МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Уманський державний педагогічний університет
імені Павла Тичини

ВИБРАНІ ПИТАННЯ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник

Укладач Ткачук Г.В.

Умань
2021

УДК 004.04(075.8)
В41

Рецензенти:

Войтович І. С., доктор педагогічних наук, професор, завідувач кафедри інформаційно-комунікаційних технологій та методики викладання інформатики Рівненського державного гуманітарного університету;

Лещук С. О., кандидат педагогічних наук, доцент, доцент кафедри інформатики та методики її навчання Тернопільського національного педагогічного університету імені Володимира Гнатюка;

Медведєва М. О., кандидат педагогічних наук, доцент, завідувач кафедри інформатики і інформаційно-комунікаційних технологій Уманського державного педагогічного університету імені Павла Тичини.

Рекомендовано до друку вченою радою
Уманського державного педагогічного університету імені Павла Тичини
(протокол № 6 від 23 листопада 2021 р.)

В 41 **Вибрані** питання інженерії програмного забезпечення : навч. посіб. / МОН України, Уманський держ. пед. ун-т імені Павла Тичини ; уклад Г. В. Ткачук. – Умань : ВПЦ «Візаві», 2021. – 191 с.

У посібнику представлено окремі аспекти інженерії програмного забезпечення та розробки програмного забезпечення від початкових стадій створення специфікації до підтримки системи після здачі в експлуатацію. Описано загальну теорію проєктування, конструювання та моделювання програмного забезпечення, моделі життєвого циклу програмного продукту, методи управління програмним проєктом, CASE-засоби та вимоги до програмного забезпечення тощо.

Посібник розрахований на студентів, які навчаються за спеціальністю «014 Середня освіта (Інформатика)», а також для технічних спеціальностей, спеціалістів ІТ-галузі, учителів інформатики, викладачів закладів вищої освіти.

УДК 004.04 (075.8)

© Ткачук Г. В., уклад., 2021

ЗМІСТ

ВСТУП 4

РОЗДІЛ І. ТЕОРЕТИЧНІ АСПЕКТИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 7

Тема 1.1. Вступ до інженерії програмного забезпечення 7

Тема 1.2. Стандарти в галузі проєктування програмного забезпечення 17

Тема 1.3. Життєвий цикл програмного забезпечення 37

Тема 1.4. Моделі життєвого циклу програмного забезпечення .. 49

Тема 1.5. CASE-засоби та вимоги до програмного забезпечення..... 67

Тема 1.6. Управління програмним проєктом 74

РОЗДІЛ ІІ. ПРАКТИЧНІ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 97

Тема 2.1. Розробка технічного завдання на виконання проєкту..... 97

Тема 2.2. Введення до моделювання в середовищі ArgoUML 105

Тема 2.3. Розробка діаграм варіантів використання (прецедентів) 112

Тема 2.4. Розробка діаграм послідовності 121

Тема 2.5. Розробка кооперативної діаграми.....131

Тема 2.6. Проєктування діаграм зміни станів об'єктів.....137

Тема 2.7. Створення діаграм UML в онлайновому середовищі Gliffy.....145

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 149

ДОДАТКИ 153

ВСТУП

Нині технології та засоби розробки програмних продуктів вважаються пріоритетним напрямом наукових досліджень і науково-технічних розробок. Тенденції та швидкість поширення цих технологій є свідченням нагальної суспільної потреби підготовки компетентних фахівців з галузі інженерії програмного забезпечення на основі кращих світових стандартів та передового зарубіжного досвіду. Такий фахівець повинен володіти компетентностями в галузі проєктування, апробації, упровадження та комерціалізації інноваційних технологій інженерії програмного забезпечення. В даному посібнику відібрано основні питання інженерії програмного забезпечення, які потрібні сучасному фахівцю для формування згаданих компетентностей.

Зокрема, у розділі І «Теоретичні аспекти інженерії програмного забезпечення» висвітлено питання поняття та специфіки програмної інженерії, її становлення та розвитку, виділені головні умови успішної інженерії програмного забезпечення. Описана класифікація ПЗ дає уявлення про місце інженерії ПЗ в системі програмних засобів обчислювальної техніки. Характеристика стандартів в галузі проєктування ПЗ та базового стандарту розроблення ПЗ дають змогу зрозуміти основні концепції інженерії ПЗ, мінімальні вимоги з точки зору безпеки, надійності, ефективності ПЗ. Аналіз моделей ЖЦ ПЗ дає уявлення про різні способи та методи розробки ПЗ. Важливим також є розгляд питань, що стосуються управління проєктом та роботі в команді, визначення ролей та компетентностей кожного з учасників виконання проєкту.

У розділі ІІ «Практичні задачі інженерії програмного забезпечення» окрім теоретичних відомостей, подано завдання для відпрацьовування навичок розробки проєктів. Завдання мають на меті сформувати такі вміння:

- брати участь у проєктування програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування;
- застосовувати специфікації, стандарти, правила і рекомендації в професійній галузі при реалізації процесів життєвого циклу;

– документування інформації про проєкт, обґрунтування експлуатаційного та функціонального призначення;
– ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення;
– створювати інформаційні моделі даних та інформаційні сутності проєкту;
– розробки діаграм варіантів використання (прецедентів), діаграм послідовності, розробки кооперативних діаграм, проєктування діаграм стану для класів.

Вивчення матеріалів посібника дасть змогу отримати наступні знання:

– поняття та специфіки інженерії програмного забезпечення;
– стандартів в галузі проєктування програмного забезпечення;
– кодексу професійної етики, соціальної значимості та культурних аспектів інженерії програмного забезпечення;
– поняття якості програмного забезпечення;
– поняття життєвого циклу програмного забезпечення;
– різних моделей життєвого циклу програмного забезпечення;
– CASE-засобів розробки програмного забезпечення;
– вимог до програмного забезпечення відповідно до стандартів;
– різниці між функціональними та нефункціональними вимогами;
– поняття технічного завдання та його основних розділів;
– функціонального призначення програмного забезпечення.

Посібник розрахований на студентів технічних спеціальностей, спеціалістів ІТ-галузі, учителів інформатики, а також викладачів закладів вищої освіти, які здійснюють підготовку зазначених фахівців.

СПИСОК СКОРОЧЕНЬ

Україномовні скорочення:

ДСТУ	–	державний стандарт України
ЖЦ ПЗ	–	життєвий цикл програмного забезпечення
ІС	–	інформаційна система
ІТ	–	інформаційні технології
КС	–	комп'ютерна система
ООП	–	об'єктно-орієнтоване програмування
ПЗ	–	програмне забезпечення
ПП	–	програмний продукт
ПС	–	програмна система
ТЗ	–	технічне завдання

Англомовні скорочення:

ISO	–	Міжнародна організація зі стандартизації
MSF	–	Методологія розробки програмного забезпечення від корпорації Microsoft
PMBOK	–	Довідник з управління проєктами
PMI	–	Інститут управління проєктами
SQI	–	Інститут якості програмного забезпечення

РОЗДІЛ І. ТЕОРЕТИЧНІ АСПЕКТИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тема 1.1. Вступ до інженерії програмного забезпечення

План

1. Програмна інженерія: поняття та специфіка.
2. Становлення і розвиток інженерії ПЗ.
3. Головні умови успішної інженерії ПЗ.
4. Класифікація ПЗ.

1. Програмна інженерія: поняття та специфіка

Сучасне виробництво ПЗ – це найбільша галузь світової економіки, в якій задіяно мільйони фахівців. У даний час практично немає жодної сфери людської діяльності, де б не застосовувалось ПЗ. При цьому багато компаній, що працюють у різних галузях матеріального виробництва й «традиційного» бізнесу, починають усвідомлювати, що забезпечити більш високі темпи зростання, домогтися конкурентної переваги вони можуть як за рахунок апаратно-орієнтованих, так і за рахунок програмно-орієнтованих рішень, і все частіше віддають перевагу останнім.

Звичайно ж, професійних розробників ПЗ не може не тішити той факт, що світова економіка стає все більш залежна від ПЗ. Завдяки розвитку комп'ютерної техніки й технологій розроблення ПЗ стало можливим створення програмних систем, які постійно ускладнюються і знаходять все більше поширення. Крім того, зростає і суспільний попит на ці системи.

Інженерія ПЗ (програмна інженерія) – це галузь інформаційних технологій, завданням якої є створення методів розроблення якісного ПЗ в умовах обмежених часових та фінансових ресурсів. Під інженерією ПЗ також розуміють інженерну дисципліну, яка стосується всіх аспектів виготовлення ПЗ.

У західній літературі часто використовуються терміни: системна інженерія (від англ. «system engineering»), програмна інженерія (віл англ. «software engineering») і комп'ютерна наука (від англ. «computer science»). Проаналізуємо ці поняття і виявимо різницю між ними.

Системна інженерія пов'язана з питаннями розробки систем за допомогою комп'ютерів (архітектура, дизайн, інтеграція ПЗ).

Програмна інженерія – частина системної інженерії, що має справу з розробкою ПЗ.

Комп'ютерна наука систематизує знання основ розробки ПЗ, тобто надає переважно теоретичну основу. На практиці його недостатньо для вирішення таких проблем як:

1. Пошук фінансування.
2. Робота із замовником.
3. Підбір персоналу.
4. Етичні питання. Мікроклімат в колективі. Команда.
5. Забезпечення якості програмного продукту.

Ці питання вирішує програмна інженерія і інженери ПЗ.

Також розглянемо різницю між поняттями «інженерія ПЗ» і «програмування», які іноді помилково вважають синонімами.

Інженерія ПЗ – це насамперед колективна діяльність, тоді як програмування може бути діяльністю однієї особи. Програмування – це один із аспектів розробки ПЗ. Інженер-програміст працює над компонентами разом з іншими інженерами для побудови системи. Програмісти пишуть код відповідно до специфікацій, наданих їм інженерами ПЗ. Вони є фахівцями з основних мов програмування, тоді як інженер ПЗ виготовляє довгі конструкторські документи за допомогою засобів автоматизованої програмної інженерії, перетворює проєктну документацію в специфікацію, яка використовується для проєктування коду. Хоча вони зазвичай не беруть участь на ранніх етапах проєктування, вони можуть брати участь у тестуванні, модифікації, оновленні та редагуванні коду.

Однією відмітною рисою програмної інженерії є паперовий слід, який вона виробляє. Проєкти підписуються менеджерами та технічними фахівцями, а роль забезпечення якості полягає у перевірці проєктної документації. Багато інженерів ПЗ визнають, що їх роботою є 70% оформлення документів та 30% коду. Це дорогий, але відповідальний спосіб написання ПЗ, саме тому електронні системи в таких сферах як авіація чи медицина є високовартісними.

Складні життєво важливі програмні системи (для авіації чи медичної сфери) потребують ретельного проєктування з боку інженерів-програмістів. Це дає змогу оцінити бюджет, набрати персонал та мінімізувати ризик невдач або критичних помилок.

Відмова ПЗ у системах для авіації чи космосу, на атомних електростанціях чи в медицині, в системах виявлення пожежі чи на американських гірках може бути фатальною і загрожувати життю людей. Тому здатність інженера-програміста передбачити проблеми та усунути їх до їх виникнення є надзвичайно важливою.

Оскільки інженерія в даному випадку стосується ПЗ, доцільно розглянути дане поняття.

Уперше термін «програмне забезпечення» (software) увів відомий статист Джон Т'юкей (John Tukey) у 1958 р. для позначення різниці апаратного забезпечення електронно-обчислювальних машин (hardware) від засобів обробки даних [35].

Б'ярне Страуструп (Bjarne Stroustrup) зазначив, що якісне ПЗ непомітне, але стає помітним тоді, коли працює із помилками [35].

2. Становлення і розвиток інженерії програмного забезпечення

Термін «програмна інженерія» (від англ. «software engineering») вперше було вжито в 1966 році [9], але ідея відокремлення інженерії ПЗ як окремої галузі запропонована в 1968 р. в Німеччині на конференції, що проводилася під егідою NATO і присвяченій так званій кризі ПЗ («software crisis») [34].

Головною тематикою конференції було проєктування, виробництво та обслуговування ПЗ. Один із головних учасників конференції П. Наур (Peter Naur) зазначав, що робота проєктувальників ПЗ схожа на роботу архітекторів та інженерів-будівельників, особливо тих, хто займається проєктуванням великих гетерогенних конструкцій, таких як міста та промислові підприємства [10, с. 13]. Термін «програмна інженерія» був навмисне обраний як провокативний: «це поняття означало, що виробництво ПЗ має базуватися на тому ж типі теоретичних засад та практичних застосувань, що й у традиційних галузях інженерії» [4, с. 1; 10, с. 13].

Походження програмної інженерії учасники конференції пов'язували із кризою ПЗ – терміном, запропонованим головою програмного комітету Ф. Л. Бауером (Friedrich Ludwig «Fritz» Bauer)). На його думку, криза полягала у неможливості застосування «кустарних» (напівінтуїтивних) методів розробки для виробництва великих масштабованих програмних систем: «Існуюче ПЗ розробляється аматорами (незалежно від того, де – в університетах, компаніях чи на виробництві) за допомогою майстерності одинаків (в університетах) або великої кількості працівників («мільйон мавп») на виробництві, є ненадійним і потребує постійного «технічного обслуговування» (причому слово «обслуговування» неправильно використовується для позначення збоїв та відмов, які очікуються від виробника із самого початку), є неохайним, непрозорим та невдосконалюваним (або принаймні занадто вартісним, щоб це зробити). І нарешті, існуюче ПЗ надходить занадто пізно і коштує дорожче, ніж очікувалося, та не виправдовує сподівань, які на нього покладалися» [1].

Криза ПЗ була викликана появою потужної обчислювальної техніки третього покоління, що дозволяла втілити в життя більш складні, не реалізовані раніше програмні застосунки.

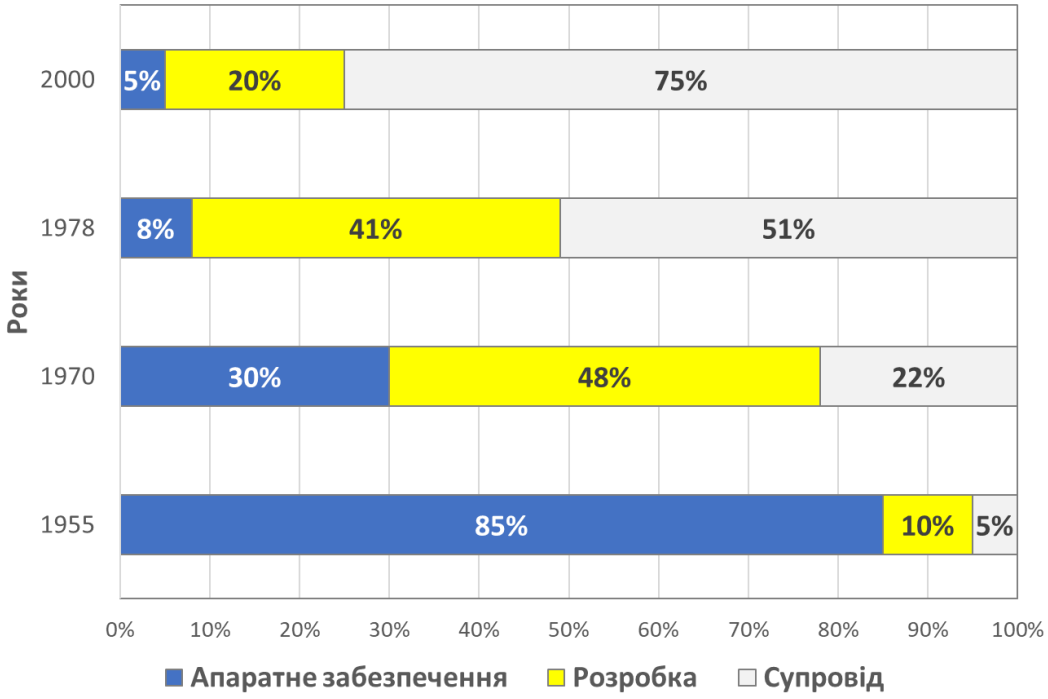


Рис. 1.1. Криза програмного забезпечення

В якості основних інструментів створення програмних продуктів почали застосовуватися алгоритмічні мови високого

рівня. Ці мови розширили можливості програмістів, але, в свою чергу, призвели до збільшення розмірів і рівня складності програмних систем, набагато перевищивши аналогічні показники при реалізації розробок на обчислювальній техніці попередніх поколінь.

Виявилося, що неформальний підхід недостатній для розробки складних систем нового покоління. На реалізацію складних програмних проєктів іноді йшли багато років. Вартість таких проєктів багаторазово зростала в порівнянні з початковими розрахунками, самі програмні системи виходили ненадійними, складними в експлуатації і супроводі. Розробка ПЗ опинилася в кризі. Вартість апаратних засобів поступово знижувалася, тоді, як вартість ПЗ стрімко зростала.

Процес становлення і розвитку програмної інженерії характеризується двома етапами: структурний та об'єктно-орієнтований підхід.



Рис. 1.2. Підходи до розроблення ПЗ

Сутність структурного підходу до розроблення ПЗ полягав у його декомпозиції на автоматизовані функції: система поділяється на функціональні підсистеми, які у свою чергу поділяються на підфункції, які поділяються на задачі і такий поділ відбувається аж до конкретних процедур. При цьому ПЗ зберігає цілісність подання, де всі складові взаємопов'язані. Таким чином, основним принципом структурного підходу був відомий вислів «поділяй та пануй».

Саме з початку 1990 років відбулась перша реальна спроба перетворити розробку ПЗ в інженерну дисципліну за допомогою концепцій CBSE (component-based software engineering – «компонентна розробка ПЗ») та COTS (commercial off-the-shelf – «готові комерційно доступні компоненти»).

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого підходу, кожен об'єкт здатний отримувати повідомлення, обробляти дані та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

3. Головні умови успішної інженерії ПЗ

Визначення інженерії ПЗ вказує на головні три умови успішного проєктування – розробка якісного ПЗ, обмежені часові та фінансові ресурси.



Рис. 1.3. Головні умови успішної інженерії

Саме тому, перед інженерами ПЗ ставляться такі основні завдання як створити якісний продукт, не вийти за межі бюджету та дотриматись терміну виконання. Розглянемо ці умови детальніше.

Поняття «якості» ПЗ

Кожний ПЗ повинен виконувати певні функції, тобто робити те, що задумано. ПЗ повинне володіти ще цілим рядом властивостей, що дозволяють успішно його використовувати протягом тривалого періоду – тобто володіти певною якістю.

Якість ПЗ *– це сукупність його рис і характеристик, які впливають на його здатність задовольняти задані потреби користувачів.*

Це не означає, що різне ПЗ повинне володіти однією і тією ж сукупністю таких властивостей в їх найвищому ступені. Цьому

перешкоджає той факт, що підвищення якості ПЗ за однією з таких властивостей часто може бути досягнуто лише ціною зміни вартості, термінів завершення розробки і зниження якості цього ПЗ за рахунок інших властивостей.

Якість ПЗ є задовільною, коли воно володіє вказаними властивостями в такому ступені, щоб гарантувати успішне його використання.

Сукупність властивостей ПЗ, яка утворює задовільну для користувача якість, залежить від умов і характеру експлуатації цього ПЗ, тобто від позиції, з якою повинна розглядатися ця якість. Тому, при описі якості ПЗ повинні бути перш за все фіксовані критерії відбору необхідних властивостей ПЗ. В даний час критеріями якості ПЗ прийнято вважати: функціональність, надійність, легкість застосування, ефективність, підтримка, мобільність.

Функціональність передбачає наявність сукупності функцій, які будуть потрібні кінцевому користувачу. Саме вони змушують користувачів використовувати та купувати розроблений продукт. Якщо ПЗ неможливо продати, то відповідно затрати на його розробку не будуть виправдані.

Надійність вказує на працездатність ПЗ та передбачає безвідмовність, стійкість, точність виконання тощо. Можливі неполадки в роботі не повинні нанести істотний, тим більше непоправний збиток.

Зручність роботи ПЗ одна із важливих властивостей, оскільки робота в комфортних умовах передбачає більшу працезданість користувача і його мотивованість. Виконання дій у ПЗ повинно бути не тільки зручним, але й природнім, простим та інтуїтивно зрозумілим.

Ефективність в розробці ПЗ має визначальне значення, оскільки прямо впливає на результат. ПЗ повинно ефективно використовувати наявні ресурси. Навряд чи клієнт погодиться повністю поміняти свою апаратну базу без належного обгрунтування. Ось у цей момент і доведеться задуматися про алгоритмічну оптимізацію, оптимізацію кодів тощо.

Підтримка передбачає супровід ПЗ розробником. Цей процес має бути максимально зручним та простим для замовника. Крім того, ПЗ повинен допускати розвиток у зв’язку із зміною потреб

користувачів та змін в ІТ-індустрії. Тому при проєктування ПЗ потрібно передбачити його майбутнє оновлення та удосконалення.

Мобільність – це здатність ПЗ бути перенесеним з однієї апаратно-програмної платформи на іншу.

Функціональність і надійність є обов'язковими критеріями якості ПЗ, причому забезпечення надійності буде червоною ниткою проходити по всіх етапах і процесах розробки ПЗ. Решта критеріїв використовується залежно від потреб користувачів відповідно до вимог до ПЗ – їх забезпечення обговорюватиметься у відповідних розділах підручника.

Фінансові ресурси та бюджет ПЗ

Якщо розробник не вкладається в бюджет, йому доведеться просити додаткові кошти на розробку. Але замовники не завжди йдуть на зустріч. Все залежить від того, наскільки замовник зацікавлений у розробці і наскільки високо він оцінює можливості компанії розробника. У будь-якому випадку, репутація компанії розробника може постраждати. Витрати на створення ПЗ найчастіше можуть бути поділені таким чином: 60% – розробка ПЗ та 40% – тестування ПЗ. Тому, витративши значні фінансові ресурси на розробку, не потрібно забувати про етап тестування та налагодження, адже вони також можуть забрати чимало коштів.

Крім того, витрати на розвиток іноді можуть перевищити витрати на створення, втім, багато що тут залежить від того, як виконано проєктування. Деталі залежать від специфіки галузі, вимог до ПЗ, використовуваних підходів до організації розробки.

Часові ресурси та терміни виконання замовлення

Розробка ПЗ повинна вкладатися в терміни. Запорука успіху – строге дотримання наступних принципів:

- Грамотне планування.
- Аналіз можливих ризиків і розробка способів реагування.
- Боротьба за чіткі межі проєкту.
- Мотивування співробітників.

4. Класифікація програмного забезпечення

Програмний засіб є однією із складових інформаційної системи і головним об'єктом інженерії ПЗ.

Стандарт ISO/IEC визначає ПЗ (software) як всі або частина програм, процедур, правил і пов'язаної з ними документації інформаційної обчислювальної системи.

Український стандарт визначає ПЗ (software) як програмний засіб, тобто взаємозв'язану сукупність програм, процедур, правил, документації і даних, необхідних для функціонування обчислювальної системи.

В даному визначенні документація являє собою сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програм. Програмна документація є одним з видів технічної документації.

Розрізняють: системне ПЗ, системи програмування (інструментальне ПЗ) та прикладне ПЗ.

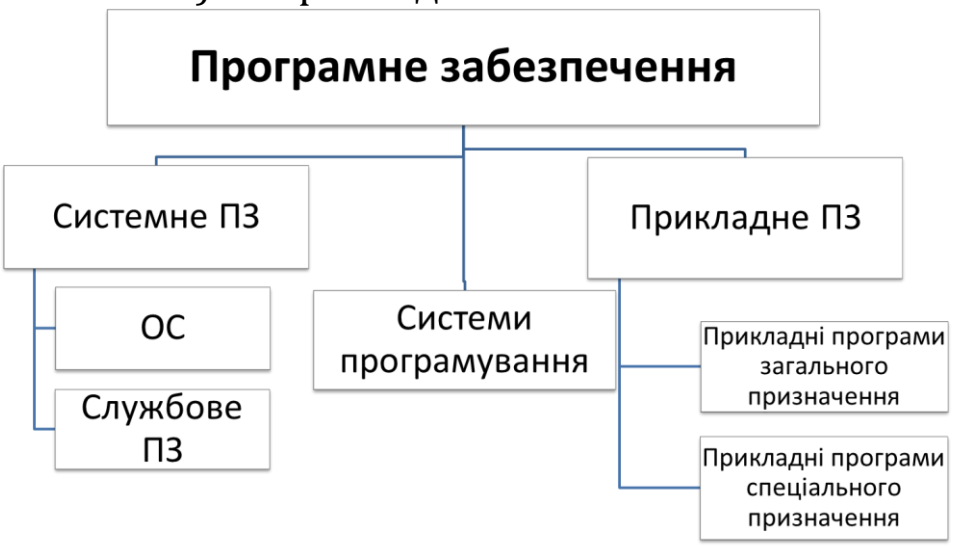


Рис. 1.4. Загальна класифікація ПЗ

До **системного ПЗ** відносять операційні системи, драйвери, файлові менеджери, службове ПЗ, утиліти тощо. Цей вид ПЗ виконує наступні функції:

– забезпечення функціонування КС;
– керування складовими КС та забезпечення їх взаємодії;
– забезпечення взаємодії елементів КС з програмами;
– розподіл ресурсів КС між програмами;
– забезпечення взаємодії користувача з КС;
– здійснення діагностики КС.

Прикладне ПЗ використовується для виконання конкретних завдань. Сюди відносять програми загального призначення для опрацювання тексту (текстові редактори та текстові процесори),

електронних таблиць, керування базами даних, створення презентацій, редагування графіки, перегляду вебсторінок (браузери) тощо. А також програми спеціального призначення: програми для математичних розрахунків, системи автоматизованого проєктування, програми для проведення бухгалтерських розрахунків, програмни для 3D комп'ютерної графіки тощо.

Системи програмування іноді називають інструментальне ПЗабо інструментальна система. Це комплекс ПЗ, призначених для проєктування, розробки, адміністрування і супровід системного та прикладного ПЗ. Системи програмування обов'язково містять компілятор з мови програмування й середовище для розробки та налаштування застосунків.

Контрольні запитання

1. Що розуміють під інженерією програмного забезпечення?
2. Яка різниця між системною та програмною інженерією?
3. Яка різниця між поняттями «інженерія ПЗ» і «програмування»?
4. Яка історія становлення та розвитку інженерії ПЗ?
5. Що таке криза програмного забезпечення і коли вона виникла?
6. Якими двома етапами характеризується процес становлення і розвитку програмної інженерії?
7. В чому суть структурного підходу розроблення ПЗ?
8. В чому суть об'єктно-орієнтованого підходу інженерії ПЗ?
9. Що таке якість ПЗ?
10. Що таке мобільність ПЗ?
11. Які ПЗ засоби існують відповідно до класифікації ПЗ?
12. Що таке прикладне ПЗ?
13. Що розуміють під системами програмування?
14. Які типи програм відносять до системного ПЗ?

Тема 1.2. Стандарти в галузі проєктування програмного забезпечення

План

1. Поняття стандарту проєктування ПЗ.
2. Міжнародні організації в галузі стандартизації.
3. Базовий стандарт розроблення ПЗ ISO 12207.
4. Характеристика областей знань інженерії ПЗ – SWEBOK.
5. Стандарт зрілості компанії-розробника ПЗ CMM.

1. Аналіз стандартів у галузі проєктування ПЗ

Розвиток будь-якої галузі обов'язково супроводжується формалізацією використовуваних підходів та появою стандартів різного рівня.

Стандарт – це узгоджений спосіб робити щось послідовно та повторювано. Стандарти встановлюють мінімальні вимоги з точки зору безпеки, надійності, ефективності та довіри. Вони можуть містити інструкції, вказівки, правила чи визначення, які потім використовуються для проєктування, виробництва, встановлення, випробування та сертифікації, обслуговування та ремонту пристроїв та систем.

Для укладання контракту замовник має бути впевнений, що розробник впорається з проєктом. Саме тому у світовій практиці промислового виробництва існують стандарти, якими керуються розробники і на основі яких відбувається сертифікація виробників на відповідність цим стандартам.

Підприємство, яке виготовляє товари або послуги, застосовує деяку технологію виготовлення. Ця технологія повинна відповідати стандартам на товари чи послуги. Для цього вона проходить обов'язкову сертифікацію на відповідність цим стандартам.

Слово «стандарт» походить від англійського «standart» та перекладається як норма, зразок, мірило.

Стандарт – це нормативно-технічний документ, що затверджується компетентним органом і встановлює комплекс норм, правил по відношенню до предмету стандартизації.

Серед всього різноманіття стандартів прийнято виділяти наступні основні типи стандартів: корпоративні, галузеві, державні, регіональні, міжнародні.

Корпоративні стандарти розробляються крупними фірмами (корпораціями) з метою підвищення якості своєї продукції. Такі стандарти розробляються на основі власного досвіду і з урахуванням вимог світових стандартів. Корпоративні стандарти не сертифікуються, але є обов'язковими для застосування всередині корпорації. В умовах ринкової конкуренції можуть мати закритий характер. У сфері ІТ відомі стандарти, розроблені Microsoft, Intel, IBM тощо.

Галузеві стандарти діють в межах організацій деякої галузі (міністерства). Є, як правило, обов'язковими для галузі та підлягають сертифікації.

Розвиток кожного виду діяльності приводить до потреби у державних засобах забезпечення якості продукції або процесу, тому розробляються та затверджуються ***державні стандарти***. Державні стандарти мають силу закону і розробляються з урахуванням світового досвіду або на основі галузевих стандартів. Можуть мати як рекомендаційний, так і обов'язковий характер (стандарт безпеки). Для сертифікації створюються державні або органи сертифікації, які проводять ліцензію.

Для поліпшення умов співробітництва, розроблення загальнозрозумілих правил конкуренції на міжнародному ринку створюються об'єднання галузевих органів стандартизації, результатом діяльності яких є ***регіональні стандарти*** (діють у обмеженому переліку держав, які приєдналися) або ***міжнародні стандарти*** [35]. Міжнародні стандарти розробляються, як правило, спеціальними міжнародними організаціями на основі світового досвіду і кращих корпоративних стандартів. Мають суто рекомендаційний характер. Право сертифікації отримують організації (державні і приватні), що пройшли ліцензування в міжнародних організаціях.

Одним із перших стандартів, що мав істотний вплив на розвиток теорії проєктування та розроблення ІС, був стандарт BSP (Business System Planning). Даний стандарт був розроблений компанією IBM у середині 70-х рр. XX ст. Процес BSP передбачав виділення в ході розроблення ІС таких кроків: отримання

підтримки керівництва, визначення процесів підприємства, визначення класів даних, проведення інтерв'ю, обробка та організація результатів інтерв'ю. Найважливіші кроки процесу BSP спостерігаються у більшості формальних методик [35].

Нині в галузі проєктування ПЗ відомі наступні стандарти:

- ISO/IEC 12207 «Процеси життєвого циклу програмного забезпечення».
- Серія стандартів ДСТУ ISO/IEC 14598 «Оцінювання програмного продукту».
- Стандарт ДСТУ ISO 15939 «Процес вимірювання».
- Серія стандартів ISO/IEC 15504 «Оцінювання процесів ЖЦ ПЗ».
- Базові стандарти з якості – ISO 9001 «Системи керування якістю. Вимоги».
- ДСТУ 2844–94, ДСТУ 2850–94, що регламентують різні аспекти забезпечення якості ПП.
- Проєкт нової серії стандартів ISO/IEC TR 9126 «Програмна інженерія. Якість продукту».
- Галузевий стандарт IEEE – SWEBOK.

Стандарти, які позначені ДСТУ відносяться до державних стандартів України. Ці стандарти, розроблені відповідно до чинного законодавства України, що встановлюють для загального і багаторазового застосування правила, загальні принципи або характеристики, які стосуються діяльності чи її результатів, з метою досягнення оптимального ступеня впорядкованості, розроблені на основі консенсусу та затверджені уповноваженим органом.

Міжнародні стандарти мають важливе значення для якості та управління ризиками; вони допомагають дослідникам зрозуміти цінність інновацій і дозволяють виробникам виробляти продукцію постійної якості та продуктивності. Міжнародні стандарти завжди використовуються технічними експертами; вони зазвичай добровільні та ґрунтуються на міжнародному консенсусі експертів з багатьох країн.

Міжнародні стандарти також становлять основу для випробувань та сертифікації.

Міжнародні стандарти також часто приймаються країнами чи регіонами, щоб стати національними чи регіональними стандартами. Наприклад, близько 80% європейських електричних та електронних стандартів насправді є міжнародними стандартами, задекларованими в організації IEC.

З іншого боку, нормативні акти – це правила чи директиви, які розробляються та підтримуються національним чи регіональним органом влади. Як правило, дотримання норм є обов'язковим.

Однак досить часто технічні регламенти посилаються на міжнародні стандарти, оскільки стандарти допомагають уникнути занадто детального чи описового характеру закону. Такий підхід дозволяє законодавству залишатися чинним, оскільки стандарти регулярно переглядаються та оновлюються.

2. Міжнародні та закордонні організації в галузі стандартизації

Відповідно до стандартів, які розробляються і затверджуються існують організації, які систематизують усі види діяльностей в галузі інженерії ПЗ, ратифікують їх та організовують зустрічі для обговорення.

Міжнародна організація зі стандартизації, ISO

Найбільш відома установа в галузі стандартизації є міжнародна організація ISO (від англ. International Organization for Standardization, ISO). Щодо скороченої назви організації, необхідно зазначити, що в ній використовується не абревіатура, а саме скорочена назва – ISO. Якщо користуватись абревіатурою, то це мало б бути IOS, проте ані розробники, ані члени організації таку абревіатуру не використовують. Вся технічна документація і стандарти містить скорочену назву ISO. Насправді, назва ISO походить від грецького слова ισος (isos), що означає «рівний». Така назва була вибрана для того, щоб будь-якою мовою вона звучала по можливості однаково, тоді як абревіатура на різних мовах виходить різна [32].

Організація ISO була заснована 23 лютого 1947 р. двадцятьма п'ятьма національними організаціями зі стандартизації, як координуючий орган. Штаб-квартира знаходиться у Швейцарії (м. Женева).

