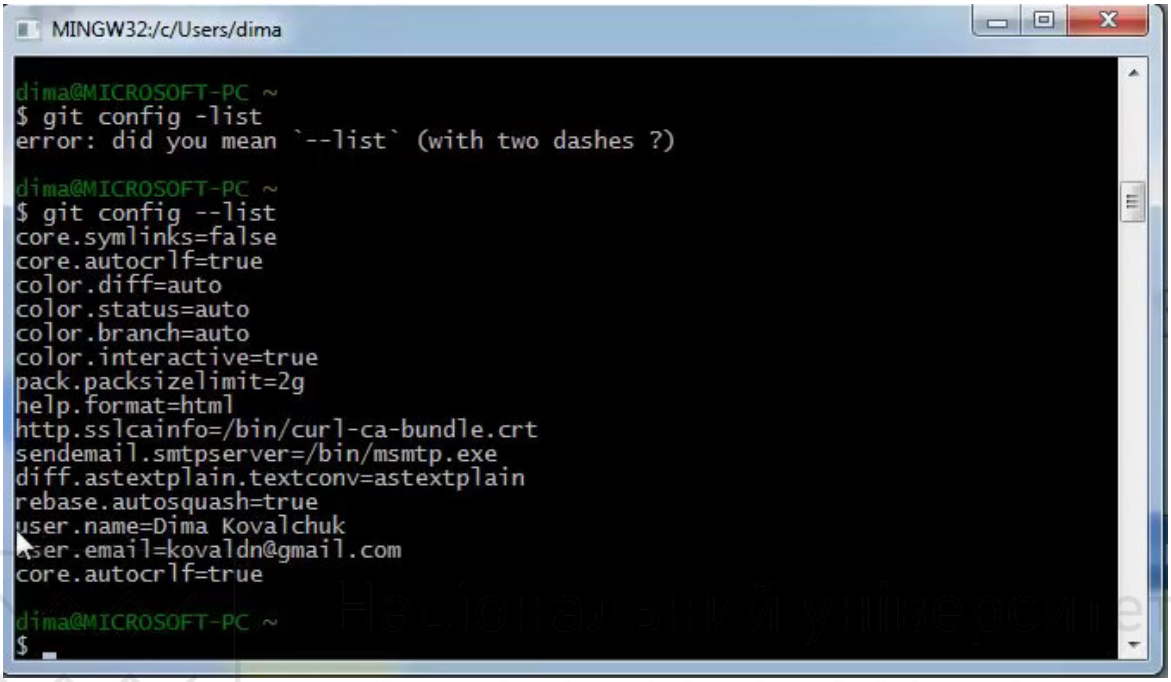


6) Прописуємо в консолі свої дані та налаштування переносів рядків:

git config --global user.name "ваше им'я" git config --global user.email "ваша пошта" git config --global core.autocrlf true git config --global core.safecrlf true

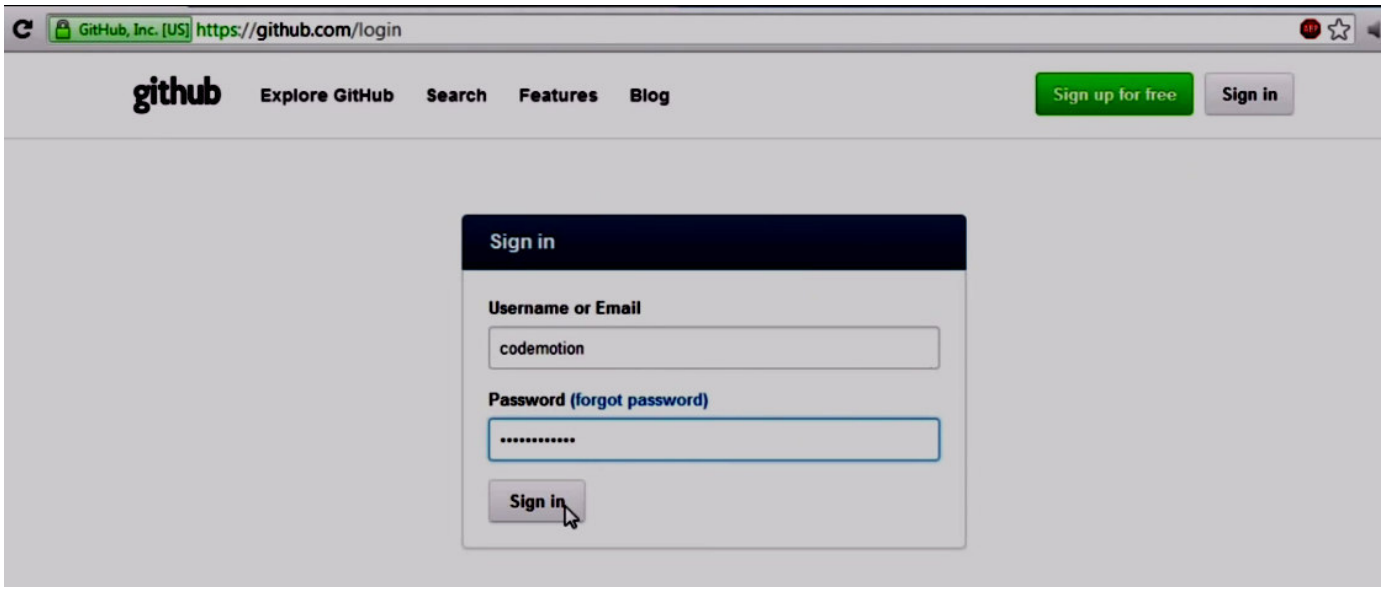
Для того щоб переглянути налаштування: git config –list

7) Налаштування у панелі MINGWIN32 (Git Bash)

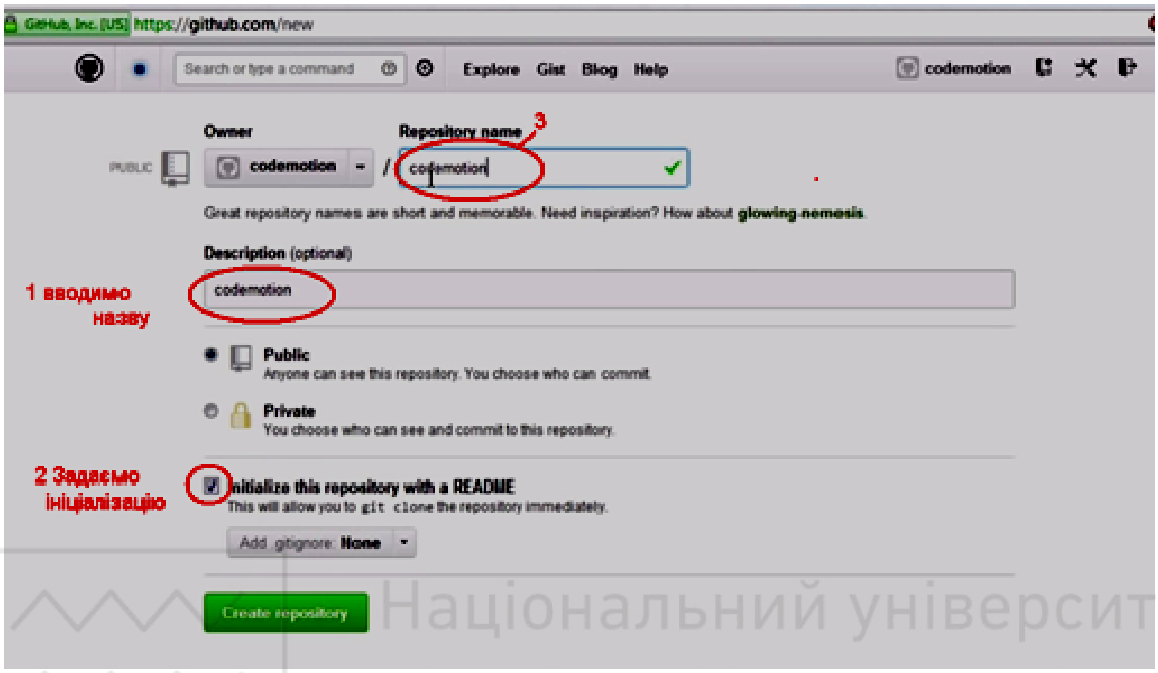


Закриваємо MINGWIN32 (Git Bash)

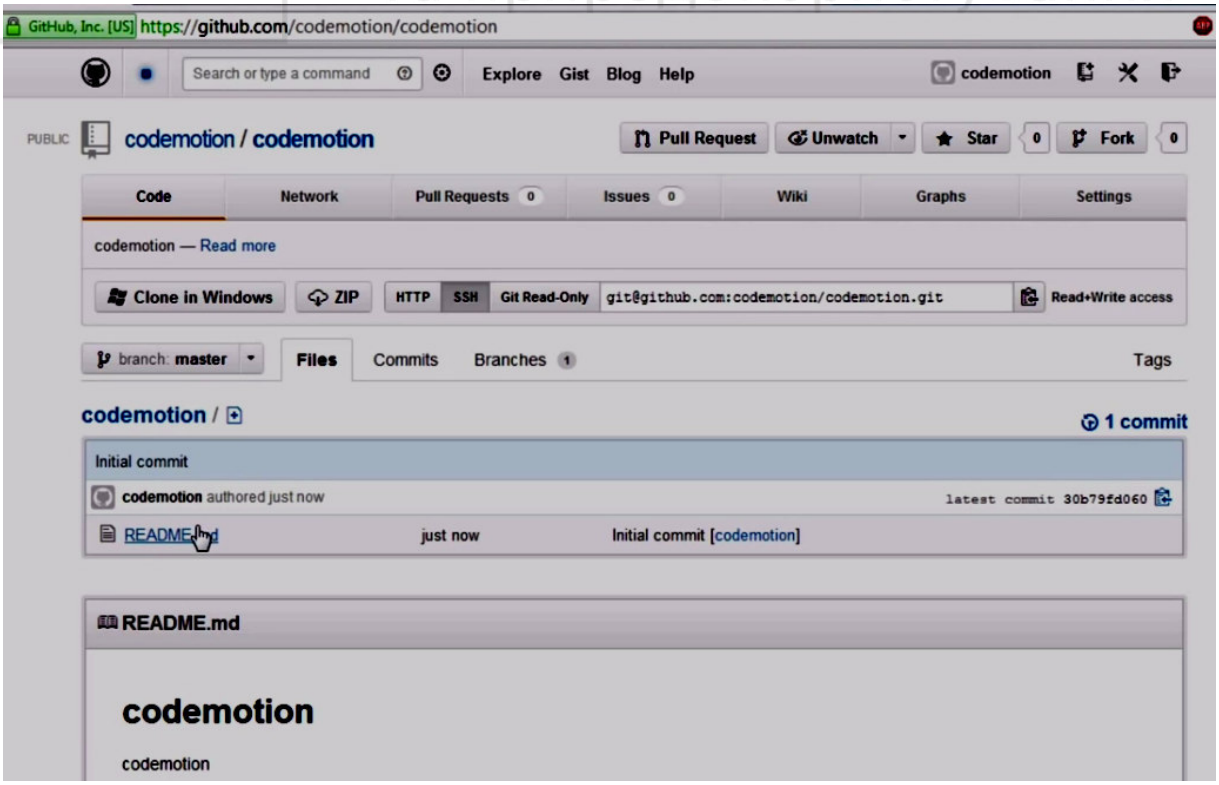
8) Реєстрація на сайті



9) Створюємо новий репозиторій (Create a new repo)



10) Репозиторій створено. В ньому бачимо файл README.md.



11) GitHub дозволяє працювати з репозиторіями трьома способами: **Git Read-Only**, **HTTP** та **SSH** і, відповідно надаючи посилання трьох видів для нашого репозиторію:

1. git@github.com:habrauser/Hello-world.git
2. https://habrauser@github.com/habrauser/Hello-world.git
3. git://github.com/habrauser/Hello-world.git

Для того, щоб просто забрати репозиторій на локальну машину, достатньо внутрішнього протоколу git (третя посилання). Це найбільш швидкий та ефективний спосіб, який забезпечує анонімний доступ тільки для читання.

Якщо ж ми захочемо внести зміни в репозиторій на github, потрібно користуватися HTTP або SSH.

Робота з використанням http ніяких труднощів не викликає, в потрібний момент просто використовується пароль облікового запису на **github**. Для цього у MINGWIN32 (Git Bash) вводимо такий код:

а) для контролю змін в одному файлі:

- вказуємо папку для проекта на комп'ютері

cd c:/ gitproj

- створюємо внутрішню папку git

git init

- зв'язуємо проект codeproj1 на сервері з комп'ютером

git remote add codeproj1 https : // github.com/login/ codeproj1.git

адресу копіюємо з сайту

- скачуємо з сервера

git pull codeproj1 master

- в скачаному файлі README.md змінюємо вміст та запам’ятовуємо
- вказуємо на цей файл для контролю

git add README.md

- здійснюємо контроль (commit) за змінами

git commit – ‘коментар до змін’

- відправляємо зміни на сервер

git push codeproj1 master

вводимо по запросу username password

- створюємо один файл, наприклад index.php, на комп’ютері, зберігаємо. Готуємо до контролю та відправки на сервер

git add index.php
git status
git commit –m ‘file index.php add’
git push codeproj1 master

б) для контролю змін у декількох файлах одночасно:

- створюємо три нових файла, наприклад файл 1.php в корні проекту, файл style.css в папці CSS та файл 1.js в папці JS, на комп’ютері, зберігаємо. Готуємо до контролю та відправки на сервер

git add . (тут крапка вказує на те, що треба контролювати декілька нових файлів)
git status
git commit –m ‘ new 3 files’
git push codeproj1 master

Тема 5. Планування проектною діяльністю

1. Метод сіткового планування.
2. Умови побудови сіткового графіка.
2. Побудова сіткового графіка.
3.Технологія Agile.

Метод сіткового планування

Особливе місце у процесі управління проектами займає календарне планування. Це викликано важливістю збалансованості робіт в часі та необхідністю своєчасного координування дій багатьох учасників проекту. ***Календарний план*** – це проектно-технологічний документ, що визначає послідовність, інтенсивність та тривалість робіт, їх взаємозв'язки, а також розподілені в часі ресурсні потреби.

Основним завданням календарного планування є складання таких розкладів робіт, які задовольняють усім висунутим обмеженням щодо термінів та інтенсивності проведення робіт, відображають технологічну взаємозалежність робіт та забезпечують раціональне використання ресурсів.

З метою складання оптимального календарного плану слід, на сам перед, визначити критерій його якості. Такими критеріями, як правило, виступають:

- мінімальна тривалість проекту;
- мінімальна вартість робіт;
- рівномірність використання трудових ресурсів тощо.

Залежно від обраного показника проводять оптимізацію плану виконання проекту в часі. Зазвичай для таких розрахунків необхідно використовувати математичні методи та обчислювальну техніку.

Ефективність календарних планів значною мірою визначається якістю технологічної моделі, покладеної в їх основу. Відомо декілька видів таких моделей, що різняться за галузями використання: лінійні графіки, циклограми, сіткові моделі. Кожна модель повинна супроводжуватись усіма необхідними документами, такими як технологічні карти, графіки поставок, схеми руху ресурсів тощо.

Лінійні графіки - прості графічні методи, що дозволяють у масштабі часу показувати послідовність та терміни виконання робіт. Вони мають широке використання там, де необхідно наочно зобразити однозначну залежність та черговість робіт, проте їх використання є недоцільним при наявності складних

взаємозв'язків. Більш складними є **циклограми**, які дозволяють зобразити розвиток проекту в часі та у просторі. При цьому хід робіт зображають у вигляді нахилених ліній в системі координат. Використання такого графічного методу доцільне при наявності типових робіт або їх комплексів. З зростанням складності проекту наочність циклограм знижується і їх застосовування стає недоцільним.