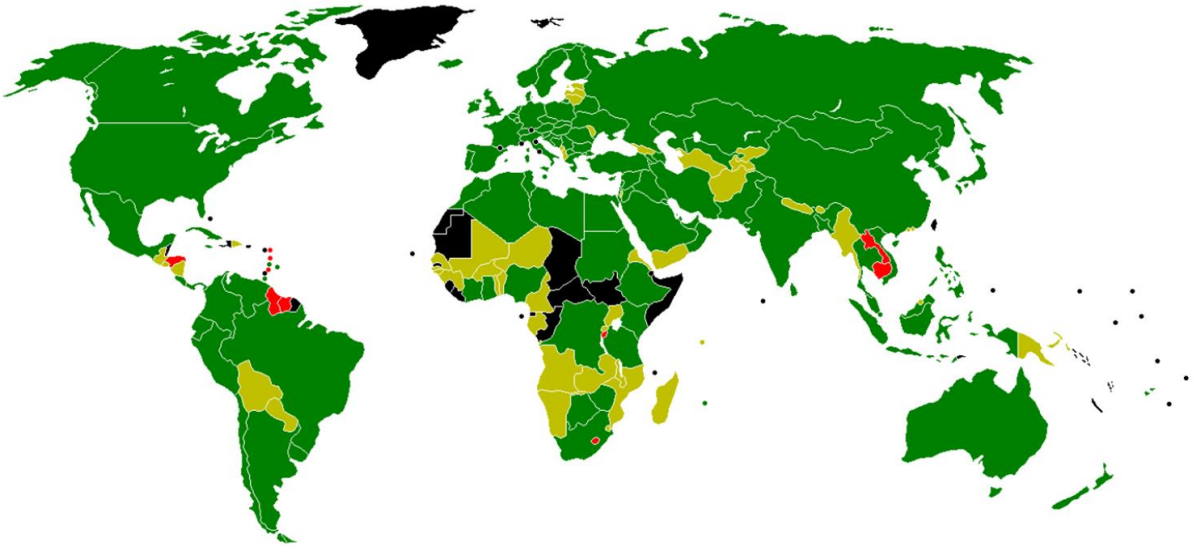


Рис. 1.5. Членство країн в ISO

– *повноправні члени*
– *члени-підписники*
– *члени-кореспонденти*
– *не є членами*

Важко повірити, що розробка стандартів починалась у маленькому офісі у м. Женева, до якої члени організації переїхали у 1949 році і налічувала всього лише 5 співробітників.



Рис. 1.6. Перший офіс ISO, 1949 р.

Нині до ISO входять 164 країни.

Україна є повноправним членом ISO з 1993 року. До 2011 р. Україну в ISO представляв Державний комітет України з питань

технічного регулювання та споживчої політики, поки не був ліквідований в ході адміністративної реформи В.Януковича. Станом на 2014 рік Україну в ISO представляло Міністерство економічного розвитку і торгівлі. З квітня 2016 р. і нині представником України є Державне підприємство «Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості» (ДП «УкрНДНЦ»).

Основною метою діяльності ISO є ратифікація розроблених спільними зусиллями делегатів від різних країн стандартів. Відповідно завданням своєї діяльності ISO визначає сприяння розвитку стандартизації і суміжних видів діяльності у світі з метою забезпечення міжнародного обміну товарами і послугами, а також розвиток співробітництва в інтелектуальній, науково-технічній і економічній галузях.

ISO є неурядовою організацією, тому її можливості у розробці та ратифікації стандартів значно більші, ніж в інших подібних організацій. Це сприяло тому, що багато її стандартів стали державними у багатьох країнах, і дозволяють досить ефективно координувати дії багатьох національних організацій з стандартизації.

ISO тісно співпрацює з Міжнародною електротехнічною комісією (IEC), котра відповідає за галузь електротехніки й електроніки. Деякі види робіт виконуються спільними зусиллями ISO та IEC. Наприклад, питання інформаційних технологій, мікропроцесорної техніки тощо – це об’єкти спільних розробок ISO/IEC.

Для досягнення поставленої мети ISO здійснює свою діяльність за такими напрямами:

- розробка й публікація міжнародних стандартів;
- розробка й поширення документів, що сприяють гармонізації стандартів різних національних систем стандартизації;
- організація обміну інформацією про роботу центральних та технічних органів ISO, а також національних організацій з стандартизації країн-членів ISO;
- співпраця з іншими міжнародними організаціями у суміжних із стандартизацією сферах діяльності.

Стандарти ISO носять рекомендаційний характер і ніякого контролю за виконанням стандартів чи сертифікації на відповідність своїм стандартам вони не проводять – це суверенне право країн. Використання логотипу ISO на виробах або на підприємствах є незаконним, оскільки дає відчуття «схвалення» даного продукту.

Проєкти міжнародних стандартів, які прийняті технічними комітетами ISO, розсилаються комітетам-членам на голосування. Їх публікація як міжнародних стандартів вимагає схвалення щонайменше 75% комітетів-членів, які беруть участь у голосуванні.

Міжнародна електротехнічна комісія, IEC

Міжнародна електротехнічна комісія (від англ. International Electrotechnical Commission, IEC) є міжнародною некомерційною організацією зі стандартизації у сфері електричних, електронних і суміжних технологій. Як описано вище, у співпраці з ISO, нею було розроблено деякі із стандартів. Штаб-квартира розташована у Швейцарії (м. Женева).

Початком міжнародного співробітництва з електротехніці вважається 1881 р., коли відбувся перший Міжнародний конгрес з електрики. Пізніше, у 1904 р., урядові делегати чергового конгресу вирішили, що необхідна спеціальна організація, яка б займалася термінологією в галузі виробництва електричних машин і стандартизацією їх параметрів. IEC була заснована в 1906 р. і складалась з представників національних служб стандартів. Це перша, дійсно міжнародна організація з стандартизації, яка виникла в той час, коли окремі держави світу ще не мали своїх централізованих органів, які б займалися стандартизацією. Спершу до складу IEC увійшли 13 країн світу а на початок 2013 р. в її складі були 82 країни. Україна входить до складу з 1993 р., представником якої є Національний електротехнічний комітет України (www.uas.org.ua).

Комісія IEC сприяла розвитку і розповсюдженню стандартів для одиниць вимірювання, особливо це стосується таких одиниць, як гаус, герц, і вебер. Також комісія IEC запропонувала систему одиниць, яка, зрештою, стала системою SI. В 1938 році був виданий Міжнародний словник з метою уніфікації електричної термінології. Ці зусилля продовжуються і Міжнародний

електротехнічний словник залишається важливою роботою в електричних і електронних галузях промисловості.

IEC має чіткий, усталений та суворо контрольований процес розробки стандартів. Він регулює розробку та публікацію Міжнародних стандартів IEC та інших типів публікацій і включає достатньо часу для коментарів усіх зацікавлених сторін на різних етапах розробки.

Національні комітети IEC (члени IEC) беруть участь на кожному етапі процесу.

Публікації IEC розробляються у понад 200 технічних комітетах та підкомітетах (ТК/СК) та сотнях робочих груп, кожна з яких відповідає за певну технологічну область.

Інститут інженерів з електротехніки та електроніки, IEEE

Інститут інженерів з електротехніки та електроніки (від англ. Institute of Electrical and Electronics Engineers, IEEE) є міжнародною організацією інженерів у галузі електротехніки, радіоелектроніки та радіоелектронної промисловості. Даний інститут є світовим лідером в галузі розроблення стандартів з електроніки та електротехніки. Штаб-квартира організації розташована в США (с.Піскатавей, штату Нью-Джерсі).

Відповідно до інформації, поданої на сайті інституту основна мета IEEE – сприяти технологічним інноваціям та досконалості на благо людства (https://www.ieee.org/about/vision-mission.html).

IEEE керує діяльністю близько 38 товариств, серед яких Товариство інженерів аерокосмічних та електронних систем, Товариство спеціалістів в області лазерів та електрооптики, Товариство інженерів по робототехніці і автоматиці, Товариство інженерів з самохідної техніки та інші.

IEEE видає третину технічної літератури, що стосується застосування комп'ютерів, управління, електроінженерії, та понад 100 журналів, популярних у середовищі професіоналів, проводить на рік понад 300 великих конференцій.

Серед прийнятих стандартів, які стосуються інженерії ПЗ можна відмітити наступні:

– IEEE 830 – Специфікація вимог до ПЗ (Software Requirements Specification).

- IEEE 829 – Тестова документація ПЗ (Software Test Documentation).
- IEEE 1059 – Керівництво щодо верифікації і плану валідації ПЗ (Guide for Software Verification and Validation Plans).
- IEEE 1074 – Процеси життєвого циклу для розвитку ПЗ (Developing a Software Project Life Cycle Process).

Інститут програмної Інженерії, SEI

Інститут програмної інженерії (від англ. Software Engineering Institute, SEI) – американський науково-дослідний центр з штаб-квартирою в Піттсбурзі, штат Пенсільванія, США. Веде діяльність в сфері кібербезпеки, перевірки якості ПЗ, програмної інженерії, придбанні ПЗ, і інших аспектів, що мають вирішальне значення для Міністерства оборони США (SEI знаходиться на федеральному фінансуванні і спонсорований МО США). Відомий розробкою стандартів щодо якості ПЗ і зрілості організацій, що розробляють ПЗ.

SEI ставить своїм основним завданням створення методик для оцінки рівня розвитку внутрішніх процесів в організації. Як підрозділ широко відомий завдяки розробкам в галузі обчислювальної техніки і програмної інженерії, SEI має доступ до самих передових технічних інновацій.

З 1984 р. SEI розвиває і пропагує методики для розробки високоякісного ПЗ. Перша версія моделі технологічної зрілості компанії – розробника ПЗ – CMM (Capability Maturity Model for Software) була створена в 1991 р.

3. Стандарти розроблення ПЗ ISO

Базовий стандарт розроблення ПЗ ISO 12207

Основним стандартом розроблення ПЗ є ISO/IEC 12207, що описує процеси ЖЦ ПЗ. Назва стандарту англійською мовою записується як «Systems and software engineering – Software Life Cycle» [35].

Усі процеси ЖЦ ПЗ в стандарті ISO/IEC 12207 поділяються на три категорії:

- основні процеси;
- процеси підтримки;
- організаційні процеси.



Рис. 1.7. Процеси ЖЦ ПЗ стандарту ISO/IEC 12207

Для кожного з процесів визначені види діяльностей (дії), задачі (вхід), сукупність результатів діяльності і розв'язання задач (результат), а також деякі специфічні вимоги. У стандарті наведено перелік робіт для основних, організаційних процесів і допоміжних процесів – процесів підтримки, але не спосіб їх виконання і не форма поданім результатів.

Наприклад, для процесу «Купівля» визначено такі види діяльностей (дії) (додаток 5):

- Ініціювання.
- Підготовка вимог заявки
- Підготовка угоди.
- Контроль діяльності постачальника.
- Приймання ПЗ.

На вході розробник отримує наступні дані:

- Рішення про початок впровадження ПЗ.
- Результати дослідження діяльності замовника.
- Результати аналізу ринку ПЗ / тендера
- План постачання/розроблення
- Комплексний тест ПЗ

Як результат отримують наступні дані:

- Техніко-економічне обгрунтування впровадження ПЗ.
- Технічне завдання на ПЗ.
- Угода на постачання/ розроблення ПЗ.
- Акти приймання етапів роботи.
- Акт приймально-передавальних випробувань.

Допоміжні процеси призначені для підтримки виконання основних процесів, забезпечення якості проєкту, організації верифікації та тестування ПЗ. ***Організаційні процеси*** визначають дії та завдання замовників та розробників для керування процесами у ході проєкту.

Для підтримки практичного використання стандарту ISO 12207 розроблені такі технологічні документи:

– Керівництво для ISO/IEC 12207 (ISO/IEC TR 24748-3:2011 Systems and software engineering – Life cycle management – Part 3: Guide to the application of ISO/IEC 12207 (Software life cycle processes))

– Керівництво з використання ISO/IEC 12207 в керуванні проєктами (ISO/IEC TR 16326:2009 Systems and software engineering – Life cycle processes - Project management).

У 2002 р. був опублікований стандарт на процеси ЖЦ систем ISO/IEC 15288 Systems and software engineering – System life cycle processes [5], у розробленні якого брали участь фахівці різних галузей: системної інженерії, програмування, управління якістю, людськими ресурсами, безпекою та ін. Даний документ враховує практичний досвід створення систем в урядових, комерційних, військових та академічних організаціях і може бути застосований для широкого класу систем, але його основне призначення – підтримка створення ПЗ. Нині діє версія стандарту 2008 р. У стандарті ISO/IEC 15288:2008 у структурі ЖЦ виділені групи процесів за видами діяльності.



Рис. 1.8. Процеси ЖЦ ПЗ стандарту ISO/IEC 12207

Стадії створення ПЗ, передбачені у стандарті ISO/IEC 15288:2008, та основні результати, які повинні бути досягнуті до моменту їх завершення, наведені в таблиці 1.1.

Таблиця 1.1

Стадії створення ПЗ за стандартом ISO/IEC 15288:2008

№	Стадія	Опис
1	Формування концепції	Аналіз потреб, вибір концепції та проєктних рішень
2	Розроблення	Проєктування ПЗ
3	Реалізація	Створення ПЗ
4	Експлуатація	Введення в експлуатацію та використання ПЗ
5	Підтримка	Забезпечення функціонування ПЗ
6	Зняття з експлуатації	Зупинення використання, демонтаж, архівування ПЗ

Стандарти ISO/IEC 12207 та ISO/IEC 15288 мають єдину термінологію і розроблені таким чином, щоб могли використовуватись одночасно у проєкті.

Стандарт якості ПЗ серії ISO 9000

У процесі промислового розроблення ПЗ обов'язково використовуються стандарти якості серії ISO 9000. Серія ISO 9000 [22] (управління якістю) містить у собі такі стандарти:

– ISO 9000-1. Керування якістю і гарантії якості. Частина 1. Посібник з вибору й використання.

– ISO 9000-2. Керування якістю й гарантії якості. Частина 2. Загальний посібник із застосування стандартів ISO 9001, ISO 9002 і ISO 9003.

– ISO 9000-3. Керування якістю й гарантії якості. Частина 3. Посібник із застосування стандарту ISO 9001 при розробленні, встановленні й супроводі ПЗ.

– ISO 9000-4. Керування якістю й гарантії якості. Частина 4. Посібник з керування надійністю програм.

Основний стандарт ISO 9001:2009 задає модель системи якості для процесів проєктування, розроблення, виробництва, встановлення й обслуговування (продукту, системи, послуги). У моделі ISO 9000 лише згадуються вимоги, які повинні бути реалізовані, але не говориться, як це можна зробити. Тому для побудови повноцінної системи якості за ISO, крім основної моделі

ISO 9001, необхідно використовувати допоміжні галузеві та рекомендаційні стандарти

4. Стандарти організації IEEE

В основі будь-якої професії лежать теоретичні знання і практичний досвід. Незважаючи на велику кількість знань і досвіду накопичених в галузі управління проєктами, проєкти розробки ПЗ або IT-проєкти залишаються, в більшості випадків, загадками – важкопередбачуваними, важкореалізованими і важкокерованими. Пов'язано це з тим, що розробка ПЗ має свою специфіку, яка погано вкладається в класичне управління проєктами. Тому абсолютно логічною була поява стандартизованого посібника знань в цій галузі, який об'єднав всі знання в галузі програмної інженерії [27].

У 2004 році була завершена робота над Посібником з програмної інженерії (The Guide to the Software Engineering Body of Knowledge, SWEBOK), а в 2013 р. з'явилася оновлена версія цього посібника – SWEBOK V3. Даний посібник є спільною працею кількох професійних організацій та представниками промисловості, який був опублікований організацією IEEE.

Нині SWEBOK V3 є міжнародним стандартом ISO / IEC TR 19759:2005, в якому описано 15 галузей знань програмної інженерії:

1) Software Requirements – вимоги до ПЗ.
2) Software Design – проєктування ПЗ.
3) Software Construction – розробка ПЗ.
4) Software Testing – тестування ПЗ.
5) Software Maintenance – супровід ПЗ.
6) Software Configuration Management – управління конфігурацією ПЗ.
7) Software Engineering Management – управління IT-проєктом.
8) Software Engineering Process – процес програмної інженерії.
9) Software Engineering Tools and Methods – методи та інструменти розробки ПЗ.
10) Software Quality – якість ПЗ.

11) Software Engineering Professional Practice – опис критеріїв професіоналізму і компетентності.
12) Software Engineering Economics – економічні аспекти розробки ПЗ.
13) Computing Foundations – основи обчислювальних технологій, що використовуються при розробці ПЗ.
14) Mathematical Foundations – базові математичні концепції і поняття, що застосовуються при розробці ПЗ.
15) Engineering Foundations – основи інженерної діяльності.

Якщо порівнювати з виданням SWEBOK 2004 року, то воно містило лише перші 10 галузей.

Для кожної галузі SWEBOK містить опис ключових елементів у вигляді підгалузей (subareas). Для кожної підгалузі наведена декомпозиція у вигляді списку тем (topics) із їх описом. Нині SWEBOK V3.0 є одним із новітніх загальнодоступних міжнародних стандартів, який можна вільно завантажити з мережі.

5. Стандарт зрілості компанії-розробника ПЗ

Говорячи про стандартизацію процесів підприємства доцільно розглянути модель зрілості технологічних процесів організації Capability Maturity Model (CMM) [35], розроблену Інститутом інженерів програмного забезпечення (Software Engineering Institute, SEI) та корпорацією Mitre під керівництвом Уоттса Хамфри (Watts Humphrey).

Методологія CMM розроблялася й розвивалася в США як засіб, що дозволяє вибирати кращих виробників ПЗ для виконання держзамовлень у першу чергу міністерства оборони.

Для цього були розроблені критерії оцінки зрілості ключових процесів компанії та визначений набір дій, необхідних для їхнього подальшого вдосконалювання. У підсумку методологія виявилася надзвичайно корисною для більшості компаній, що прагнуть якісно поліпшити існуючі процеси проєктування, розроблення, тестування програмних засобів і звести керування ними до зрозумілих і легко реалізованих алгоритмів і технологій, описаних у єдиному стандарті. У подальшому ця модель переросла у методологію підвищення якості процесів підприємства – CMMI (Capability Maturity Model Integration).

Застосування CMMI дозволяє поставити розроблення ПЗ на промислову основу, підвищити керованість ключових процесів і виробничу культуру в цілому, гарантувати якісну роботу й виконання проєктів у термін.

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

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

– відсутнє довгострокове й проєктне планування;

– процес розроблення ПЗ і його ключові моменти не ідентифіковані, реалізація процесу залежить від поточних умов, конкретних менеджерів і виконавців;

– методи і процедури не стандартизовані й не документовані;

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

– процес вироблення рішення відбувається стихійно, на грані мистецтва.

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

Основні ознаки зрілої організації:

– у компанії є чітко визначені й документовані процедури керування вимогами, планування проєктної діяльності,

– керування конфігурацією, створення й тестування програмних продуктів, відпрацьовані механізми керування проєктами;

– ці процедури постійно уточнюються й удосконалюються;

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

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

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

– технології незначно змінюються від проєкту до проєкту на основі стабільних і перевірених підходів і методик;

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

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

СММ визначає п'ять рівнів технологічної зрілості компанії (рис. 1.9), за якими замовники можуть оцінювати потенційних претендентів на підпис контракту, а розробники – удосконалювати процеси створення ПЗ.



Рис. 1.9. Рівні зрілості компанії за СММ

Кожний із рівнів, крім першого, складається з декількох ключових областей процесу (Key Process Area), що містять цілі (Goal), зобов'язання щодо виконання (Commitment to Perform),

можливість виконання (Ability to Perform), виконувані дії (Activity Performed), їхній вимір і аналіз (Measurement and Analysis) та перевірку впровадження (Verifying Implementation).

Таким чином, CMM фактично є комплексом вимог до ключових параметрів ефективного стандартного процесу розроблення ПЗ та засобом його постійного поліпшення. Виконання цих вимог збільшує ймовірність досягнення підприємством поставлених цілей у сфері якості.

Початковий рівень (Initial Level – Level 1).

На даному рівні компанія може отримати замовлення, розробити і передати замовникові програмний продукт. Процес у компанії не формалізований. Він не може строго плануватися і відслідковуватися, його успіх носить випадковий характер. Стабільність розробок відсутня. Лише деякі процеси визначені, результат цілком залежить від зусиль та особистих якостей окремих співробітників. При звільненні таких співробітників розробка проєкту зупиняється.

Успіх одного проєкту не гарантує успішності наступного. До цієї категорії можна віднести будь-яку компанію, що хоч якось виконує взяті на себе зобов'язання.

Ключові галузі процесу цього рівня не зафіксовані.

Повторюваний рівень (Repeatable Level – Level 2). Цьому рівню відповідають підприємства, які володіють певними технологіями керування й розроблення. Керування вимогами та планування у більшості випадків ґрунтуються на розробленій документованій політиці й накопиченому досвіді.

Установлені й уведені в повсякденну практику базові показники для оцінки параметрів проєкту. Менеджери відслідковують виконання робіт і контролюють тимчасові й виробничі витрати.

У компанії розроблені деякі внутрішні стандарти й організовані спеціальні групи перевірки якості QA (від англ. quality assurance – забезпечення якості). Зміни версій кінцевого програмного продукту й створених проміжних програмних засобів відслідковуються в системі керування конфігурацією. Є необхідна дисципліна дотримання встановлених правил. Ефективні методики й процеси установлюються, що забезпечує

можливість повторення успіху попередніх проєктів у тій самій прикладній області.

Ключові галузі процесу розроблення ПЗ цього рівня:

- Керування вимогами (Requirements management)
- Планування проєкту розробки ПЗ (Software project planning)
- Відстеження ходу проєкту та контроль (Software project tracking and oversight)
- Керування субпідрядниками розробки ПЗ (Software subcontract management)
- Забезпечення якості розробки ПЗ (Software quality assurance)
- Керування конфігурацією продукту (Software configuration management).

Визначений рівень (Defined Level – Level 3).

Даний рівень передбачає, що всі елементи процесу визначені, стандартизовані та задокументовані.

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

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

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

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

Ключові галузі процесу розроблення ПЗ цього рівня:

- Мета організаційних процесів (Organization Process Focus).
- Визначення (стандартного) процесу (Organization Process Definition).
- Програма навчання (Training Program).
- Керування інтегрованою розробкою ПЗ (Integrated Software Management).
- Технологія розробки програмних продуктів (Software Product Engineering).
- Міжгрупова координація (Intergroup Coordination).
- Експертні (спільні) оцінки колег (Peer Reviews).

Керований рівень (Managed Level – Level 4).

Рівень, на якому розроблені та закріплені у відповідних нормативних документах кількісні показники якості. Більш високий рівень керування проєктами досягається за рахунок зменшення відхилень різних показників проєкту від запланованих. При цьому тенденції зміни продуктивності процесу можна відділити від випадкових варіацій на підставі статистичної обробки результатів вимірювань у процесах.

Ключові галузі процесу розроблення ПЗ цього рівня:

- Кількісне керування процесом (Quantitative Process Management).
- Керування якістю ПЗ (Software Quality Management).

Рівень оптимізації (Optimizing Level – Level 5).

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

Ключові галузі процесу розроблення ПЗ цього рівня:

- Запобігання дефектам (Defect Prevention).

– Керування змінами технологій (Technology Change Management).
– Керування змінами процесу (Process Change Management).

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

У результаті аудиту та атестації компанії присвоюється певний рівень, що після наступних аудитів може підвищуватися або знижуватися. Кожний наступний рівень в обов'язковому порядку містить у собі всі ключові характеристики попередніх.

У зв'язку з цим сертифікація компанії щодо одного з рівнів припускає безумовне виконання всіх вимог більш низьких рівнів.

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

Основний недолік СММ полягає в тому, що модель не авторизована як стандарт ні міжнародними, ні національними органами зі стандартизації. Втім, СММ давно стала промисловим стандартом. До недоліків моделі також необхідно віднести більші зовнішні накладні витрати на приведення процесів компанії у відповідність до моделі СММ, ніж при використанні моделей Міжнародного стандарту ISO 9000.

Контрольні запитання

1. Які види стандартів вам відомі? Наведіть приклади.
2. Що відповідно до стандарту СММ повинна запровадити компанія-розробник програмних систем, щоб якісно виконувати замовлення?
3. Які міжнародні стандарти ISO регулюють розроблення інформаційних систем?
4. Які стандарти організації IEEE регламентують розроблення програмних систем?
5. Які галузі охоплює керівництво SWEBOK?

6. Які державні стандарти України регламентують розроблення інформаційних систем та ПЗ?
7. Які рівні зрілості компанії виділяє стандарт CMM? Чим вони характеризуються?
8. На якому рівні зрілості компанія може контролювати якість створюваного ПЗ?

Тема 1.3. Життєвий цикл програмного забезпечення

План

1. Поняття життєвого циклу програмного забезпечення.
2. Класичний життєвий цикл ПЗ – каскадна модель.
3. Нотації проєктування програмного забезпечення.
4. Процеси ЖЦ ПЗ за стандартом ISO/IEC 12207.

1. Поняття життєвого циклу програмного забезпечення

Поява поняття життєвого циклу програмного забезпечення (ЖЦ ПЗ) пов'язана із першою кризою програмування. Суть кризи полягала в тому, що програмні проєкти все частіше стали виходити з-під контролю: порушувалися терміни розробки, перевищувалися заплановані обсяги фінансування, результати не відповідали необхідним критеріям. Багато проєктів взагалі не були завершені. Крім того, виявилося, що недостатньо розробити програму, а треба її ще супроводжувати. Цей етап часто вимагав більше засобів, ніж її розроблення. Проблемна ситуація також наростала із появою складних проєктів. Необхідно було вживати заходів для радикального вдосконалення принципів і методів розроблення ПЗ з урахуванням розвитку й супроводу [29].

Поняття ЖЦ ПЗ виникло під впливом потреби у систематизації процесів розроблення ПЗ. Систематизація стала першим етапом на шляху до автоматизації процесу розроблення ПЗ [29.

Провідні фахівці заговорили про те, що треба враховувати досвід промислового проєктування й виробництва, де успішно реалізовано розроблення складних проєктів. Методологічну основу промислової інженерії становить поняття ЖЦ виробу (продукту) як сукупності всіх дій – процесів, які треба виконати протягом всього «життя» виробу [29].

ЖЦ промислового виробу – це набір етапів (фаз, стадій), а саме: проєктування, виготовлення зразка, організація виробництва, серійне виробництво, експлуатація, ремонт, висновок з експлуатації тощо. Відповідно ЖЦ складається з дій, технологічних процесів, операцій [29].

Уперше про ЖЦ ПЗ заговорили в 1968 р. у Лондоні, де відбулася зустріч керівників проєктів з розроблення ПЗ. На зустрічі аналізувалися проблеми й перспективи проєктування, розроблення, поширення й підтримки ПЗ. Принципи і методи розроблення ПЗ вимагали постійного вдосконалення. Саме на цій зустрічі була запропонована концепція ЖЦ ПЗ (SLC – Software Lifetime Cycle) як послідовності кроків-стадій, які необхідно виконати в процесі створення й експлуатації ПЗ [29].

Концепція не отримала одностайного схвалення. Суперечки, деякою мірою, припинилися, коли у 1970 р. У. Ройс визначив стадії типового ЖЦ ПЗ, а також висловив припущення, що контроль над виконанням цих стадій приведе до підвищення якості ПЗ та скороченню вартості та термінів виконання проєктів з розроблення ПЗ [29].

Інженерія ПЗ пройшла декілька етапів розвитку, в процесі яких були сформульовані фундаментальні принципи і методи розроблення програмних продуктів. Основний принцип інженерії ПЗ полягає в тому, що програми створюються в результаті виконання декількох взаємозв’язаних етапів. Ці етапи складають ЖЦ програмного продукту. Характер взаємозв’язаності окремих етапів ЖЦ розробки визначається моделлю ЖЦ. Тобто ЖЦ ПЗ вказує на всі процеси розроблення ПЗ, а модель ЖЦ ПЗ – на необхідні для заданого проєкту і, що важливо, на порядок виконання цих процесів [29].

Думка провідних фахівців в галузі інженерії ПЗ зводиться до того, що ключове завдання інженерії ПЗ – не використання технологій, а комбінування технологічних знань, людей і процесів для досягнення бізнес цілей організації. Це завдання вирішується на основі поняття ЖЦ ПЗ [29].

Отже, ЖЦ ПЗ визначається моделлю й описується у формі методології (методу). Модель або парадигма ЖЦ визначає загальну організацію і, як правило, основні його фази та принципи переходу між ними. Методологія визначає комплекс робіт, їх

детальний зміст і рольову відповідальність спеціалістів на всіх етапах вибраної моделі.

Типові схеми упорядкування виконання робіт з проєктування і розроблення ПЗ одержали назву ***життєвого циклу***.

Зазвичай до етапів ЖЦ відносять: аналіз вимог, проєктування, програмування, тестування і налагодження, експлуатацію, супровід і підтримку. Але це не повний перелік.

Існує деяка варіативність у проходженні етапів ЖЦ під час розробки та впровадження продукту на ринок. Для кожного продукту це відбувається по-своєму, але щоб цим якось керувати були сформульовані моделі ЖЦ ПЗ – спрощене й узагальнене уявлення про те, як розвивається продукт. У реальності життя продукту не відповідає моделі.

Життєвий цикл програмного забезпечення (ЖЦ ПЗ) – це безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

ЖЦ розбивається на окремі процеси.

Процес – це сукупність дій і завдань, які мають на меті досягнення значимого результату. Процеси інколи називають етапами або фазами ЖЦ.

ЖЦ ПЗ супроводжується розробленням, обігом та використанням програмної документації.

Програмна документація – сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програм. Програмна документація є одним з видів технічної документації.

2. Класичний ЖЦ ПЗ – каскадна модель

Класична модель ЖЦ ПЗ застосовується як для розробки програм для індивідуального використання, так і для складних програмних виробів промислового і комерційного призначення.

Як вже було зазначено, класичний ЖЦ ПЗ проходить декілька основних етапів від аналізу вимог до ПЗ та їх специфікації до повного зняття системи з експлуатації. Дуже часто класичний ЖЦ називають ***каскадною*** або ***водоспадною*** моделлю, підкреслюючи, що розробка розглядається як послідовність етапів, причому

перехід на наступний, ієрархічно нижчий етап відбувається тільки після повного завершення робіт на поточному етапі.

Каскадна модель була однією з перших моделей, яку застосували до процесу розробки ПЗ. Відповідно до неї кожен етап роботи виконується лише раз. На кожному етапі робота виконується настільки ретельно, щоб потреби повертатись до попереднього не виникало. Результат виконання кожного етапу, перед передачею в наступний, піддається верифікації.

Вважається, що розробка починається на системному рівні і проходить через такі основні етапи ЖЦ ПЗ:

– Етап аналізу та специфікації вимог – важливий і фундаментальний етап, на якому виконується збір та аналіз вимог замовника виконавцем і подання їх у нотації, яка є зрозумілою як для замовника, так і для виконавця. Окрім визначення вимог, на даному етапі здійснюється планування якості, оцінка майбутніх ризиків тощо.

– Проєктування – перетворення вимог до розробки в послідовність проєктних рішень, які визначають структуру й поведінку системи, а також інтерфейс користувача. На даному етапі створюються уявлення про архітектуру ПЗ, модульну структуру ПЗ, алгоритмічну структуру ПЗ, структуру даних, вхідного і вихідного інтерфейсу (форм даних). Початкові дані для проєктування знаходяться в специфікації аналізу, отже в ході проєктування виконується трансляція вимог до ПЗ в безліч проєктних уявлень.

– Кодування – розроблення ПЗ. На даному етапі відбувається переклад результатів проєктування в текст на відповідній мові програмування.

– Тестування – перевірка програмного продукту на наявність помилок. На даному етапі відбувається виконання програми для виявлення дефектів у функціях, логіці і формі реалізації ПС.

– Експлуатація – дії з обслуговування системи під час її використання – консультації користувачів, вивчення їхніх побажань тощо.

– Супровід – дії щодо керування модифікаціями, підтримки актуального стану та функціональної придатності, інсталяцію нових та вилучення попередніх версій програмних систем у

користувача. На даному етапі може відбуватись внесення змін до ПС. Ціль змін – виправлення помилок, адаптація до змін зовнішнього для ПС середовища, удосконалення ПС на вимогу замовника. Супровід ПЗ полягає в повторному застосуванні кожного з попередніх кроків (етапів) ЖЦ до існуючої програми, але не в розробці нової.

– Зняття системи з експлуатації – дії з поступового припинення використання програми, перехід до нової програми, навчання персоналу новій програмі.

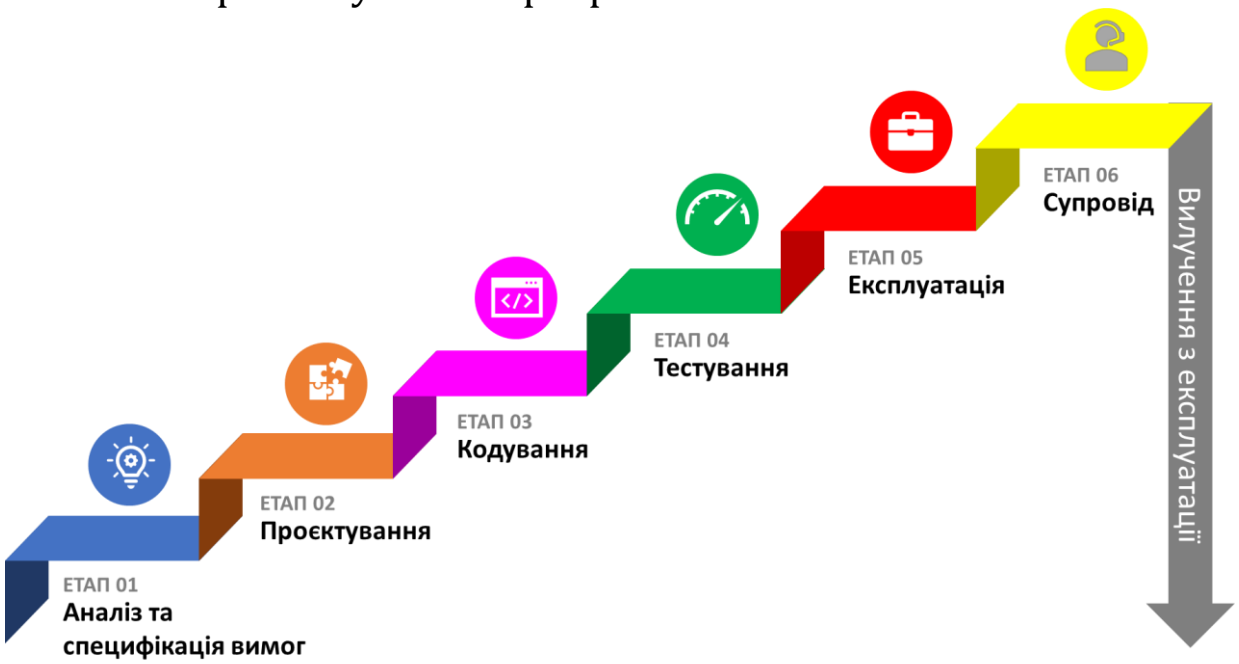


Рис. 1.10. Класичний ЖЦ ПЗ

Найважливішим видом діяльності на всіх етапах ЖЦ є критичний і всебічний огляд та оцінка матеріалів, підготовлених на відповідній фазі ЖЦ ПЗ. Під час оцінки матеріалів відбуваються такі процеси:

– розгляд і схвалення вимог користувача;
– розгляд і схвалення вимог до ПС;
– розгляд і схвалення архітектурного проєкту ПС;
– затвердження акту про попереднє приймання ПС;
– розгляд і схвалення документа про передачу ПС в експлуатацію;
– затвердження акту про остаточне приймання і передачу в експлуатацію ПС і прийняття документу про результати всіх робіт по створенню ПЗ.

Таким чином, можна визначити основні принципи каскадної моделі ЖЦ ПЗ:

– Строге і послідовне виконання фаз:
  - кожна наступна фаза починається лише тоді, коли повністю завершено виконання попередньої фази;
  - кожна фаза має певні критерії входу і виходу: вхідні і вихідні дані;
  - кожна фаза повністю документується;
  - перехід від однієї фази до іншої здійснюється за допомогою формального огляду за участю замовника

– Основа моделі – сформульовані вимоги (технічне завдання), які змінюватися не повинні.

– Критерій якості результату – відповідність продукту встановленим вимогам.

Як і будь-яка інженерна схема, класичний ЖЦ ПЗ має переваги і недоліки.

До **переваг** моделі класичного ЖЦ ПЗ доцільно віднести:

– Проста і зрозуміла замовникам, оскільки часто використовується іншими організаціями для відстеження проєктів, не пов’язаних з розробкою ПЗ.

– Проста і зручна в застосуванні:
  - процес розробки виконується поетапно;
  - її структурою може керуватися навіть слабо підготовлений в технічному плані або - недосвідчений персонал;
  - вона сприяє здійсненню суворого контролю менеджменту проєкту;

– Кожну стадію можуть виконувати незалежні команди (все документовано).

– Дозволяє досить точно планувати терміни і витрати.

Недоліки класичного ЖЦ:

– Реальні проєкти часто вимагають відхилення від стандартної послідовності кроків. Спроба повернутися на одну або дві фази назад, щоб виправити будь-яку проблему або недолік, призведе до значного збільшення витрат і збою в графіку.

– Інтеграція компонент, на якій зазвичай виявляється велика частина помилок, виконується в кінці розробки, що сильно збільшує вартість усунення помилок.

– Запізнення з отриманням результатів – якщо в процесі виконання проєкту вимоги змінилися, то вийде застарілий результат

– Цикл заснований на точному формулюванні початкових вимог до ПЗ (реально на початку проєкту вимоги замовника визначені лише частково).

– Результати проєкту доступні замовникові тільки в кінці роботи.

Недоліки каскадної моделі особливо гостро проявляються в разі, коли важко (або неможливо) сформулювати вимоги або вимоги можуть змінюватися в процесі виконання проєкту. В цьому випадку розробка ПЗ має принципово циклічний характер.

3. Нотації проєктування

Нотація дозволяють представити опис об'єкта (елемента) ПЗ і його структуру, а також поведінку системи за цим об'єктом. Нотації містять у собі мовний опис архітектури й інтерфейсу, діаграм класів і об'єктів, діаграм сутність-зв'язок, конфігурації компонентів, схем розгортання, а також структурні діаграми, що задають у наочному вигляді оператори циклу, розгалуження, вибору і послідовності.

Нотації – це особлива графічна мова, яка дозволяє описувати роботу компанії, наочно демонструвати взаємодію між різними підрозділами, тобто описувати бізнес-процеси. Нотації можуть застосовуватися для процесного або функціонального моделювання. Загалом, нотації можна назвати мовою програмування в бізнес-аналізі.

Існує два типи нотацій: ***структурна***, ***поведінкова*** та множина їх різних представлень.

До ***структурних нотацій*** відносять формальні мови специфікацій і проєктування.

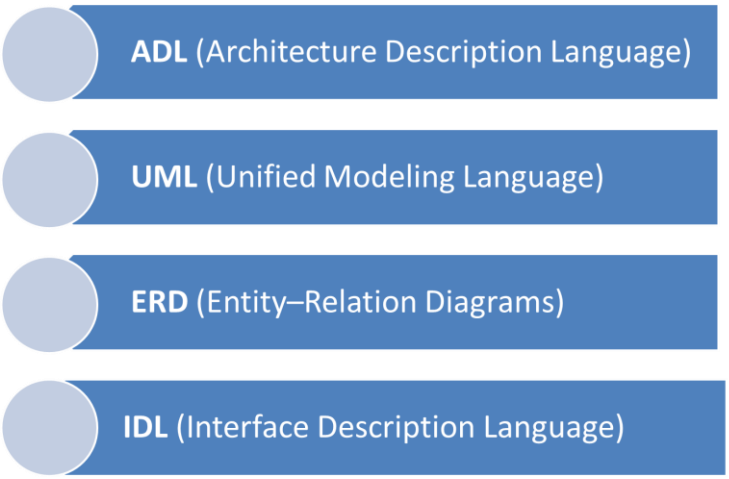


Рис. 1.11. Формальні мови специфікацій і проєктування

Можна розглянути структурну нотацію на прикладі ***методології IDEF0***, яка використовує методи функціонального моделювання і графічного опису процесів і призначені для формалізації і опису бізнес-процесів. Особливістю IDEF0 є її акцент на ієрархічне представлення об'єктів, що значно полегшує розуміння предметної галузі. В IDEF0 розглядаються логічні зв'язки між роботами, а не послідовність їх виконання в часі.

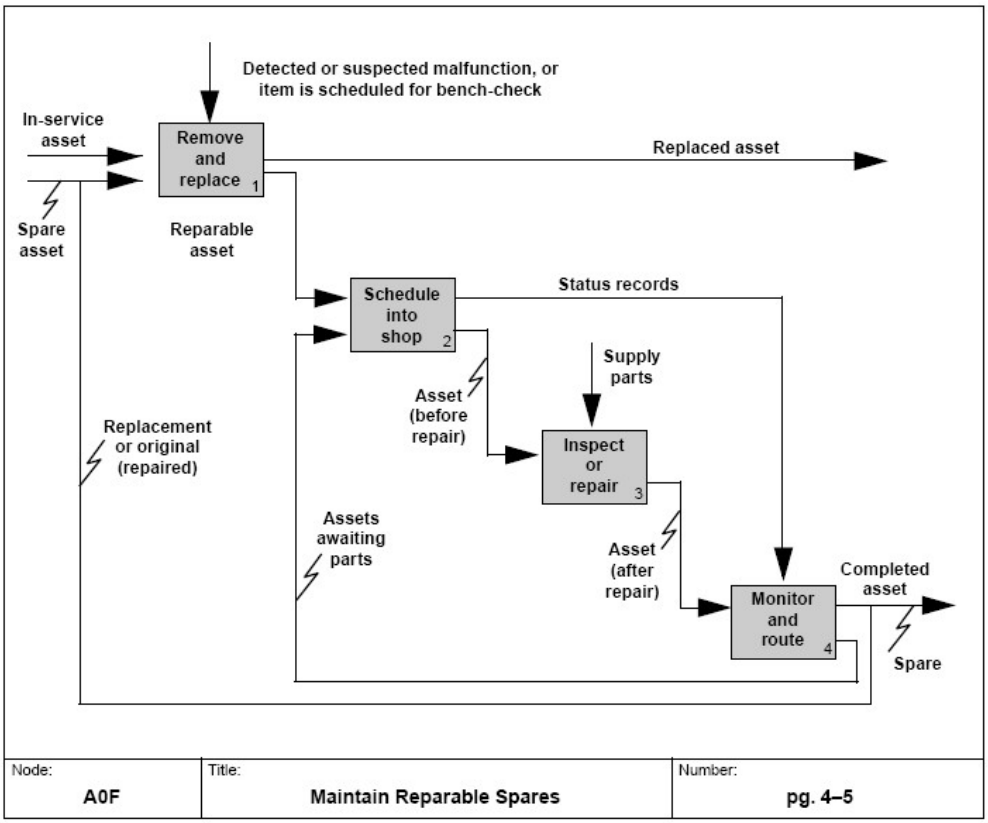


Рис. 1.12. Приклад нотації IDEF0

Бізнес-процеси в нотації IDEF0 подаються у формі прямокутника, а стрілки відображають зв'язок з іншими

процесами і зовнішнім середовищем. Особливістю нотації є можливість декомпозувати процеси на підпроцеси і, таким чином, будувати ієрархічні моделі бізнес-процесів.

Нотація IDEF0 використовується для створення верхнього рівня моделі бізнес-процесів. Побудова IDEF0-діаграми верхнього рівня забезпечує найбільш загальний або абстрактний опис об'єкта моделювання. На нижньому рівні для опису алгоритму (сценарію) виконання процесу допустимо змінити стандарт IDEF0 на нотацію Процес, Процедура, EPC або BPMN 2.0.

Поведінкові нотації відображають динамічний аспект роботи системи та її компонентів (додаток 6).

До поведінкових нотацій відносять:

- Діаграми потоків даних (Data Flow).
- Діяльності (Activity).
- Кооперації (Colloboration).
- Послідовності (Sequence).
- Діяльності (Activity),
- Таблиці прийняття рішень (Decision Tables),
- Передумови і постумови (Pre-Post Conditions
- Формальні мови специфікації (Z, VDM, RAISE) і проєктування.

4. Процеси ЖЦ ПЗ

ЖЦ ПЗ складається з опису багатьох процесів, взаємозв'язків між ними, а також сформульованих дій і задач, виконуваних у цих процесах. Іншими словами, стандартний ЖЦ визначає лише схему робіт за процесами розробки ПС, а не те, як саме виконувати ті або інші процеси. Стандарт ISO/IEC 12207 визначає загальну структуру і зміст ЖЦ ПС, починаючи з розробки концепції до утилізації системи.

Процес є важливою складовою будь-якої інженерної діяльності. Кожен процес має чіткий опис, визначені вхідні дані й результати.

Встановлення процесу передбачає:

- Опис процесу – детальний опис дій і операцій процесу.
- Навчання процесу – проведення занять з персоналом з освоєння процесу.

– Введення метрик – встановлення кількісних показників процесу виконання.
– Контроль виконання – вимір метричних показників і оцінювання процесу виконання.
– Удосконалення – зміна процесу за змінних умов застосування.

Розглянемо процеси на прикладі процесів ЖЦ ПЗ відповідно до стандарту ISO/IEC 12207:2002. Усі процеси в стандарті ISO/IEC 12207:2002 поділяються на три категорії:

– основні процеси
– процеси підтримки;
– організаційні процеси.

Для кожного з процесів визначені види діяльності (дії - activity), задачі, сукупність результатів (виходів) діяльності і розв'язання задач, а також деякі специфічні вимоги. У стандарті наведено перелік робіт для основних, організаційних процесів і процесів підтримки, але не спосіб їх виконання і не форма подання цих результатів.

Основні процеси

Згідно стандарту ISO/IEC 12207:2002 до основних процесів належать:

– процес придбання;
– процес постачання.

Процес придбання ініціює ЖЦ ПС (програмної системи) і визначає дії організації-покупця (або замовника) що отримує автоматизовану систему або сервіс. Цей процес містить у собі такі види діяльності: ініціювання і підготовка запиту, оформлення контракту і його актуалізація, моніторинг користувачів, приймання і завершення.

Процес постачання визначає дії з передачі покупцю ПС або сервісу і містить у собі такі види діяльності: підготовку пропозицій (відповідей на запити), оформлення контракту, планування, виконання і контроль, про що повідомляє, аналіз і оцінку продукту; постачання і завершення робіт і постачання. Процес постачання починається тоді, коли встановлені договірні відношення між замовником і постачальником.

Залежно від умов договору процес постачання може містити у собі *процес розроблення ПЗ*, *процес експлуатації і супроводу* для виправлення і поліпшення ПС.

Процес розроблення визначає дії підприємства-розробника програмного продукту та передбачає: аналіз вимог до системи; проєктування архітектури системи, детальне проєктування компонентів ПС, кодування і тестування ПС, інтеграція системи, кваліфікаційне тестування, установка ПС і забезпечення приймання ПС.



Рис. 1.13. Процеси розроблення, експлуатації та супроводу

Процес експлуатації визначає дії підприємства-оператора, що забезпечує обслуговування системи в ході її експлуатації користувачами (консультування користувачів, вивчення потреб операторів, задоволеності споживачів системою тощо). Цей процес регламентує задачі і дії з функціонального тестування, перевірки правильності експлуатації системи; дотримання інструкцій і настанов з її запуску.

Процес супроводу визначає дії організації, що виконує супровід програмного продукту (керування модифікаціями, підтримку поточного стану і функціональної придатності, інсталяцію програмного продукту на обчислювальній системі користувача та її вилучення при списанні). Даний процес містить у собі завдання і дії щодо аналізу питань супроводу і модифікації, розробки планів і реалізації модифікації системи, аналізу результатів супроводу після змін системи, міграції ПС в інше середовище або її виведення з експлуатації.

До категорії основних процесів належать також «первинні» процеси, що визначають порядок підготовки договору на розробку ПС, моніторинг діяльності постачальників ПС тощо. Стандарт містить у собі опис допоміжних процесів, що регламентують додаткові дії з перевірки продукту, керування проєктом та його якістю.

Процеси підтримки

До процесів підтримки розроблення ПС належать: документування, керування версіями, верифікація і валідація, перегляди, аудити, оцінювання продукту та ін. Процес керування версіями за змістом відповідає керуванню конфігурацією системи, що так само, як і продукти процесів, повинні перевірятися на правильність реалізації цілей проєкту і відповідність вимогам замовника. Завдання з перевірки рекомендується виконувати спеціальним контролерам, які знаються на методах і процесах проєктування ПС.

Організаційні процеси

До організаційних процесів належать: керування проєктом, якістю, ризиками.

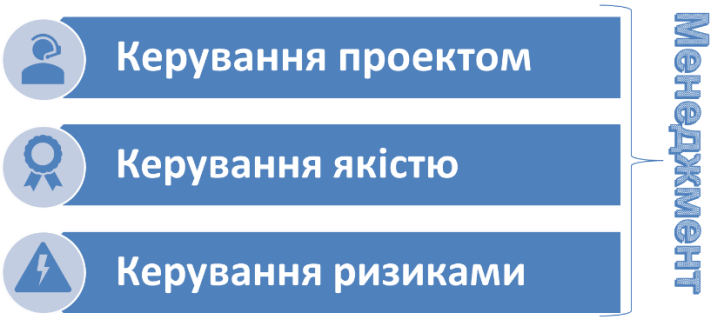


Рис. 1.14. Організаційні процеси ЖЦ ПЗ

Ці процеси виконуються спеціальними службами, що здійснюють планування робіт у проєкті, контроль процесів, визначення метрик для вимірювання продуктів, перевірку показників якості, дотримання стандартних положень тощо.

Контрольні запитання

1. Що таке життєвий цикл ПЗ?
2. Яка історія виникнення терміну ЖЦ в інженерії ПЗ?
3. Що є складовими ЖЦ ПЗ? Наведіть приклади.
4. Чим ЖЦ ПЗ відрізняється від моделі ЖЦ ПЗ?
5. Назвіть основні етапи ЖЦ ПЗ?

6. Які фази у каскадній моделі ЖЦ ПЗ?
7. Які принципи каскадної моделі ЖЦ ПЗ?
8. Які переваги та недоліки каскадної моделі ЖЦ ПЗ?
9. Що таке процес у ЖЦ ПЗ?
10. Чим основні процеси відрізняються від неосновних процесів ЖЦ ПЗ?
11. З чого складаються процеси ЖЦ ПЗ?
12. Які особливості етапу аналізу і специфікації вимог до ПЗ?
13. Які особливості етапу проєктування ПЗ?
14. Які особливості етапу тестування ПЗ?

Тема 1.4. Моделі життєвого циклу програмного забезпечення

План

1. Спіральна модель.
2. Ітераційна та V-подібна моделі.
3. Інкрементна модель.
4. Промислові технології створення ПЗ.

1. Спіральна модель

Стандарти ЖЦ ПЗ встановлюють склад і організацію процесів, які можуть виконуватися на різних етапах ЖЦ ПЗ. У багатьох випадках ці процеси виявляються збитковими і наведені стандарти адаптуються до умов виконання конкретних проєктів відповідно до вибіркових моделей ЖЦ, враховуючи специфіку цих проєктів [20].

У стандарті ISO / IEC 12207 модель ЖЦ («life cycle model») визначається як структура, що складається з процесів, робіт і завдань, що включають в себе розробку, експлуатацію та супровід програмного продукту, що охоплює життя системи від встановлення вимог до неї до припинення її використання. При цьому, конкретні моделі визначаються специфікою завдань, обмеженнями на ресурси, досвідом розробників тощо [20].

Відомі деякі типові моделі ЖЦ ПЗ, які проявили себе в певних умовах, мають певні переваги, недоліки та умови застосування. Ці типові моделі встановлюють деякі принципи організації моделі ЖЦ ПЗ. До числа основних моделей ЖЦ ПЗ відносять ***каскадну*** і

спіральну моделі. На практиці часто використовують ітераційну, V-подібну, інкрементну і модель швидкого прототипування.



Рис. 1.15. Моделі ЖЦ ПЗ

Каскадна модель є класичною моделлю ЖЦ ПЗ, яка передбачає послідовність виконання певних етапів розробки з переходом на ієрархічно нижчий етап, після виконання робіт на поточному етапі. Дана модель була детально розглянута у попередньому розділі, тому розглянемо наступну широко застосовувану модель – спіральну.

У випадку з каскадною моделлю передбачається, що розробник після завершення відповідного етапу, до нього вже не повертається. В цьому є певний недолік, оскільки ПЗ потребує постійного перегляду вимог до нього та вдосконалення.

На практиці, при вирішенні досить великої кількості завдань, розробка ПЗ має циклічний характер, коли після виконання деяких стадій доводиться повертатися на попередні.

Можна вказати дві основні причини таких повернень:

– Помилки розробників, допущені на ранніх стадіях і виявлення на пізніх стадіях – помилки аналізу, проєктування, кодування, які виявляються, як правило, на стадії тестування.
– Зміна вимог в процесі розробки («помилки» замовників). Це або неготовність замовників сформулювати вимоги («Сказати, що повинна робити програма я зможу тільки після того, як побачу як вона працює»), або зміни вимог, викликані змінами ситуації в процесі розробки (зміни ринку, нові технології тощо).

Циклічний характер розробки ПЗ відображений в спіральній моделі ЖЦ, описаної ***Баррі Боемом*** в 1988 році. Спіральна модель

була запропонована як альтернатива каскадній моделі, що враховує циклічний характер розробки ПЗ. Основні принципи спіральної моделі можна сформулювати наступним чином:

1. Розробка варіантів продукту, що відповідають різним варіантам вимог з можливістю повернутися до більш ранніх варіантів.

2. Створення прототипів ПЗ як засобу спілкування з замовником для уточнення і виявлення вимог.

3. Планування наступних варіантів з оцінкою альтернатив і аналізом ризиків, пов’язаних з переходом до наступного варіанту.

4. Перехід до розробки наступного варіанта до завершення попереднього в разі, коли ризик завершення чергового варіанту (прототипу) стає невиправдано високий.

5. Використання каскадної моделі як схеми розробки чергового варіанту.

6. Активне залучення замовника до роботи над проєктом. Замовник бере участь в оцінці чергового прототипу ПЗ, уточнення вимог при переході до наступного, оцінці запропонованих альтернатив чергового варіанту та оцінки ризиків.

Схема роботи спіральної моделі виглядає наступним чином. Розробка варіантів продукту надається як набір циклів розкручування спіралі. Кожному циклу спіралі відповідає така ж кількість стадій, як і в моделі каскадного процесу. При цьому, початкові стадії, пов’язані з аналізом і плануванням представлені більш детально з додаванням нових елементів.

У кожному циклі виділяються чотири базові фази:

1. Визначення цілей, альтернативних варіантів і обмежень.
2. Оцінка альтернативних варіантів, ідентифікація і дозвіл ризиків.
3. Розробка продукту наступного рівня.
4. Планування наступної фази.

Істотна відмінність спіральної моделі від інших моделей процесу створення ПЗ полягає в точному визначенні та оцінюванні ризиків. Якщо говорити неформально, то ризик – це ті неприємності, які можуть трапитися в процесі розробки системи. Наприклад, якщо при написанні програмного коду використовується нова мова програмування, то ризик може полягати в тому, що компілятор цієї мови може бути ненадійним

або що результуючий код може бути не достатньо ефективним. Ризики можуть також полягати в перевищенні термінів або вартості проєкту. Таким чином, зменшення (дозвіл) ризиків – важливий елемент управління системним проєктом.

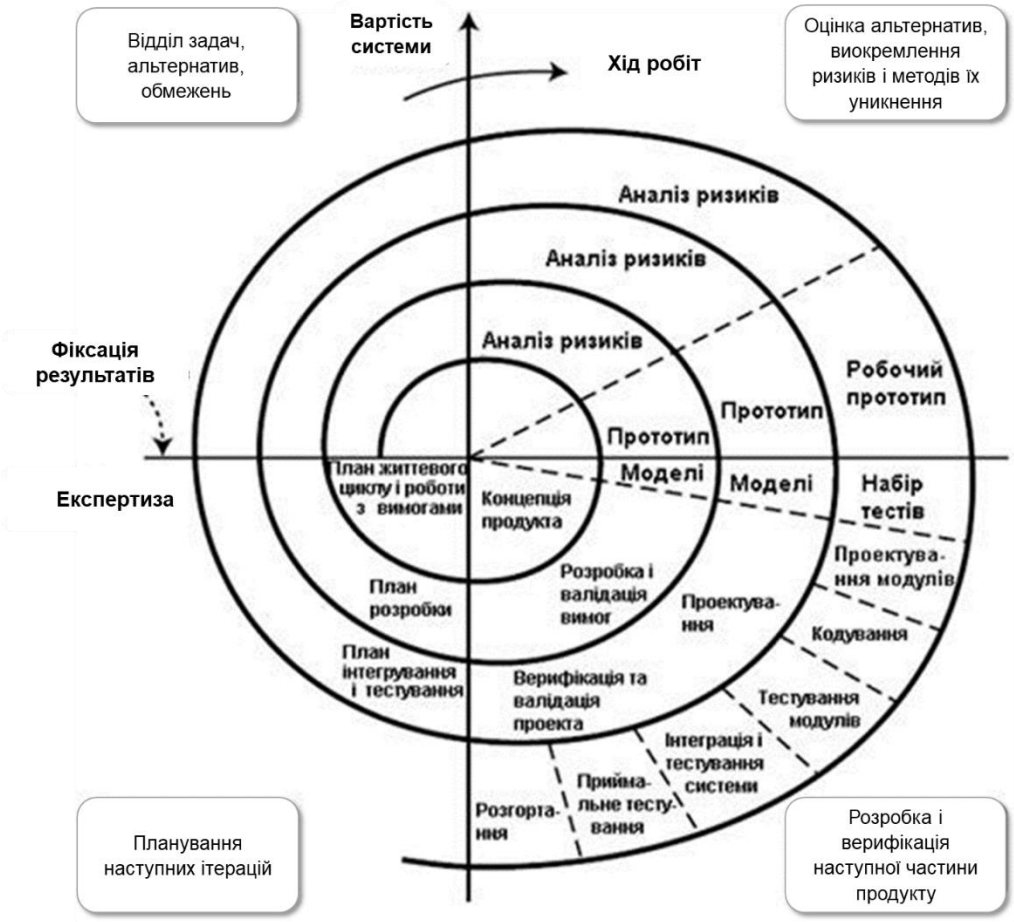


Рис. 1.16. Спіральна модель ЖЦ ПЗ

Розглянемо фази розробки ПЗ за спіральною моделлю детальніше:

1. «Розкручування» проєкту починається з аналізу загальної постановки завдання на розробку ПЗ – концепції продукту. В межах даної фази визначається загальна мета, встановлюються попередні обмеження, визначаються можливі альтернативи підходів до вирішення завдання. Далі проводиться оцінка підходів, встановлюються їх ризики. На кроці розробки створюється концепція (бачення) продукту і шляхів його створення.

2. Наступний цикл починається з планування вимог і деталей ЖЦ продукту для оцінки витрат. На фазі визначення мети встановлюються альтернативні варіанти вимог, пов’язані з

аранжуванням вимог по важливості і вартості їх виконання. На фазі оцінки встановлюються ризики варіантів вимог. На фазі розробки – специфікація вимог (із зазначенням ризиків і вартості), готується демо-версія ПЗ для аналізу вимог замовником.

3. Наступний цикл – розробка проєкту – починається з планування розробки. На фазі визначення цілей встановлюються обмеження проєкту (за термінами, обсягом фінансування, ресурсів тощо), визначаються альтернативи проєктування, пов'язані з альтернативами вимог, застосовуваними технологіями проєктування, залученням субпідрядників. На фазі оцінки альтернатив встановлюються ризики варіантів і робиться вибір варіанта для подальшої реалізації. На фазі розробки виконується проєктування і створюється демо-версія, яка відображає основні проєктні рішення.

4. Наступний цикл – реалізація ПЗ – також починається з планування. Альтернативними варіантами реалізації можуть бути вживані технології реалізації, залучені ресурси. Оцінка альтернатив і пов'язаних з ними ризиків на цьому циклі визначається ступенем «відпрацьованості» технологій і «якістю» наявних ресурсів. Фаза розробки виконується за каскадною моделлю з виходом – дієвим (прототипом) продукту.

Одним з можливих підходів до розробки ПЗ в рамках спіральної моделі ЖЦ, отримала останнім часом широкого поширення методологія швидкої розробки застосунків ***Rapid Application Development (RAD)***.

Під цим терміном зазвичай розуміється процес розробки застосунків, що сповідує наступні принципи:

– невелика команда програмістів (від 2 до 10 осіб);

– короткий, але ретельно пророблений виробничий графік (від 2 до 6 місяців);

– повторюваний цикл, при якому розробники, у міру того, як застосунок починає набувати форму, запитують і реалізують в застосунку вимоги, отримані через взаємодію із замовником.



Рис. 1.17. Принципи швидкої розробки застосунків (RAD)

ЖЦ ПЗ за методологією RAD складається з чотирьох фаз:

– фаза визначення вимог і аналізу;
– фаза проєктування;
– фаза реалізації;
– фаза впровадження.

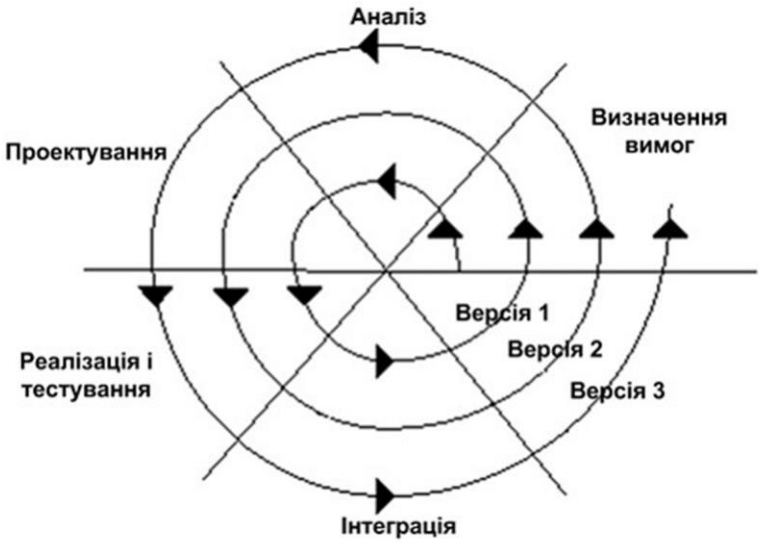


Рис. 1.18. Модель швидкої розробки додатків (RAD)

Після проходження кожної фази створюється нова версія продукту – прототип (версія 1, версія 2, версія 3).

Кожен виток спіралі відповідає створенню фрагмента або версії ПЗ, на ньому уточнюються цілі і характеристики проєкту, визначається його якість і плануються роботи наступного витка спіралі. Таким чином, поглиблюються і послідовно конкретизуються деталі проєкту і в результаті вибирається обґрунтований варіант, який доводиться до реалізації.

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При

ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання – якнайшвидше показати користувачам системи працездатний продукт, тим самим, активізуючи процес уточнення і доповнення вимог. Основна проблема спірального циклу – визначення моменту переходу на наступний етап. Для її вирішення необхідно ввести тимчасові обмеження на кожен з етапів ЖЦ ПЗ. Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота закінчена. План складається на основі статистичних даних, отриманих в попередніх проєктах, і особистого досвіду розробників.

Спіральна модель (по відношенню до каскадної) має наступні очевидні переваги:

– Більш ретельне проєктування (кілька початкових ітерацій) з оцінкою результатів проєктування, що дозволяє виявити помилки проєктування на більш ранніх стадіях.

– Поетапне уточнення вимог в процесі виконання ітерацій, що дозволяє більш точно задовольнити вимогам замовника.

– Участь замовника в виконанні проєкту з використанням прототипів програми. Замовник бачить, що і як створюється, не висуває необгрунтованих вимог, оцінює реальні обсяги фінансування.

– Планування і управління ризиками при переході на наступні ітерації дозволяє розумно планувати використання ресурсів і обґрунтовувати фінансування робіт.

– Можливість розробки складного проєкту «по частинах», виділяючи на перших етапах найбільш значущі вимоги.

Основні недоліки спіральної моделі пов’язані з її складністю:

– Складність аналізу та оцінки ризиків при виборі варіантів.

– Складність підтримки версій продукту (зберігання версій, повернення до попередніх версій, комбінація версій).

– Складність оцінки точки переходу на наступний цикл.

– Нескінченність моделі – на кожному витку замовник може висувати нові вимоги, які призводять до необхідності наступного циклу розробки.

Спіральну модель доцільно застосовувати при наступних умовах:

– Коли користувачі не впевнені в своїх потребах або коли вимоги занадто складні і можуть змінюватися в процесі виконання проєкту і необхідно робити прототипування для аналізу і оцінки вимог.

– Коли досягнення успіху не гарантовано і необхідна оцінка ризиків продовження проєкту.

– Коли проєкт є складним, дорогим і обгрунтування його фінансування можливо тільки в процесі його виконання.

– Коли мова йде про застосування нових технологій, що пов’язано з ризиком їх освоєння і досягнення очікуваного результату.

– При виконанні дуже великих проєктів, які в силу обмеженості ресурсів можна робити тільки по частинах.

Каскадна і спіральна моделі встановлюють деякі принципи організації ЖЦ створення програмного продукту. Кожна з них має переваги, недоліки і галузі застосування. Каскадна модель проста, але може бути застосована в разі, коли вимоги відомі і змінюватиась не будуть. Спіральна модель враховує такі важливі показники проєкту як змінність вимог, неможливість оцінити заздалегідь обсяг фінансування, ризики виконання проєкту. Водночас спіральна модель складна і вимагає великих витрат на супровід.

Існують деякі інші типи моделей, які можна розглядати як «проміжні» між каскадною і спіральною моделями. Ці моделі використовують окремі переваги каскадної і спіральної моделей і досягають успіху для визначених типів завдань.

2. Ітераційна та V-подібна моделі

Ітераційна модель життєвого циклу є розвитком класичної каскадної моделі і передбачає можливість повернення на раніше виконані етапи.

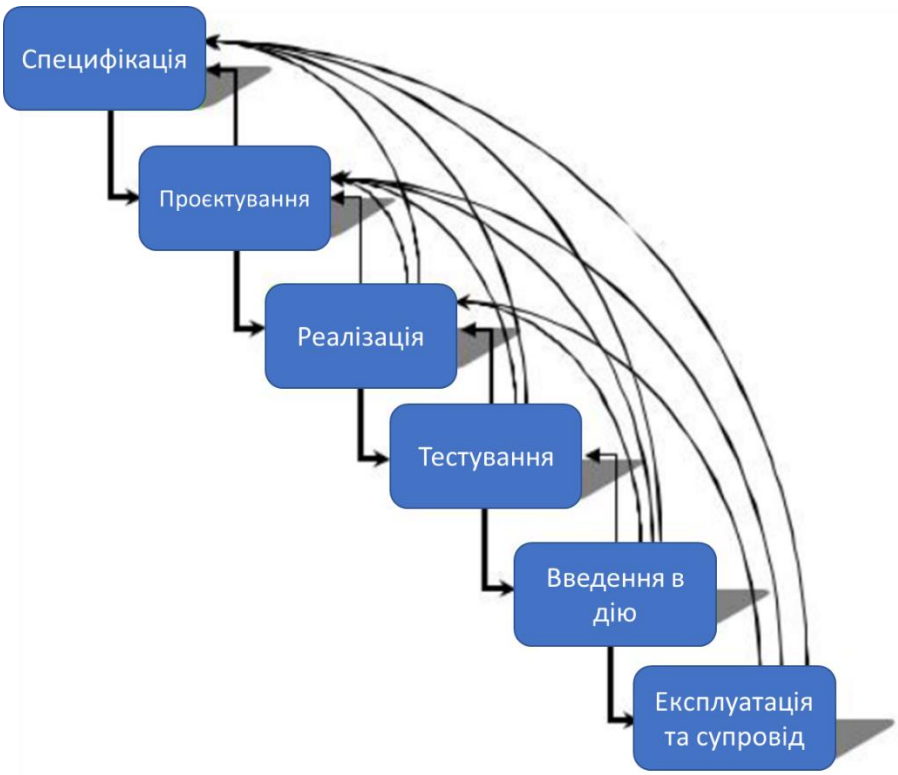


Рис. 1.19. Ітераційна модель ЖЦ ПЗ

Причинами повернень в класичній ітераційній моделі є виявлення помилок, усунення яких і вимагає повернення на попередні етапи в залежності від типу (природи) помилки – помилки кодування, проєктування, специфікації або визначення вимог. Реально ітераційна модель є більш життєвою, ніж класична (сувора) каскадна модель, тому що створення ПЗ завжди пов'язане з усуненням помилок.