Найкраще відобразити порядок реалізації проекту дозволяють **сіткові моделі**. Їх використання в календарному плануванні забезпечує розв'язання багатьох проблем, що виникають в процесі реального здійснення задумів.

Календарними планами можуть охоплюватись окремі етапи чи складові проекту, їх групи, створенні за певною ознакою, та весь проект в цілому. Незалежно від того, який з випадків розглядають, в основу закладається певна технологічна модель, а загальна задача календарного планування формулюється наступним чином.

Вихідні дані: об'єкти, що створюються, склад і характеристика необхідних для цього робіт, їх взаємозв'язки та спеціальні умови здійснення, обмеження у термінах виконання робіт, загальна кількість потрібних ресурсів в цілому та в розрізі часових проміжків, можливості використання конкретних ресурсів в певний час, а також їх характеристики. *Необхідно:* побудувати план робіт в часі, який задовольнятиме перераховані умови та буде оптимальним за наперед визначеним критерієм.

Побудований календарний план за своєю суттю визначає початок та завершення реалізації проекту та інтенсивність здійснення необхідних робіт. Одночасно з ним формуються графіки використання всіх ресурсів, передбачених технологічною моделлю. В процесі календарного планування вирішуються часові, ресурсні та вартісні задачі.

Часові задачі зводяться до узгодження робі в часі без урахування будь-яких обмежень у вартості, кількості чи рівномірності використання ресурсів. Їх результатом є розклад, що відображає терміни початку та завершення кожної роботи, певних етапів та проекту в цілому. Вихідними даними при цьому слугують тривалості кожної роботи, їх взаємозв'язки та при потребі необхідні терміни реалізації проекту.

У часових задачах можуть використовуватись і стохастичні моделі, де тривалості робіт розглядаються як вірогідні величини. В такому разі визначають ймовірності завершення проекту або його складових у встановлені строки. При розробці багатоваріантних рішень у календарному плануванні доцільно

використовувати моделі, де однозначно невизначеними є не лише тривалості робіт, а й їх склад. Звичайно, що має бути задана ймовірність виконання проекту тим чи іншим способом.

Зазвичай в реальних умовах необхідно враховувати ті чи інші обмеження щодо використання матеріальних, трудових чи фінансових ресурсів. Тому виникають ресурсні задачі календарного планування, які поділяють на задачі обліку потреби в ресурсах та задачі раціонального їх розподілу. Перші зводяться до побудови графіків загальної потреби в ресурсах для заданого варіанту календарного плану. Їх мета – дати характеристику розподілу потреби в ресурсах у часі та порівняти її з реальними можливостями. Другий тип задач, на відміну від попереднього, вирішується не на основі створеного вже календарного плану, а безпосередньо в процесі його формування. Це завдання оптимізації, оскільки при їх розв'язанні намагаються знайти найкращий за певним критерієм варіант розподілу ресурсів в часі.

Вартісні задачі календарного планування вирішуються на основі цінових характеристик, які тісно пов'язані з техніко-економічними показниками проекту або діяльністю підприємства, що його здійснює. В такому випадку у технологічних моделях мають бути відображені вартості робіт.

В процесі розв'язання ресурсних і вартісних задач календарного планування також можуть використовуватись стохастичні технологічні моделі. При цьому можуть мати місце невизначеності як самих моделей, так і потреби або наявності ресурсів, чи різні комбінування цих випадків. Завдання оптимізації календарного плану за таких умов є набагато складнішим, ніж за умови використання детермінованого підходу.

Планування послідовності робіт в часі є важливим інструментом, за допомогою якого можна вчасно довідатися про небезпеку зриву проекту вже на ранній стадії його реалізації. Це дозволяє прийняти своєчасні кроки для усунення загрози зриву проекту або коректування кінцевих результатів проекту.

Умови побудови сіткового графіка

Сіткові моделі поділяються на три основні класи через різну інтерпретацію дуг та вершин графа. Розрізняють стрілчасті графіки, графіки передування та сіткові діаграми. Для розуміння різниці між ними необхідно розрізняти такі елементи сіткових графіків як подія та робота.

Зв'язки передування (логічні залежності) - відображають природу залежностей між роботами. Вони утворюють структуру мережі. Комплекс

взаємозв'язків між роботами часто також називають логічною структурою проекту, оскільки він визначає послідовність виконання робіт.

Сіткові моделі класу **стрілчастих графіків** будуються тільки з використанням подій, що виступають вершинами графа. Кожна подія з'єднується з однією або декількома іншими подіями зв'язками, які мають оцінку в часі і цим визначають положення події в сітковій моделі. Ці зв'язки не являються роботами, а лише обумовлюють взаємозв'язок між подіями.

Сіткові моделі класу **графіків передування** базуються на роботах і визначають логіку процесу, який описується моделлю. У графічному вигляді вони зображуються за допомогою вершин, що відповідають роботам, зв'язаних лініями, які характеризують взаємозв'язки між роботами. Цей граф є найбільш розповсюдженим видом сітки на сьогоднішній день.

Третій тип діаграм поєднує два згаданих підходи. При його використанні робота на графі представляється у вигляді лінії між двома подіями (вершинами графа), які в свою чергу відображають початок та кінець даної роботи. Побудова такої сітки - досить складне завдання, тому її використання сьогодні не дуже поширене, хоча й традиційно збереглось у деяких сферах виробничої діяльності.

Сіткова діаграма не є блок-схемою в тому сенсі, у якому цей засіб використовується для моделювання ділових процесів. Її принциповою відмінністю від блок-схеми є те, що сіткова діаграма моделює тільки логічні залежності між елементарними роботами. Вона не відображає входи, процеси і виходи, і не допускає повторень або циклів.

При створенні сіткових моделей можливими є декілька способів їх формування. Перший - зведена сіткова модель будується централізовано (“зверху-вниз”), виходячи з логічної схеми виконання проекту. Другий - зведена сіткова модель створюється на основі зшивання первинних сіткових графіків, отриманих від відповідальних виконавців (“знизу-вверх”). Третім, найбільш поширеним, є спосіб, при якому будується укрупнений сітковий графік, який потім деталізується і коректується на основі первинних графіків відповідальних виконавців (“зверху - вниз” - “знизу-вверх”).

Перший спосіб отримав розповсюдження при формування сіткових моделей невеликого обсягу, а другий і третій - при розробці складних проектів. Найточнішою є сіткова модель, побудована за підходом “знизу-вверх”, оскільки безпосередній виконавець може надати найбільш деталізований графік роботи

на своїй дільниці. Але на практиці можлива ситуація проблематичності складання зведеного сіткового графіка через різний рівень деталізації первинних графіків, отриманих з різних ділянок. Дана проблема вирішується використанням підходу “зверху-вниз” - “знизу-вверх”, коли створена централізовано укрупнена сіткова модель задає необхідний рівень деталізації, що полегшує зшивання первинних графіків.

Первинні графіки робіт створюються на основі переліку необхідних робіт. При цьому побудову можна вести як зліва направо, тобто від початкової події, так і справа наліво, коли першою зображають кінцеву подію.

Побудова сіткового графіка.

Головні елементи сіткового графіка — події та роботи.

Подія — це результат, стан системи під час досягнення певної початкової, проміжної або кінцевої мети розробки. Подія не має тривалості в часі.

Початку графіка відповідає *перша* подія, кінцю — *завершальна* подія (їх може бути кілька), решта подій є *проміжними*. Події відповідають вершинам сіткового графіка і позначаються кільцями.

Робота — це тривалий за часом процес, необхідний для здійснення події.

На сіткових графіках ті чи інші роботи позначаються стрілками. Кожній роботі відповідає пара подій — *початкова* та *кінцева*.

Першим кроком в аналізі будь-якого проекту є впорядкування робіт, які в нього входять, з виокремленням безпосередньо випереджаючих робіт, тобто тих, виконання яких повинно бути закінчено раніше, ніж почнеться дана робота. Логічна послідовність виконання робіт зображується у вигляді сіткового графіка. З точки зору теорії графів, сітковий графік являє собою сітку, тобто скінчений зв’язаний орієнтований граф без петель з одним джерелом та одним або кількома стоками.

Розглянемо такі обов’язкові умови побудови сіткового графіка:

1) тільки перша подія не має вхідних стрілок, тільки завершальна — вихідних. Якщо подія за своїм характером проміжна, вона повинна мати як вхідні, так і вихідні стрілки;

2) кожна робота повинна мати початкову та кінцеву події;

3) на графіку не повинно бути окремих ділянок, які не пов’язані роботами з рештою операцій на графіку (умова зв’язаності сіткового графіка);

4) на графіку не повинно бути контурів і петель, оскільки вони по суті

означають, що умовою початку певної роботи є її закінчення. Якщо контур виникає (а в складних мережах це трапляється досить часто), необхідно повернутися до початкових даних і переглянути склад і послідовність робіт, домагаючись усунення контуру;

5)будь-які дві події повинні бути, по-перше, безпосередньо пов'язані не більш ніж з однією роботою. При виявленні на графіку паралельних робіт (рис. 5.1. *а*) вводяться фіктивна подія і фіктивна робота (подія 2′ і робота 2′-2 на рис. 5.1. *б*), а одна з паралельних робіт замикається на цю подію.

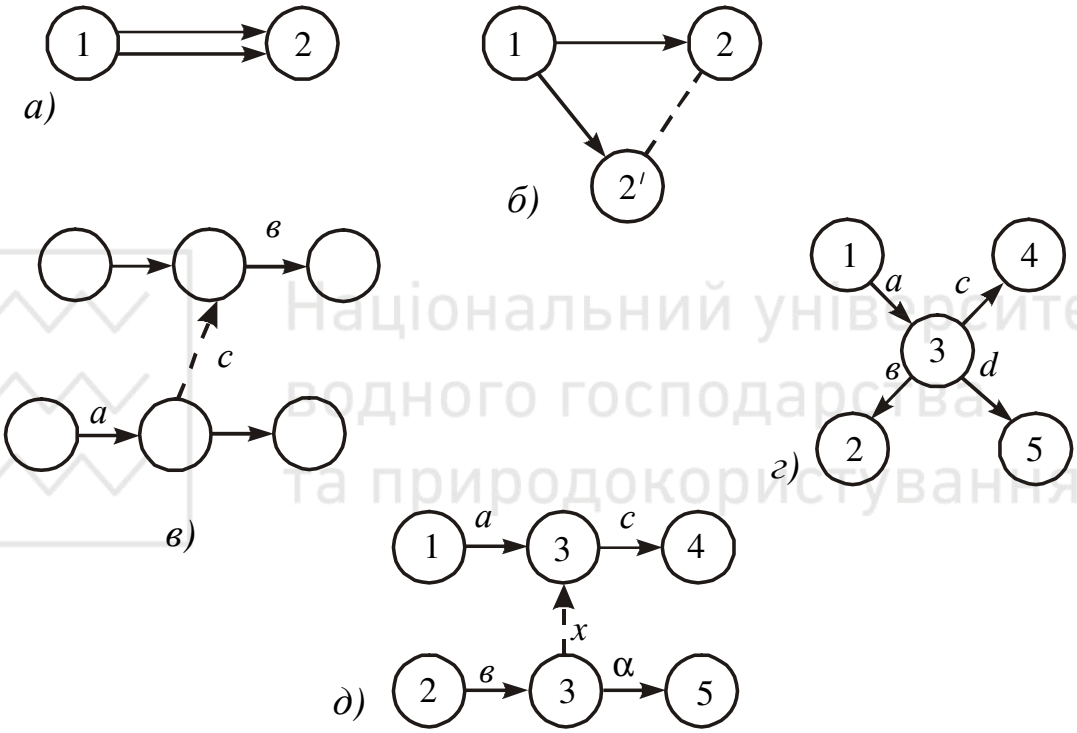


Рис. 5.1. Ситуації в сітковому графіку

Другий випадок, коли потребується введення фіктивних робіт, — це відображення залежності подій, не зв'язаних реальними роботами. Припустимо, наприклад, що роботи *а* і *в (*рис. 5.1. *в*) можуть виконуватися незалежно одна від одної, але потребують одних і тих же виконавців або устаткування. Отже, робота *в* не може початися, поки не звільняться виконавці або устаткування після виконання роботи *а.* Ці обставини вимагають введення фіктивної роботи *с.*

Третій випадок — неповна залежність робіт. Наприклад, робота *с* (рис. 5.1. *г*) вимагає для свого початку завершення робіт *а* і *в*, але робота *d* зв'язана тільки з роботою *в* і від роботи *а* не залежить. Тоді для правильного

зображення залежності робіт потрібно ввести фіктивну роботу *x* і фіктивну подію 3′, як показано на рис. 5.1. *д*.

У всіх трьох наведених випадках фіктивні роботи не мають тривалості в часі, однак без їх включення кількісний аналіз сіткового графіка може дати неправильні результати.

Четвертий випадок введення фіктивних робіт — відображення реальних відстрочок і очікувань. У ряді технологічних процесів вимагається природне затвердіння, висихання, дозрівання, бродіння тощо, коли реальна робота людей та обладнання не виконується, але наступний етап робіт до певного моменту починатися не може. В подібному випадку в сітковий графік вводяться фіктивні роботи, які мають відповідну тривалість у часі.

Технологія Agile (планування та управління розробкою ПП)

Гнучка методологія розробки (англ. agile software development) - це концептуальний каркас, у рамках якого виконується розробка програмного забезпечення.

Такий підхід викликаний тим, що вимоги до програмного забезпечення дуже мінливі в процесі його створення. Для цього продукт і система його створення мають бути гнучкими для своєчасних змін і обліку нових вимог.

Методики **Agile** дозволяють розробляти таке гнучке програмне забезпечення, яке відповідатиме вимогам сучасного бізнесу.

Agile (як і її варіації - методологія **Scrum**, методологія **Extreme Programming** тощо) має на увазі розробку програмного забезпечення короткими циклами (ітераціями).

Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає усі завдання, необхідні для видачі міні-приросту по функціональності, :

- планування
- аналіз вимог
- проектування
- кодування
- тестування
- документування.

Після закінчення етапу замовник отримує працюючу версію IT-системи, якщо вимагається - переглядає подальші пріоритети проекту, і цикл розробки запускається знову.

Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі що гнучкий програмний проект готовий до випуску у кінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Переваги Agile як гнучкої розробки програмного забезпечення :

1. Швидкий і постійний зворотний зв'язок команди розробників із замовником - зміни у вимогах користувачів оперативно враховуються.

2. Гнучкий графік реалізації функціональності - при використанні Agile, Scrum або Extreme Programming процес розробки легко перебудовується для кращої реакції на потребі бізнесу.

3. Акцент на ергономічність IT-системи - вимоги замовника до ергономіки системи мають рівний пріоритет з вимогами до її функціональності.

4. Відсутність витрат на формалізацію процесів і документації - Agile дозволяє уникнути витрат на роботи, які не завжди потрібні (наприклад, складання і узгодження детальної технічної документації).

5. Можливість зупинки проекту без збитку для здійснених вкладень в розробку IT-системи - кожна стадія проекту (за винятком початкових) закінчується створенням цілком готової версії IT-системи; є можливість (наприклад, при заморожуванні бюджету) зупинити проект і при цьому отримати працездатне рішення.

На практиці методологія Agile може використовуватися в декількох інтерпретаціях.

Забезпечити високу якість при реалізації проектів і при цьому укластися в стислі терміни нам допомагає:

1. Особлива увага до створення і оформлення програмного коду

2. Поєднання різної техніки розробки IT-системи: парне програмування, розробка через тестування, колективне володіння кодом тощо.

3. Використання методів безперервної інтеграції для зниження кількості помилок і постійного моніторингу виробничих процесів

4. Застосування апробованих технологій для управління проектом

5. Контроль над розподілом ресурсів і роботою IT-команди

У проектах для замовників найчастіше застосовує:

Scrum;

Extreme Programming;

Lean Software Development (LSD);

Dynamic Systems Development Method (DSDM);

Open Unified Process (OpenUP);

Agile Project Management (APM);

Microsoft Solutions Framework для Agile (MSF).

SCRUM

На думку ряду експертів одне з кращих рішень для підтримки методологій гнучкої розробки програмного забезпечення є Scrum.

SCRUM - одна з гнучких методологій розробки IT-систем. Уперше була використана в 1993 з метою поліпшити продуктивність команди розробників, зробивши упор не на якісно визначений, а на якісно контрольований процес розробки.

Ця методологія дозволяє надавати готовий продукт замовникам і кінцевим користувачам програмного забезпечення швидше, якісніше і дешевше, ніж при використанні звичайних методологій розробки.

Переваги цього підходу :

1. За дуже короткий проміжок часу project manager і усі члени команди можуть оцінити статус проекту.

2. Усі виниклі проблеми вирішуються дуже швидко оскільки доносяться до усіх учасників проекту (серед яких потенційно є компетентні в цьому питанні люди).

3. Співробітники вчаться слухати інших, розуміти їх, а також чітко виражати свої власні думки.

4. часники проекту вчаться ставити перед собою реальні завдання і відповідати за статус їх виконання.

В результаті замовник у будь-який момент бачить виконаний етап розробки і може своєчасно коригувати початкові характеристики продукту.

Основна термінологія SCRUM

Продукт - це IT-система, наприклад, яка розробляється на фреймворк .

Product backlog - по суті наші плани. Плани складаються на місяць.

User story - окремий пункт плану: проект або велике доопрацювання. Може розтягнутися на декілька місяців.

Sprint backlog - завдання, на які розбиваються пункти плану. Може мінятися в наслідок пересогласования плану.

Етапи при застосуванні SCRUM

Product backlog

На початку місяця після складання планів оголошуємо усі пункти і привласнюємо їм пріоритет. Пріоритет вплине на порядок виконання завдань.

Sprint backlog

Далі розбиваємо пункти на підзадачі. Формований Sprint backlog. На кожному завданні повинно бути відмічено скільки часу людина планує їй займатися. Виходячи з цього часу вибиратимуться завдання на спринт (Не можна, щоб час перевищив відведений на спринт час).

Sprint пропонується робити місяць, виключаючи вихідні дні. Це дозволить бачити чіткіше прогрес робіт на burndown chart і дошці.

За підсумками спринту усі завдання мають бути зроблені. Має бути проведена демонстрація замовникові поточного стану робіт і отриманий зворотний зв'язок.

Якщо завдання в поточному спринті не зроблене переносимо її на наступний.

Оскільки спринт займає місяць, а це достатньо довго, то кожну п'ятницю проводиться **зріз робіт**.

На зрізі обговорюється поточний стан справ, у разі відхилення від ідеального вигорання завдань обговорюється куди пішов час. Також необхідно провести демонстрацію робіт, що виконуються в цей тиждень, замовникові і отримати від нього зворотний зв'язок.

Проблеми, що виникають при виконанні Sprint

Це можуть бути завдання в зборку, доопрацювання і виправлення помилок. Те, що з'являється "несподівано". В плані ці пункти мають бути присутніми. Тобто, в кожному спринті з'являються два зарезервовані завдання: "доопрацювання" і "виправлення помилок". На цих завданнях ми відмічаємо витрачений час. У кінці спринту таке завдання буде гарантовано закрито. Потрібновиконувати збір метрик та фіксація витраченого часу на помилки/доопрацювання часу.

Daily Meeting. Наради (щоденні мітинги) членів команди. Може проходити щодня в 16:00. Кожному учасникові команди відводиться 1 хвилина. Розповідаючи, він відповідає на питання:

- що робив;
- з якими проблемами зіткнувся;
- що робитиме далі.

На щоденні мітинги запрошуються всі члени команди, для того, щоб бути в курсі того, що відбувається у проекті.

Scrum –дошка. Відображає план роботи над проектом та рівень його виконання. Відображається графа для черги. Це ті завдання, які ще не включені ні в один спринт. Дошка містить графи:

1. Чекає обробки
2. На виконанні
3. Чекає тестування/перевірки
4. На перевірці. Цей стан означає, що хтось з розробників перевіряє завдання.
5. Тестування
6. Зроблено

Sprint review. По суті це здача планів. У них входить демонстрація результату замовникові.

Sprint retrospective. Проводиться після закінчення кожного спринту, тобто в п'ятницю після щоденного мітингу. В цей час кожен може висловити свою думку/побажання/враження про роботу виконаної їм особисто або відділом за тиждень. Також на п'ятничний мітинг запрошується група автоматизації тестування.

Product retrospective. Демонстрація результату роботи над проектами. Можливо, також підключення зацікавлених людей. Пропонується за підсумками кожного місяця проводити внутрішній семінар, на якому кожен розповість що нового він зробив. Також можливе обговорення проблем з якими зіткнулися.

Змістовий модуль 2. Програмні засоби моделювання в IT-проектах

Тема 6. Поняття CASE-технологій та їх призначення.

План:

1.Поняття CASE-технологій та їх призначення.

2. Концептуальні основи CASE-технології.

Поняття CASE-технологій та їх призначення.

Тенденції розвитку сучасних інформаційних технологій приводять до постійного зростання складності інформаційних систем (ІС), створюваних у різних галузях.

Для успішної реалізації проекту об'єкт проектування (ІС) повинен бути насамперед адекватно описаний, повинні бути побудовані повні і несуперечливі функціональні та інформаційні моделі ІС. Крім того, у процесі створення і функціонування ІС інформаційні потреби користувачів можуть змінюватися чи уточнюватися, що ще більш ускладнює розробку і супровід таких систем.

Приблизно чверть століття тому швидко зростаючий обсяг і складність систем вступили в явне протиріччя з відсутністю єдиного підходу до їх аналізу і проектування, неучастю користувача в процесі розробки, непогодженістю різних етапів розробки. Помилок було багато й обходилися вони дуже дорого. Модульне і структурне програмування, логічне моделювання структур баз даних, схеми потоків даних і проектування "зверху вниз" при всій початковій ейфорії, узагалі ж, залишилися внутрішньою справою розроблювачів. Проблема була глибше - потрібно було якось об'єднати замовників, розроблювачів, програмістів, користувачів - причому в умовах постійно мінливої ситуації. А для того, щоб про щось домовитися, потрібна якась спільна мова. Природна мова в силу малої наочності, неоднозначності, надмірності і багатослівності для цієї ролі не пасувала, і, зрештою, почалися спроби створення чіткої графічної мови.

Перераховані фактори сприяли появі програмно-технологічних засобів спеціального класу - CASE-засобів, що реалізують CASE-технологію створення і супроводу ІС. Термін CASE (Computer Aided Software Engineering) використовується в даний час у дуже широкому сенсі. Первісне значення терміна CASE, обмежене питаннями автоматизації розробки тільки лише ПП, у

даний час набуло нового сенсу, що охоплює процес розробки складних ІС у цілому. Тепер під терміном CASE-засобу розуміються програмні засоби, що підтримують процеси створення і супроводу ІС, включаючи аналіз і формулювання вимог, проектування прикладного ПП (додатків) і баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне управління і управління проектом, а також інші процеси. CASE-засоби разом із системним ПП і технічними засобами утворять повне середовище розробки ІС.

Появі CASE-технології і CASE-засобів передували дослідження в області методології програмування. Програмування знайшло риси системного підходу з розробкою і впровадженням мов високого рівня, методів структурного і модульного програмування, мов проектування і засобів їх підтримки, формальних і неформальних мов описів системних вимог і специфікацій і т.д. Крім того, появі CASE-технології сприяли і такі фактори, як:

- підготовка аналітиків і програмістів, сприйнятливих до концепцій модульного і структурного програмування;
- широке впровадження і постійний ріст продуктивності комп'ютерів, що дозволили використовувати ефективні графічні засоби й автоматизувати більшість етапів проектування;
- упровадження мережної технології, що надала можливість об'єднання зусиль окремих виконавців у єдиний процес проектування шляхом використання поділюваної бази даних, що містить необхідну інформацію про проект.

CASE-технологія являє собою методологію проектування ІС, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки і супроводу ІС і розробляти додатка відповідно до інформаційних потреб користувачів. Більшість існуючих CASE-засобів засновано на методологіях структурного (в основному) чи об'єктно-орієнтованого аналізу і проектування, що використовують специфікації у виді чи діаграм текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поводження системи й архітектури програмних засобів.

При використанні методологій структурного аналізу з'явився ряд обмежень (складність розуміння, велика трудомісткість і вартість використання, незручність внесення змін у проектні специфікації тощо). Із самого початку

CASE-технології і розвивалися з метою подолання цих обмежень шляхом автоматизації процесів аналізу й інтеграції підтримуючих засобів.

Концептуальні основи CASE-технології

У випадку взаємодії розроблювачів системи з користувачами, основною на етапі аналізу є проблема, пов'язана з взаємодією з замовником, тому що вони розмовляють "на різних мовах", що мають надто багато подробиць, що є несуттєвими для іншої сторони. У цьому відношенні неоціненною перевагою CASE-підходу є надання "універсальної" графічної мови різного роду діаграм, що підпорядковані певним нотаціям. Застосування стандартних нотацій забезпечує читабельність програм, а також можливість переносу частин (чи всього проекту) в інші CASE-засоби. Таким чином, кінцевому користувачу продукту (замовнику) зовсім не обов'язково володіти всіма премудростями розроблювача, за допомогою CASE-підходу і замовник і розроблювач будуть розуміти один одного однозначно.

У структурному аналізі використовуються в основному три групи засобів, що ілюструють функції, виконувані системою і відносини між даними. Кожній групі засобів відповідають певні види моделей (діаграм), найбільш розповсюдженими серед який є наступні:

- SADT (Structured Analysis and Design Technique) моделі і відповідні функціональні діаграми;
- DFD (Data Flow Diagrams) діаграми потоків даних;
- STD (State Transition Diagrams) діаграми переходів станів;
- ERD (Entity-Relationship Diagrams) діаграми "сутність-зв'язок".

У залежності від спрямованості CASE-продукту, він може підтримувати різного роду діаграми.

На стадії проектування ІС моделі розширюються, уточнюються і доповнюються діаграмами, що відбивають структуру програмного забезпечення: архітектуру ПП, структурні схеми програм і діаграми екранних форм.

Перераховані моделі в сукупності дають повний опис ІС незалежно від того, чи є вона існуючої чи знову розроблювальної. Склад діаграм у кожнім конкретному випадку залежить від необхідної повноти опису системи.

Функції CASE-технологій в управлінні IT-проектами.

CASE-технологія являє собою методологію проектування ІС, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки і супроводу ІС і розробляти додатка відповідно до інформаційних потреб користувачів. Більшість існуючих CASE-засобів засновано на методологіях структурного (в основному) чи об'єктно-орієнтованого аналізу і проектування, що використовують специфікації у виді чи діаграм текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поводження системи й архітектури програмних засобів.

При використанні методологій структурного аналізу з'явився ряд обмежень (складність розуміння, велика трудомісткість і вартість використання, незручність внесення змін у проектні специфікації і т.д.) Із самого початку CASE-технології і розвивалися з метою подолання цих обмежень шляхом автоматизації процесів аналізу й інтеграції підтримуючих засобів. Вони мають достоїнства і можливостями, перерахованими нижче.

Єдина графічна мова.

CASE-технології забезпечують всіх учасників проекту, включаючи замовників, єдиною строгою, наочною і інтуїтивно зрозумілою графічною мовою, що дозволяє одержувати доступні для огляду компоненти з простій і ясну структуру. При цьому програми представляються двовимірними схемами (які простіше використовувати, ніж багато сторінок текстового опису), що дозволяють замовнику брати участь у процесі розробки, а розроблювачам - спілкуватися з експертами предметної області, розділяти діяльність системних аналітиків, проектувальників і програмістів, полегшуючи їм захист проекту перед керівництвом, а також забезпечуючи легкість супроводу і внесення змін у систему.

Єдина БД проекту.

Основа CASE-технології - використання бази даних проекту (репозиторію) для збереження всієї інформації про проект, що може розділятися між розроблювачами відповідно до їхніх прав доступу.. Уміст репозиторію включає не тільки інформаційні об'єкти різних типів, але і відносини між їх компонентами, а також правила чи використання обробки цих компонентів..Репозиторій може зберігати понад 100 типів об'єктів: структурні

діаграми, визначення екранів і меню, проекти звітів, опису даних, логіка обробки, моделі даних, їхні організації й обробки, вихідні коди, елементи даних тощо.

Інтеграція засобів.

На основі репозиторію здійснюється інтеграція CASE-засобів і поділ системної інформації між розроблювачами.. При цьому можливості репозиторію забезпечують кілька рівнів інтеграції: спільний користувальницький інтерфейс по всіх засобах, передачу даних між засобами, інтеграцію етапів розробки через єдину систему представлення фаз життєвого циклу, передачу даних між різними платформами.

Підтримка колективної розробки й управління проектом. CASE-технологія підтримує групову роботу над проектом, забезпечуючи можливість роботи в мережі, експорт-імпорт будь-яких фрагментів проекту для їхнього розвитку і/чи модифікації, а також планування, контроль, адміністрування і взаємодія, тобто функції, необхідні в процесі розробки і супроводу проектів. Ці функції також реалізуються на основі репозиторію. Зокрема, через репозиторій може здійснюватися контроль безпеки (обмеження і привілеї доступу), контроль версій і змін і ін.

Макетування. CASE-технологія дає можливість швидко будувати макети (прототипи) майбутньої системи, що дозволяє замовнику на ранніх етапах розробки оцінити, наскільки вона прийнятна для майбутніх користувачів і влаштовує його.

Генерація документації. Уся документація у проекті генерується автоматично на базі репозиторію (як правило, відповідно до вимог діючих стандартів). Безсумнівне достоїнство CASE-технології полягає в тім, що документація завжди відповідає поточному стану справ, оскільки будь-які зміни в проекті автоматично відбиваються в репозиторії (відомо, що при традиційних підходах до розробки ПП документація в кращому випадку запізнюється, а ряд модифікацій узагалі не знаходить у ній відображення).

Верифікація проекту. CASE-технологія забезпечує автоматичну верифікацію і контроль проекту на повноту і переконливість на ранніх етапах розробки, що впливає на успіх розробки в цілому - по статистичним даним аналізу п'яти великих проектів фірми TRW (США) помилки проектування і кодування складають відповідно 64% і 32% від спільного числа помилок, а

помилки проектування в 100 разів сутужніше знайти на етапі супроводу ПП, чим на етапі аналізу вимог.

Автоматична генерація об'єктного коду. Генерація програм у машинному коді здійснюється на основі репозиторію і дозволяє автоматично побудувати до 85-90% об'єктного чи коду текстів на мовах високого рівня.

Супровід і реінжинірінг. Супровід системи в рамках CASE-технології характеризується супроводом проекту, а не програмних кодів. Засоби реінжинірінгу та зворотного інжинірингу дозволяють створювати модель системи з її кодів і інтегрувати отримані моделі в проект, автоматично обновляти документацію при зміні кодів, автоматично змінювати специфікації при редагуванні кодів і т.п.

Критерії оцінки ефективності CASE - системи

1. CASE системи повинні забезпечувати економію часу проектування.
2. Повинні забезпечувати поліпшення якості створюваного програмного продукту.
3. Повинні забезпечувати економію ресурсу (економічні, людські, технічні)

Багато сучасні CASE - системи мають:

1. Графічний інтерфейс, який дозволяє в тому або іншому вигляді описати проект.
2. Загальну БД проекту або репозиторій, де зберігається вся інформація про проект.
3. Інтеграцію засобів проектування.
4. Засоби генерації документації.
5. Засоби верифікації проекту.
6. Засоби автоматичної генерації коди.