Серед особливостей доцільно навести те, що стадії повторюються неодноразаво. Все інше – так само як у класичній каскадній моделі.

Недоліками є те, що:

– система часто погано структурована, проєкт «непрозорий»;

– після уточнення вимог відкидається частина раніше виконаної роботи;

– потрібні засоби для швидкого розроблення.

Ітераційна модель підходить для малих та середніх проєктів.

V-подібна модель ЖЦ ПЗ була створена як ітераційний різновид каскадної моделі.

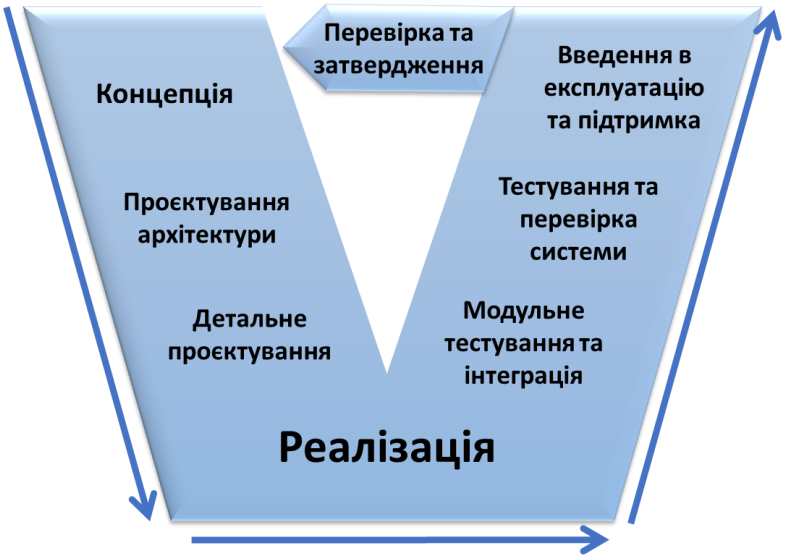


Рис. 1.20. V-подібна модель ЖЦ ПЗ

Мета ітерацій в цій моделі є забезпечення процесу тестування. В моделі особливе значення надається діям, спрямованим на верифікацію й атестацію продукту. Тестування продукції обговорюється, проєктується і планується на ранніх етапах життєвого циклу розробки. План приймання кінцевого програмного продукту замовником розробляється на етапі планування, а планування решти системи – на фазах аналізу, розробки проєкту тощо. Кожна наступна фаза починається лише по завершенню попередньої. Тестування здійснюється після того, як продукт створено. Виявлені помилки спричинять повернення на попередні етапи життєвого циклу.

3. Інкрементна модель

Інкрементна (покрокова) розробка являє собою процес поетапної реалізації всієї системи і поетапного нарощування (збільшення) функціональних можливостей. На першому кроці необхідний повний заздалегідь сформульований набір вимог, які діляться за певною ознакою на частини. Далі вибирається перша група вимог і виконується повний прохід по каскадній моделі. Після того, як перший варіант системи, що виконує першу групу вимог здано замовникові, розробники переходять до наступного кроку (другого інкременту) по розробці варіанту, що виконує другу групу вимог.

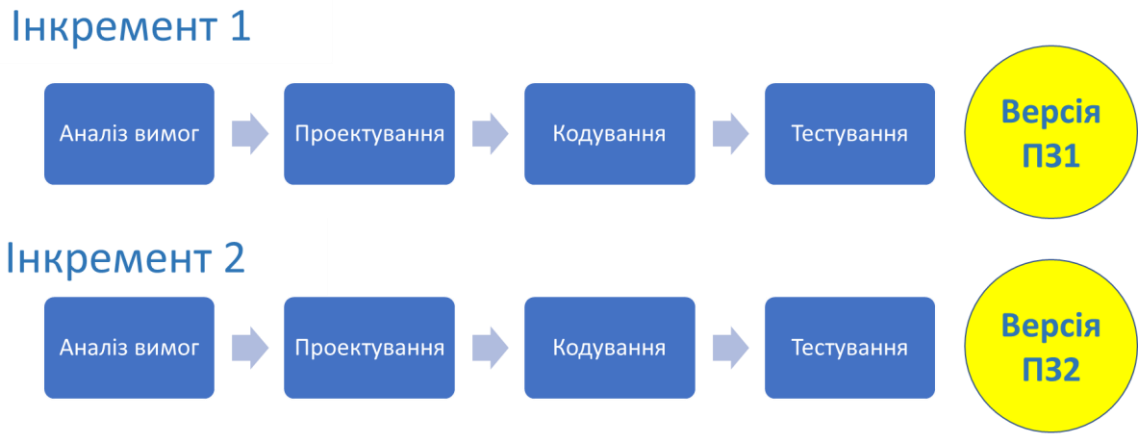


Рис. 1.21. Інкрементна модель ЖЦ ПЗ

Особливістю інкрементної моделі є розробка приймальних тестів на етапі аналізу вимог, що спрощує приймання варіанту замовником і встановлює чіткі цілі розробки чергового варіанту системи.

Інкрементна модель особливо ефективна в разі, коли задача розбивається на кілька відносно незалежних підзадач (розробка підсистем «Зарплата », «Бухгалтерія», «Склад», «Постачальники»). Крім того, інкрементна модель для внутрішньої ітерації може використовувати не тільки каскадну, а й інші типи моделей.

4. Промислові технології створення ПЗ

В даний час широке застосування отримують так звані промислові технології створення програмних продуктів. Ці технології були розроблені фірмами, які нагромадили великий досвід створення ПЗ. Технології представлені описами принципів, методів, застосовуваних процесів і операцій. Такі технології, як правило, підтримуються набором CASE-засобів, охоплюють всі етапи ЖЦ продукту і успішно застосовуються для вирішення практичних завдань.

Розглянемо особливості моделей ЖЦ трьох найбільш відомих промислових технологій. Кожна з цих технологій має свої особливості організації моделі ЖЦ створення продукту.

Microsoft Solution Framework (MSF)

Це методологія розробки програмного забезпечення, запропонована корпорацією Microsoft і призначена для вирішення широкого кола завдань. Технологія масштабована, тобто

настроюється на рішення задач будь-якої складності колективом будь-якої чисельності.

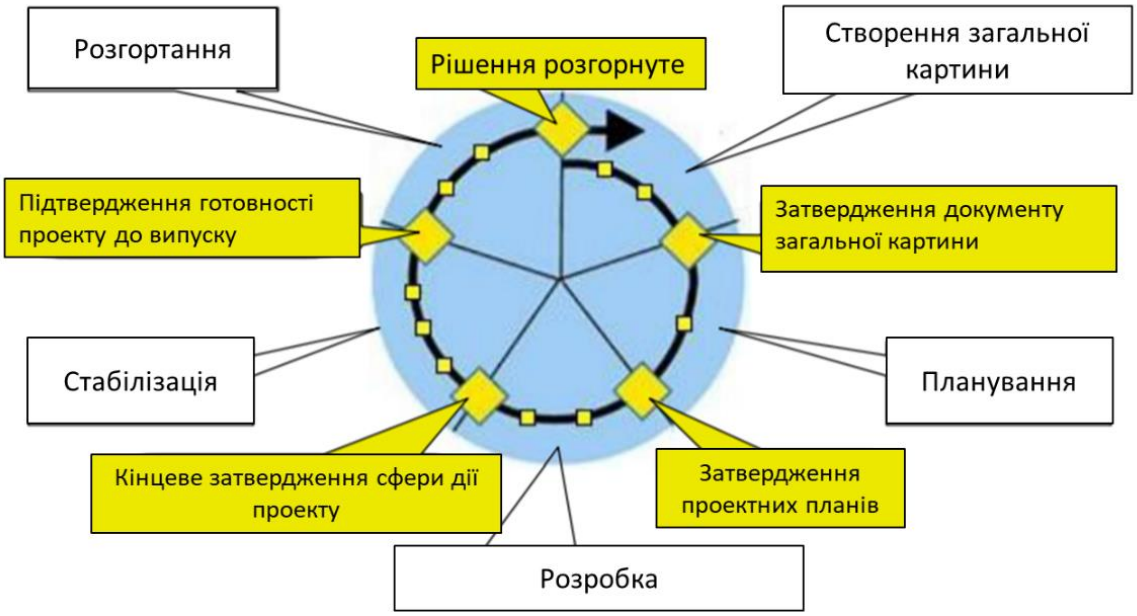


Рис. 1.22. Модель ЖЦ ПЗ Microsoft Solution Framework, MSF

Одна з особливостей технології MSF складається в тому, що вона орієнтована не просто на створення програмного продукту, що задовольняє певним вимогам, а на пошук рішення проблем, що стоять перед замовником. Різниця полягає в тому, що вимоги, які перераховуються замовником є проявами деяких глибших проблем і неточність, неповнота, зміна вимог в процесі розробки – наслідок не зовсім чіткого розуміння проблем. Тому, в технології MSF велика увага приділяється аналізу проблем замовника і розробці варіантів системи для пошуку вирішення цих проблем.

Модель життєвого циклу MSF є деяким гібридом каскадної і спіральної моделей, поєднуючи простоту управління каскадної моделі з гнучкістю спіральної.

Модель життєвого циклу MSF орієнтована на «віхи» (milestones) – ключові точки проєкту, що характеризують досягнення якого-небудь істотного результату. Цей результат може бути оцінений і проаналізований, що має на увазі відповідь на питання: «А чи досягли ми мети, поставленої на цьому кроці?». У моделі передбачається наявність основних віх (завершення головних фаз моделі) і проміжних, що відбивають внутрішні етапи головних фаз.

Rational Unified Process (RUP)

Це технологія розробки ПЗ, заснована на досвіді корпорації Rational, яка довгий час успішно займалася створенням CASE-засобів, що застосовуються на різних етапах ЖЦ продукту від аналізу до тестування і документування. Аналогічно MSF, RUP універсальна, масштабована і настроюється на застосування в конкретних умовах.

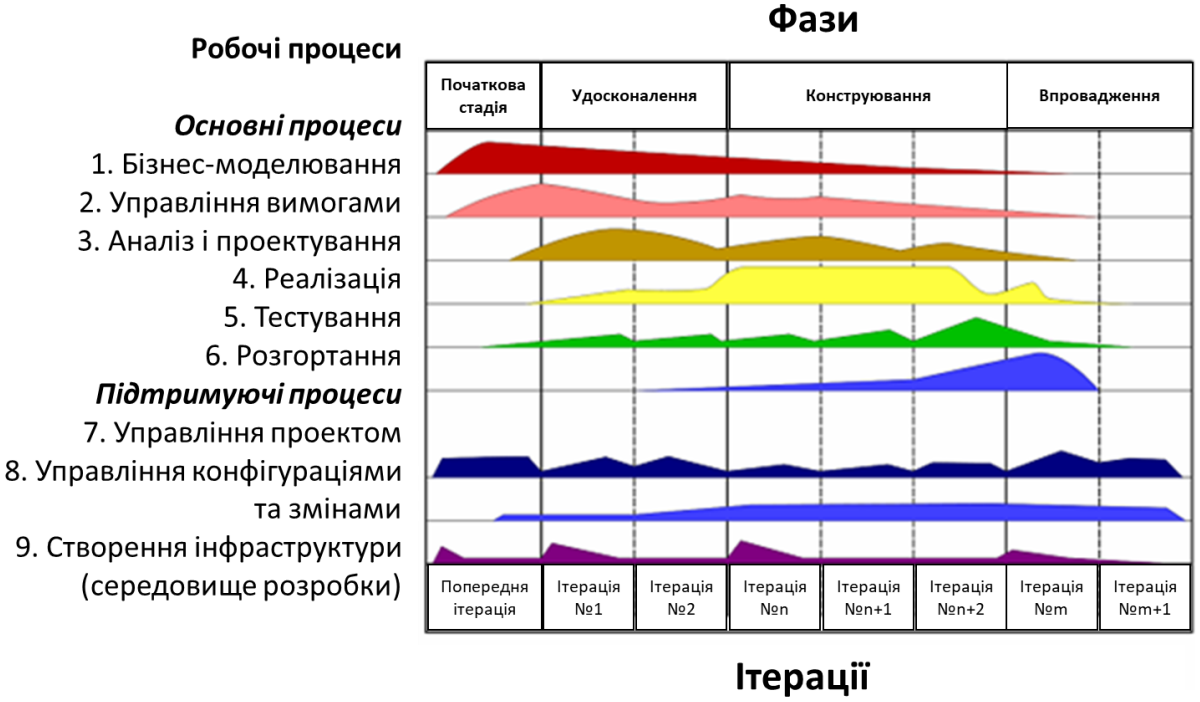


Рис. 1.23. Модель ЖЦ ПЗ Rational Unified Process, RUP

Модель ЖЦ ПЗ RUP є досить складною, детально відпрацьованою ітеративно-інкрементною моделлю з елементами каскадної моделі. У моделі RUP виділяються 4 основні фази (початкова стадія, удосконалення, конструювання, впровадження), 9 видів діяльності (процесів), які розподілені на основні процеси та підтримуючі. Крім того, в моделі описується ряд практик, які слід застосовувати або керуватися для успішного виконання проєкту. RUP орієнтований на поетапне моделювання створюваного продукту за допомогою мови UML.

Розглянемо основні фази RUP:

1. ***Фаза початку проєкту*** (Inception). Визначаються основні цілі проєкту, бюджет проєкту, основні засоби його виконання – технології, інструменти, ключовий персонал, складаються попередні плани проєкта. Основна мета цієї фази –

досягти компромісу між усіма зацікавленими особами щодо завдань проєкту.

2. ***Фаза удосконалення*** (Elaboration). Основна мета цієї фази – на базі основних, найбільш істотних вимог розробити стабільну базову архітектуру продукту, яка дозволяє вирішувати поставлені перед системою завдання і в подальшому використовується як основа розробки системи.

3. ***Фаза конструювання*** (Construction). Основна мета цієї фази – детальне прояснення вимог і розробка системи, що задовольняє цим вимогам, на основі спроєктованої раніше архітектури.

4. ***Фаза впровадження*** (Transition). Мета фази – зробити систему повністю доступною кінцевим користувачам. Тут відбувається остаточне розгортання системи в її робочому середовищі, доопрацювання дрібних деталей під потреби користувачів.

У рамках кожної фази можливо проведення кількох інтерацій, кількість яких визначає складність виконання проєкту.

Робочі процеси RUP поділяютсья на шість основних та три підтримуючі. До *основних* процесів відносяться:

1. ***Моделювання предметної галузі*** (бізнес-моделювання, Business Modeling). Мета цієї діяльності – зрозуміти бізнес-контекст, в якому повинна буде працювати система (і переконатися, що всі зацікавлені особи розуміють її однаково), зрозуміти можливі проблеми, оцінити можливі їх рішення і їх наслідки для бізнесу організації, в якій буде працювати система.

2. ***Визначення вимог*** (Requirements). Мета – зрозуміти, що повинна робити система, визначити межі системи і основу для планування проєкту та оцінок ресурсозатрат в ньому.

3. ***Аналіз і проєктування*** (Analysis and Design). Вироблення архітектури системи на основі ключових вимог, створення проєктної моделі, представленої у вигляді діаграм UML, що описують продукт з різних точок зору.

4. ***Реалізація*** (Implementation). Розробка вихідного коду, компонент системи, тестування і інтегрування компонент.

5. ***Тестування*** (Test). Загальна оцінка дефектів продукту, його якість загалом; оцінка ступеня відповідності вихідним вимогам.

6. ***Розгортання*** (Deployment). Мета – розгорнути систему в її робочому оточенні і оцінити її працездатність.

До підтримуючих процесів відносять:

7. ***Управління проєктом*** (Project Management). Включає планування, управління персоналом, забезпечення зв'язків з іншими зацікавленими особами, управління ризиками, відстеження поточного стану проєкту.

8. ***Управління конфігураціями і змінами*** (Configuration and Change Management). Визначення елементів, що підлягають зберіганню і правил побудови з них узгоджених конфігурацій, підтримку цілісності поточного стану системи, перевірка узгодженості внесених змін.

9. ***Управління середовищем проєкту*** – створення інфраструктури (Environment). Налаштування процесу під конкретний проєкт, вибір і зміна технологій та інструментів, які використовуються в проєкті.

Extreme Programming (XP)

Це методологія швидкої розробки ПЗ для вирішення відносно невеликих завдань, відносно невеликими колективами професійних розробників в умовах жорстко обмеженого часу.

Екстремальне програмування є прикладом так званого методу «живої» розробки.

Модель життєвого циклу XP є ітераційно-інкрементною моделлю швидкого створення (і модифікації) прототипів продукту, що задовольняють чергову вимогу (requirement).

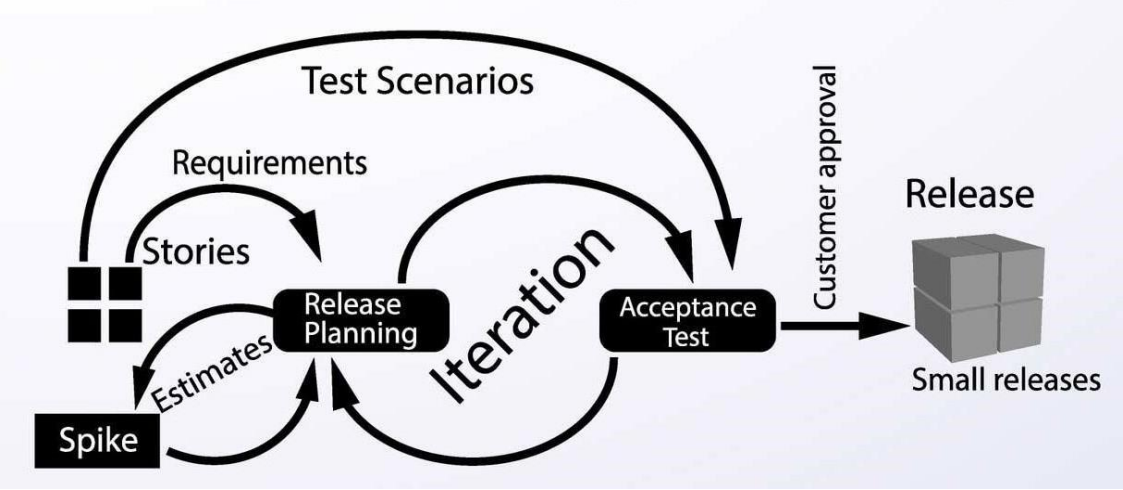


Рис. 1.24. Модель ЖЦ ПЗ Extreme Programming, XP

Основними фазами моделі можна вважати:

1. «Вкид» архітектури (spike) – початковий етап проєкту, на якому створюється бачення продукту, приймаються основні рішення по архітектурі і вибору технологій розробки ПЗ. Результатом початкового етапу є метафора (metaphor) системи, яка в досить простій і зрозумілій для команди розробників вигляді повинна описувати основний механізм роботи системи.

2. Історії використання (Stories) – етап збору вимог, що записуються на спеціальних картках у вигляді сценаріїв виконання окремих функцій. Історії використання є вимогами для планування чергової версії і одночасної розробки приймальних тестів (Acceptance tests) для її перевірки.

3. Планування версії (Release). Проводиться на зборах за участю замовника шляхом вибору історій використання, які увійдуть в наступну версію. Одночасно приймаються рішення, пов'язані з реалізацією версії. Мета планування – отримання оцінок того, що і як можна зробити за 1-3 тижні створення наступної версії продукту.

4. Розробка (Iteration) проводиться відповідно до плану і включає тільки ті функції, які були відібрані на етапі планування.

5. Тестування проводиться за участю замовника, який бере участь в складанні тестів.

6. Випуск релізу – розроблена версія передається замовнику для використання або для бета-тестування.

По завершенню циклу робиться перехід на наступну ітерацію розробки.

Особливості моделі ЖЦ ПЗ Extreme Programming відображені в наступних принципах цього методу. Перш за все, це принципи «живої» розробки ПО, зафіксовані в маніфесті «живої» розробки:

– Люди та їх спілкування більш важливі, ніж процеси і інструменти.

– Працююча програма важливіша, ніж вичерпна документація.

– Співпраця із замовником більш важливо, ніж обговорення деталей контракту.

– Відпрацювання змін важливіша, ніж проходження планам.

Крім того, в XP є кілька правил (технік), що характеризують особливості моделі його життєвого циклу.



Рис. 1.25. Правила розробки ПЗ відповідно до XP

Живе планування (planning game) – якомога швидше визначити обсяг робіт, який потрібно зробити до наступної версії ПЗ. Рішення приймається на основі, в першу чергу, бізнес-пріоритетів замовника і, по-друге, технічних оцінок. Плани змінюються, як тільки вони починають розходиться з дійсністю або побажаннями замовника.

Швидка зміна версій (small releases) – перша працююча версія повинна з'явитися якомога швидше, і тут же повинна почати використовуватися.

Наступні версії готуються через досить короткі проміжки часу.

Прості проєктні рішення (simple design) – в кожен момент часу система повинна бути сконструйована настільки просто, наскільки це можливо. Нові функції додаються тільки після цілеспрямованого прохання про це (якщо замовник конкретно каже – це потрібно). Вся зайва складність видаляється, як тільки її виявляють в системі.

Розробка на основі тестування (test-driven development) – спочатку пишуться тести, потім реалізуються модулі так, щоб тести спрацьовували. Замовники заздалегідь пишуть тести, що демонструють основні можливості системи, щоб можна було побачити, що система дійсно заробила.

Постійна переробка системи (refactoring) – для усунення зайвої складності, збільшення зрозумілості коду, підвищення його гнучкості. При цьому перевага віддається більш елегантним і гнучким рішенням, в порівнянні з тими, які просто дають потрібний результат.

Парне програмування (pair programming) – весь код пишеться двома програмістами на одному комп'ютері, що

підвищує його якість (відсутність помилок, зрозумілість, читабельність тощо).

Постійна інтеграція (continuous integration) – система збирається і проходить інтеграційне тестування якомога частіше, по кілька разів на день, кожного разу, коли пара програмістів закінчує реалізацію чергової функції.

40-годинний робочий тиждень – понаднормова робота розглядається як ознака великих проблем в проєкті. Не допускається понаднормова робота 2 тижні поспіль – це виснажує програмістів і робить їх роботу значно менш продуктивною.

Контрольні запитання

1. Які бувають моделі ЖЦ ПЗ?
2. Яка особливість спіральної моделі ЖЦ ПЗ?
3. В чому основна суть моделі швидкої розробки застосунків?
4. Які переваги і недоліки спіральної моделі ЖЦ ПЗ?
5. Коли доцільно обирати спіральну модель ЖЦ ПЗ?
6. В чому особливість ітераційної моделі ЖЦ ПЗ?
7. В чому особливість V-подібної моделі ЖЦ ПЗ?
8. В чому суть інкрементної моделі ЖЦ ПЗ?
9. Які є промислові технології створення ПЗ?
10. Які процеси задіяні в моделі Rational Unified Process (RUP)?
11. В чому особливість екстремального програмування?
12. На які позиції вказує маніфест екстремального програмування?

Тема 1.5. CASE-засоби та вимоги до програмного забезпечення

План

1. Поняття та класифікація CASE-засобів.
2. Вимоги до ПЗ.
3. Професійні і етичні вимоги.
4. Кодекс етики IEEE-CS / ACM.

1. Поняття та класифікація CASE-засобу

Проаналізуємо поняття «кейс» (від англ. case – коробка, чохол, ящик, міркування, аргументація) з точки зору мовної практики. Досить часто люди використовують слово «кейс» для позначення речей, які знаходяться в певній ємності – папка, сумка, скринька, коробка тощо. Говорячи це слово, люди розповідають, як мінімум, про дві речі.

В більш сучасному значенні «кейс» це випадок з практики, який наочно демонструє якусь теорію. Розгляд та вивчення випадків з практики (кейсів) – важлива технологія для запобігання помилок та удосконалення процесу.

І одне, і інше твердження є справедливим, коли мова йде про CASE-засоби. В першу чергу CASE (від англ. Computer Aided System Engineering) – це один з напрямків інженерії ПЗ, який передбачає використання різних методологій аналізу, проєктування, розробки і супроводу складних систем програмного забезпечення, яка підтримується комплексом взаємопов’язаних засобів автоматизації

Коли мова йде про CASE-засоби, то це набір інструментів і методів програмної інженерії для проєктування ПЗ, що допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів.

За допомогою CASE автоматизують процеси проєктування інтерфейсів, документування й генерування структурованого коду бажаною мовою програмування.

CASE-засоби – це інструментарій для системних аналітиків, розробників і програмістів, який змінює папір і олівець на комп’ютер для автоматизації процесу проєктування і розробки ПЗ.

Таблиця 1.2

Відмінності CASE від традиційної розробки

Традиційна розробка	CASE-розробка
Основні зусилля – на кодування і тестування	Основні зусилля – на аналіз і проєктування
«Паперові» специфікації	Швидке ітеративне прототипування
Ручне кодування	Автоматична кодогенерацію
Ручне документування	Автоматична генерація документації
Тестування кодів	Автоматичний контроль проєкту
Супровід кодів	Супровід специфікацій проєктування

Більшість CASE-засобів засновані на парадигмі методологія / метод / нотація / засіб.

Методологія визначає керівні вказівки для оцінки і вибору проєкту ПЗ, кроки роботи і їх послідовність, а також правила розподілу і призначення методів.

Метод – це систематична процедура або техніка генерації описів компонент ПЗ (наприклад, проєктування потоків і структур даних).

Нотації призначені для опису структури системи, елементів даних, етапів обробки і включають графи, діаграми, таблиці, блок-схеми, формальні і природні мови.

Засоби – інструментарій для підтримки і посилення методів. Ці інструменти підтримують роботу користувачів при створенні і редагуванні графічного проєкту в інтерактивному режимі, вони сприяють організації проєкту у вигляді ієрархії рівнів абстракції, виконують перевірки відповідності компонентів.

При використання CASE-засобів зазнає змін модель ЖЦ ПЗ. CASE-технології пропонують новий, заснований на автоматизації підхід до концепції ЖЦ ПЗ – змінюються всі фази ЖЦ, при цьому найбільші зміни стосуються фаз аналізу і проєктування.



Рис. 1.26. Модель ЖЦ ПЗ з використанням CASE-технологій

На першому етапі створюється прототип програмної системи. Процес прототипування пов'язаний з потребою виявлення вимог клієнта до функціоналу його продукту. Основне, що потрібно донести клієнту на цьому етапі – це те, що остаточний продукт може дуже сильно відрізнятися візуально.

Власне, прототип – це готовий до безпосередніх випробувань реальний, наочний, предметний макет продукту, комплексного сервісу чи його частини.



Рис. 1.27. Шлях від прототипу до готового продукту

Прототипи можна відрізняти за складом та комплексністю, але всі вони мають одну особливість – вони дозволяють протестувати товар чи послугу майже нічим ще не ризикуючи.

Прототипування дозволяє творчо та наявно висловлювати свої ідеї і надає можливість вчитися та знайти помилки на практиці.

CASE-засоби можуть бути **класифіковані** за кількома ознаками:

За рівнем застосування:

– Upper CASE-засоби аналізу вимог.
– Middle CASE-засоби проєктування.
– Low CASE-засоби розробки застосунків.

Спеціалізовані:

– Засоби проєктування баз даних.
– Засоби реінжинірингу (відновлення) моделі.

Допоміжні:

– Планування та управління проєктом.
– Конфігураційного управління.
– Тестування.

Інтегровані CASE-засоби охоплюють всі етапи і процеси створення ПЗ, починаючи від аналізу вимог до тестування та випуску документації. Інтегровані CASE виступають, як правило, у вигляді набору узгоджених за інтерфейсом коштів, призначених для підтримки окремих етапів процесу.

2. Вимоги до програмного забезпечення

Перш за все, хороша програма повинна робити те, що очікує від неї замовник – тобто задовольняти вимогам замовника. Такі вимоги називають ***функціональними***.

Але крім функціональних вимог, існує ще ряд загальних характеристик, яким в тій чи іншій мірі повинна володіти кожна програма. Ці характеристики прийнято називати ***нефункціональними вимогами***.

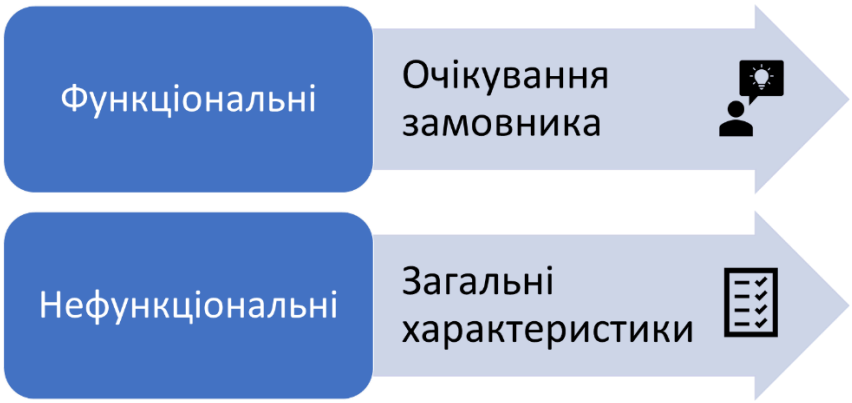


Рис. 1.28. Вимоги до ПЗ

Проаналізуємо загальні характеристики, які висуваються до ПЗ. До них відносять наступні:

– Можливість супроводу (Maintainability)

– Надійність (Dependability)
– Ефективність (Efficiency)
– Зручність використання (Usability)

Можливість супроводу означає, що програма повинна бути написана з розрахунком на подальший розвиток. Це критична властивість системи, тому що зміни ПЗ неминучі внаслідок зміни бізнесу. Супровід програми виконують, як правило, не ті люди, які її розробляли. Можливість супроводу включає такі елементи як наявність і зрозумілість проєктної документації, відповідність проєктної документації вихідному коду, зрозумілість вихідного коду, простота змін вихідного коду, простота додавання нових функцій.

Надійність ПЗ включає такі елементи як:

– Відмовостійкість – можливість відновлення програм і даних у разі відмови в роботі
– Безпечність – збій та помилки у програмі не повинні призводити призводити до небезпечних наслідків (аварій)
– Захищеність від випадкових або навмисних зовнішніх впливів (захист від вірусів, спами, захист від дурня).

Ефективність означає, що ПЗ не повинне даремно витрачати системні ресурси, такі як пам’ять, процесорний час, канали зв’язку. Тому ефективність ПЗ оцінюється такими показниками як час виконання коду, завантаженість процесора, обсяг необхідної пам’яті, час виклику тощо.

Зручність використання (юзабіліті) вказує на те, що ПЗ повинне бути легким у використанні, зокрема саме для тих типів користувачів, для яких розроблено застосунок. Дана вимога включає в себе вимоги до інтерфейсу користувача та адекватну документацію. Причому, користувацький інтерфейс повинен бути не просто інтуїтивно зрозумілим, а професійно зрозумілим для користувача.

Варто відмітити, що реалізація нефункціональних вимог часто вимагає більших затрат, ніж функціональних. Так, супровід (можливість супроводу) вимагає значних зусиль щодо підтримки відповідності проєкту вихідного коду та застосування спеціальних методів створення модифікованих програм. Надійність – додаткових засобів відновлення системи при відмовах (збоях). Ефективність – пошук спеціальних

архітектурних рішень та оптимізації коду. Зручність – проєктування не «інтуїтивно» зрозумілого, а професійно зрозумілого інтерфейсу користувача.

3. Професійні і етичні вимоги

Розвиток засобів обчислювальної техніки, розробка комунікативних технологій та програмних систем з кожним роком здійснює значний вплив на суспільство. Роль спеціалістів в галузі розробки ПЗ при цьому весь час зростає. Вони працюють у визначеному правовому та соціальному оточенні, знаходяться під дією міжнародних, національних та місцевих законодавчих органів.

Зрозуміло, що програмісти, як і фахівці інших професій, повинні бути чесними та порядними людьми. Але разом з тим, програмісти не можуть керуватися лише моральними нормами або юридичними обмеженнями, оскільки вони зазвичай зв'язані з більш тонкими професійними зобов'язаннями:

Конфіденційність – програмні фахівці повинні поважати конфіденційність своїх роботодавців або замовників незалежно від того, підписували вони відповідну угоду чи ні.

Компетентність – програмний фахівець не повинен завищувати свій справжній рівень компетентності і не повинен свідомо братися за роботу, яка цього рівня не відповідає.

Захист інтелектуальної власності – фахівець повинен дотримуватися законодавства і принципів захисту інтелектуальної власності при використанні чужої інтелектуальної власності. Крім того, він повинен захищати інтелектуальну власність роботодавця і клієнта. Потрібно пам'ятати про те, що створювана ним інтелектуальна власність є власністю роботодавця або клієнта.

Зловживання комп'ютером – програмний фахівець не повинен зловживати комп'ютерними ресурсами роботодавця або замовника; під зловживаннями розуміємо широкий спектр – від ігор в комп'ютерні ігри на робочому місці до поширення вірусів.

4. Кодекс етики IEEE-CS / ACM

У розробці етичних зобов'язань провідну роль відіграють професійні спільноти. Такі організації як Асоціація з

обчислювальної техніки (ACM), Інститут інженерів з електротехніки та електроніки (IEEE) і Британська комп'ютерна спільнота (BCS) спільно розробили і опублікували Кодекс етики та професійної практики програмної інженерії (IEEE-CS / ACM Software Engineering Code of Ethics and Professional Practices). Члени цих організацій приймають зобов'язання дотримуватися цього кодексу в момент вступу в організацію.

Кодекс містить вісім принципів, пов'язаних з поведінкою і рішеннями, прийнятими професійними програмістами, включаючи практиків, викладачів, менеджерів і керівників вищої ланки. Кодекс поширюється також на студентів і «новачків», які вивчають дану професію.

Розглянемо 8 принципів кодексу етики IEEE-CS / ACM:

1. Суспільство – програмні інженери будуть діяти відповідно до суспільних інтересів.

2. Клієнт і роботодавець – програмні інженери будуть діяти в інтересах клієнтів і роботодавця, відповідно суспільним інтересам.

3. Продукт – програмні інженери будуть домагатися, щоб вироблені ними продукти і їх модифікації відповідав найвищим професійним стандартам.