Репозиторій є об'єктно-орієнтованою базою даних. Засоби перегляду забезпечують "навігацію" за проектом, зокрема, переміщення по ієрархіях класів і підсистем, перемикання від одного вигляду діаграм до іншого і так далі Засоби контролю і збору статистики дають можливість знаходити і усувати помилки у міру розвитку проекту, а не після завершення його опису. Генератор звітів формує тексти вихідних документів на основі інформації, що міститься в репозиторії.

Засоби автоматичної генерації код програм на мові програмування, використовуючи інформацію, що міститься в логічній і фізичній моделях проекту, формують файли заголовків і файли описів класів і об'єктів.

Створюваний таким чином скелет програми може бути уточнений шляхом прямого програмування на мові програмування. Аналізатор код, як правило, реалізований у вигляді окремого програмного модуля. Його призначення полягає в тому, щоб створювати модулі проектів і здійснювати контроль правильності початкових текстів і діагностику помилок. Модель, отримана в результаті його роботи, може цілком або фрагментарно використовуватися в різних проектах. Аналізатор володіє широкими можливостями настройки по входу і виходу. Наприклад, можна визначити типи початкових файлів, базовий компілятор, задати, яка інформація повинна бути включена у формовану модель і які елементи вихідної моделі слід виводити на екран.

В результаті розробки проекту за допомогою CASE-средств формуються наступні документи:

- діаграми класів;
- діаграми станів;
- діаграми сценаріїв;
- діаграми модулів;
- діаграми процесів;
- специфікації класів, об'єктів, атрибутів і операцій
- заготовки текстів програм;
- модель програмної системи, що розробляється.

Тема 7. Засоби CASE-технологій в управлінні IT-проектами

План:

1. Засоби CASE-технологій в управлінні IT-проектами.
2. Класифікація CASE засобів.
3. Характеристика сучасних CASE-засобів.
3. Microsoft Project .

Засоби CASE-технологій в управлінні IT-проектами

Засоби функціонального моделювання

Для рішення задачі функціонального моделювання на базі структурного аналізу традиційно застосовуються два типи моделей: SADT-діаграми і діаграми потоків даних.

DFD - показують зовнішні джерела і стоки даних, визначають процеси обробки і потоки даних, ідентифікують сховища даних (накопичувачі). Структура потоків даних зберігається в Словнику даних. Будь-яка DFD може бути деталізована DFD нижнього рівня і т.д. поки доцільна деталізація.

У випадку наявності в модельованій системі програмної/програмувальної частини (тобто практично завжди) перевага, як правило, віддається DFD по наступним розумінням.

1. DFD із самого початку створювалися як засіб проектування програмних систем (тоді як SADT - як засіб проектування систем узагалі) і мають більш багатий набір елементів, що адекватно відбивають їхню специфіку (наприклад, сховища даних є прообразами файлів чи баз даних).
2. Наявність міні-специфікацій DFD-процесів нижнього рівня дозволяє перебороти логічну незавершеність SADT (а саме обрив моделі на деякому досить низькому рівні, коли подальша її деталізація стає безглуздою) і побудувати повну функціональну специфікацію розроблювальної системи.
3. Існують (і підтримуються поруч CASE-пакетів) алгоритми автоматичного перетворення ієрархії DFD у структурні карти, що демонструють міжмодульні і внутрімодульні зв'язки, а також ієрархію модулів, що в сукупності з міні-специфікаціями є завершеним завданням для програміста.

Нарешті, у частині автоматизованої підтримки моделей приблизно 85-90% існуючих CASE-пакетів підтримують DFD і лише 2-3% - SADT.

Засоби подійного моделювання

Традиційний підхід до моделювання аспектів поведінки системи ґрунтується на розширенні діаграм потоків даних за рахунок введення керуючих потоків (сигналів) і керуючих процесів, що фактично є інтерфейсом між DFD і специфікаціями управління, власне моделююче поводження. Найбільше часто специфікації управління формалізуються за допомогою діаграм переходів станів (STD - state transition diagrams), що дозволяють задавати стану різних об'єктів системи (наприклад, особовий рахунок може мати стану ВІДКРИТИЙ, ЗАКРИТИЙ, ЗАБЛОКОВАНИЙ і т.п.), умови переходів з одного стану в інше (як зовнішні стосовно системи, так і внутрішні, виникаючі в самій системі), а також чинені при переходах дії.

Засоби інформаційного моделювання

Для цілей інформаційного моделювання на сьогоднішній день не існує альтернативи діаграмам "сутність-зв'язок" (ERD - entity-relationship diagrams).

Вміст накопичувача даних зберігається в Словнику даних і розкривається за допомогою ERD (даної діаграми в основному використовуються при проектуванні БД, зокрема продуктом Logic Works - ERWin- засобом для розробки моделей даних). У випадку наявності реального часу DFD доповнюються STD.

Характеристика сучасних CASE-засобів

Сучасні CASE-засоби охоплюють велику галузь підтримки численних технологій проектування ІС: від простих засобів аналізу і документування до повномасштабних засобів автоматизації, що покривають весь життєвий цикл ПП.

До числа CASE-засобів попадають як відносно дешеві системи для персональних комп'ютерів з дуже обмеженими можливостями, так і дорогі системи для неоднорідних обчислювальних платформ і операційних середовищ. Так, сучасний ринок програмних засобів нараховує близько 300 різних CASE-засобів, найбільш могутні з який так чи інакше використовуються практично усіма ведучими західними фірмами.

Звичайно до CASE-засобів відносять будь-який програмний засіб, що автоматизує ту чи іншу сукупність процесів життєвого циклу ПП та мають наступні основні характерні риси:

- могутні графічні засоби для опису і документування ІС, що забезпечують зручний інтерфейс із розроблювачем і розвиваючі його творчі можливості;
- інтеграція окремих компонентів CASE-засобів, що забезпечує керованість процесом розробки ІС;
- використання спеціальним образом організованого сховища проектних метаданих (репозиторію).

Інтегрований CASE-засіб (чи комплекс засобів, що підтримують повний ЖЦ ПП) містить наступні компоненти;

- репозиторій, що є основою CASE-засобу. Він повинен забезпечувати збереження версій проекту і його окремих компонентів, синхронізацію надходження інформації від різних розроблювачів при груповій розробці, контроль метаданих на повноту і несуперечність;

- графічні засоби аналізу і проектування, що забезпечують створення і редагування ієрархічно зв'язаних діаграм (DFD, ERD і ін.), що утворять моделі ІС;
- засоби розробки додатків, включаючи мови 4GL і генератори кодів;
- засоби конфігураційного управління;
- засоби документування;
- засоби тестування;
- засоби управління проектом;
- засоби реінжинірінга.

Класифікація CASE-засобів

Усі сучасні CASE-засоби можуть бути класифіковані в основному за типами і категоріями. Класифікація по типах відбиває функціональну орієнтацію CASE-засобів на ті чи інші процеси ЖЦ. Класифікація по категоріях визначає ступінь інтегрованості по виконуваних функціях і включає окремі локальні засоби, що вирішують невеликі автономні задачі (tools), набір частково інтегрованих засобів, що охоплюють більшість етапів життєвого циклу ІС (toolkit) і цілком інтегровані засоби, що підтримують весь ЖЦ ІС і пов'язані спільним репозиторієм. Крім цього, CASE-засоби можна класифікувати за наступними ознаками:

- застосовуваним методологіям і моделям систем і БД;
- ступенем інтегрованості із СУБД;
- доступним платформам.

Класифікація за типами в основному збігається з компонентним складом CASE-засобів і включає наступні основні типи:

- засоби аналізу (Upper CASE), призначені для побудови й аналізу моделей предметної галузі (Design/IDEF, BPwin);
- засоби аналізу і проектування (Middle CASE), що підтримують найбільш розповсюджені методології проектування і, що використовуються для створення проектних специфікацій (Vantage Team Builder, Designer/2000, Silverrun, PRO-IV, CASE.Аналітик). Виходом таких засобів є специфікації компонентів і інтерфейсів системи, архітектури системи, алгоритмів і структур даних;
- засоби проектування баз даних, що забезпечують моделювання даних і генерацію схем баз даних (як правило, мовою SQL) для найбільш розповсюджених СУБД. До них відносяться ERwin, S-Designor і DataBase

Designer (ORACLE). Засобу проектування баз даних маються також у складі CASE-засобів Vantage Team Builder, Designer/2000, Silverrun і PRO-IV;

- засоби розробки додатків. До них відносяться засоби 4GL (Uniface, JAM, PowerBuilder, Developer/2000, New Era, SQLWindows, Delphi і ін.) і генератори кодів, що входять до складу Vantage Team Builder, PRO-IV і частково - у Silverrun;
- засоби реінжинірінга, що забезпечують аналіз програмних кодів і схем баз даних і формування на їхній основі різних моделей і проектних специфікацій. Засоби аналізу схем БД і формування ERD входять до складу Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin і S-Designor. У галузі аналізу програмних кодів найбільше поширення одержують об'єктно-орієнтовані CASE-засоби, що забезпечують реінжинірінг програм мовою C++ (Rational Rose, Object Team).

Допоміжні типи включають:

- засоби планування й управління проектом (SE Companion, Microsoft Project і ін.);
- засоби конфігураційного управління (PVCS, SCCS і ін.);
- засоби тестування (Quality Works і ін.).

Характеристика сучасних CASE-засобів

На сьогодні у світі розроблено кілька сотень систем, які реалізують функції календарного планування і контролю проектів. Але реально на вітчизняному і російському ринках представлені не більш як 10 програм, серед яких — Microsoft Project, Open Plan Professional, Spider Project, Sure Trek Project Manager, Primavera Project Planner (P3), Time Line, CA Super Project, Project Scheduler, Turbo Project, Artemis Views.

Open Plan — це професійна система управління проектами, яка характеризується, зокрема, потужними засобами ресурсного і бюджетного планування, що дозволяють значно полегшити знаходження якнайефективнішого розподілу ресурсів і складання робочого розкладу їх.

Основні характеристики Open Plan

1. Створення моделі проекту.

Open Plan має найпотужніші засоби структуризації моделі проекту, які

базуються на:

- ієрархічній структурі робіт (Work Breakdown Structure);
- сітковій моделі (PERT-діаграма);
- ієрархічній структурі ресурсів;
- ієрархічній системі кодування робіт.

Система Open Plan надає гнучкі й зручні засоби для формування ієрархічної структури робіт. Менеджер може формувати необмежену кількість рівнів ієрархії проекту. Open Plan забезпечує широкі можливості для створення логічної структури проекту, включаючи будь-які типи зв'язку між завданнями. Під час планування допускається складання календаря для робіт і зв'язків між ними, а також урахування цільових дат початку і завершення окремих робіт.

2. Управління ресурсами

Система Open Plan дозволяє управляти всіма видами ресурсів, а саме: відновлюваними ресурсами (люди, обладнання); невідновлюваними ресурсами (матеріали), у тому числі ресурсами з обмеженим терміном придатності; і фінансами, які описуються в ресурсному файлі.

Кількість наявних ресурсів на будь-який момент реалізації проекту описується параметром доступності. Для відновлюваних ресурсів (наприклад будівельники) цей параметр визначається доступною їх чисельністю в певні часові інтервали. Для невідновлюваних ресурсів (наприклад будівельні матеріали) — це загальна кількість і дата, з якої ресурс надходить у розпорядження, для ресурсів з обмеженим терміном — загальна кількість і часовий проміжок, упродовж якого ресурс можна використати.

3. Планування і контроль витрат

Open Plan дозволяє реалізувати такі функції з планування і контролю витрат:

1 розрахунок витрат за проектом з урахуванням і без урахування змін у вартості ресурсів;

2 «запам'ятовування» кількох прогнозних варіантів виконання проекту в різні терміни для пошуку «найекономнішого» часу реалізації;

3 можливість автоматичного розрахунку витрат на основі кількості відпрацьованих ресурсних одиниць;

4 аналіз вартості за фактичним обсягом.

4. Аналіз ризиків

Система Open Plan має у своєму розпорядженні аналітичні інструменти, які базуються на методі Монте-Карло і дозволяють визначити можливі ризики в

оцінці термінів завершення окремих робіт, етапів і всього проекту. Таким чином, оцінюється ймовірність відхилення термінів виконання робіт від графіка і, звідси, перевищення бюджету, а також інші негативні наслідки.

Аналіз ризиків у Open Plan реалізується такими засобами:

1 процедурами введення оптимістичних і песимістичних оцінок параметрів для певних чи всіх робіт проекту;

2 виконанням аналізу ризиків за методом Монте-Карло для обчислення ймовірності завершення робіт за проектом у визначені терміни;

3 підготовка звітів, які використовуються для аналізу впливу невизначеності на реалізацію проекту.

5. Багатопроектне планування

Можливість роботи в багатопроектному режимі дозволяє користувачам розглядати великий проект як проект, який складається з менших субпроектів, і здійснювати більш гнучке управління ним на різних рівнях. Робота в багатопроектому режимі надає засоби для контролю і розподілу єдиних ресурсів організації за всіма проектами, які вона здійснює.

Об'єднання проектів, таким чином, слугує двом цілям: по-перше, можна здійснювати аналіз завантаження ресурсів у масштабах проектів усього підприємства, по-друге, є можливість забезпечити середовище для інтегрованого програмного управління великими комплексними проектами, поділеними на субпроекти. В другому випадку в кожного підпроекту може бути свій файл ресурсів.

Основні характеристики Spider Project

1. Роботи і взаємозв'язки між роботами

Існуючі пакети з управління проектами роботи здебільшого характеризуються тривалістю їх виконання. У Spider Project замість тривалості можна задавати фізичні обсяги робіт. Тоді тривалість визначається програмою в процесі створення розкладу робіт залежно від продуктивності необхідних ресурсів. У Spider Project використовуються ті самі типи взаємозв'язків, що і в інших пакетах. Відмінності є у визначенні затримок: поряд з часовими затримками можна використовувати і затримки за обсягами робіт.

2. Формування розкладу проекту і розрахунок критичного шляху

Програма Spider Project дозволяє, крім традиційного критичного шляху, визначити ресурсний критичний шлях і резерви виконання робіт, враховуючи обмеженість ресурсів. Розклад виконання проекту можна обчислити, зважаючи

не тільки на обмеженість відновлюваних ресурсів, а й на графіки постачання і фінансування проекту, причому не тільки за сумарними витратами, а й за окремими складовими і центрами витрат і матеріалів.

3. Ієрархічні структури

У Spider Project можна використовувати необмежену кількість різних ієрархічних структур робіт і ресурсів. Окрім того, можна створювати так звані неповні структури, які не включають в себе усі роботи проекту. Неповні структури — зручний інструмент для підготовки звітів та аналізу окремих аспектів проекту. Прикладом такої неповної структури може бути структура постачання, в яку входять лише ті операції, які відображають постачання матеріалів для проекту.

4. Ресурси

Ресурси — їх поділяють на відновлювані (люди, обладнання) і невідновлювані (матеріали) — задають окремо. При цьому можна додатково вказувати, які матеріали використовуються як відновлювані ресурси, тоді, визначивши останні, можна автоматично передбачити використання необхідних матеріалів.

Крім окремих ресурсів можна задавати мультиресурси і пули. Мультиресурси — це групи ресурсів, які виконують роботу спільно (наприклад, бригада, програміст з комп'ютером і т. ін.). Їх можна призначати на виконання роботи повністю, що означає призначення всіх ресурсів, які входять до мультиресурсу. Пули — це групи взаємозамінних ресурсів. Використання ресурсних пулів позбавляє менеджера необхідності жорстко призначати виконавців на роботи проекту. Йому достатньо вказати загальну чисельність необхідних для виконання робіт ресурсів, а також те, з яких ресурсів цю кількість вибрати. Це дозволяє скоротити непродуктивні простої ресурсів і полегшити роботу менеджера проекту. Основна відмінність від підходів, що їх використовують в інших пакетах, полягає в тому, що ресурси пула можуть мати різну продуктивність.

5. Призначення ресурсів

У Spider Project в ході призначення ресурсів на виконання робіт проекту з'являється поняття команди, тобто групи ресурсів, які виконують роботи спільно. До команди можуть входити як окремі ресурси, так і мультиресурси, і пули. Ресурси можуть бути призначені на виконання робіт частково, тоді задається завантаження призначених ресурсів у відсотках поряд із кількістю. Матеріали можуть бути призначені прямо на операцію або на ресурс (тоді

можна отримати звіт про використання матеріалів окремими ресурсами).

6. Витрати

Крім призначення вартості години роботи відновлюваного ресурсу і вартості одиниці матеріалів, витрати можна розподіляти безпосередньо по роботах. Наприклад, якщо робота виконується за контрактом із фіксованою ціною, то немає сенсу визначати вартість години роботи ресурсу, а треба просто використовувати сумарну вартість призначення ресурсу (підрядчика) на роботу.

7. Аналіз ризиків

Алгоритм аналізу ризиків відрізняється від реалізованих в інших системах тим, що під час моделювання ризиків як початкову інформацію використовують не оцінки тривалості робіт (оптимістичні, песимістичні), а оцінки продуктивності ресурсів.

8. Групова робота над проектом

Програма Spider Project не передбачає одночасного доступу до зміни даних. Відповідальний за певну частину проекту (фазу) надає менеджерові проекту свої файли, і рішення прийняти чи відкинути зміни залишається за менеджером проекту. Саме таке рішення, на думку розробників, дозволяє уникнути плутанини під час зміни проектних даних. З цих позицій розроблена і система групової роботи через Інтернет.

Microsoft Project

Основні характеристики Microsoft Project

На сьогодні найбільш поширена у світі система управління проектами завдяки поєднанню простоти використання, дружнього інтерфейсу і найнеобхідніших інструментів для управління проектами, розрахованих передусім на користувачів, які не є професіоналами у сфері управління проектами. В багатьох західних компаніях Microsoft Project — це звичний додаток до Microsoft Office навіть для рядових працівників, які використовують його для планування графіків нескладних комплексів робіт.

Microsoft Project — один із лідерів за можливостями об’єднання учасників проекту засобами електронної пошти або Інтранет. При описанні ресурсу для кожного виконавця може бути вказана адреса його електронної пошти. Тоді для поширення інформації серед учасників проекту досить виконати команду Team Assing, а для отримання інформації про стан робіт — команду Team Status. Інформація про роботи проекту може зберігатися у форматі NTML і друкуватися на внутрішньому Web-сервері.

Серед переваг Microsoft Project — досить гнучкі й зручні засоби створення звітів. Основні типи звітів можуть бути вибрані з (Report Gallary).

Крім стандартних форматів файлів Microsoft Project: MPP і MPX, користувач може зберігати інформацію по проекту в форматах ODBC, Excel і Access. Формат MPD (Microsoft Project Database) дозволяє зберігати всі дані про проект у структурі, доступній як з MS Project 98, так і з Access 8.0.

Для швидкого включення в роботу початківців Microsoft Project надає крім звичайних засобів допомоги також можливість покрокової розробки проекту (Create Your First Project) та інтелектуальної підказки (Answer Wizard). Microsoft Project не русифікований, тому для ефективного використання цих засобів потрібне знання англійської мови, зокрема термінології управління проектами.

Основним недоліком Microsoft Project є те, що цей пакет надає мінімальний набір засобів для планування й управління ресурсами. У Microsoft Project 98 як ресурси можна планувати лише людей і обладнання.

Принципи сіткового планування і управління та їх реалізація у системі Microsoft Project

Будь-який проект ґрунтується на стандартному ЖЦ, який може включати наступні етапи:

1. формування мети проекту;
2. формування списку мироприємств (завдань) та визначення їх логічних взаємозв’язків;
3. визначення часу, що потрібен для виконання робіт та ресурсів на кожне завдання;
4. планування робіт проекту з врахуванням потреби та можливості застосування потрібних ресурсів та визначення терміну реалізації проекту та ресурсів в цілому;
5. реалізація проекту, що полягає у контролі за термінами виконання, ресурсами. Що виконуються, вартістю проекту на підставі чого здійснюється коригування планів;
6. завершення проекту і оцінювання результатів.

Процеси корегування плану являють собою процес багатократного повторення планування. На відміну від планування початкового, яке охоплює весь проект коли він ще не почав виконуватись корегування планування здійснюється на етапі виконання і призводить до оптимізації проектних робіт.

MS Project забезпечує реалізацію таких завдань:

1. покрокову розробку проекту та інтелектуальне його ведення;
2. створення завдань, визначення ресурсів, оцінку термінів, виконання проектних робіт на рівнях, наближених до реальності;
3. можливість присвоєння ієрархічних кодів для завдань та ресурсів;
4. встановлення зв'язків між окремими завданнями та визначення пріоритетів, що дає можливість автоматизовано вирівнювати ресурсні потреби;
5. використовувати всі ресурси, що є наявними на підприємстві і задіяні в різних проектах з елементами оптимізації в межах свого підприємства;
6. розрахунок практичного шляху певного ланцюга проектів;
7. наочна ілюстрація процесу виконання проектів, визначення вузьких проблемних місць за допомогою графічних індикаторів;
8. дозволяє проводити обчислення за формулами, що визначені користувачем;
9. встановлення наближених періодів виконання завдань з наступним їх уточненням;
10. має потужні вбудовані засоби аналізу поточного стану виконання завдань та тенденцій розвитку проекту;
11. автоматизує підготовку звітів на будь-який поточний момент та на перспективу;
12. створення шаблонів проектів;
13. можливість створення макросів та VBA програм.

Кожен проект створюється для досягнення певних результатів за певний період та за певні кошти. Проект буде вміщувати:

1. потрібні параметри, що дозволяють визначити якість та масштаб проектів у вигляді послідовності завдань;
2. час на виконання кожного завдання та проекту в цілому;
3. ресурси, що потрібні для реалізації кожного завдання та проекту в цілому.

Під завданням будемо розуміти будь-які дії або події, що потрібні для реалізації проектних робіт. деталізація завдань визначається складністю продукту, що отримується.

Знайомство з MS Project.

Запуск проекту

При першому запуску Project 2010, ви можете бути здивовані тим, що ви бачите. Меню і панелі інструментів замінені стрічкою, яка допоможе вам швидко знаходити команди, необхідні для виконання завдання. Команди організовано в логічні групи, зібрані разом під вкладками.

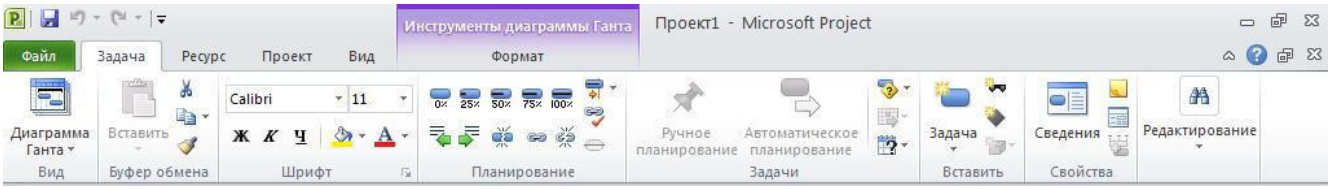


Рис. 7.1.

Для проекту 2010 року, всі вкладки та групи на стрічці, що повністю налаштовується. Якщо ваша організація має унікальний у своєму бізнесі, ви можете згрупувати їх за власною вкладці стрічки.

Відкрийте вкладку Файл і ви перебуваєте в Backstage, єдиний графічний призначення для управління файлами проекту. Backstage містить ті ж основні команди, доступні в меню Файл у попередніх версіях Microsoft Project для відкриття, збереження, друку і файлів проекту. Project Professional 2010 користувачі можуть використовувати Backstage для управління з'єднаннями Project Server, а також перевірити й опублікувати проекти.

Опції команди, які були в меню Сервіс була переміщена в Backstage. Ця команда відкриває діалогове вікно Project Options, де ви можете ввести, огляд, або змінити настройки управління тим, як Microsoft Project працює і з'являється.

Найбільш часто використовувані команди тепер можна знайти з одного кліка - одна клацніть правою кнопкою миші, то є. При клацанні правою кнопкою миші на будь-який пункт у поданні, таких як бар, осередки таблиці або діаграми, міні-панель зі списком найбільш часто використовуваних команд на дисплеї. Коли ви перебуваєте в поспіху, це один із способів використання проектів, які будуть платити Вам в економії часу.

Команда планування

Project Professional 2010 користувачів тепер є команда планування, з метою планування ресурсів, який дозволяє вам працювати з вашим графіком таким чином, щоб не було можливо раніше в більш ранніх версіях проекту. З

метою Планувальник команди ви можете відразу побачити, що ваші члени команди працюють над завданнями і перейти від однієї людини до іншої. Ви також можете переглянути та призначити нерозподілені роботи, представлення overallocations, і подивитися, назви завдань, а також імена ресурсів - все це в один ефективний вид. Управління завдань і ресурсів, ніколи не було так просто. Наприклад, якщо ресурс перевищення, все що вам потрібно зробити, це перетягнути завдання від одного ресурсу до іншого, і overallocation зникає.

Планування вручну

У програму Project 2010 внесено суттєві зміни щодо способу планування проектів. Якщо завдання заплановане вручну, то його дати не змінюватимуться автоматично під впливом змін таких факторів, як залежності завдань і календар проекту.

Заплановане вручну завдання можна помістити в будь-яке місце на плані, і програма Project не переміщатиме його.

Керівники проектів, які звикли до автоматичного планування в попередніх версіях Project, можуть вимкнути нову функцію планування вручну для певних завдань або всього проекту. Для деяких (особливо складних) проектів може знадобитися потужний засіб планування програми Project, що візьме на себе функцію планування.

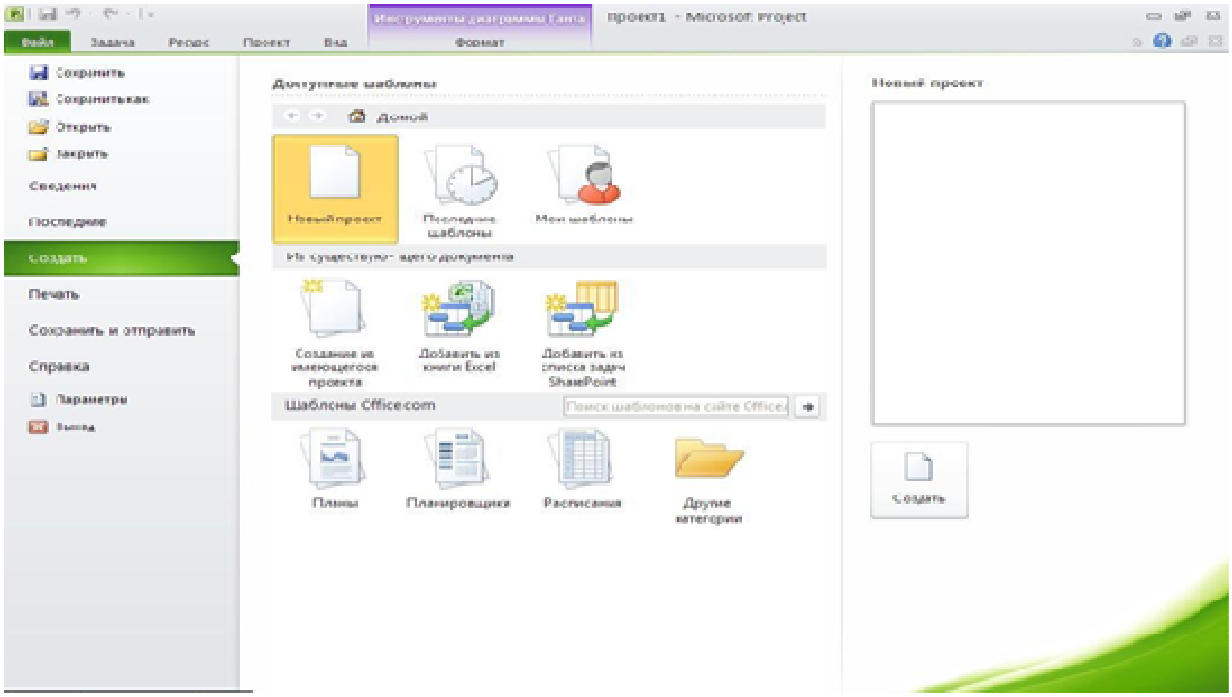


Рис. 7.2.

Організація проектних робіт за допомогою програмних засобів

Перед початком роботи над проектом необхідно розбити проект його задачі, описати їх зв'язки, оцінити трудомісткість задач і описати ресурси, необхідні для реалізації проекту. Це є вихідною інформацією для роботи Microsoft Project, і, як правило, цю роботу виконує менеджер. На основі цієї інформації система автоматично складає докладний календарний план ходу виконання робіт, визначає критичні шляхи, виконує розрахунки бюджетних витрат, надає членам команди всю необхідну інформацію і відображає її в зручному для аналізу вигляді діаграми Гантта

За умовчанням для нового проекту встановлюється представлення у вигляді Діаграми Гантта (Gantt Chart). * **Генрі Гантт** (1861-1919) – учень основоположника теорії управління Ф.У. Тейлора, перший розробник графічної інтерпретації тактичного плану реалізації проекту.

Критичний шлях - шлях у сітковій моделі, тривалість якого дорівнює критичній. Роботи, що лежать на критичному шляху, називаються критичними.

Метод критичного шляху є основним для розрахунку ранніх та пізніх початків та закінчень робіт та резервів часу. Календарний план, як перелік тільки планових параметрів проектних робіт, втрачає свій сенс без порівняння з фактичними термінами виконання, тому частіше говорять про календарний графік. Він відбиває планові та фактичні дані про початок, кінець і тривалість кожного робочого елемента.

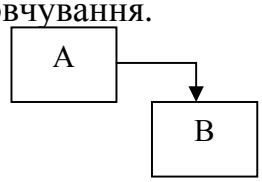
Визначення зв’язків між задачами на діаграмі Ганта

Зв’язки між двома завданнями можуть бути визначенні різним чином, наприклад, час початку або завершення однієї задачі буде впливати на час початку або завершення іншої. Задача, що впливає на іншу отримала назву попередньої (Predecessor) і інша задача, яка залежить від неї отримала назву послідовної (Successor). При встановленні зв’язків визначаються правила, що один зв’язок може об’єднувати лише дві задачі, а в однієї задачі може бути декілька зв’язків з іншими.

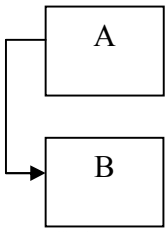
Застосування зв’язків між задачами дозволяє визначити всі основні дати проекту (наприклад, початок і закінчення), а також дозволяє здійснювати автоматичне перепланування усіх дат проектів у разі зміни будь-якого часового інтервалу. Наприклад, початок, кінець або тривалість етапу.

У Microsoft Project є **чотири типи зв'язків між завданнями**:

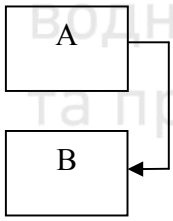
1. Finish to Short (FS), який визначає, що завдання В не може розпочатись раніше ніж завершиться завдання А. Даний зв'язок встановлюється в режимі замовчування.