4. Судження – програмні інженери будуть домагатися чесності і незалежності в своїх професійних судженнях.

5. Менеджмент – менеджери і лідери програмних інженерів будуть керуватися етичним підходом до керівництва розробкою і супроводом ПЗ, а також будуть просувати і розвивати цей підхід.

6. Професія – програмні інженери будуть покращувати цілісність і репутацію своєї професії відповідно до інтересів суспільства.

4. Колеги – програмні інженери будуть чесними по відношенню до своїх колег і будуть всіляко їх підтримувати.

8. Особистість – програмні інженери протягом всього свого життя навчатимуться практиці свою професію і просуватимуть етичний підхід до практики своєї професії.

Контрольні запитання

1. Що таке CASE-засоби?
2. Які є типові інструменти CASE-розробки?

3. Як класифікують Case-засоби?

4. Яка різниця між функціональними і нефункціональними вимогами до ПЗ?

5. Які є нефункціональні вимоги до ПЗ?

6. Що таке професійні та етичні вимоги до спеціаліста в галузі розробки ПЗ?

Тема 1.6. Управління програмним проєктом

План

1. Проєкт і управління проєктом.
2. Менеджер IT-проєкту та його компетентності.
3. Команда IT-проєкту та рольова модель.
4. Моделі організації команд.
5. Спілкування в команді.

1. Проєкт і управління проєктом

Розглянемо поняття «управління».



Рис. 1.29. Порівняння різних понять терміну «управління»

У різних джерелах можна знайти різні визначення поняття «управління»:

– Елемент, функція організованих систем різної природи (біологічних, соціальних, технічних), що забезпечує збереження їх певної структури, підтримання режиму діяльності, реалізацію їх програм і цілей.

– Керівництво, спрямування діяльності інших.

– Зміна стану об’єкта, системи або процесу, що веде до досягнення поставленої мети (словник з кібернетики).

З точки зору останнього (найбільш прийнятного для управління в галузі інженерії ПЗ) визначення, істотним є:

– наявність вимірюваної мети управління (функції мети);

– наявність (можливість) управління (керуючого впливу, що впливає на зміну функції мети);

– наявність вимірювань стану об'єкта або процесу;

– обмеженість управління.

Слово «проєкт» походить від латинського projectus, що означає «кинутий вперед».

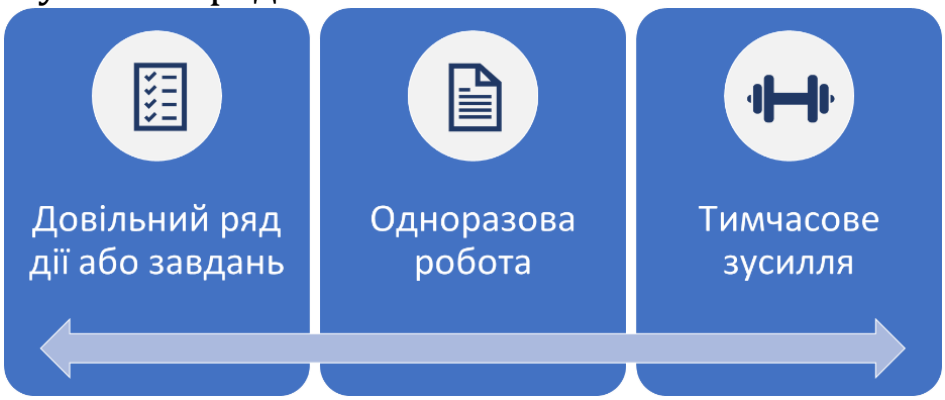


Рис. 1.30. Зміст поняття «проєкт»

Останнім часом слово «проєкт» вживається досить часто (і часто просто так): проєкт озеленення вулиць міста, проєкт підвищення кваліфікації співробітників, проєкт реорганізації діяльності фірми тощо. Відомі кілька визначень проєкту:

Проєкт – це довільний ряд дій або завдань, що має певну мету, яка буде досягнута в рамках виконання деяких завдань, що характеризуються певними датами початку і закінчення, межами фінансування і ресурсами (Г.Керцнер).

Проєкт – одноразова робота, яка має певні дати початку і закінчення, чітко визначені цілі, можливості і, як правило, бюджет (Д. Льюїс).

Проєкт – тимчасове зусилля, що застосовується для того, щоб створити унікальний продукт або послугу з певною датою початку і закінчення дії, що відрізняється від довготривалих, повторних дій і вимагає прогресивного вдосконалення характеристик.

У цих визначеннях у тій чи іншій мірі висвітлюють визначальні характеристики проєкту.



Рис. 1.31. Характеристики проєкту

Мета проєкту. Наявність чітко вираженого кінцевого результату, продукції, що визначаються в термінах витрат, якості та часу реалізації.

Унікальність. Проєкт – це разове починання, яке не буде повторюватися. Навіть «повторювані» проєкти, наприклад, з будівництва ще одного підприємства за тією ж проєктною документацією, значно відрізняються один від одного використовуваними ресурсами і середовищем реалізації.

Обмеженість в часі. Проєкт має початок і кінець. Для його реалізація необхідна тимчасова концентрація ресурсів. Якщо така потреба зникає, ресурси використовуються на інші цілі.

Обмеженість ресурсів, що виділяються на виконання проєкту (фінансових, людських, матеріальних).

Складність. Для досягнення цілей проєкту необхідно вирішити безліч завдань. Зв'язки між завданнями можуть бути досить складними, особливо, якщо в проєкті багато завдань.

Невизначеність. Можливість досягнення мети в зазначені терміни з виділеними ресурсами заздалегідь не гарантована.

Передбачуваність. У міру реалізації проєкту, змінюється потреба в тих чи інших ресурсах. Ця зміна йде в певній передбачуваній послідовності, яка визначається ЖЦ проєкту. Іншими словами, проєкт – це досить складний вид діяльності, яким складно управляти в силу його унікальності і обмеженості ресурсів і часу. Ця обставина вносить в проєкт елемент невизначеності, а правильно організоване управління робить результати передбачуваними. Також потрібно пам'ятати про те, що передбачуваний – не означає успішний. Це означає – вчасно завершений (успішно) або вчасно припинений (неуспішно).

До проєктів можна віднести далеко не будь-який вид діяльності. До таких видів діяльності зазвичай відносять: програму, виконання встановленого процесу, творча задача.

Програма – широкомасштабне зусилля, спрямоване на досягнення певної комплексної мети: програма космічних досліджень, програма меліорації земель Середньої Азії. Мета програми не конкретна, терміни і ресурси не визначені.

Але програма може розбиватися на окремі конкретні цілі, для яких встановлюються терміни і виділяються ресурси. Тобто програма може розбиватися на окремі проєкти: проєкт Апполон, проєкт Карпати, проєкт будівництва каналу тощо.

Виконання встановленого процесу – діяльність, яка виконується багаторазово і постійно: конвеєрне виробництво, обробка замовлень, ведення бухгалтерії. Має конкретну мету (випуск запланованої кількості продукції, отримання встановленої прибутку, ведення звітності) і виділені ресурси, але не є унікальною або складною і не пов’язана з конкретними термінами. Управління такими повторюваними процесами відносно просте.

Зміна параметрів сталих процесів може перетворитися в проєкт (підвищення прибутковості фірми) з конкретними цілями (до 45%), термінами і виділеними ресурсами.

Творча задача (наукова чи художня). Тут є конкретна мета, унікальність і складність, але, як правило, немає обмежень за часом і ресурсами (об’єднаємо зусилля нашого колективу і доведемо теорему до Нового року). Занадто велика ступінь невизначеності.

Рішення складних наукових проблем може розбиватися на окремі дослідження (експерименти) з конкретно встановленими цілями, термінами і ресурсами.

Поняття управління проєктами

Серед різних визначень управління проєктами [27], найбільш повним слід вважати визначення, сформульоване в Зведенні знань з управління проєктами (PMBOK): «Управління проєктом (Project Management – PM) – це наука і мистецтво керівництва і координації людських і матеріальних ресурсів протягом ЖЦ проєкту шляхом застосування сучасних методів і техніки управління для досягнення визначених у проєкті результатів за

складом і обсягом робіт, вартості, часу, якості й задоволення учасників проєкту».

Управління проєктом засновано на двох китах (принципах):

Уміння – знання принципів і методів управління проєктом (планування, організація, складання графіків, контроль, управління і відстеження).

Навички – досвід в галузі управління – застосування вміння для досягнення цілей в конкретних умовах.

Хоча розробка методів і прийомів управління була розпочата ще в початку минулого століття, як дисципліна управління проєктами почала розвиватись в 50-х роках XX століття, що було викликано необхідністю координації робіт у великих проєктах щодо розробки озброєнь і освоєння космосу (США). Розроблялися методи управління великими проєктами, серед яких найбільш відомими є:

– Метод критичного шляху (Critical Path Method, CPM).

– Метод аналізу і оцінки програм (Program Evaluation and Review Technique, PERT).

60-80 рр. минулого століття характеризуються значним поширенням методів управління проєктами, створенням комп'ютерних програм на базі CPM та PERT, а також розробкою нових методів і програм правління проєктами.

З 90 рр. XX ст. головним чином завдяки зусиллям Інституту управління проєктами (Project Management Institute, PMI) управління проєктами стає професією і областю знань.

Категорії управління проєктами

Категорії – від грецького «kategoria» означає висловлювання; ознака. У філософії найбільш загальні і фундаментальні поняття, що відображають істотні, загальні властивості і відносини явищ дійсності і пізнання. Категорії утворилися як результат узагальнення історичного розвитку пізнання і практики: матерія і свідомість, простір і час, причинність, необхідність і випадковість, можливість і дійсність тощо.

Аналогічно філософським категоріям, в галузі управління проєктами існують категорії, що відображають основні поняття цієї галузі. У загальному випадку виділяють наступні групи категорій:

– Цілі, які визначаються очікуваними результатами проєкту.
– Критерії успіху і обмеження: вартість, терміни, якість.
– Основні важелі управління: ресурси (що є також обмеженням) і технології.
– Допоміжні важелі управління: контракти, організація, взаємодія, персонал.
– Невизначеність, пов'язана з ризиками виконання проєкту.

Ключовою категорією, яка бере участь в процесі управління проєктами, є обмеження. Відомий закон Лермана говорить: «Будь-яку технічну проблему можна подолати, маючи достатньо часу і грошей», але одразу йде уточнення: «Вам ніколи не буде вистачати або часу, або грошей». Якщо попросити менеджера описати, як він розуміє своє основне завдання у виконанні проєкту, то він відповість: «Забезпечити виконання робіт у термін, в межах виділених коштів, відповідно до технічного завдання».

Саме ці три моменти: ***час***, ***бюджет*** і ***якість робіт*** знаходяться під постійною увагою керівника проєкту. Їх також можна назвати основними обмеженнями, що накладаються на проєкт.

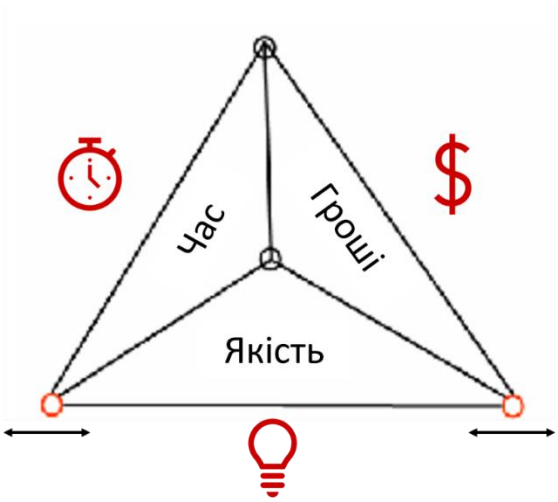


Рис. 1.32. Трикутник обмежень проєкту

Ці три основні обмеження (терміни, витрати і якість результату) взаємопов'язані.

Для ілюстрації взаємозв'язку використовують трикутник обмежень, в якому якість, час і гроші інтерпретуються площами внутрішніх трикутників. У цьому трикутнику центр і верхня вершини фіксовані, а нижні вершини можуть переміщатися. Трикутник ілюструє, що будь-яке скорочення фінансів або часу

веде до скорочення якості, а збільшення якості може бути досягнуто за рахунок збільшення фінансування або термінів.

2. Менеджер IT-проєкту та його компетентності.

Головна дійова особа проєкту – менеджер. Він повинен мати знання і навички. Хто він у програмному проєкті – програміст, проєктувальник, дизайнер? Спочатку так і було. Відігравали значну роль знання в предметній галузі (проєктування і розробка ПЗ). Але потім на перше місце стали виходити знання і навички управління.



Рис. 1.33. Галузі управлінських знань менеджера проєкту згідно з PMBOK

Міжнародний стандарт складу знань з управління проєктами PMBOK (Project Management Body of Knowledge – Зведення знань з управління проєктами), який розроблений і розвивається Інститутом проєктного менеджменту (Project Management Institute – PMI) описує зміст знань за наступними 9 галузями знань управління проєктами, кожен з яких ділиться на відповідні підрозділи:

1. Управління інтеграцією проєкту (Integration):

– Створення плану проєкту (Project Plan Development).
– Виконання плану проєкту (Project Plan Execution).
– Контроль змін у проєкті (Integrated Change Control).

2. Управління обсягом робіт (Scope):

– Ініціювання (Initiation) – формальне ухвалення рішення про початок проєкту (наступної фази проєкту).

– Планування обсягу робіт (Scope Planning) – розробка документа, що описує обсяг робіт.
– Формалізація обсягу робіт (Scope Definition) – декомпозиція всього обсягу робіт (основних необхідних результатів) на дрібні, вимірні завдання.
– Верифікація (Scope Verification) – підтвердження обсягу робіт – формальна перевірка прийнятності результатів роботи.
– Управління змінами обсягу робіт (Scope Change Control) – контроль та затвердження змін.

3. Управління часом виконання (Time):

– Визначення складу робіт (Activity Definition).
– Визначення взаємозв'язків робіт (Activity Sequencing).
– Оцінка тривалості робіт (Activity Duration Estimating).
– Складання розкладу проєкту (Schedule Development).

4. Управління вартістю (Cost):

– Планування ресурсів (Resource Planning) – які ресурси і в якій кількості потрібні для робіт.
– Оцінка вартостей (Cost Estimating) ресурсів, необхідних для робіт.
– Розробка бюджету (Cost Budgeting) – бюджетування – розподіл витрат по компонентах проєкту.
– Контроль вартості (Cost Control) – управління змінами бюджету.

5. Управління якістю (Quality):

– Планування якості (Quality Planning) – визначення стандартів якості та засобів для їх досягнення.
– Забезпечення якості процесу (Quality Assurance) – планова, регулярна оцінка виконання – перевірка виробничих процесів
– Контроль якості результатів (Quality Control) – моніторинг результатів проєкту, визначення їх відповідності стандартам, виявлення і усунення причин невідповідності якості.

6. Управління персоналом (Human Resource):

– Організаційне планування (Organizational Planning) – ідентифікація, документування, і призначення проєктних ролей, обов'язків і структури звітності.
– Підбір кадрів (Staff Acquisition) – отримання необхідних для проєкту людських ресурсів, призначення персоналу в команду проєкту.
– Розвиток команди проєкту (Team Development) – підвищення продуктивності праці: індивідуальної і команди в цілому (поліпшення взаємодії).

7. Управління комунікаціями (Communications):

– Планування взаємодії (Communications Planning) – визначення потреб учасників проєкту в інформації і планування інформаційних потоків.
– Розподіл інформації (Information Distribution) – регулярне і своєчасне забезпечення учасників проєкту необхідною інформацією.
– Оцінка виконання (Performance Reporting) – збір і поширення звітності про поточний стан проєкту, досягнутий прогрес і очікуваних результатах.
– Адміністративне завершення (Administrative Closure) - створення, поширення (знищення) інформації, необхідні для формального завершення проєкту / фази.

8. Управління ризиками (Risk):

– Планування управління ризиками (Risk Management Planning).
– Ідентифікація ризиків (Risk Identification).
– Якісний аналіз ризиків (Qualitative Risk Analysis).
– Кількісний аналіз ризиків (Quantitative Risk Analysis).
– Планування реагування на ризики (Risk Response Planning).
– Моніторинг і контроль ризиків (Risk Monitoring and Control).

9. Управління закупівлями і поставками (Procurement):

– Планування закупівель (Procurement Planning) – визначення які продуктів і послуг потрібні ззовні.

– Планування пропозицій (Solicitation Planning) – документування вимог до продуктів і послуг від зовнішніх постачальників.
– Отримання пропозицій (Solicitation).
– Вибір постачальників (Source Selection).
– Управління контрактами (Contract Administration) – регулювання відносин з постачальниками.
– Завершення контрактів (Contract Closeout) – підтвердження виконання, вирішення спорів.

SQI: 34 компетенції IT менеджера

Інститутом якості ПЗ (Software Quality Institute, SQI) розроблений керівний документ для сертифікації менеджерів програмних проєктів (SoftWare Project Management, SWPM).



Рис. 1.34. SQI: 34 компетенції IT менеджера

У цьому документі міститься список 34 компетенцій, якими повинен володіти менеджер програмного проєкту (додаток 7). Список розділений на три основні категорії: методика розробки продукту, навички управління проєктами та навички управління персоналом.

3. Команда IT-проєкту та рольова модель.

Успіх проєкту безпосередньо пов'язаний з наявними талантами у учасників, і, що ще важливо, способом, відповідно до якого керівництво використовує ці таланти в проєкті.

Управління програмним проєктом включає рішення трьох основних завдань: підбір і управління командою, вибір процесу і

вибір інструментальних засобів. Хоча всі три завдання однаково важливі для успіху проєкту, провідну роль відіграє правильний підбір і управління командою. Успіх проєкту багато в чому залежить від того, наскільки склад учасників проєкту зможе бути перетворений в команду однодумців, наскільки ця команда буде активною та ініціативною з одного боку і керованою з іншого. З безлічі моделей управління командою проєкту ми розглянемо три: рольова модель команди, моделі організації команд, спілкування в команді.

Рольова модель команди

Склад команди визначається досвідом і рівнем колективу, особливостями проєкту, застосовуваними технологіями і рівнем цих технологій. «Класичний» варіант складу команди входять такі позиції:

Менеджер проєкту – головна дійова особа, що володіє знаннями і навичками, необхідними для успішного управління проєктом. Його основні функції:

- Підбір і управління кадрами.
- Підготовка та виконання плану проєкту.
- Керівництво командою.
- Забезпечення зв'язку між підрозділами.
- Забезпечення готовності продукту.

Проєктувальник – це функція проєктування архітектури високого рівня і контролю її виконання. У невеликих командах функція розподіляється між менеджером і розробниками. У великих проєктах це може бути цілий відділ. Основними функціями проєктування є:

- Аналіз вимог.
- Розробка архітектури та основних інтерфейсів.
- Участь в плануванні проєкту.
- Контроль виконання проєкту.
- Участь в підборі кадрів.

Розробник – роль, що передбачає відповідальність за безпосереднє створення кінцевого продукту. Крім власне програмування (кодування) в його функції входить:

- Контроль архітектурних і технічних специфікацій продукту.

– Підбір технологічних інструментів і стандартів.
– Діагностика і вирішення всіх технічних проблем.
– Контроль за роботою розробників документації, тестування, технологів.
– Моніторинг стану продукту (ведення списку виявлених помилок).
– Підбір інструментів розробки, метрик і стандартів. Контроль їх використання.

Тестувальник – роль, відповідальна за задоволення вимог до продукту (функціональних і нефункціональних). У функції тестувальника входить:

– Складання плану тестування. План тестування становить один з елементів проєкту і складається до початку реалізації (розробки) проєкту.
– Контроль виконання плану. Найважливіша функція контролю – підтримка цілісності бази даних зареєстрованих помилок. У цій базі реєструється: хто, коли і де виявив, опис помилки, опис стану середовища; статус помилки: пріоритет, хто дозволяє; стан помилки: висить, в розробці, дозволена, проблеми. Ця база повинна бути доступна всім, тому що в тестуванні беруть участь всі члени команди.
– Розробка тестів. Сама трудомістка частина в роботі тестувальника. Тестування повинне забезпечити повну перевірку функціональності при всіх режимах роботи продукту.
– Автоматизація тестування включає автоматизацію складання тестів, автоматизацію пропуску тестів і автоматизацію обробки результатів тестування. З огляду на важливість автоматизації тестування, іноді вводять нового учасника – ***інженера з автоматизації***.
– Вибір інструментів, метрик, стандартів для організації процесу тестування.
– Організація бета тестування – тестування майже готового продукту зовнішніми тестерами (користувачами). Цю важливу процедуру треба продумати і організувати в разі розробки коробкового продукту.

Інженер з якості. У сучасному уявленні розглядається три аспекти (рівня) якості:

- якість кінцевого продукту – забезпечується тестуванням;
- якість процесу розробки (теза: для підвищення якості продукту треба підвищить якість процесу розробки);
- якість (рівень) організації (теза: для підвищення якості процесу треба підвищити якість організації робіт).

У деяких випадках функції інженера з якості покладаються на тестувальника. Насправді вони ширші – два наступні рівні якості.

Тут наведені функції, відмінні від функцій тестувальника:

- Складання плану якості. План якості включає всі заходи щодо підвищення якості (на всіх рівнях). Має довготривалий характер. План тестування – його оперативна складова.
- Опис процесів. Опис процесів є їх формалізацією.

При описі вводяться метрики процесу, що впливають на якість продукту. Оцінка процесів включає реєстрацію ходу виконання процесів і оцінку значень встановлених метрик процесів. Виявлення «слабких» місць і вироблення рекомендацій щодо поліпшення процесів.

- Поліпшення процесів – перевизначення процесу, автоматизація частини робіт, навчання персоналу.

Підвищення якості процесів вимагає участі всіх дійових осіб. Прийняте рішення повинно бути обґрунтоване, всім зрозуміле і всіма прийняте.

При підвищенні якості організації робота щодо поліпшення процесів проводиться за певною схемою. На кожному кроці підвищення рівня організації робіт виділяються ключові процеси і виконуються роботи щодо поліпшення цих процесів.

Технічний редактор або ***розробник користувацької (та іншої) документації*** як частини програмного продукту. Функціями технічного письменника є:

- Розробка плану документування, який включає склад, терміни підготовки і порядок тестування документів.
- Вибір і розробка стандартів і шаблонів підготовки документів.
- Вибір засобів автоматизації документування.

– Розробка документації.
– Організація тестування документації.
– Участь в тестуванні продукту. Технічний редактор весь час працює з продуктом (його готовими версіями) і виступаючи від імені користувача бачить всі недоліки і невідповідності.

Технолог розробки ПЗ забезпечує виконання наступних завдань:

– Підтримка моделі ЖЦ – створення служб і структур з підтримки працездатності прийнятої моделі ЖЦ ПЗ. У підтримці моделі ЖЦ беруть участь всі. Але контроль покладено на технолога.
– Створення та супровід середовища збірки продукту. Функція особливо важлива на завершальних етапах розробки або при використанні моделі прототипування. У такій ситуації збірка буде проводитися досить часто (в деяких випадках – щодня). Середовище збірки повинне бути підготовлене заздалегідь, збірка повинна проводитися швидко і без збоїв. З урахуванням збірки версій це не просте завдання.
– Створення і супровід процедури установки з тим, щоб кожна збірка встановлювалася автоматично з урахуванням версії і конфігурацій середовищ.
– Управління вихідними текстами – супровід і адміністрування системи управління версіями вихідних текстів.

Виділені позиції не обов’язково представлені конкретними людьми. Це список основних функціональних ролей в команді (рольова модель команди). У малих командах ролі можуть поєднуватися. У великих – виділятися групи або відділи (відділ проєктування, відділ тестування, відділ контролю якості, відділ підготовки документації тощо).

Склад команди визначається також типом виконуваних робіт: під замовлення або коробкове виробництво (продукт на ринок).

Інженерний психолог і інженер з маркетингу потрібні в останньому випадку.

4. Моделі організації команд

Слід зазначити, що рольові моделі проєктних команд можуть бути найрізноманітнішими. Нижче приведена рольова модель команди, рекомендована в методології MSF (Microsoft Solution Framework). Ця модель заснована на шести рольових кластерах, кожен з яких має свою мету, компетенції та функції:

Кластер 1. Управління продуктом *(product management).*

Мета: задоволені замовники.

Компетенції. Маркетинг. Бізнес-віддача (бізнес-пріоритети). Представлення інтересів замовника. Планування продукту.

Функції. Виступає в ролі представника замовника. Формує загальне бачення і межі проєкту. Організовує роботу з вимогами замовника. Розвиває сфери застосування в бізнесі. Формує очікування замовника. Визначає компроміси між параметрами «можливості продукту / час / ресурси».

Організовує маркетинг, PR і євангелізацію. Розробляє, підтримує і виконує план комунікацій.

Кластер 2. Управління програмою *(program management).*

Мета: Досягнення результату в рамках проєктних обмежень.

Компетенції. Управління проєктом. Вироблення архітектури рішення. Контроль виробничого процесу. Адміністративні служби.

Функції. Управляє процесом розробки з метою отримання готового продукту в відведені терміни. Формулює специфікацію продукту і розробляє його архітектуру. Регулює взаємовідносини і комунікацію всередині проєктної групи. Стежить за тимчасовим графіком проєкту і готує звітність про його стан. Проводить в життя важливі компромісні рішення. Розробляє, підтримує і виконує зведений план і календарний графік проєкту. Організовує управління ризиками.

Кластер 3. Розробка *(development).*

Мета: створення продукту відповідно до специфікації.

Компетенції. Технологічне консультування. Проєктування і здійснення реалізації. Розробка додатків. Розробка інфраструктури.

Функції. Визначає деталі фізичного дизайну. Оцінює необхідні час і ресурси на реалізацію кожного елемента дизайну.

Розробляє або контролює розробку елементів. Готує продукт до впровадження. Консультує команду з технологічних питань.

Кластер 4. Тестування *(test).*

Мета: схвалення випуску продукту тільки після того, як всі дефекти виявлені і улагоджені.

Компетенції. Планування тестів. Розробка тестів. Звітність за тестами.

Функції. Забезпечує виявлення всіх дефектів. Розробляє стратегію і плани тестування. Здійснює тестування.

Кластер 5. Задоволення споживача *(user experience – досвід користувача).*

Мета: підвищення ефективності користувача, збільшення споживчої цінності продукту.

Компетенції. Забезпечення технічної підтримки. Навчання. Ергономіка.

Графічний дизайн. Інтернаціоналізація. Загальнодоступність (забезпечення можливості роботи для користувачів з обмеженими фізичними можливостями).

Функції. Представляє інтереси споживача в команді. Організовує роботу з вимогами користувача. Проєктує і розробляє системи підтримки продуктивності. Визначає компроміси, які стосуються зручності використання і споживчих якостей продукту. Визначає вимоги до системи допомоги та її зміст. Розробляє навчальні матеріали та здійснює навчання користувачів.

Кластер 6. Управління випуском *(release management).*

Мета: безпроблемне впровадження і супровід продукту.

Компетенції. Інфраструктура. Супровід. Бізнес процеси. Управління випуском готового продукту.

Функції. Представляє інтереси відділів постачання і обслуговування продукту. Організовує постачання проєктної групи. Організовує впровадження продукту. Виробляє компроміси в керованості і зручність супроводу продукту. Організовує супровід і інфраструктуру постачання. Організовує логістичне забезпечення проєктної групи.

Перш ніж обрати відповідну модель організації команди потрібно детально продумати таку важливу річ як людський фактор та дати відповіді на такі питання: Як організувати роботу

команди? Команди з 10 чоловік і команди з 500 чоловік? Чи є відмінності і в чому вони полягають? Чи треба організовувати роботу за жорсткою технологію або треба надати свободу дій? Чи можна знайти методологію (технологію) виконання проєкту, що забезпечує успіх?



Рис. 1.35. Основні питання вибору моделі організації команди

Алістер Коубі – фахівець в галузі технологій виконання IT-проєктів наводить дані 23 проєктів різного ступеня складності, що виконувалися за різними технологіями і мають різні результати. Намагаючись проаналізувати результати застосування різних технологій в тих чи інших умовах, він приходить до висновку, що практично будь-яку методологію можна з успіхом застосовувати в будь-якому проєкті. Проте, будь-яка методологія може привести як до успіху проєкту, так і до його провалу.

Головну причину він бачить в тому, що прямо перед нами завжди знаходиться щось, чого ми не помічаємо – люди. Саме людські якості забезпечують успіх того або іншого проєкту, саме вони є чинником першорядної важливості, ґрунтуючись на якому треба будувати прогнози про проєкт.

Дослідженню питань людського фактора (peopleware) приділяється досить багато уваги. Проблеми людського фактора пов’язані з тим, що люди, які беруть участь в проєкті:

- Всі різні – за характером, темпераментом, активності, цілям – немає двох однакових людей.
- Всі схожі – участь у проєкті об’єднує людей спільними метою, пошуком шляхів досягнення цих цілей.
- Розрізняються за типом: індивідуалісти – члени команди; генератори ідей – виконавці; відповідальні – безвідповідальні.

– Постійні і мінливі – люди, як правило, виявляють сталість своїх звичок і властивостей характеру, але при цьому здатні виявляти «протилежні» якості: індивідуаліст – командні якості, виконавець – генерувати ідеї.

Різноманітні – треба розуміти, що різноманіття людей є основною гарантією виживання людства взагалі і можливості виконувати ІТ-проєкти зокрема. Якби всі були індивідуалісти або всі керівниками, всі генератори ідей або всі виконавці, то навряд чи вдалося виконати хоча б один проєкт, а світ став би жахливий. Як же управляти такими людьми?

Розглянемо три основні моделі управління командою.

Адміністративна модель – теорія X

Це традиційний стиль управління, пов'язаний з ієрархічною адміністративно-командної моделлю, яку використовують військові організації.



Рис. 1.36. Адміністративна модель (теорія X)

В основі лежить теорія X, яка вказує на те, що такий підхід необхідний, оскільки більшість людей за своєю природою не любить роботу, і буде прагнути уникнути її, якщо у них є така можливість. Однак менеджери повинні примушувати, контролювати, направляти співробітників і погрожувати їм, щоб отримати від них максимальну віддачу. Девіз теорії і моделі: Люди роблять тільки те, що ви контролюєте. Або в більш м'якому варіанті: Люди роблять те, що вони не хочуть робити, тільки якщо ви їх контролюєте.

Зрештою, теорія стверджує, що більшість людей вважають за краще, щоб їм говорили, що слід робити і їм не доведеться нічого вирішувати самим.

Характерні риси моделі:

– Владна піраміда – рішення приймаються зверху-вниз.
– Чіткий розподіл ролей і обов'язків.
– Чіткий розподіл відповідальності.
– Дотримання інструкціям, процедурам, технологіями.
– Роль менеджера: планування, контроль, прийняття основних рішень.

Переваги моделі: ясність, простота, прогнозованість. Модель добре поєднується з каскадною моделлю ЖЦ і може бути застосована в тих же випадках, що і каскадна модель. Модель ефективна в разі усталеного процесу.

Недоліки моделі пов'язані з тим, що адміністративна система прагне самозбереження (стабільності) і погано сприйнятлива до зміни ситуації – нові типи проєктів, застосування нових технологій, оперативна реакція на зміну ринку.

Крім того, в адміністративній моделі погано уживаються індивідуалісти і генератори ідей.

Адміністративна система (модель) – це важкий потяг, що йде тільки по центру і не вдається до крайнощів для пошуку нових шляхів і рішень. Вона сприймає нові рішення і технології, але тільки перевірені, відпрацьовані і стандартизовані. У цьому її сила, слабкість і прояв принципу різноманіття. Мабуть, саме до неї найбільшою мірою можна застосувати термін «промислове програмування».

Модель хаосу – Теорія Y

В основі моделі хаосу лежить Теорія Y, яка є повною протилежністю Теорії X. Основна теза Теорії Y: робота – природна і приємна діяльність і більшість людей, насправді, дуже відповідальні і не ухиляються від роботи.



Рис. 1.37. Модель хаосу – Теорія Y

Характерними рисами моделі хаосу є:

– Відсутність явно виражених ознак влади.
– Роль менеджера – поставити завдання, забезпечити ресурсами, не заважати і стежити, щоб не заважали інші.
– Відсутність інструкцій і регламентованих процедур.
– Індивідуальна ініціатива – рішення по проблемі приймається там, де проблема виявлена.
– Процес нагадує творчу гру учасників на основі дружньої змагальності.

Переваги такої моделі в тому, що творча ініціатива учасників нічим не пов'язана і потенціал учасників розкривається в повній мірі. Це буває особливо ефективно в разі, коли для вирішення проблеми потрібно шукати нових підходів, методів, ідей і засобів. Команда стає командою «прориву», а робота проходить у формі гри, мета якої – пошук найкращого результату. Процес нагадує випадковий пошук, коли ідеї і рішення народжуються при живому і ніби як випадковому обговоренні проблем в коридорі, їдальнею на пікніку. Зібрати таку команду в робочій кімнаті і влаштувати обговорення за регламентом часто просто не вдається – це команда творчих індивідуалістів.

Недоліки моделі пов'язані з тим, що при певних умовах команда прориву може стати командою провалу. Причинами провалу можуть бути:

– Творча змагальність переходить в конкуренцію спочатку ідей, а потім – особистостей. Процес починає переважати над метою проєкту – висловлені ідеї не доводяться до кінця і змінюються новими ідеями, переважання отримують «красиві» ідеї, що лежать в стороні від основних цілей проєкту.
– Люди, здатні до генерації ідей, рідко мають терпіння доведення ідей до повної реалізації.

Модель хаосу – це те, що потрібно для освоєння нових земель. Модель хаосу чи не суперечить адміністративній моделі – вона її доповнює і може ефективно з нею бути сусідами (але в різних кімнатах). Більшість потужних корпоративних розробників покладаються на дослідні «відділи скунсів», звідки вони черпають нові ідеї, технології і продукти.



Рис. 1.38. Порівняння моделей X та Y

Якщо порівняти дві розглянуті моделі, то бачимо, що для моделі X характерний авторитарний, репресивний стиль. В цій моделі переважає жорсткий контроль, не існує ніякого розвитку, депресивна культура. Основа моделі Y – це ліберальний і розвивальний стиль, постановка задача і вільне її виконання.

Відкрита архітектура – Теорія Z

Адміністративна і хаотична моделі є двома «крайнощами», між якими знаходяться безліч моделей, що поєднують переваги «крайніх» моделей.

Однією з таких моделей є модель відкритої архітектури, заснована на теорії Z.



Рис. 1.39. модель відкритої архітектури – Теорія Z

Ця теорія була сформульована Уїльямом Оучи на основі вивчення досвіду японського стилю управління [13]. Теорія Z передбачає (але не декларує) наявність внутрішнього механізму управління, заснованого на вплив з боку колег і групи загалом. Додатковий вплив надають культурні норми конкретної корпорації. Основний принцип моделі можна сформулювати так: «Працюємо спокійно. Працюємо разом».

Особливостями цієї моделі є:

Адаптація до умов роботи – якщо робимо незалежні модулі, то розходимося і робимо, якщо потрібна архітектура бази даних, то збираємося разом і обговорюємо ідеї.

Колективне обговорення проблем, вироблення консенсусу і прийняття рішення – не всі можуть погодиться, але прийняте рішення є колективним і в силу цього – обов'язковим для всіх.

Розподілена відповідальність – відповідають всі, хто обговорював, виробляв, брав.

Динаміка складу робочих груп в залежності від поточних завдань.

Відсутність спеціалізації – учасники міняються ролями і функціями і можуть при необхідності замінити один одного.

Завдання менеджера – активна (але рядове, що не керівне) участь в процесі, контроль конструктивності обговорень, забезпечення можливості активної участі всіх.

Відкрита архітектура є більш гнучкою, адаптується, що настроюється на ситуацію. Вона дає можливість проявити себе всім членам команди – в ній можуть уживатися і індивідуалісти і колективісти. Колективне обговорення висловлених ідей дозволяє залишати тільки прагматичні ідеї.

5. Спілкування в команді

Основним фактором у розробці ПЗ є можливість комунікації (спілкування учасників проєкту). Спілкування може проводитися в різних формах від строго формалізованого (стандартизована документація) до повністю неформалізованого (питання-відповідь сусідові, обговорення в неформальній обстановці).

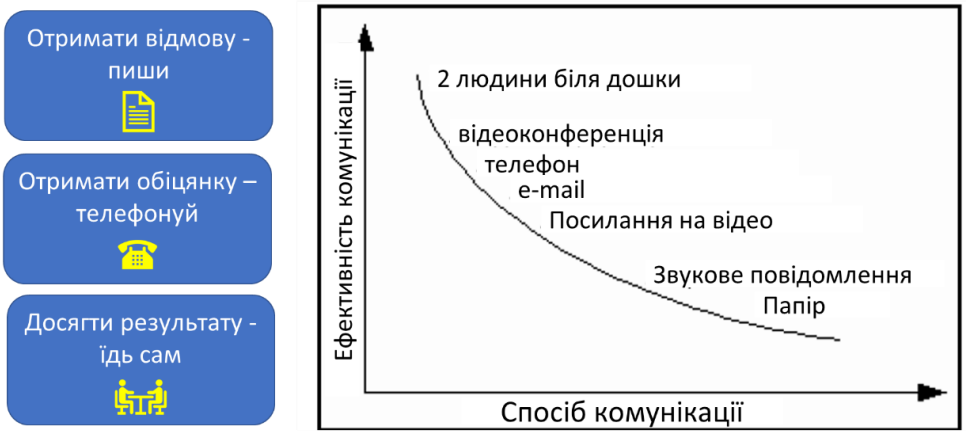


Рис. 1.40. Ефективність різних способів спілкування

На рис. 1.40 зображена деяка крива, що ілюструє ефективність різних способів спілкування. Крива є узагальненням ряду досліджень в цій галузі. Видно, що ефективність спілкування падає в міру зростання ступеня його формалізованості. З одного боку, в цьому немає нічого дивного – старий принцип бюрократа говорить: хочеш отримати відмову – пиши лист, хочеш отримати обіцянку – телефонуй, хочеш досягти результату – їдь сам.

Але з іншого боку, треба пам'ятати, що комунікації в команді визначаються кількістю учасників (робочих зв'язків): за двох учасників – це один зв'язок, при n учасниках – n*(n-2)/2. При цьому, будь-який з цих зв'язків може давати збої і вони не рівні: з того, що учасник А добре контактує з Б, а Б – з В зовсім не випливає, що А контактує з В. Тобто неформальне, «живе» спілкування ефективне тільки у відносно невеликих, добре організованих (спрацьованих) колективах.

Контрольні запитання

1. Які є визначення поняття «проєкт»?
2. Які основні характеристики проєкту?
3. Який вид діяльності можна вважати проєктом?
4. Які принципи та методи управління проєктом існують?
5. Що таке категорія управління проєктом?
6. Про що говорить закон Лермана?
7. Що таке трикутник обмежень проєкту?
8. Які компетентності повинні бути сформовані у менеджера ІТ-проєкту?
9. Які галузі управлінських знань менеджера проєкту існують (згідно з PMBOK)?
10. Які ролі існують у класичному варіанті моделі команди?
11. Яка різниця ролей «проєктувальник» та «розробник» ПЗ?
12. Які функції покладено на тестувальника ПЗ?
13. Хто забезпечує якість проєкту?
14. Хто забезпечує документування проєкту?
15. Яку роль виконує технолог розробки ПЗ?
16. Які існують моделі організації команд? Охарактеризуйте кожну з них.

РОЗДІЛ ІІ. ПРАКТИЧНІ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тема 2.1. Розробка технічного завдання на виконання проєкту

Практична робота №1

Мета: сформувати навички розробки технічного завдання щодо розробки програмного забезпечення, вміти подавати відомості у вигляді загальної інформації про проєкт, обґрунтовувати експлуатаційне та функціональне призначення, визначати термін виконання, формувати вимоги до ПЗ.

Теоретичні відомості

Складання технічного завдання (ТЗ) – це один з перших етапів при створенні сайту. При формулюванні завдання на розробку сайту необхідно оформити своє уявлення про сайт документально (додаток 4).

ТЗ розробляється не тільки для загального уявлення про проєкт, воно також повинне певною мірою регламентувати всі відносини замовника і виконавця.

У ТЗ зазвичай містяться вимоги по дизайну, навігації, способам представлення інформації, термінам і об’ємам робіт кожного етапу розробки сайту. Чітко сформульоване ТЗ допоможе уникнути конфліктів «нерозуміння» між виконавцем і замовником.

Технічне завдання найчастіше представлено наступними розділами:

- загальні відомості;
- експлуатаційне призначення;
- функціональне призначення;
- терміни та визначення;
- дані і списки;
- функціональні характеристики;
- сторінки;
- вимоги до надійності;
- вимоги до хостингу;

– наповнення контентом;
– здача і приймання.

Розглянемо їх детальніше.

1. Загальні відомості.

Тут можна описати назву компанії, сферу діяльності, товари або послуги, досягнення, дата заснування, унікальні пропозиції та інші відмінності. Наприклад, можливий такий опис:

ТОВ «Прінтфоревері», компанія заснована в 2007 році, виготовляємо рекламну (брендовану) і фанатську продукцію не тільки на продаж, але і під замовлення. Є всі види друку на тканинах, канцелярія, кераміка і друкарня. Є доставка по країні. Регулярно спонсоруємо фестиваль «Літній». Конкуренти: «Прінтпрізер», «Яскравофарбинка».

2. Експлуатаційне призначення.

Експлуатаційне призначення – це вигода, яку повинен принести сайт. Взагалі, найчастіше, вигода зводиться до грошей (якщо сайт комерційний, а таких більшість), і в розділі можна було б завжди обмежитися написанням того, що експлуатаційне призначення сайту – отримання матеріальної вигоди. Проте, зазвичай в даному розділі потрібно деталізувати призначення. Наприклад, для інтернет-магазину це буде продаж товару (з якого і буде матеріальна вигода – заробіток), для сайту візитки – це провести рекламу в Інтернеті (а реклама потрібна для отримання матеріальної вигоди) тощо.

Якщо говорити про сайт турбази, то потрібен сайт, щоб на нього заходили туристи, дивилися фотографії, читали відгуки («спалахували» бажанням побувати в цих місцях) і купували путівки (матеріальна вигода – купівля путівки). Отже, експлуатаційним призначенням сайту може бути надання інформації про туристичний сайт, залучення нових клієнтів.

3. Функціональне призначення.

Цей пункт ще більш деталізує уявлення про ПЗ. Тут повинен бути короткий перелік того, якими технічними засобами потрібно отримати те, що було описано в попередньому пункті. Наприклад, для інтернет-магазину – це каталог товарів, кошик замовлення, сторінки з інформацією про доставку, повернення і про компанію.

Приклад про сайт турбази. Сайт повинен надавати загальну інформацію про турбазу; можливість переглядати фотографії; на

сайті повинна бути карта, яка показує дістатися до місця призначення; форма бронювання номерів; і відгуки людей, які відпочили гостей з можливістю залишити свій відгук (гостьова книга).

4. Терміни та визначення.

Цей розділ дає впевненість, що замовник і виконавець говорять про одне й те ж.

Терміни можуть «вводиться» з двох сторін: від розробника до замовника, наприклад розробник пояснює замовнику, що таке хостинг і SMTP-сервер і від замовника до розробника. Найчастіше, це стосується предметної галузі та сфери, в якій працює замовник.

Але, як правило, у другому випадку не потрібно описувати терміни специфічні для предметної галузі, які не мають відношення до реалізації проєкту. Наприклад, для магазину, який продає запчастини для вітрильних суден, не варто виносити такі терміни, як «стаксель» і «ванти». Тут потрібні розшифровки термінів, якими оперує замовник і вкладає в них якийсь сенс і які можуть бути розробниками витлумачені невірно. Це можуть бути і прості слова, але в даному контексті, вони мають особливе значення.

Наприклад, замовник каже: «Сеанс роботи з сайтом коштує 100 гривень». Фраза «сеанс роботи з сайтом» – потребує опису. Цей термін може означати тривалість часу від входу на сайт до виходу, або ж період роботи поки на рахунку користувача є гроші. Тобто в даному випадку потрібно точно знати, що таке «Сеанс роботи». Помилкове розуміння такого простого терміну може створити реальну проблему.

5. Дані і списки.

Ключовий розділ ТЗ. Можна сказати його серце. Це не самий багатослівний, але найважливіший і важкий пункт ТЗ. Якщо він зроблений як слід, можна бути впевненим, що автор завдання зрозуміє, що саме потрібно зробити. Наявність цього пункту накладає дуже великі обмеження на створюваний продукт. Один тільки цей пункт, «важить» більше половини всього ТЗ.

Дані

Цей розділ містить перелік інформаційних сутностей, які використовуються в проєкті (додаток 8). Це дуже близько до опису таблиць в базі даних або моделей, якщо говорити про

фреймворк з MVC. Наприклад, у на сайті є новини. Новина – це сукупність заголовка, тексту і дати публікації. Для чого потрібно це визначення? Як і все в ТЗ – прояснити, що робити і підстрахуватися від непорозумінь.

Перерахування атрибутів сутності дозволяє помітити дрібниці, які, залишившись непоміченими, могли б привести до ускладнень.

Для прикладу, та ж сама новина: заголовок, текст, дата публікації.

Припустимо, в процесі роботи з'ясовується, що забули анонс новини (короткий текст, який відображається в списку новин). Додати його не проблема: потрібно в таблицю додати поле «анонс» типу «текст» і додаткове поле введення в створенні/редагуванні новини. Доопрацювання нескладне.

А тепер, припустимо, з'ясовується, що забули додати атрибут «Категорія новини». Просто додати одне поле в таблицю бази даних, як це було з анонсом, вже недостатньо. Доведеться додавати ще одну сутність, таблицю категорій і відповідний розділ в адмінпанелі з управління категоріями новин. Ось такого роду пункти, залишаючись непоміченими при оцінці проєкту, призводять до невірних результатів і, як наслідок, до зриву термінів. І саме цей пункт ТЗ дозволяє виявляти подібні проблеми. Тобто краще помітити брак «Категорій» на етапі написання ТЗ, ніж в процесі роботи.

Списки

Новина – це одна сторінка з текстом, заголовком і датою публікації (анотацією), а список – це набір новин.

Навіщо це описувати? Припустимо потрібно відобразити на головній сторінці «останні новини». Ось останні новини, це як раз такий список. А що таке «останні новини»? Це вже можна зрозуміти по-різному, це можуть бути останні 5 новин або новини за останні 24 години. Наведений приклад простий, його нескладно виправити і при здачі проєкту. Але є більш важкі випадки.

Наприклад, замовник хоче свій сайт з колективними блогами. І він хоче, що б на сторінці, де відображається одна стаття, збоку був список «схожих статей». Що таке схожі статті? Це питання вимагає окремого розгляду і опису. І не звернувши уваги на цей список ми ризикуємо вже досить серйозно. Тобто тут потрібно

детально описувати алгоритм за визначенням подібності статей. Пропустивши цей пункт на етапі оцінки термінів можна досить сильно помилитись.

6. Функціональні характеристики.

Описується безпосередньо робота (частин) сайту. Оскільки докладний опис всіх сторінок сайту (компонентів) і всіх сутностей описується окремо, сюди потрібно виносити або загальні принципи роботи, або ту функціональність, яку не можна прив'язати до конкретної сторінки (додаток 9).

7. Сторінки.

Це розділ з описом всіх сторінок і їх змісту. У більшості випадків це досить короткий опис, тому що завжди можна використовувати розділ даних і списків. Наприклад, «на сторінці відображається список останніх новин». Що таке новина, вже було описано, що таке список новин – теж. Якщо потрібно, можна уточнити, що відображаються не всі дані новини, а тільки назва і анонс.

Тут буде доречно описати не тільки, що відображається, але і як. Слід зауважити, що це не стосується дизайну, наприклад: «Великими червоними літерами відображається назва новини», а стосується функціональності, тобто як це працює: «Зліва плавно виїжджає віконце з пропозицією ввести логін і пароль». Або так: «при натисканні кнопки "Відправити коментар", новий запис з'являється на сторінці без перезавантаження, за допомогою AJAX».

Чи потрібно тут описувати контент сторінки? Ні, не потрібно. Лише ТЗ. Описуючи каталог, не потрібно описувати всі товари в ньому. Основне завдання – описати функціонал, який дозволить замовнику самостійно заповнити контентом цю сторінку. Якщо планується наповнення сайту виконавцем, то цей процес краще винести в окремий документ.

В цьому випадку доцільно створити ескіз до кожної сторінки:

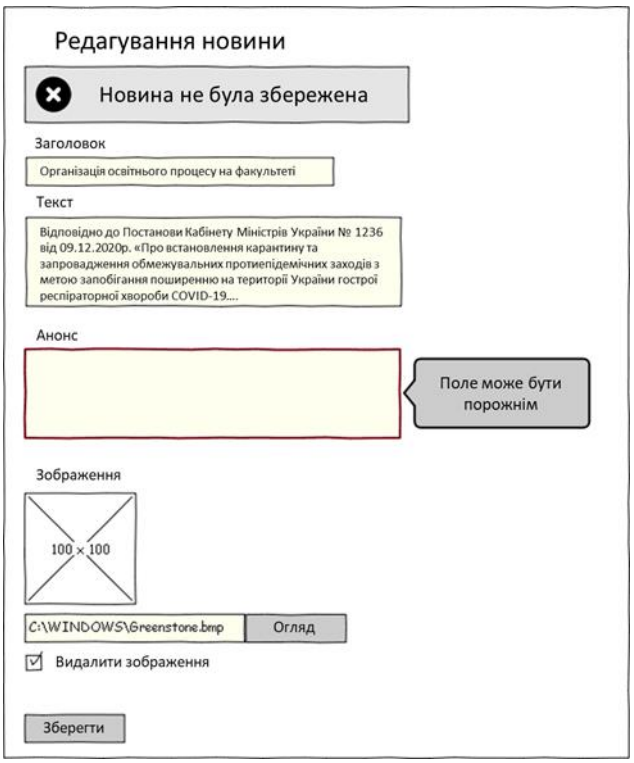


Рис. 2.1. Ескіз сторінки додавання новини на сайт

Потрібно також слідкувати, щоб текстовий опис не вступав в протиріччя з тим, що намальовано в ескізі.

Тобто якщо на ілюстрації новина має «Категорію новини», а в розділі «Дані та списки» новина не має категорії, то це проблема. Дуже висока ймовірність, що вивчаючи ТЗ, замовник запам'ятає саме картинку з ескізом новин, в якій є категорія, і якщо в готовому проєкті не буде категорії (відповідно до текстового опису новини) можуть бути неприємності.

Звісно, в тексті документу можна прямо вказати, що це первинний текст, а ілюстрації просто для полегшення розуміння. Хоча це не завжди спрацьовує.

Для сайтів з чітко вираженим поділом на адмінпанель і публічну частину, має сенс згрупувати всі сторінки в дві великі категорії: публічна частина і адмінпанель. Якщо чіткого поділу немає, потрібно вказати права доступу для кожної сторінки.

Всі сторінки сайту працюють взаємопов'язано, тому доцільно також пропрацювати схему цих зв'язків (додаток 10).

8. Вимоги до надійності.

Якщо планується сайт з високим навантаженням, про це варто сказати заздалегідь. Високонавантажений сайт цілком може потребувати специфічних дій з налаштування серверів або написання коду. І з'ясувати, що сайт повинен тримати велике

навантаження в момент здачі проєкту, не найкращий розвиток ситуації.

Варто окремо прописати, що для надійності, необхідно налаштувати бекапи, оскільки ніхто не застрахований від хакерів, які можуть зіпсувати базу даних або хостингу, який може вийти з ладу.

9. Вимоги до хостингу.

Очевидно, що цілком може виникнути, наприклад, така ситуація. Веб-студія робить гарні сайти, але розробляє їх виключно на Django. Замовник зробив замовлення. І коли сайт готовий, то його необхідно викласти на хостинг, де знаходяться інші сайти замовника, а там, природно – PHP. Виявляється, замовник думав, що всі роблять сайти на PHP і іншого хостингу він не має, тому сайт доведеться переробляти на PHP.

Крім таких очевидних проблем є проблеми і більш глибокого спрямування. Наприклад, для нормальної роботи сайту потрібен cron (утиліта, яка дозволяє користувачам виконувати команди або скрипти (групи команд) автоматично в заданий час), а такої послуги хостинг не надає.

Сюди варто включити вимоги до інтерпретаторів, бібліотек, пакетів; вимоги до дискового простору, пам’яті, smtp, pop, ftp, зовнішніх програм і до решти, що має значення для роботи проєкту.

10. Наповнення контентом.

Цей пункт обумовлює обсяг наповнення контентом. Як мінімум, потрібно створити той контент, який дозволить замовнику почати експлуатацію сайту. Принаймні створити обліковий запис для адміністратора сайту і сказати замовнику логін і пароль.

Якщо потрібно, наприклад, завантажити в каталог 500 фотографій, попередньо їх обробивши, то це слід описати саме тут.

Опис цього розділу застереже від різного розуміння того, хто повинен завантажити 500 фотографій і наповнити каталог товарами.

11. Здача і приймання.

Тут доцільно описати ті умови, при виконанні яких має відбутися розрахунок за роботу.

Можливі варіанти, наприклад, перенесення сайту на хостинг замовника лише після 100% оплати. Або ж оплата після перенесення на сайт замовника плюс тиждень на тестування.

Крім того, 100% оплата, не завжди може означати закінчення виправлення помилок. Загалом, помилки повинні виправлятися завжди і безкоштовно.

Практичні завдання

Завдання 1. Розробити попередній варіант ТЗ на прикладі готового сайту за своїм варіантом. При розробці ТЗ орієнтуватись на власні вподобання, виконуючи роль замовника проєкту.

Таблиця 2.1

Варіанти завдань

Вар.	***Назва сайту***	***Адреса***
1	Інтернет-магазин «Розетка»	https://rozetka.com.ua/
2	Інтернет-платформа «robota.ua»	https://about.rabota.ua/
3	Всеукраїнський портал «jobs.ua»	https://jobs.ua/
4	Біопарк «Золотий фазан»	https://gold-fazan.com.ua/
5	Готельний комплекс «GOL&Wood»	https://goliwood.com/
6	Інтернет-платформа «restoran.ua»	https://restoran.ua/
7	Інтернет-платформа «Work.ua»	https://www.work.ua/
8	Інтернет-платформа «BUKI»	https://buki.com.ua/
9	Інтернет-платформа «Preply»	https://preply.com/ua/
10	Інтернет-платформа «Асоціація репетиторів України»	https://repetitor.org.ua/uk

Завдання 2. Деталізувати ТЗ, а саме:

- Визначити інформаційні сутності готового сайту.
- Описати функціональні характеристики сайту.
- Розробити структуру окремого функціонального компоненту сайту, який зобразити у вигляді схеми.
- Розробити ескізи 3 сторінок сайту.

Контрольні запитання

1. Що таке ТЗ проєкту?
2. Які відомості містять в ТЗ?
3. Що таке функціональне призначення ПЗ?

4. Які можуть бути вимоги до надійності ПЗ?
5. Що описують дані і списки у ТЗ?
6. Що таке експлуатаційне призначення ПЗ?
7. Для чого потрібно робити ескізи сторінок ПЗ?

Тема 2.2. Введення до моделювання в середовищі ArgoUML

Практична робота №2

Мета: сформувати навички встановлення та використання середовища моделювання ArgoUML, використання провідника та панелі інструментів програми, роботи з елементами.

Теоретичні відомості

ArgoUML – UML засіб моделювання, який є відкритим ПЗ, що розповсюджується під ліцензією EPL (http://argouml.tigris.org). Посібник користувача знаходиться за посиланням: http://argouml-stats.tigris.org/documentation/manual-0.34.

ArgoUML повністю написаний на Java і для роботи йому підходить будь-яка операційна система з встановленою Java 2 JRE або JDK версії 1.4 або вище. Перевірити версію Java можна в командному рядку (cmd), ввівши команду: java -version.

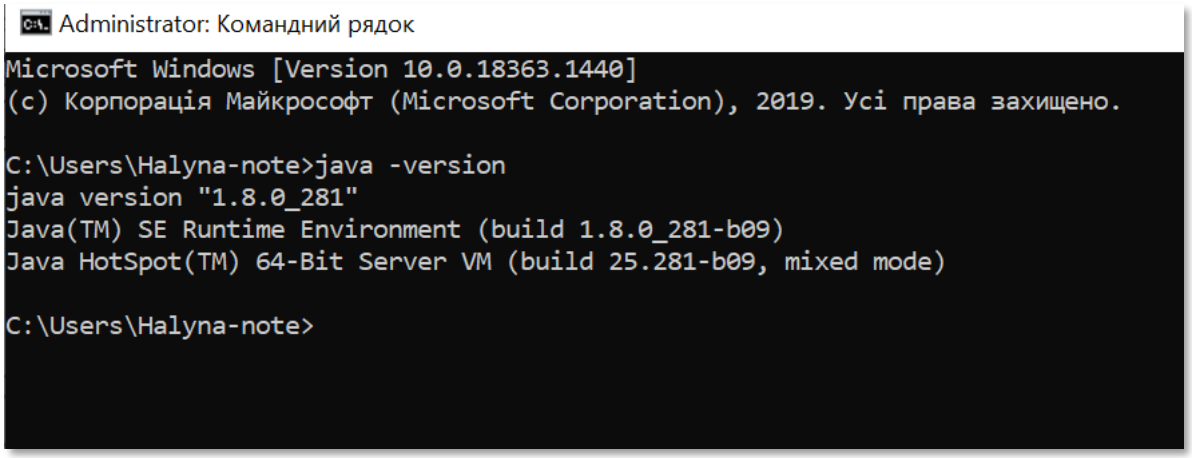


Рис. 2.2. Перевірка версії Java в командному рядку

Нижче наведено головне вікно ArgoUML.

Інтерфейс користувача ArgoUML розділений на 4 панелі:

– Зліва вгорі: ієрархічний вигляд поточного файлу проєкту.

– Угорі праворуч: редактор для вибраної частини проєкту, в даному випадку діаграма класів.
– Внизу ліворуч: список «Робити» дизайнера.
– Внизу праворуч: деталі вибраного об’єкта на діаграмі або вибраний елемент «Робити».

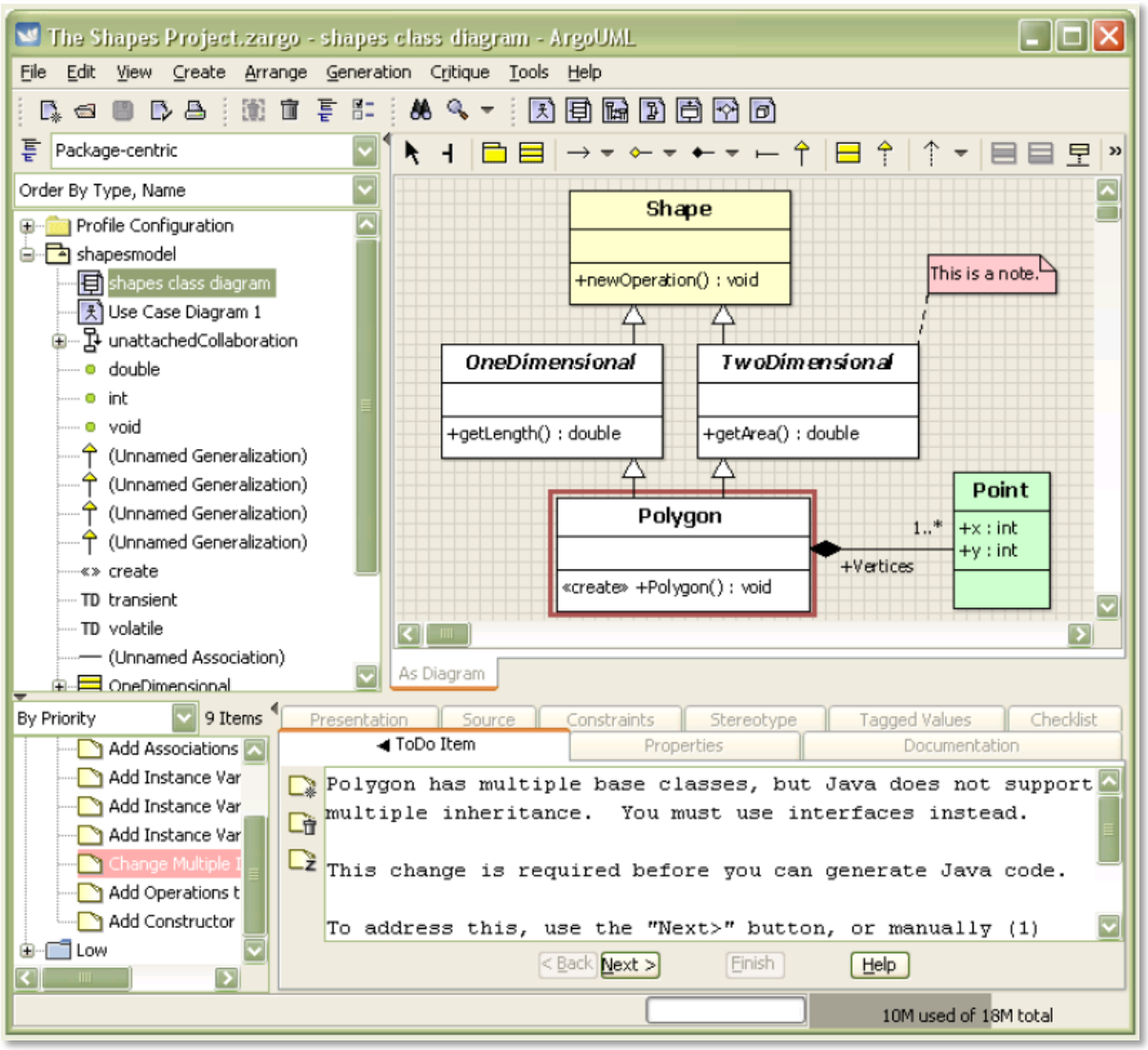


Рис. 2.3. Інтерфейс ArgoUML

ArgoUML підтримує наступні діаграми:

– Діаграма класів (Class diagram).
– Діаграма станів (State chart diagram).
– Діаграма використання (Use Case diagram).
– Діаграма діяльності (Activity diagram).
– Діаграма кооперації (Collaboration diagram).
– Діаграма розміщення (Deployment diagram).
– Діаграма послідовності (Sequence diagram).

Панель меню ArgoUML складається з меню для File (Файл), Edit (Редагування), View (Перегляд), Create (Створення), Arrange (Упорядкування), Generation (Генерація), Critique (Критика), Tools (Інструменти) і Help (Допомога).



Рис. 2.4. Рядок меню ArgoUML

Основною робочою областю є Editing Pane (Панель редагування) ArgoUML. Ця панель використовується для редагування діаграм. Для цього вона містить панель інструментів у верхній частині, яка містить всі елементи (ModelElement), які можна намалювати на поточному типі діаграми. Натискання на значок інструмента на панелі інструментів вибирає тип ModelElement, який буде поміщений простим клацанням миші в області діаграм. Панель інструментів також містить деякі загальні засоби малювання, для оформлення діаграм з лініями, текстом і фігурами, які не є частиною моделі UML.

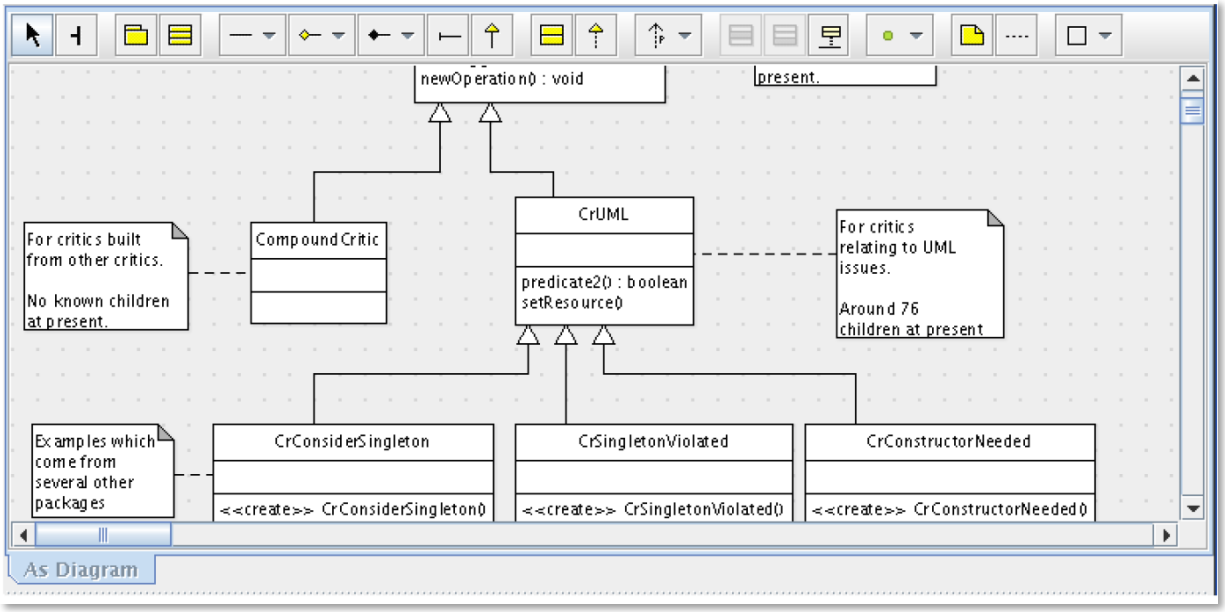


Рис. 2.5. Робоча область Editing Pane в ArgoUML

Explorer (Провідник) ArgoUML показує вміст та структуру дизайну. Цей елемент інтерфейсу користувача ArgoUML повинен бути знайомим кожному, хто використовував Провідник Microsoft Windows або будь-який з ряду комерційних інструментів CASE. Одноразове натискання на об’єкті в області провідника вибирає об’єкт у панелі головного редактора і показує деталі цього об’єкта

в області відомостей. Подвійне клацання по об'єкту в провіднику змінить об'єкт, показаний на головній панелі редагування.

Сортування елементів у деревній структурі може здійснюватися за алфавітом або (за замовчуванням) за типом.

Зазвичай потрібно двічі клацнути, щоб змінити тип діаграми, і клацнути один раз, щоб змінити властивості інших об'єктів.

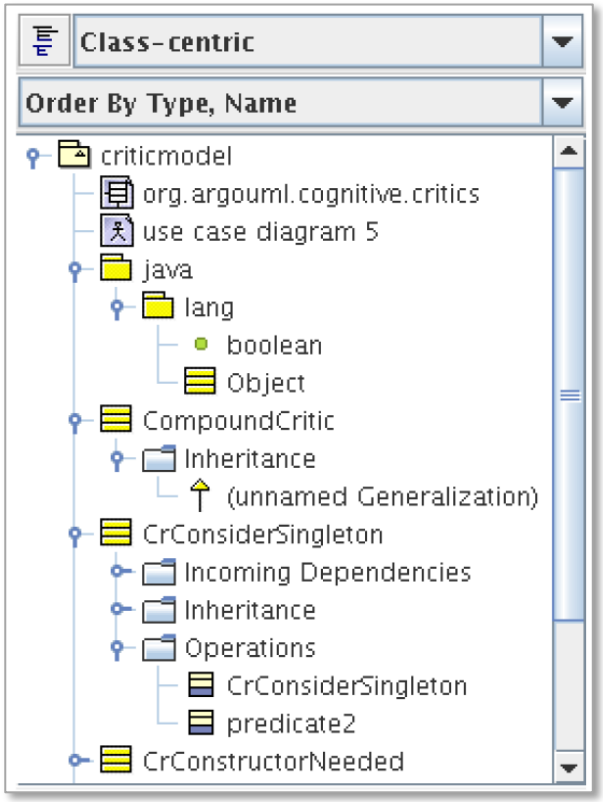


Рис. 2.6. Провідник Explorer в ArgoUML

Панель інструментів ArgoUML "To Do" (Виконати) допомагає дизайнерам зрозуміти, що вже зроблено і що можна зробити. Елементи панелі To Do (Виконати) можуть бути особистими нагадуваннями, введеними дизайнером, але більшість з них створюються критиками дизайну. Дизайнерські критики в ArgoUML постійно аналізують дизайн, шукаючи неповні або проблемні області. Коли виявляється потенційна проблема, критик створює пункт To Do (Виконати) і додає його до списку в панелі завдань. Аналогічно, якщо вирішується виявлена проблема, елемент буде видалено зі списку.