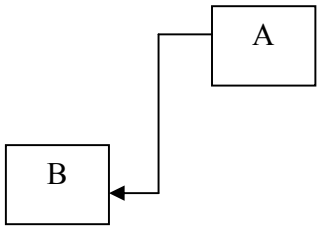
2. Start to Start (SS). Визначає залежність за якою завдання В не може розпочатися до тих пір, поки не розпочнеться А.



3. Finish to Finish (FF). визначає залежність за якою завдання В не може бути завершено до тих пір, поки не завершено завдання А.



4. Start to Finish (SF). Встановлює зв'язок коли завдання В не може завершитись до тих пір, поки не розпочнеться завдання А.



Для завдання будь-якого з цих залежностей використовується команда **Меню/Правка/Связать задачи або Edit/Link tasks**.

Основні стандартні представлення для введення, редагування і отримання інформації про завдання

Таблиця 7.1

Gantt Chart Діаграма Гантта	Список завдань і відповідні тимчасові відрізки в області діаграми. Зручно використовувати при введенні і планеруванні завдань.
Tracking Gantt Діаграма Гантта з відстежуванням	Зручно використовувати при відстежуванні ходу виконання проекту, для порівняння запланованих термінів виконання проекту і реальних в ході виконання проектних робіт.
Pert Chart Мережевий графік	Завдання представлені у вигляді блоків, сполучених стрілками в блок-схему відповідно до взаємозв'язків завдань в плані проекту. Зручно планувати проекти з великим числом зв'язків між завданнями.
Calendar Календар	Відображує інформацію у вигляді таблиці з колонками, відповідними дням тижня. Завдання на календарі позначені відрізками, розташованими у відповідних колонках (від дня початку робіт над завданням до дня закінчення).
Task Usage Використання завдань	Список робіт з призначеними завданням ресурсами.

Основні стандартні представлення для введення, редагування і отримання інформації про ресурси

Таблиця 7. 2

Resource Sheet Лист ресурсів	Список ресурсів і інформації, що відноситься до них. Зручно використовувати для введення і редагування даних про ресурси.
Resource Graph Графік ресурсів	На діаграмі відображується різна інформація про вибрані ресурси: процентне завантаження, наднормова робота для ресурсів і так далі
Resource Usage Использование ресурсів	Список ресурсів і даних про призначення, вартість, перевантаження ресурсу.

Перед створенням нового проекту необхідно набудувати робочу середу проекту. Більшість налаштувань встановлюються в діалоговому вікні **Сервіс - Параметри** (Tools-Options).

Вибрати в меню команду **Сервіс - Параметри** (Tools-Options), перейти на вкладку **Планування** (Schedule) і встановити наступні опції:

- **Показувати одиниці призначень у вигляді: відсотків** (Show assignment units as: Percentage)
- **Нові роботи зачинаються: В день початку проекту** (New tasks start on: Project start date)
- **Тривалість вводиться: у днях** (* можна задати тривалість в інших вимірах – хвилинах, годиннику, тижнях) (Duration is entered: in Days)
- **Трудовитрати вводяться в годиннику** (* можна задати тривалість в інших вимірах) (Work is entered: in Hours)
- **Тип завдань за умовчанням: Фіксована тривалість** (* можна задати іншого типа завдань – Фіксований об'єм ресурсів або Фіксовані трудовитрати) (Default task type: Fixed duration);
- Вибрати в меню команду **Сервіс-параметри** (Tools-Options), перейти на вкладку **Планування** (Schedule), зняти прапорець біля опції **Нові завдання мають фіксований об'єм робіт** (New tasks are effort driven) – інакше призначення нового ресурсу завданню приведе до зменшення часу на її виконання.

У MS Project існує три типи базових календарів:

Standart – відображає загальноприйнятий розклад робочого часу – з понеділка по п'ятницю, з 8:00 до 17:00, обідня перерва з 12:00 до 13:00.

24 Hours – безперервний робочий час з понеділка по до воскресіння, з 9 до 24 годин. Цей календар зазвичай використовується для устаткування або для ресурсів, що працюють безперервно (наприклад, служба охорони).

Night Shift – задає робочий час з ночі понеділка до ранку суботи, з 23:00 до 8:00, перерва з 3:00 до 4:00.

Приклад розробки плану створення ПП

Даний проект полягає у виконанні взаємопов’язаних робіт, які повинні бути враховані при побудові сіткового графіка. Проект, що розглядається маємо поділити відповідно на виконання таких робіт:

1. Вимого до проекту;
2. розробка ТЗ проекту, заключення договору;
3. Розробка блок-схем модулів (низький рівень)
4. навчання персоналу.

Етапи робіт описанні в табл. 7.3

Таблиця7.3

Етапи робіт

1. вимоги до проекту	1д
2. розробка ТЗ проекту та заключення договору	6д
2.1 підготовка матеріалів, апаратні, технічні засоби	2д
2.2 створення функціональної специфікації (високий рівень)	2д
2.3 розробка інтерфейсу програми	2д
3. Розробка блок-схем модулів (низький рівень)	5д
3.1 Кодування	3д
3.2 Тестування програми	2д
4. Навчання персоналу	4д
4.1 Монтаж апаратного забезпечення	6д
4.2 Програмне встановлення	4д
4.3 Супровід ПП	1д

Для задач сіткового графіка необхідно задати параметри часу виконання робіт, що ми й зробили в табл. 7.4.

Таблиця 7.4

Термін виконання робіт на всіх етапах.

1. вимоги до проекту	**3.03.12**
2.розробка ТЗ проекту, заключення договору	4.03.12
2.1 підготовка матеріалів, апаратні, технічні засоби	10.03.12
2.2 створення функціональної специфікації (високий рівень)	12.03.12
2.3 розробка інтерфейсу	14.03.12
3. Розробка блок-схем модулів (низький рівень)	**16.03.12**
3.1 Кодування	21.03.12

3.2 Тестування програми	28.03.12
4. навчання персоналу	**01.04.12**
4.1 Монтаж апаратного забезпечення	07.04.12
4.2 Програмне встановлення	11.04.12
4.3 Супровід ПП	12.04.12

Для кожної задачі можуть бути визначені ресурси. Ресурси поділяються на дві категорії: трудові і матеріальні ресурси.

До трудових ресурсів відносяться співробітники і обладнання.

До матеріальних ресурсів відносяться елементи постачання та інші розхідні матеріали.

Ресурси, що будуть задані в проекті подаємо у вигляді таблиці (табл. 7.5).

Таблиця 7.5

Ресурси, що будуть задані в проекті

Ресурс	**Тип**	**Макс. Доступ**	**Ставка Грн/год**	**Позауроч ний час Грн/год**	**Розхід**	**Тип календаря**
Керівник	робітник	100%	850	850	Prorate	Стандартний
Менеджер	Робітник	100%	400	425	Prorate	Стандартний
Програміст	Робітник	100%	630	650	Prorate	Стандартний
Дизайнер	Робітник	100%	300	375	Prorate	Стандартний
Комп’ютер	робота	100%	5	5	Prorate	Погодинно
Матеріали інформації	Матеріал	100%	5	5	Prorate	Погодинно

Таблиця 7.6

Попередній розподіл ресурсів

Завдання	Ресурси	К-сть одиниць
Вимоги до проекту	Керівник, Програміст	100%, 100%
Розробка ТЗ проекту, заключення договору	Керівник, Програміст, Менеджер	100%, 100%, 100%
Підготовка матеріалів, апаратні, технічні засоби	Програміст,	100%
Створення функціональної специфікації (високий рівень)	Програміст,	100%
Розробка інтерфейсу	Програміст, дизайнер	100%, 100%
Розробка блок-схем модулів (низький рівень)	Програміст	100%,
Кодування	Програміст	100%
Тестування	Програміст	100%
Навчання персоналу	Керівник	100%
Монтаж апаратного забезпечення	Програміст	100%
Програмне встановлення	Програміст	100%
Супровід ПП	Керівник, програміст	100%, 100%

Реалізація плану забезпечення життєвого циклу ПП у MS Project

Основні області вікна системи MS Project 2007 показані на Рис 7.3.

Діаграма Гантта (Gantt Chart, Рис 7.4) виводиться за умовчанням і інформація про завдання представляється одночасно в текстовому і графічному вигляді. У цьому режимі можна додавати нові завдання, встановлювати зв'язки між завданнями і виділяти ресурси для завдань.

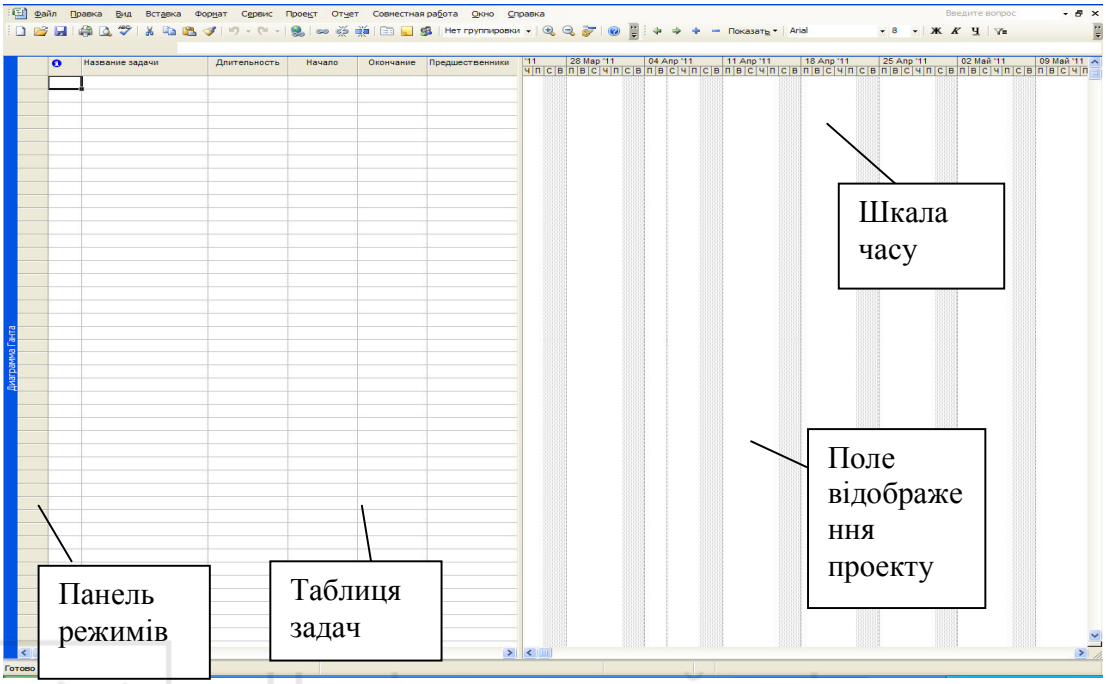


Рис.7.3. Основні області вікна системи MS Project

Таблиця 7.7

Основні піктограми меню

	Зв'язати завдання, видалити зв'язок між завданнями, розбиття завдань
	Інформація до завдання, замітки до завдання, призначення ресурсів
	Розміщення ресурсів, Інформація про завдання (Task Entry View), Перехід до наступного перевантаження ресурсів, призначення ресурсів, розділення ресурсів, змінити пул ресурсів, відновити пул ресурсів.
	Збільшити або зменшити масштаб зображення проекту, перейти до вибраного завдання
	Підвищити рівень завдання, Знизити рівень завдання, показати підзадачі, прибрати підзадачі, показати всі підзадачі або завдання різних рівнів
	Показати всі завдання або по певному критерію, вибір завдань по автофільтру

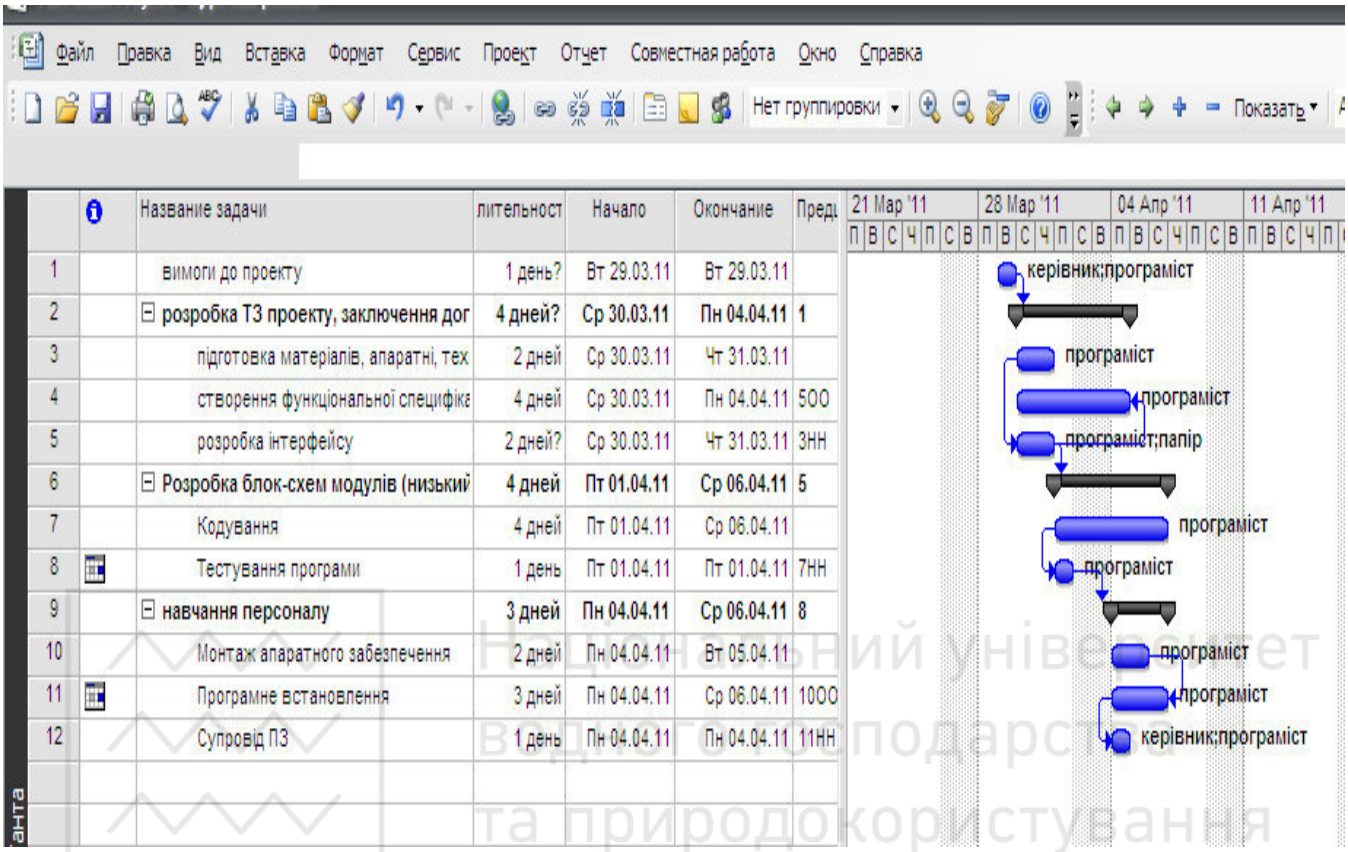


Рис.7.4. Діаграма Гантта

Мережева діаграма (Network Diagram, *Рис 7.5*) показує завдання і їх взаємозв'язки у вигляді блок схеми. Критичні роботи виділяються червоним кольором.