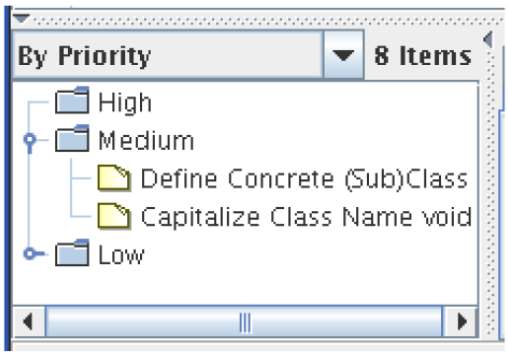


Рис. 2.7. Панель інструментів To Do в ArgoUML

Якщо натиснути на елемент у списку To Do (Виконати), буде відображено інформацію про потенційну проблему на вкладці ToDoItem панелі відомостей і буде виділено червону «критичну» частину проєкту.

При роботі з великою конструкцією може існувати велика кількість невирішених проблем і проблем, які необхідно розглянути до того, як дизайн буде повністю виконаний. Це може зробити список «to do» досить довгим. Щоб керувати списком «to do», можна групувати елементи за пріоритетом, типом підтримуваних рішень, порушуючим елементом дизайну або іншими способами.

Details Pane (Панель відомостей) ArgoUML дозволяє редагувати деталі вибраного елемента дизайну або елемента «to do» («робити»). Вкладка ToDoItem має головну стрілку на захід, яка вказує, що вона показує деталі об'єктів, вибраних в області «To Do» («Зробити»). Більшість інших вкладок мають північні стрілки, що вказують на те, що вони показують деталі об'єктів, вибраних на діаграмі або досліднику.

Натисніть одну з наведених нижче позицій, щоб дізнатися більше про вкладки, позначені як ToDoItem, Properties (Властивості), Documentation (Документація), Presentation (Уявлення), Source (Джерело), Constraints (Обмеження) і TaggedValues (Помічені Значення).

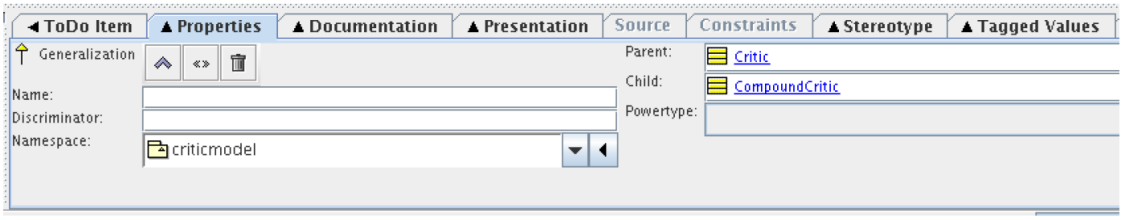


Рис. 2.8. Панель відомостей в ArgoUML

Робота з програмою починається з порожнього документу.

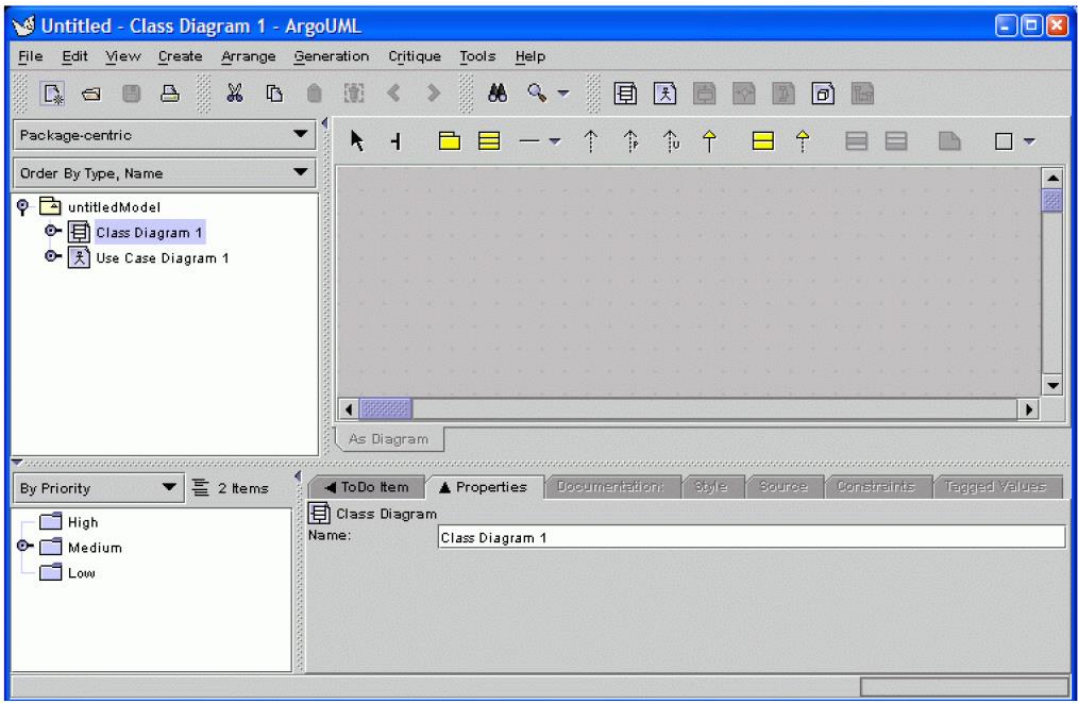


Рис. 2.9. Порожній документ в ArgoUML

Класи можна додавати до діаграми без назви, натиснувши на значок класу на панелі інструментів. Подвійне клацання на кнопці панелі інструментів «блокує» цей інструмент, доки не буде вибрано інструмент стрілки.

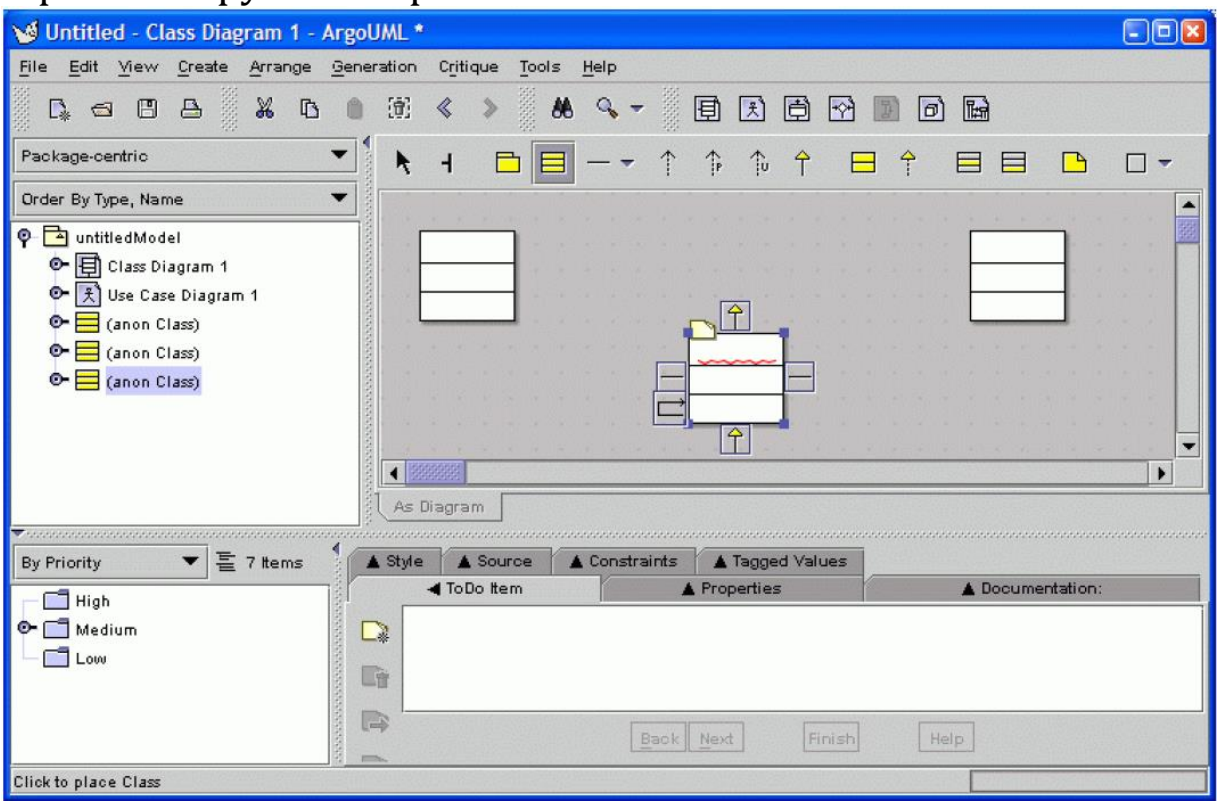


Рис. 2.10. Створення діаграм класів в ArgoUML

Жовті наліпки і хвилясті, червоні підкреслення вказують на неповні або проблемні частини конструкції. Розміщення миші над

клейкою приміткою або хвилеподібною лінією відображатиме синій наконечник, який коротко описує проблему.

Користувач може встановити ім'я класу або асоціації, просто вибравши його і ввівши потрібний текст.

Коли елемент елемента обраний в провіднику, його деталі відображаються на вкладці «Властивості». Зауважимо, що критика, що клас Shape не має операцій, автоматично видаляється.

Функціональність ArgoUML включає в себе:

- підтримку специфікацій UML 1.3, 1.4, XMI 1.0, 1.1, 1.2;
- 9 видів діаграм UML (діаграми класів, станів, кооперації, послідовності, діяльності, прецедентів, об'єктів, компонентів, розгортання);
- підтримку OCL для класів;
- генерацію вихідного коду Java, C ++, C # і PHP;
- зворотний інжиніринг з вихідного коду і байткода Java;
- автоматичну верифікацію моделі UML (design critics).

Засіб UML-моделювання ArgoUML є відкритим програмним забезпеченням та поширюється ліцензією EPL.

Більш детальніше з особливостями роботи з ArgoUML можна отримати на сторінці http://argouml.tigris.org/tours/.

Практичні завдання

Завдання 1. Встановити програмний засіб ArgoUML на робочий комп'ютер та зробити основні налаштування у програмі.

Завдання 2. Розглянути інтерфейс програми, навчитись використовувати панель редагування для створення та зміни діаграм.

Контрольні запитання

1. Для чого призначено програмний засіб ArgoUML?
2. Яка мови використовувалась для написання ПЗ ArgoUML?
3. З яких основних панелей складається ПЗ ArgoUML?
4. Які види діаграм підтримуються в ArgoUML?
5. Яка панель використовується для редагування діаграм?
6. Для чого використовується панель інструментів «To Do»?
7. Що таке дизайнерські критики?

Тема 2.3. Розробка діаграм варіантів використання (прецедентів)

Практична робота №3

Мета: сформувати навички розробки діаграм варіантів використання (прецедентів) у середовищі ArgoUML.

Теоретичні відомості

Діаграма – це графічне представлення безлічі елементів. Зазвичай подається у вигляді графа з вершинами (сутностями) і ребрами (відносинами). За допомогою діаграм можна візуалізувати систему з різних точок зору.

Існують такі види діаграм:

- Діаграма варіантів використання (Use Case diagram).
- Діаграма класів (Class diagram).
- Діаграма об'єктів (Object diagram).
- Діаграма станів (State chart diagram).
- Діаграма діяльності (Activity diagram).
- Діаграма послідовності (Sequence diagram).
- Діаграма кооперації (Collaboration diagram).
- Діаграма компонентів (Component diagram).
- Діаграма розміщення (Deployment diagram).

Функціональні вимоги до програмного продукту в нотації UML описують за допомогою діаграми варіантів використання (Use Case Diagram).

Варіант використання (прецедент) – це послідовність дій, які повинні бути виконані системою при взаємодії з відповідним користувачем – «актором». Діаграма варіантів використання також перекладається як діаграма прецедентів.

Прецедент – є засобом специфікації **дій** (**функцій**) системи з точки зору отримання основними акторами деяких суттєвих для них результатів. Простий спосіб визначення прецедентів полягає у тому, що треба розглянути кожного основного актора та визначити суттєві результати (суттєві цілі), які він може переслідувати, використовуючи систему. Для іменування

прецедентів використовують дієслова чи короткі дієслівні фрази, наприклад, «створення замовлення», «оновлення замовлення» тощо.

Прецеденти дають змогу специфікувати функціональну поведінку розроблюваної системи та отримати відповідь на запитання «Що має робити система?». Варто також пам'ятати, що прецеденти не визначають реалізацію поведінки системи – не торкаються питань «Яким чином відповідні функції будуть реалізовані?».

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами.

Суть даної діаграми полягає в наступному: проєктована система представляється у вигляді безлічі сутностей (акторів), що взаємодіють із системою за допомогою так званих варіантів використання (прецедентів). Варіант використання (від англ. use case[1]) використовують для описання послуг, які система надає «актору». Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з «актором». При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

Актор («actor» перекладається як «діяч») – це зовнішня до системи сутність (об'єкт), яка взаємодіє з нею для досягнення певних цілей. В діаграмах прецедентів вони позначаються стилізованими людськими фігурками. Прикладами акторів можуть бути: клієнт, адміністратор, бухгалтерська система тощо. Варто зауважити наступні особливості даного поняття:

- несе рольову ознаку;
- акторами можуть виступати інші програмні системи.

Ім'я актора записується з великої літери, не повинно бути власним; зазвичай це посада. Імена абстрактних акторів записуються курсивом.

[1] Термін "use case" запропонований Айвером Якобсоном в середині 1980-х роках. Спочатку цей термін звучав "usage case" і дослівно переводився як "ситуація використання". Проте, працюючи над англійським перекладом своєї статті, А.Якобсон вирішив, що термін "usage case" звучить не по-англійськи і переробив його в "use case".

У мові UML є кілька стандартних **видів відношень** між акторами і варіантами використання:

- асоціації (англ. association relationship);
- розширення (англ. extend relationship);
- узагальнення (англ. generalization relationship);
- включення (англ. include relationship).

Ці відношення застосовують тоді, коли необхідно виділити деяку загальну поведінку системи чи навпаки, створити декілька варіантів її використання.

Асоціації – вказує конкретну роль, яку грає актор при взаємодії з варіантом використання.

Розширення (extend) – визначає такий тип відношення, коли один варіант за певних умов повністю використовує інший (розширює його). Наприклад, оператор Інтернет магазину може видалити товар («Вилучити товар»), знаючи його ідентифікатор, або попередньо здійснивши пошук товару («Пошук товару»).

– – – ▷“extend”	Стрілка направлена до більш загального варіанту використання.

Інший приклад розширення. На рисунку зображено два відношення, які розширюють базовий прецедент *Зареєструвати клієнта* прецедентами *Зареєструвати корпоративного клієнта* і *Зареєструвати фізичну особу*. На основі базового прецеденту можна розробити головну процедуру реєстрації клієнта, а потім розробити спеціальні процедури, які доповнюватимуть головну процедуру.

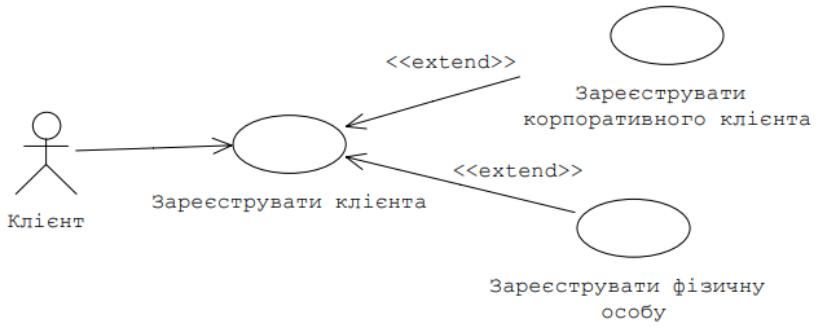


Рис. 2.11. Зразок діаграми прецедентів – «розширення»

Відношення розширення застосовують для моделювання таких частин прецеденту, які користувач сприймає як необов’язкову поведінку системи. Відношення розширення

використовують також для моделювання окремих субпотоків, які виконуються лише за визначених обставин. Зрештою, їх застосовують для моделювання декількох потоків, які можуть зливатися в деякій точці.

Узагальнення (generalization) означає, що прецедент нащадок успадковує поведінку і семантику свого предка, може заміщати його чи доповнювати його поведінку та, крім цього, може фігурувати усюди, де тільки з'явився його предок (як батько, так і нащадок можуть мати конкретні екземпляри). Таким чином, варіант використання А може бути узагальненим до варіанту використання В:



А – є спеціалізацією або нащадком, В – пращур.

Отже, таке відношення показує зв'язок між загальним і частковим. Наприклад, актор «Адміністратор» може виконувати всі функції актора «Користувач», тобто виступає частковим випадком користувача, але й виконує більш специфічні функції (що стосується адміністрування). Отже, узагальнення вказує на спільність ролей.

Включення (include) – вказує що деяка поведінка варіанту використання являється складовою поведінки іншого варіанту використання. Наприклад, варіант використання «Замовлення товару» завжди містить такі варіанти використання як «Пошук товару», «Порівняння товару», «Метод оплати» тощо.

– – – ▷ “include”	Стрілка направлена від базового елементу до менш загального.

Інший приклад включення – система бронювання квитків. Між базовим прецедентом *Забронювати місця* та автономним прецедентом *Переглянути інформацію про рейси* створено відношення, яка вказує на те, що забронювати місця можна, переглянувши інформацію про рейси.

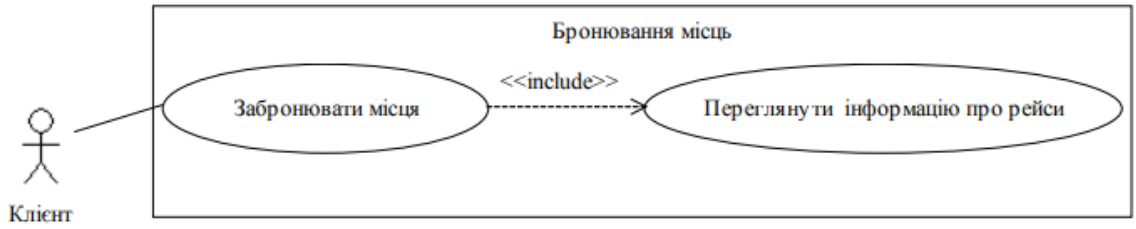


Рис. 2.12. Зразок діаграми прецедентів – «включення»

Мета використання діаграм варіантів використання для моделювання виробництва:

- визначення структури й робочих процесів організації, у якій буде використовуватися розроблювальна система;
- осмислення поточних проблем організації й шляхів поліпшення;
- забезпечення загального розуміння роботи організації замовниками й кінцевими користувачами;
- визначення вимог до системи, необхідних для підтримки виробничих процесів організації.

Практичні завдання

Завдання 1. Виконати практичне завдання за інструкцією

Постановка завдання (опис предметної області).

Магазин здійснює продаж товарів клієнту шляхом оформлення документів «Замовлення». Директор магазину, прийняв рішення автоматизувати документообіг продажів товару і запросив для виконання робіт програміста. У відповідності з концепцією життєвого циклу (ЖЦ) програми програміст приступив до опису бізнес процесів, що супроводжують продаж товару. Взявши за основу мову UML, він почав з побудови контекстної діаграми процесів – Use Case diagram. Діаграма повинна відповісти на питання – «**Що повинно робитися в системі і хто учасник цих процесів?**».

1. Створити нову модель. У вікні характеристик перейменувати ім’я моделі на «<<useCaseModel>> Use Case Model – OWN».
2. Виділити та клацнути правою кнопкою миші на назві створеної моделі, обирати пункт «CREATE DIAGRAM \ NEW USE CASE DIAGRAM» (або натиснути кнопку New Use Case Diagram на панелі інструментів) та назвати діаграму «Main - OWN».
3. На панелі інструментів у вікні редагування вибрати «New Use Case» і назвати «Ввести нове замовлення». Таким чином створюються нові варіанти використання.

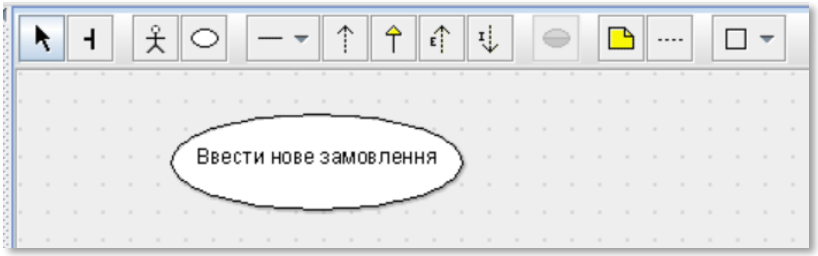


Рис. 2.13. Створення нового варіанту використання

4. Створити аналогічно такі варіанти використання:

- *Змінити наявне замовлення.*
- *Надрукувати інвентарний опис.*
- *Оновити інвентарний опис.*
- *Оформити замовлення.*
- *Відхилити замовлення.*
- *Виконати доставку замовлення.*

5. За допомогою кнопки «New Actor», на панелі інструментів, розмістити на діаграму нову діючу особу. Назвати її «Продавець».

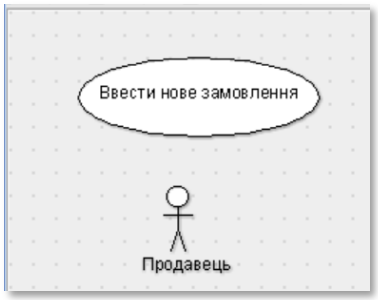


Рис. 2.14. Додавання нового актора – «Продавець»

6. Виконати аналогічні пункту 5 дії та створити на діаграмі наступні діючі особи:

- *Менеджер*
- *Клерк*
- *Бухгалтерська система*

7. Для створення абстрактного варіанту використання (тобто варіанту, що не потребує подальшої декомпозиції) потрібно виконати наступне:

- Вибрати на діаграмі «Main – OWN» варіант використання «Відхилити замовлення».
- У вікні характеристик активувати властивість «IsAbstract» (поставити позначку напроти цієї опції).

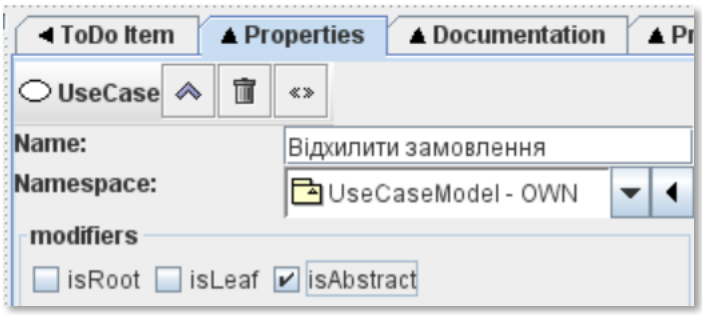


Рис. 2.15. Створення абстрактного варіанту використання

8. Створення асоціації:

За допомогою кнопки «New UniAssosiation» на панелі інструментів провести зв'язок між діючою особою «Продавець» і варіантом використання «Ввести замовлення».

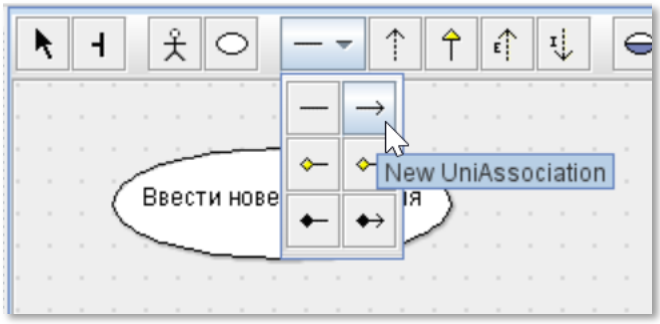


Рис. 2.16. Вибір інструменту створення асоціації

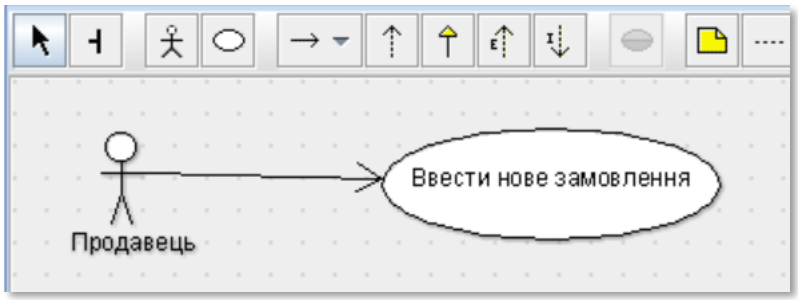


Рис. 2.17. Результат створення асоціації

9. Виконати аналогічні дії пункту 8 та створити зв'язки для:

- *Продавець – «Змінити наявне замовлення»*
- *«Ввести нове замовлення» – Бухгалтерська система*
- *Менеджер – «Надрукувати інвентарний опис»*
- *Менеджер – «Оновити інвентарний опис»*
- *Клерк – «Оформити замовлення»*
- *Клерк – «Виконати доставку замовлення»*

10. Створити зв'язки розширення (extend). За допомогою кнопки «New Generalization», розташованої на панелі

інструментів, проведемо зв’язок між прецедентами «Відхилити замовлення» і «Оформити замовлення».

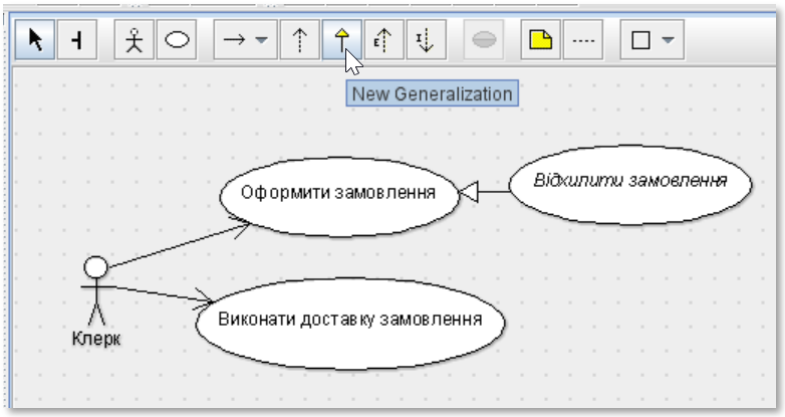


Рис. 2.18. Зв'язок розширення між прецедентами «Відхилити замовлення» і «Оформити замовлення»

11. Виділити зв’язок між варіантами використання «Відхилити замовлення» і «Оформити замовлення». У вікні «Properties» натиснути кнопку «New Stereotype».

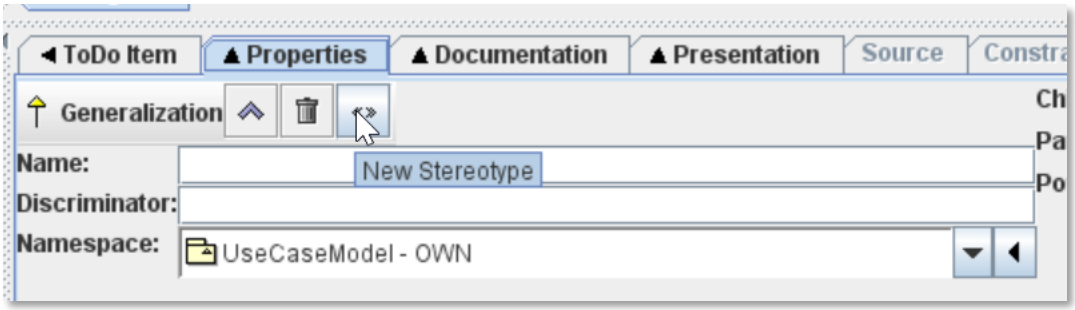


Рис. 2.19. Створення позначення зв’язку розширення

У вікні, що з’явилось, у полі «Name» впишемо «extends»

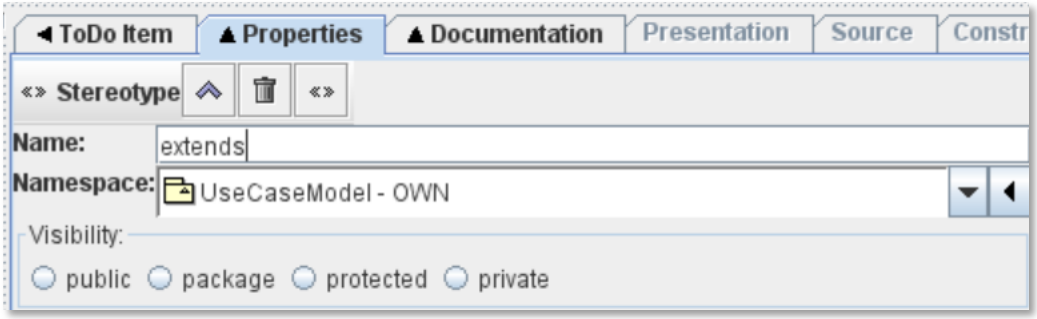


Рис. 2.20. Введення назви зв’язку розширення

12. Таким чином, позначення цього виду зв’язку з’явиться на діаграмі:



Рис. 2.21. Результат назви зв'язку розширення

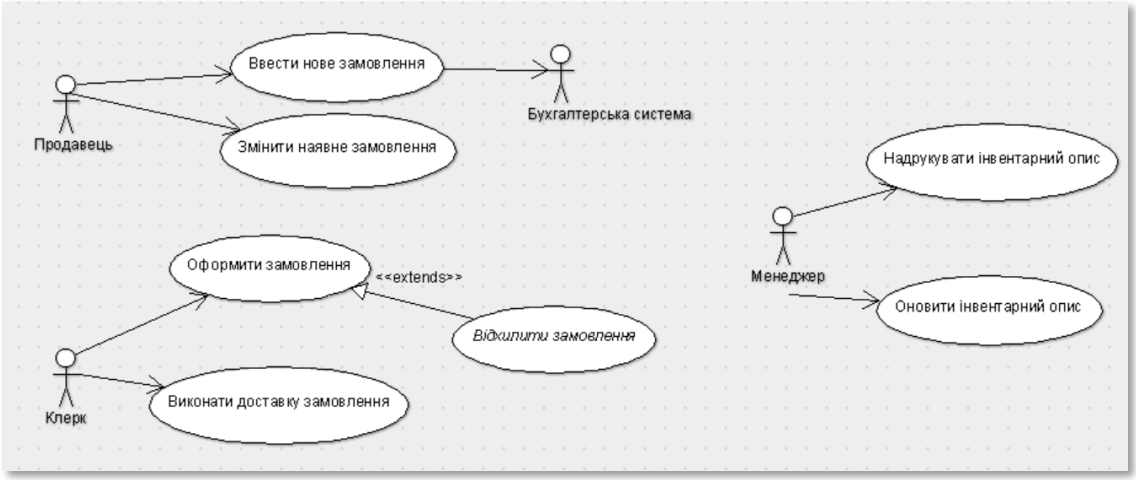


Рис. 2.22. Кінцевий вигляд діаграми USE CASE

Завдання 2. Самостійно розробити діаграми варіантів використання та продумати види зв'язків (розширення, узагальнення, включення) між акторами та прецедентами і позначити їх на діаграмі (додаток 13)

Контрольні запитання

1. Для чого призначено призначена діаграма варіантів використання?
2. Дайте визначення актора (actor) програмної системи. Хто може виступати в ролі акторів?
3. Дайте визначення варіанта використання (прецеденту). Яким чином варіант використання подається на діаграмі?
4. Перелічіть відношення, що можуть використовуватися на діаграмі прецедентів?
5. Дайте визначення відношення узагальнення (generalization). Наведіть приклад.

6. У чому відмінність між відношенням узагальнення між акторами та між варіантами використання?
7. Дайте визначення відношення асоціації (association). Наведіть приклад.
8. Дайте визначення відношення включення (include). Наведіть приклад.
9. Дайте визначення відношення розширення (extend). Наведіть приклад.
10. Які відношення можуть використовуватися для поєднання тільки акторів, акторів та варіантів використання, тільки варіантів використання?

Тема 2.4. Розробка діаграм послідовності

Практична робота №4

Мета: сформувати навички розробки діаграм послідовності у програмі ArgoUML.

Теоретичні відомості

Діаграма послідовності (англ. sequence diagram) – це графічне представлення взаємодії об'єктів програми, впорядкованих за часом. Іншими словами, діаграма послідовності відображає ***часові особливості передачі*** і ***прийому повідомлень*** об'єктами.

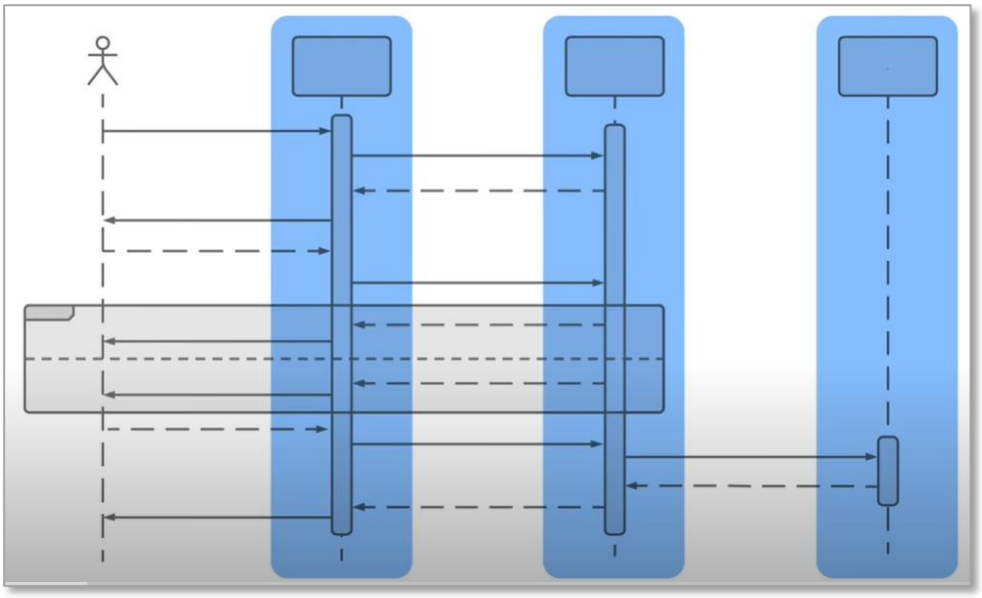


Рис. 2.23. Графічне представлення діаграми послідовності

Як правило, в цій діаграмі демонструється, як користувачі (актори з діаграми прецедентів) взаємодіють з іншими компонентами програми під час реалізації тих чи інших варіантів використання програми, та як при цьому взаємодіють інші компоненти програмної системи. Таким чином, діаграма послідовностей відображає ***взаємодію об'єктів в динаміці***.