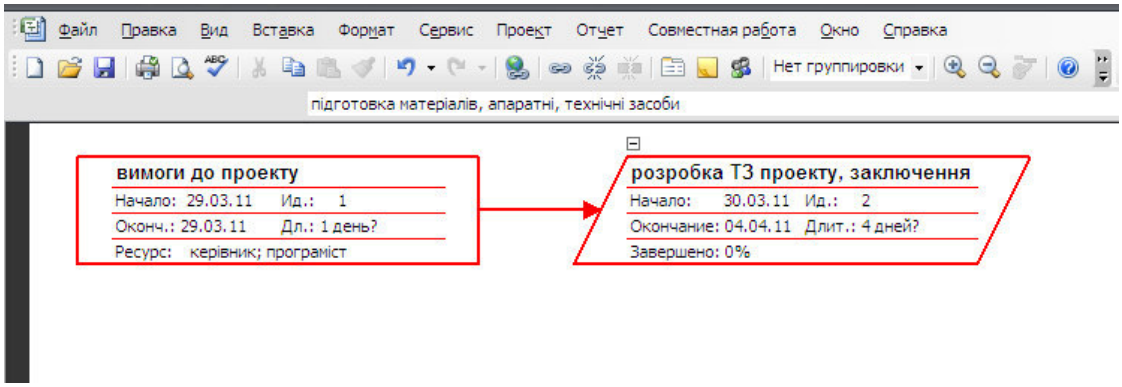


Рис.7.5. Мережева діаграма

У режимі Використання завдань (Task Usage, Рис 7.6) можна визначити об'єм роботи, виконаний кожним ресурсом і порівняти фактично виконані об'єми з плановими.

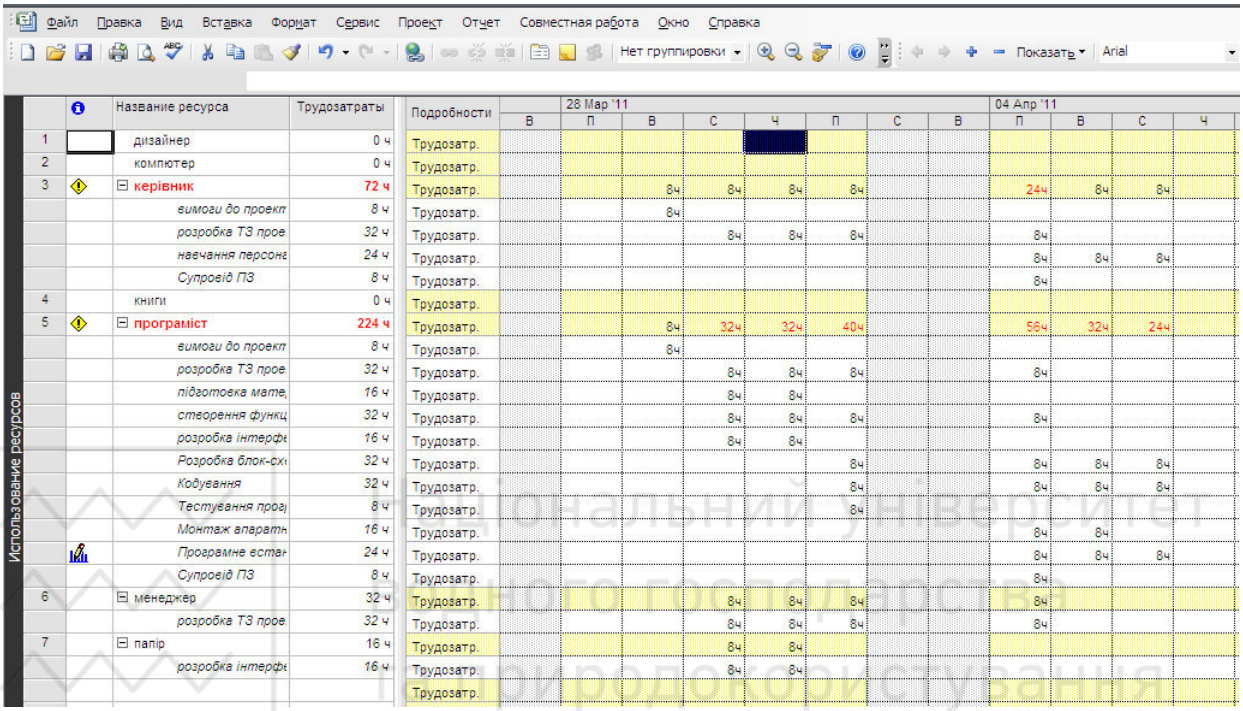


Рис.7.6. Режим Використання завдань

Модифікована діаграма Ганта (Tracking Gantt, Рис 7.7) дозволяє додатково порівняти планові дати почала і закінчення робіт із значеннями контрольного плану, а також обчислити відсоток завершення роботи. Використовується при реалізації проекту.

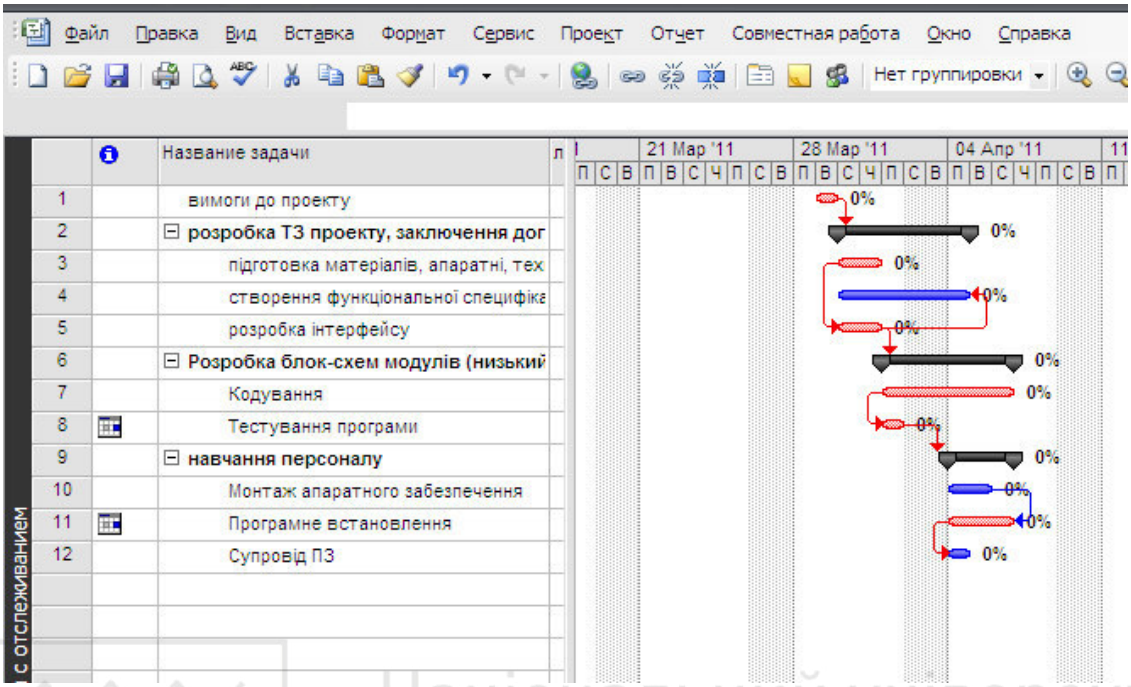


Рис.7.7. Модифікована діаграма Ганта

Графік завантаження ресурсів (Resource Graph, Рис 7.8) показує розподіл ресурсів, робіт і вартості ресурсів.

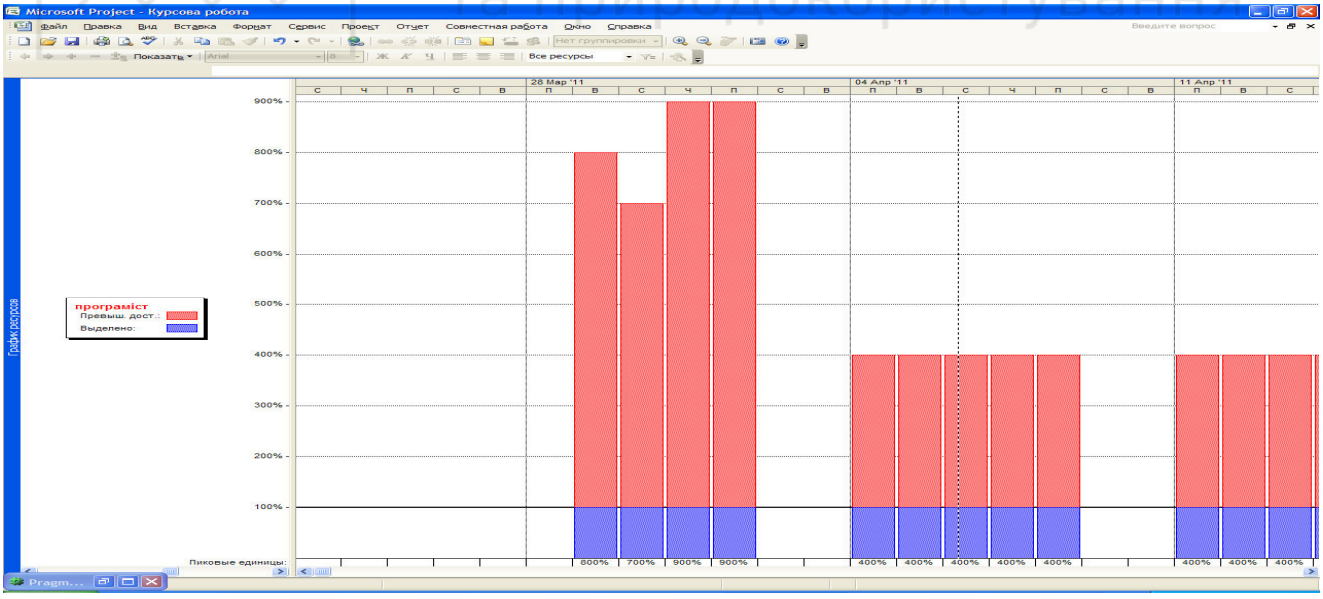


Рис.7.8. Графік завантаження ресурсів

Режим Таблиця ресурсів (Resource Sheet, Рис 7.9) показує загальну інформацію про ресурси у вигляді таблиці.

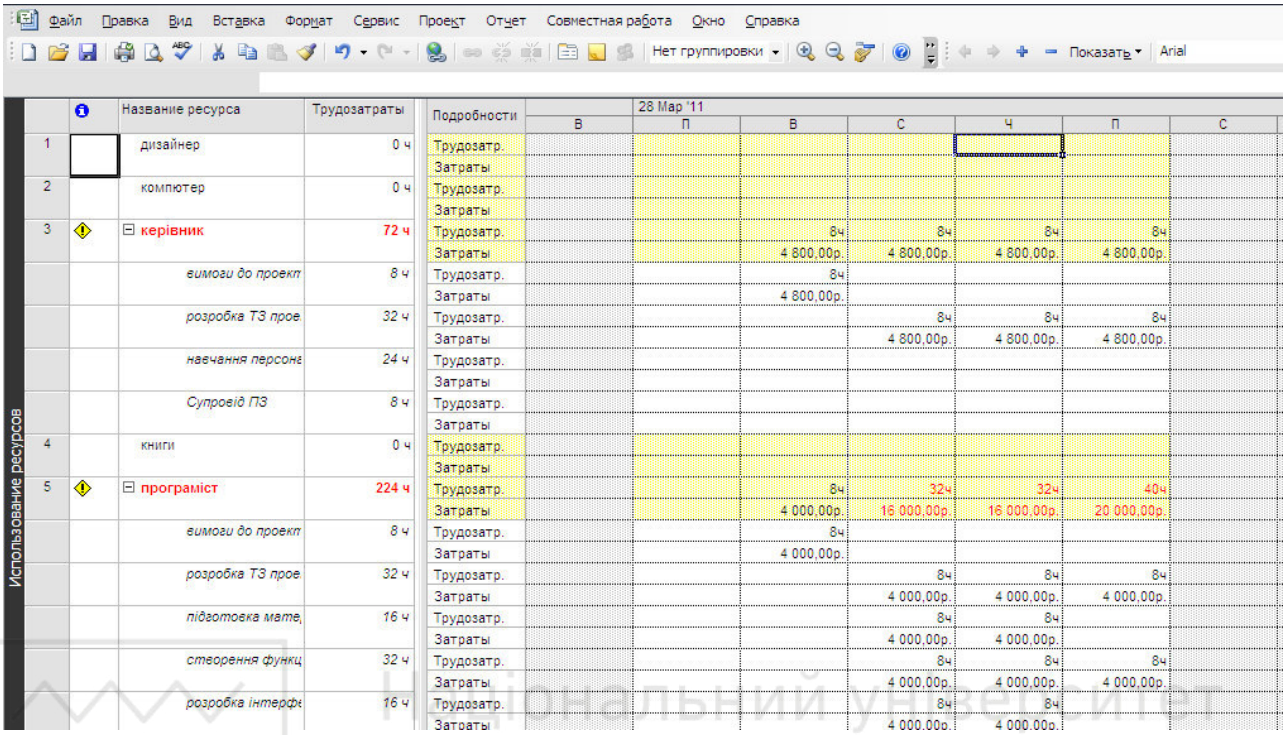


Рис.7.9. Режим Таблиця ресурсів

У розглянутих уявленнях завдання виконуються паралельно, оскільки не задані зв'язки між завданнями. Використовуватимемо спосіб зв'язку між завданнями «Фініш - Старт», який встановлений за умовчанням.
Можна зв'язати два завдання або цілу послідовність. Для установки зв'язку двох завдань, виділити їх і використовувати кнопку «Зв'язати завдання» (Link Task, Рис 7.10). Зв'язок встановлений - роботи виконуються послідовно, що показане на Gantt Chart (Рис7.11).

Для зв'язку декількох завдань, їх необхідно виділити. Якщо виділяти завдання, протягаючи мишею по заголовках, то завдання з меншим номером буде оголошено попередником, а завдання з великим номером - наступником. Незв'язані завдання виділяються за допомогою клавіші Ctl і послідовність їх виділення визначає їх логічну послідовність.

Для видалення зв'язку виділених завдань використовується кнопка «Видалити зв'язки» (Unlink Task).

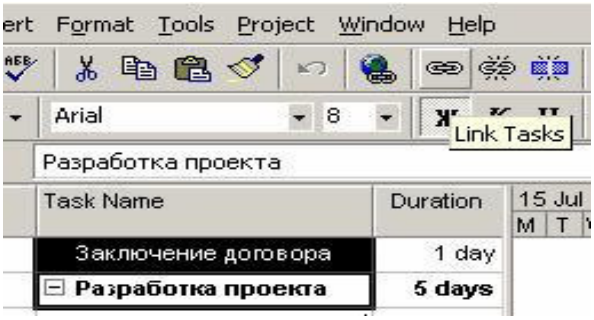


Рис.7.10. Встановлення зв'язку між двома завданнями

Рис.7.11. Результат встановлення зв'язку між двома завданнями

Для завдання інших типів зв'язків між завданнями, перетягнете розділову лінію між списком робіт і графіком управо, до появи стовпця Predecessors (Попередники). Для зв'язку указується номер задачі-попередника і скорочена назва типу зв'язку (FS - фініш - старт, SS- старт - старт, SF - старт - фініш, FF - фініш-фініш).

Всі зв'язані роботи проекту показані на Рис 7.12

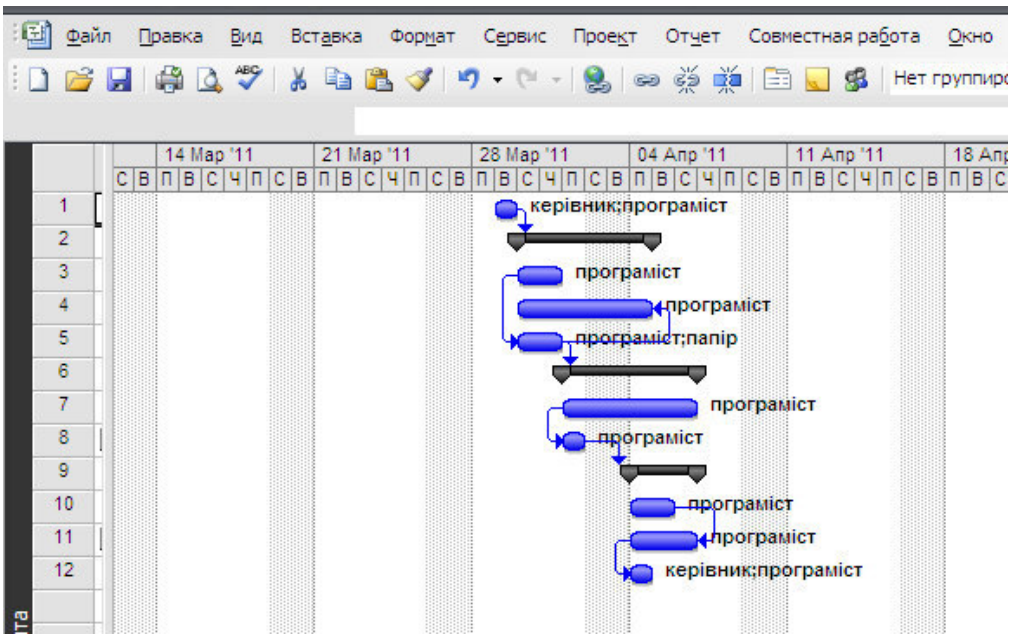


Рис.7.12. Зв'язані роботи проекту показані

Заздалегідь, до призначення ресурсів роботам, задамо загальний список, використовуваних в проекті ресурсів. У прикладі проекту використовуються об'єднані імена ресурсів - менеджер, архітектор, різноробочий і ін. Передбачається, що в організації, що виконує будівельні проекти, може бути декілька менеджерів, архітекторів або різноробочих.

Рис.7.13

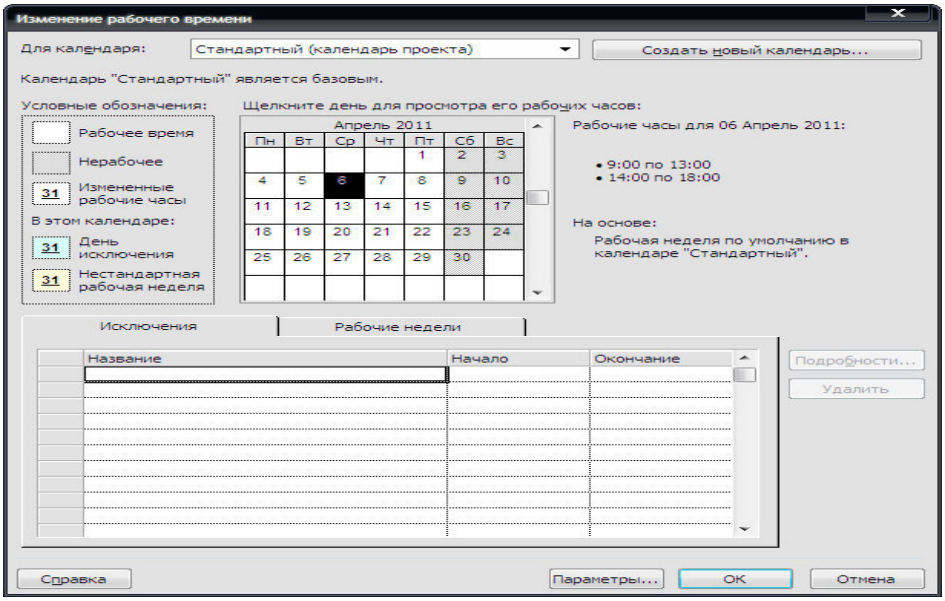


Рис.7.14

Можна змінити робочі дні і робочий годинник у вибраному дні.

На основі загального списку ресурсів можна задати їх використання в роботах проекту (проводиться розподіл ресурсів по роботах). При цьому розподілі встановлюється і тип планованого завдання (типи завдань планування були розглянуті вище). Використовуватимемо в проекті наступні типи завдань планування: фіксована кількість одиниць, планування по ресурсах (використовується в Project за умовчанням), фіксована тривалість і фіксований об'єм роботи, планування по ресурсах.

Для виклику вікна призначення ресурсів (Assign Resources): в Главном меню Tools à Resources à Assign Resources (Рис. 7.15)

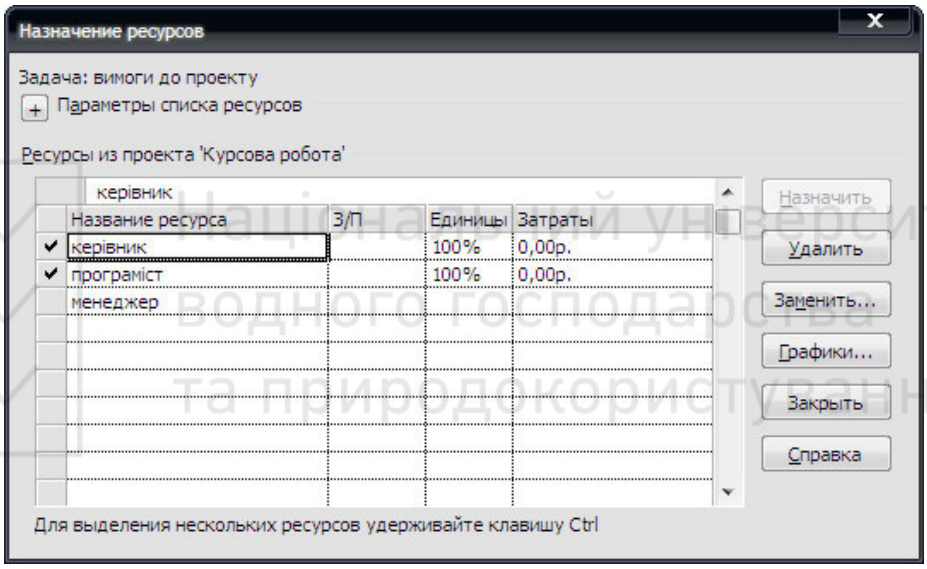


Рис.7.15. Вікно призначення ресурсів

При активізації осередку рядка завдання (роботи) і счелчке правої миші з'являється контекстне меню, в якому можна вибрати пункт «Task information.”. З'являється вікно з якнайповнішою інформацією про завдання (Рис. 7.16). Це вікно дозволяє також змінювати задану інформацію.

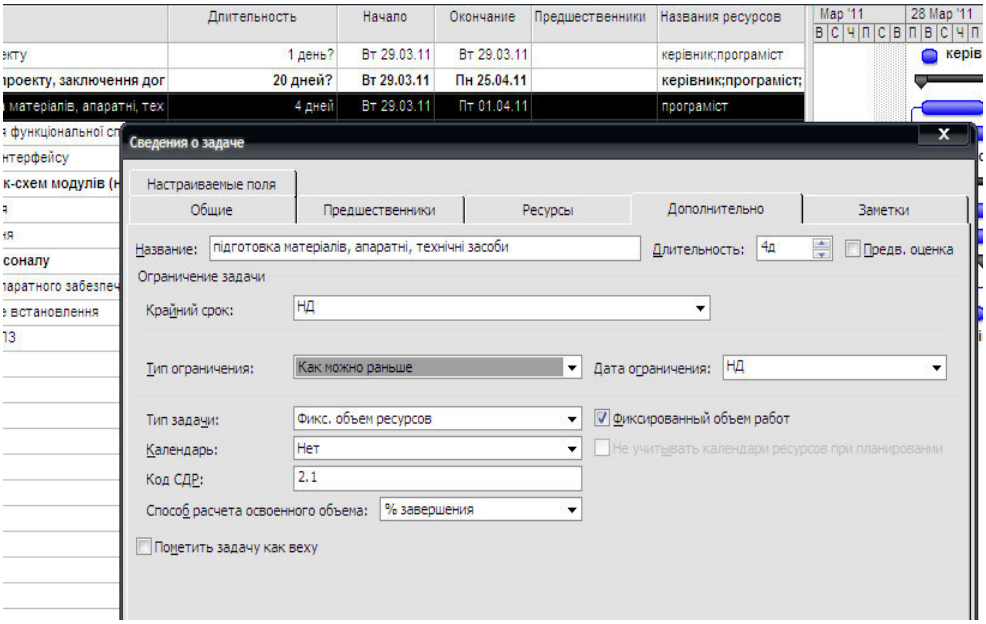
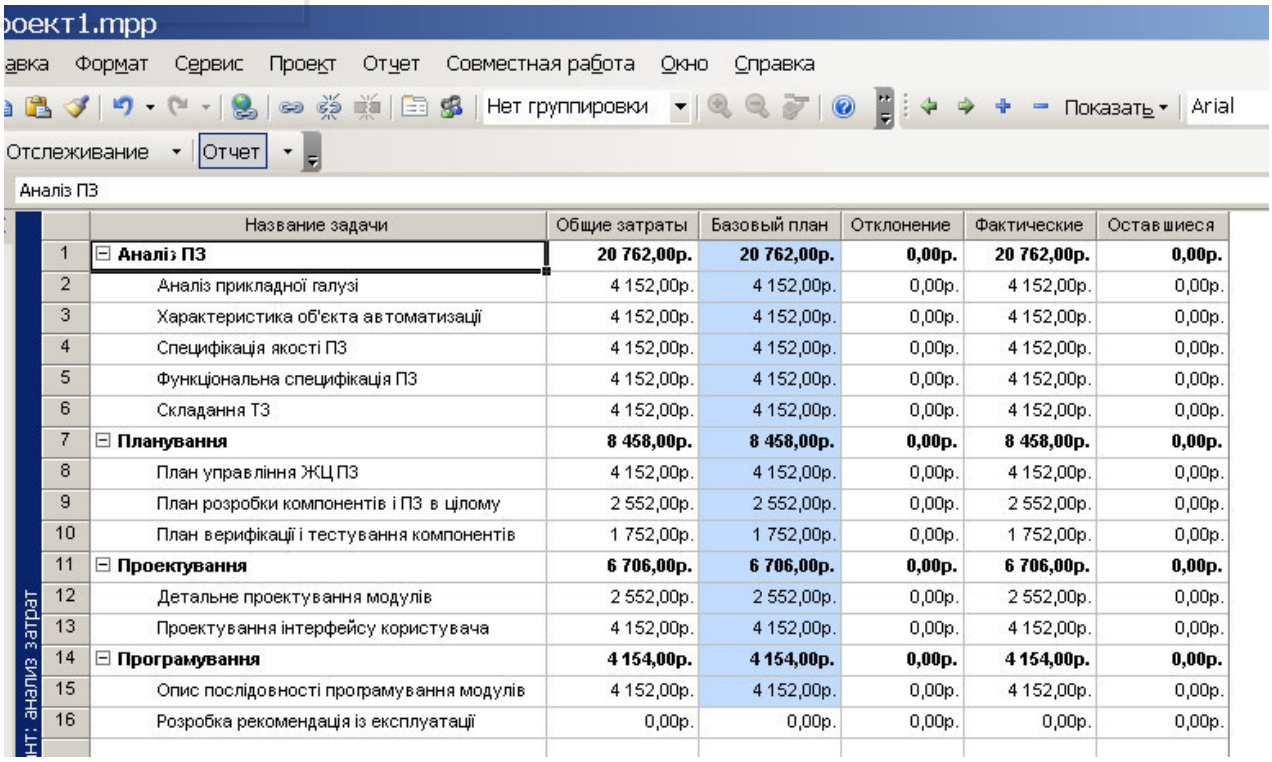


Рис.7.16. Вікно з інформацією про завдання

Визначаємо бюджет проекту. Для цього активізуємо команду Звіт та обираємо Перегляд витрат проекту. На рисунку 7.17 подано вигляд Звіту про бюджет проекту та відхилення його в процесі виконання.



	Название задачи	Общие затраты	Базовый план	Отклонение	Фактические	Оставшиеся
1	**Аналіз ПЗ**	**20 762,00р.**	**20 762,00р.**	**0,00р.**	**20 762,00р.**	**0,00р.**
2	Аналіз прикладної галузі	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
3	Характеристика об'єкта автоматизації	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
4	Специфікація якості ПЗ	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
5	Функціональна специфікація ПЗ	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
6	Складання ТЗ	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
7	**Планування**	**8 458,00р.**	**8 458,00р.**	**0,00р.**	**8 458,00р.**	**0,00р.**
8	План управління ЖЦ ПЗ	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
9	План розробки компонентів і ПЗ в цілому	2 552,00р.	2 552,00р.	0,00р.	2 552,00р.	0,00р.
10	План верифікації і тестування компонентів	1 752,00р.	1 752,00р.	0,00р.	1 752,00р.	0,00р.
11	**Проектування**	**6 706,00р.**	**6 706,00р.**	**0,00р.**	**6 706,00р.**	**0,00р.**
12	Детальне проектування модулів	2 552,00р.	2 552,00р.	0,00р.	2 552,00р.	0,00р.
13	Проектування інтерфейсу користувача	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
14	**Програмування**	**4 154,00р.**	**4 154,00р.**	**0,00р.**	**4 154,00р.**	**0,00р.**
15	Опис послідовності програмування модулів	4 152,00р.	4 152,00р.	0,00р.	4 152,00р.	0,00р.
16	Розробка рекомендація із експлуатації	0,00р.	0,00р.	0,00р.	0,00р.	0,00р.

Рис.7.17. Звіт про бюджет проекту

Використана література:

1. Гецци К. Основы инженерии программного обеспечения / К. Гецци, М. Джазайери, Д. Мандриоли. – СПб: БХВ-Петербург, 2005. – 832 с.
2. Андон Ф.И. Основы инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль, Т.М. Коротун, В.Ю. Суслов. – К. : Академпериодика, 2002. – 504 с.
3. Соммервилл Иан Инженерия программного обеспечения / Иан Соммервилл. – М. : Издательский дом «Вильямс», 2002. – 624с.
4. Бабенко Л.П. Основи програмної інженерії / Л.П. Бабенко, К.М. Лавріщева. – К. : Т-во «Знання», КОО, 2001. – 269 с.
5. Матвєєва Л.Є. Процес розробки програмного забезпечення. Від теорії до практики / Л.Є. Матвєєва, В.А. Волков. – К. : ТОВ «Інформаційні програмні системи», 2008. – 117 с.
6. Боэм Б.У. Инженерное проектирование программного обеспечения / Б.У. Боэм. — М. : Радио и связь, 1985. – 511с.
7. Липаев В.В. Выбор и оценивание характеристик качества программных средств. Методы и стандарты / В.В. Липаев. – М. : СИНТЕГ, 2001. – 228 с.
8. Липаев, В.В. Программная инженерия. Методологические основы : Учеб. пособие / В. В. Липаев ; Гос. ун-т – Высшая школа экономики. – М. : ТЕИС, 2006. – 608 с.
9. Рудаков А.В. Технология разработки программных продуктов : учебное пособие / А.В. Рудаков. – М. : Издательский центр «Академия», 2006. – 208 с.
10. Канер С. Тестирование программного обеспечения : Пер. с англ. / С. Канер, Д. Фолк, Кек Нгуен Е. – Киев : ДиаСофт, 2000. – 544 с.
11. Орлов С. А. Технологии разработки программного обеспечения: Учебник / С. А. Орлов. – СПб.: Питер, 2002. – 464 с.
12. Git-scm [Електронний ресурс]. – Режим доступу: http://git-scm.com/book/ru/v1. – Назва з екрану.
13. Scott Chacon. Pro Git [Електронний ресурс] / Scott Chacon. – 2012. – Режим доступу: http://www.sis.uta.fi/~csolsp/shared/temp/progit.ru.pdf. – Назва з екрану.