Головними елементами діаграм послідовності є ***об'єкти***, які є логічними сутностями, що представляють окремі елементи системи та ***повідомлення***, якими вони обмінюються. В якості об'єктів можуть виступати також актори.

Діаграми послідовності є одним із способів *формалізації сценаріїв використання*. Її перевага полягає в тому, що на ранніх стадіях опису сценаріїв можливо з'ясувати склад взаємодіючих компонентів та описати потік повідомлень від одних компонентів до інших. Ці компоненти та потоки повідомлень в подальшому будуть трансформовані в конкретні класи (об'єкти), методи цих об'єктів (якщо говорити термінологією мови Java). Відповідно, одразу ж з'ясовується і модель системи подій (Actions), які дані класи (об'єкти) будуть підтримувати та обробляти.

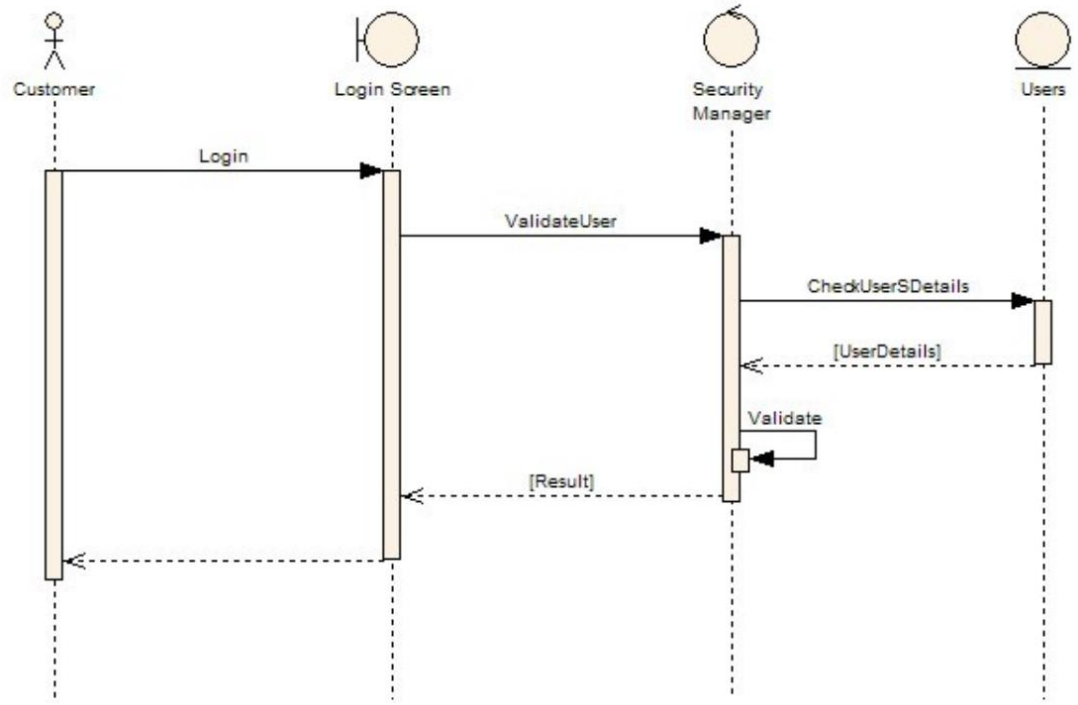


Рис. 2.24. Приклад діаграми послідовності

В наведеному на рисунку 2.24 прикладі діаграми послідовності демонструється послідовність дій, які ініціюється актором (деяким користувачем системи) ліворуч (customer). Ці дії

та відповідні повідомлення у самій програмі описують один з варіантів використання програми – процес авторизації (login). Повідомлення, якими при цьому обмінюються об'єкти програмної системи, стають ***методами відповідних класів***.

На діаграмі послідовності об'єкт відображається на вершині пунктирної вертикальної лінії. Ця вертикальна лінія називається ***лінією життя об'єкту***. Вона являє собою фрагмент життєвого циклу об'єкту в процесі взаємодії. Кожне повідомлення представляється у вигляді стрілки між лініями життя двох об'єктів.

Об'єкти діаграми

На діаграмі послідовності зображуються тільки ті об'єкти, які безпосередньо беруть участь у взаємодії. Ключовим моментом для діаграм послідовності є *динаміка взаємодії об'єктів у часі*. В UML діаграма послідовності має ніби як два виміри. Перший зліва направо у вигляді вертикальних ліній, кожна з яких відображає лінію життя окремого об'єкта, який бере участь у взаємодії. Крайнім зліва на діаграмі зображується об'єкт, який є *ініціатором взаємодії*. Правіше зображується об'єкт, який безпосередньо взаємодії з першим. Таким чином, всі об'єкти на діаграмі взаємодії утворюють деякий порядок, обумовлений черговістю або ступенем активності об'єктів при взаємодії один з одним.

Лінія життя об'єкту

Лінія життя об'єкту зображується вертикальною пунктирною лінією, асоційована з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі і, отже, потенційно, може брати участь у взаємодії. Якщо об'єкт існує в системі постійно, то і його лінія життя повинна продовжуватися по всій площині діаграми послідовності від верхньої до найнижчої її частини.

Як, видно, з рисунку, діаграма послідовності відповідає на питання – «**Як це повинно робитися в системі?**» і призначена для опису *поведінкових аспектів системи*.

Зазвичай, одна діаграма послідовності присвячена опису одного з варіантів використання, зазначеного у діаграмі прецедентів. Діаграми послідовностей можна використовувати для уточнення діаграм прецедентів, більш детального опису

логіки сценаріїв використання. Це ефективний засіб документування проєкту з точки зору сценаріїв використання.

Практичні завдання

Завдання 1. Виконати практичне завдання відповідно до інструкції.

Інструкція:

1. Виділити і клацнути правою кнопкою миші по назві створеної раніше моделі «<<useCaseModel>> Use Case Model - OWN». Вибрати пункт «CREATE DIAGRAM \ NEW SEQUENCE DIAGRAM» і назвати діаграму «Введення замовлення». Якщо є певні труднощі з відображенням діаграми в навігаторі, можна обрати режим відображення Diagram-Centric.

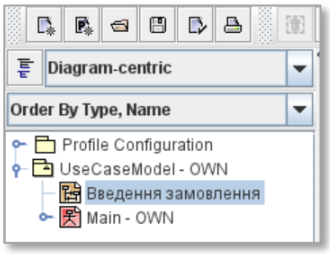


Рис. 2.25. Створена діаграма в навігаторі

2. Відкрити новостворену діаграму.
3. Додати до діаграми діючих осіб та об’єкти.
   – Перетягнути діючу особу «Продавець» із навігатора на діаграму (актора брати з діаграми Use Case Diagram).

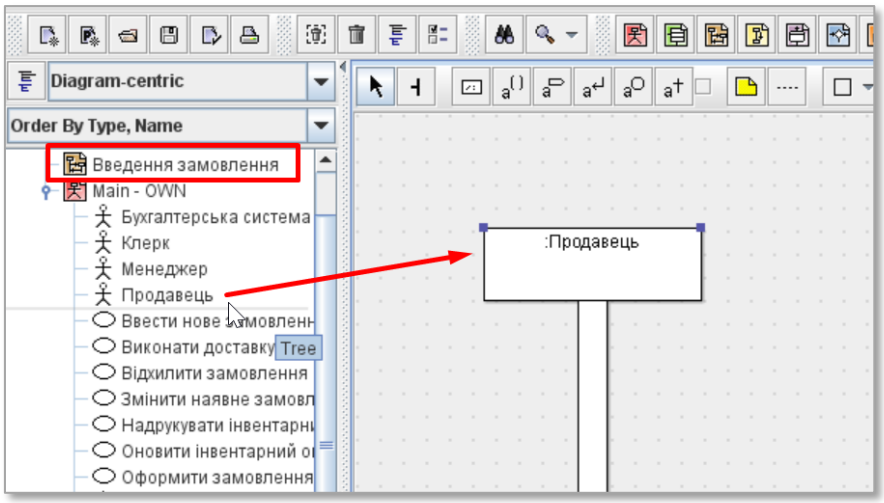


Рис. 2.26. Розташування актора «Продавець» на робочому полі

– Для створення об’єктів натиснути кнопку «New Classifier Role» панелі інструментів. Назвати об’єкт *Варіант замовлення*.

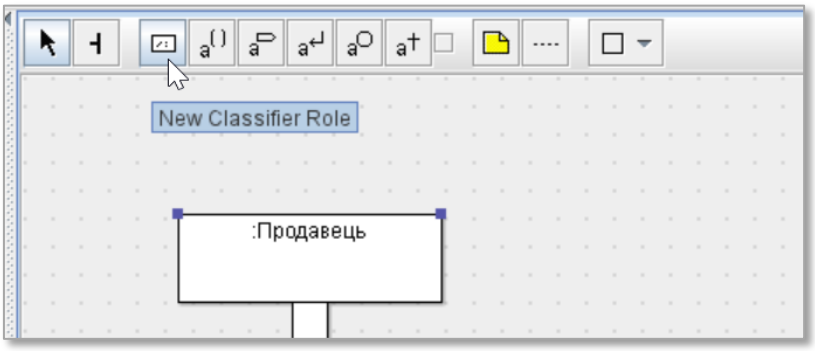


Рис. 2.27. Створення нового об’єкту

– Повторити аналогічні дії для об’єктів *Форма деталей замовлення, Замовлення №1*.

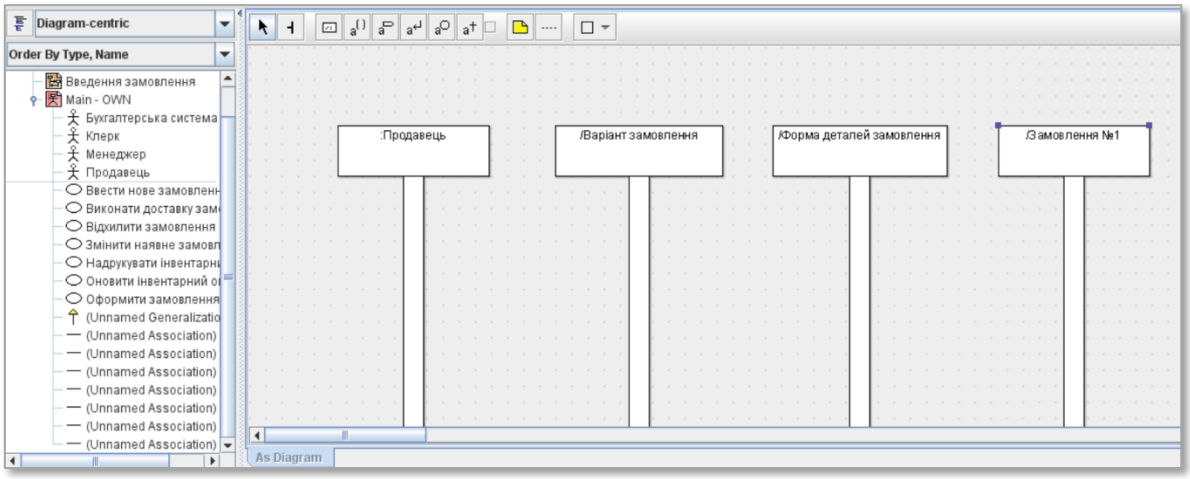


Рис. 2.28. Створення інших об’єктів

4. Додати повідомлення на діаграму.

– Натиснути кнопку **New Send Action** на панелі інструментів.

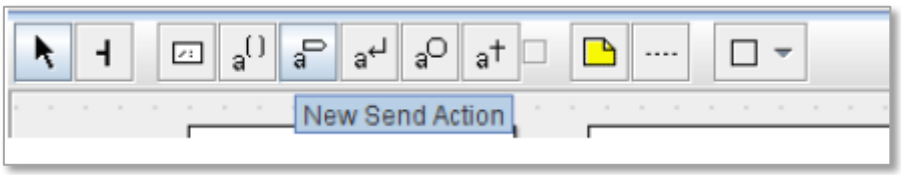


Рис. 2.29. Створення повідомлень на діаграмі

– Провести мишею від лінії життя (вертикальна широка лінія білого кольору) діючої особи «Продавець» до лінії життя об’єкту «Варіант замовлення».

– Виділивши повідомлення, ввести його ім’я – «Створити нове замовлення».



Рис. 2.30. Результат створення повідомлення

5. Виконавши аналогічні дії пункту 4, додати наступні повідомлення на діаграму:
   – «Відкрити форму» — між «Варіант Замовлення» та «Форма деталей Замовлення».
   – «Ввести номер Замовлення, замовника і число замовлених предметів» — між «Продавець» і «Форма деталей Замовлення».
   – «Зберегти замовлення» — між «Продавець» та «Форма деталей Замовлення».
   – «Створити порожнє замовлення» — між «Форма деталей Замовлення» та «Замовлення №1».
   – «Ввести номер замовлення, замовника і число замовлених товарів» — між «Форма деталей Замовлення» та «Замовлення №1».
   – «Зберегти замовлення» — між «Форма деталей Замовлення» і «Замовлення №1».
6. Діаграма послідовності без управляючих елементів виглядає таким чином:

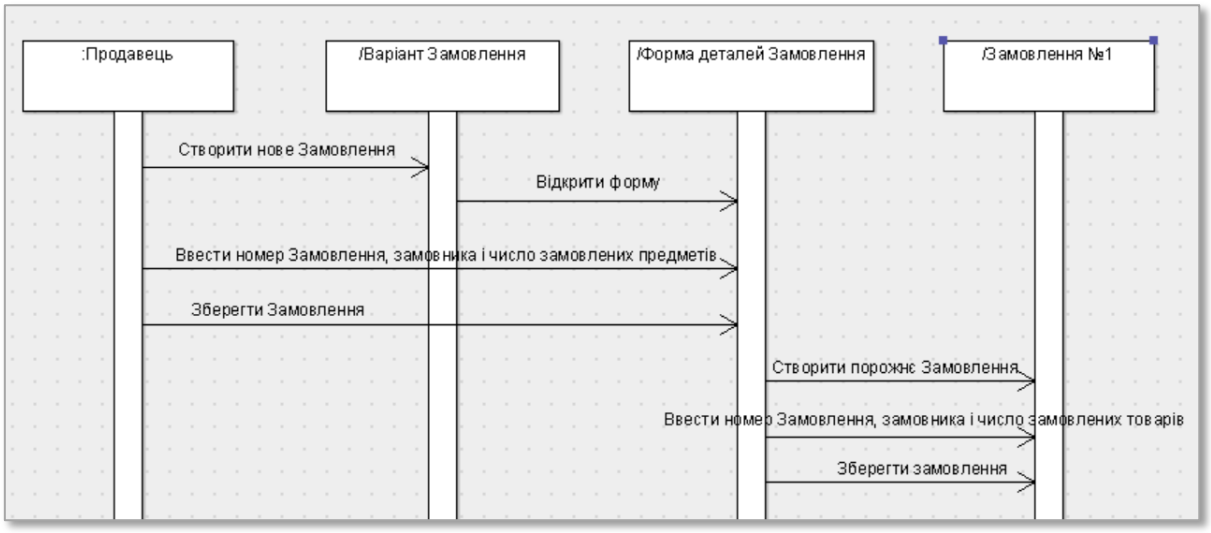


Рис. 2.31. Створення повідомлень на діаграмі

7. Додати на діаграму нові об’єкти:
   - «Менеджер замовлень» між об’єктами «Форма деталей Замовлення» і «Замовлення №1».
   - «Менеджер транзакцій» справа від об’єкту «Замовлення №1».
8. Згідно з новими доданими об’єктами необхідно визначити нову відповідність:
   - Помістити на діаграму нове повідомлення «Зберегти замовлення», розташувавши його під повідомленням 4, між «Форма деталей замовлення» та «Менеджер замовлень».
   - «Створити порожнє Замовлення» – між «Менеджер замовлень» та «Замовлення №1».
   - «Ввести номер Замовлення, замовника і число замовлених предметів» – між «Менеджер замовлень» та «Замовлення №1».
   - «Зберегти замовлення» – між «Менеджер замовлень» та «Менеджером транзакцій».
   - «Інформація про замовлення» – між «Менеджером транзакцій» та «Замовлення №1».
9. Для співвіднесення об’єктів з класами необхідно:

Створити діаграму класів:

- Виділити та клацнути правою кнопкою миші на назві створеної моделі «<<useCaseModel>> Use Case Model -

OWN», обрати пункт «CREATE DIAGRAM \ NEW CLASS DIAGRAM» і назвати діаграму «Введення замовлення».

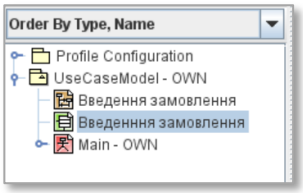


Рис. 2.32. Створення діаграми класів

– Відкрити діаграму, обравши її у провіднику.
– Створити у вікні діаграми класів (за допомогою кнопки ***New Class*** на панелі інструментів) класи *Варіант замовлення*, *Форма деталей Замовлення*, *Замовлення*, *Менеджер замовлень* і *Менеджер транзакцій*.

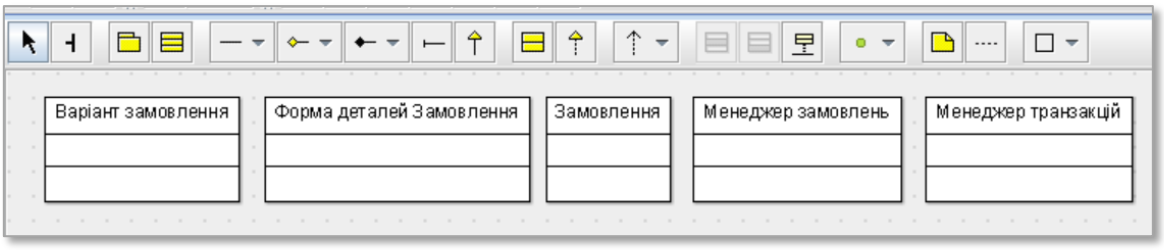


Рис. 2.33. Створення класів

Співвідношення об'єктів з класами:

– Обрати об'єкт *Варіант замовлення* у діаграмі послідовності «Введення замовлення».
– В панелі ***Properties*** відкрити список ***Base*** та натиснути на кнопку з плюсом.
– У вікні ***Add/Remove Bases*** обрати клас *Варіант замовлення* і додати його до списку ***Selected***.
– Натиснути кнопку ОК.

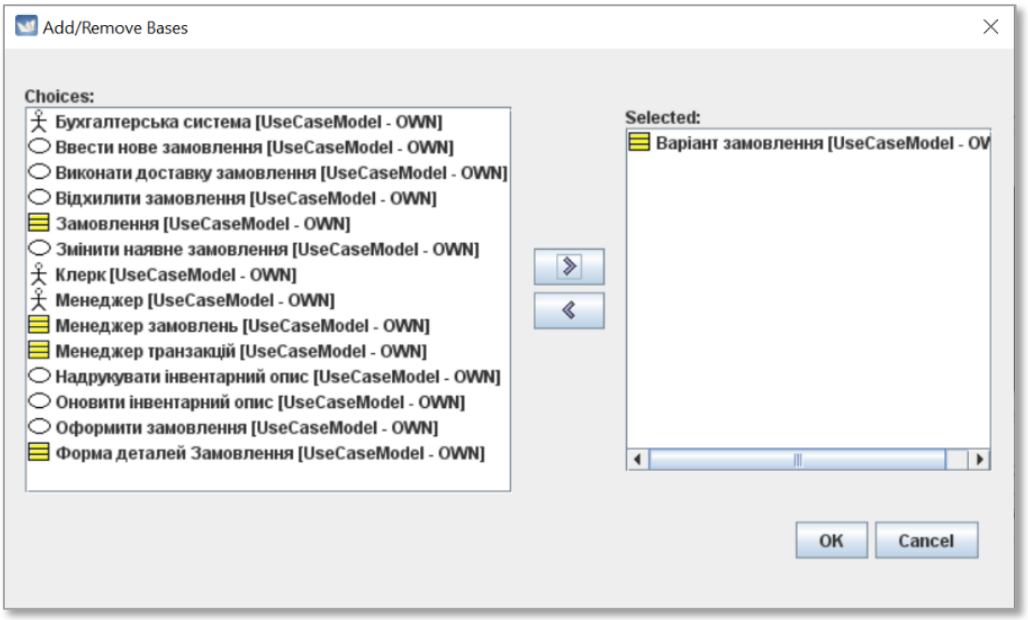


Рис. 2.34. Створення класів

10. Аналогічно пункту 9 співвіднести інші об'єкти з класами:
    - Клас «Форма деталей Замовлення» – з об'єктом «Форма деталей Замовлення».
    - Клас «Менеджер Замовлень» – з об'єктом «Менеджер Замовлень».
    - Клас «Замовлення» — з об'єктом «Замовлення №1».
    - Клас «Менеджер транзакціями» – з об'єктом «Менеджер транзакціями».
11. Для співвідношення повідомлень з операціями необхідно виконати наступні дії:
    - Натиснути на повідомлення «Створити нове замовлення».
    - У вікні Properties двічі клацнути в полі Action. У вікні, що відкрилось ввести *Створити ()*.

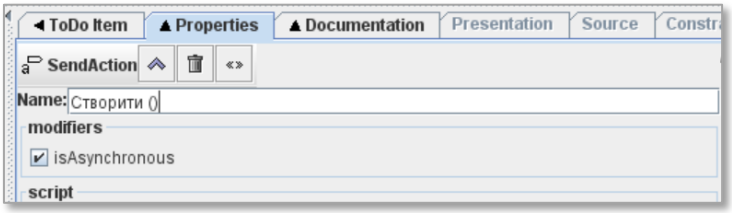


Рис. 2.35. Створення співвідношення «Створити()»

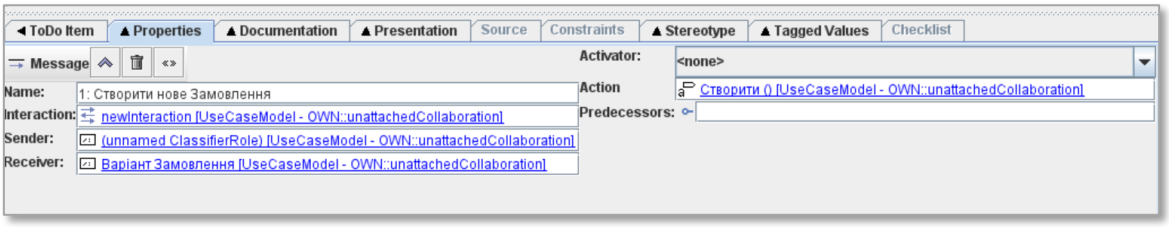


Рис. 2.36. Результат створення співвідношення «Створити()»

12. Повторивши етапи аналогічно пункту 11, співвіднести повідомлення з операціями:

- Повідомлення 2: «Відкрити форму» співвіднести з операцією «Відкрити()».
- Повідомлення 3: «Ввести номер Замовлення, замовника і число замовлених предметів» — з операцією «Ввести номер Замовлення, замовника і число замовлених предметів ()».
- Повідомлення 4: «Зберегти замовлення» — з операцією «Зберегти Замовлення()».
- Повідомлення 5: «Зберегти замовлення» — з операцією «Зберегти Замовлення()».
- Повідомлення 6: «Створити порожнє замовлення» – з операцією «Створити порожнє замовлення()».
- Повідомлення 7: «Ввести номер замовлення, замовника і число замовлених предметів» - з однойменною операцією.
- Повідомлення 8: «Зберегти замовлення» – з операцією «Зберегти замовлення()».
- Повідомлення 9: «Зібрати інформацію про замовлення» – з однойменною операцією.
- Повідомлення 10: «Зберегти інформацію про замовлення» з однойменною операцією.

13. Кінцевий вигляд створеної діаграми послідовності можна переглянути у додатку 11.

Завдання 2. Розробити діаграму послідовності приготування страви: включити плиту перед приготуванням; добре розігріти

масло; першим покласти масло, потім яйця, а не навпаки; сіль можна додати асинхронно, в процесі готування.

Для позначення дій, що повертають певний результат (Розігріто, Посолено, Вимкнути плиту) використовувати інструмент New Return Action.

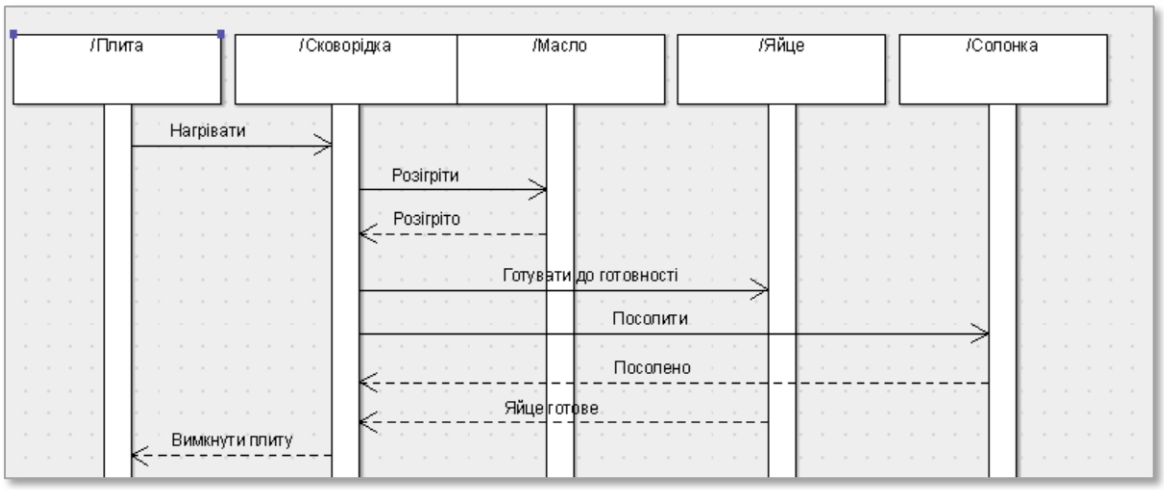


Рис. 2.37. Діаграма послідовності приготування страви

Контрольні запитання

1. Що таке діаграма послідовності? Що на ній відображено?
2. Що таке об’єкт діаграми послідовності?
3. Що таке повідомлення діаграми послідовності?
4. Що таке лінія життя об’єкту?
5. Яке основне питання вирішує діаграма послідовності?
6. Охарактеризуйте діаграму послідовності на прикладі авторизації користувача в системі (скільки об’єктів, які повідомлення, що відбувається в системі).

Тема 2.5. Розробка кооперативної діаграми

Практична робота №5

Мета: сформувати навички розробки коопертивної діаграми, проєктування структури поведінки системи в термінах взаємодії учасників проєкту.

Теоретичні відомості

Поняття кооперації (collaboration) є одним з фундаментальних понять у мові UML. Воно служить для

позначення безлічі взаємодіючих з певною метою об’єктів в загальному контексті модельованої системи.

Мета самої кооперації полягає в тому, щоб специфікувати особливості реалізації окремих найбільш значущих операцій в системі. Кооперація визначає структуру поведінки системи в термінах взаємодії учасників цієї кооперації.

Кооперативна діаграма (англ. collaboration diagram, інша назва – діаграми співробітництва) – це графічне представлення, що відображає потік подій в конкретному сценарії варіанту використання системи. Головна особливість діаграм кооперації полягає в можливості графічно подати не тільки послідовність взаємодії, але й усі ***структурні відносини між об’єктами***, які беруть участь у цій взаємодії.

Кооперативні діаграми відображають ту ж саму інформацію, що і діаграми послідовності. Але роблять вони це по іншому та з іншими цілями. Як і раніше, актори відображені у вигляді фігури. Якщо діаграма послідовності показує взаємодію між діючими особами і об`єктами у часі, то на кооперативній діаграмі зв`язок з часом відсутній.

Головна мета кооперативних діаграм полягає у представленні розкритих відношень між об’єктами, а також для виявлення завантаженості об’єкта з метою оптимального розподілу.

Діаграма кооперації насамперед відображає структуру взаємодії та містить такі елементи:

– Екземпляри акторів і класів, що беруть участь в реалізації варіанту використання;

– Асоціацію між екземплярами акторів і класів;

– Повідомлення, що передаються між екземплярами акторів і класів.

Кооперація може бути представлена на двох рівнях:

– рівні специфікації – показує ролі класифікаторів та ролі асоціацій у розглянутому взаємодії;

– рівні прикладів – вказує екземпляри і зв’язки, що утворюють окремі ролі в кооперації.

На відміну від діаграми послідовності, на діаграмі кооперації представлено тільки відносини між об’єктами, що грають певні

ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого виміру. Тому послідовність взаємодій і паралельних потоків може бути визначена за допомогою порядкових номерів.

Отже, якщо необхідно явно специфікувати взаємозв'язок між об'єктами в реальному часі, краще це робити на діаграмі послідовності.

За допомогою діаграми кооперації можна описати повний контекст взаємодій як своєрідний часовий "зріз" сукупності об'єктів, взаємодіючих між собою для виконання певного завдання або бізнес-цілі програмної системи.

Нижче наведено діаграму кооперації для системи «Магазин».



Рис. 2.38. Діаграма кооперації для системи «Магазин»

Ідея. Клієнт прийшовши в магазин обирає запчастину. Потім він замовляє її у працівника, той в свою чергу оформляє замовлення. Після того як працівник отримав запчастину зі складу, клієнт зобов'язаний підписати замовлення та розрахуватися на касі. Далі працівник видає запчастину клієнту.

Практичні завдання

Завдання 1. Виконати завдання за інструкцією.

Інструкція

1. В моделі «Use Case Model – OWN» створити новий тип діаграми «CREATE DIAGRAM \ NEW COLLABORATION DIAGRAM» і назвати її «Введення замовлення». Відкрити новостворену діаграму.

2. З діаграми *Main-OWN* (use case) перетягнути діючу особу *Продавець* на робоче поле.

3. За допомогою кнопки ***New Classifier Role*** на панелі інструментів створити на діаграмі *Варіант замовлення, Форма деталей замовлення, Замовлення №1, Менеджер замовлень, Менеджер транзакцій*.



Рис. 2.39. Створення об'єктів для діаграми кооперації

4. Далі необхідно пов'язати об'єкти відповідними асоціаціями:

- На панелі інструментів обрати кнопку ***New association role***.
- Зв'язати *Продавець* з об'єктом *Варіант замовлення*.

Аналогічно пункту 4 виконати встановлення зв'язків між:

- Діюча особа *Продавець* і об'єкт *Форма деталей замовлення*.
- Об'єкт *Форма деталей замовлення* і об'єкт *Варіант замовлення*.
- *Форма деталей замовлення* і *Менеджер замовлень*.
- *Менеджер замовлень* і *Замовлення №1.*
- *Форма деталей Замовлення* і *Замовлення №1*.
- *Замовлення №1* і *Менеджер транзакціями*.
- *Менеджер замовлень* і *Менеджер транзакціями*.

5. Обрати зв'язок між *Продавець* і *Варіант замовлення*. На панелі інструментів обрати кнопку *Add Message*. Додати повідомлення та ввести його ім'я *Створити нове замовлення().*

6. Аналогічно пункту 5 помістити на діаграму повідомлення:

- *Відкрити форму* – між *Варіант замовлення* і *Форма деталей замовлення*.
- *Ввести номер замовлення, замовника і число замовлених предметів* – між *Продавець* і *Форма деталей Замовлення*.
- *Зберегти замовлення* – між *Продавець* і *Форма деталей замовлення*.
- *Створити порожнє замовлення* – між *Форма деталей замовлення* і *Замовлення №1*.
- *Ввести номер замовлення, замовника і число замовлених предметів* – між *Форма деталей замовлення* та *Замовлення №1*.
- *Зберегти замовлення* – між *Форма деталей замовлення* і *Замовлення №1*.
- *Форма деталей Замовлення* і *Менеджер замовлень* для введення повідомлень *Зберегти замовлення*.
- *Зберегти нове замовлення* – між *Менеджер замовлень* і *Замовлення №1*.
- *Ввести номер замовлення, замовника і число замовлених предметів* – між *Менеджер замовлень* і *Замовлення №1*.
- *Зберегти замовлення* – між *Менеджер замовлень* і *Менеджер транзакцій*.
- *Інформація про замовлення* – між *Менеджер транзакцій* і *Замовлення №1*.

7. Тепер необхідно додати повідомлення «*Зберегти інформацію про замовлення в базі даних*» на рефлексивний зв'язок у об'єкта «*Менеджер транзакцій*». Щоб створити рефлексивний зв'язок потрібно обрати об'єкт, по бокам від об'єкту будуть показані маркери, серед яких необхідно обрати .

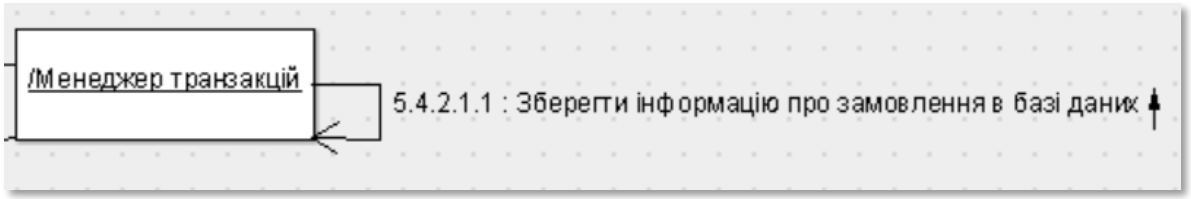


Рис. 2.40. Додавання повідомлення на рефлексивний зв'язок

8. Співвідношення об’єктів з класами:
   - Обрати об’єкт *Варіант замовлення*.
   - В панелі Properties відкрийте список Base.
   - Обрати кнопку з плюсом.
   - Обрати клас *Варіант замовлення* і натиснути на кнопку зі стрілкою для додання в список.
   - Натисніть кнопку ОК.
9. Аналогічно пункту 8 співвіднести інші об’єкти з класами.
10. Кінцевий вигляд створеної кооперативної діаграми можна переглянути у додатку 12.

Завдання 2. Побудувати кооперативну діаграму несанкціонованого входу в систему. На даній діаграмі представлені актори – користувач, підсистема захисту, система навчання, адміністратор з інформаційної безпеки.

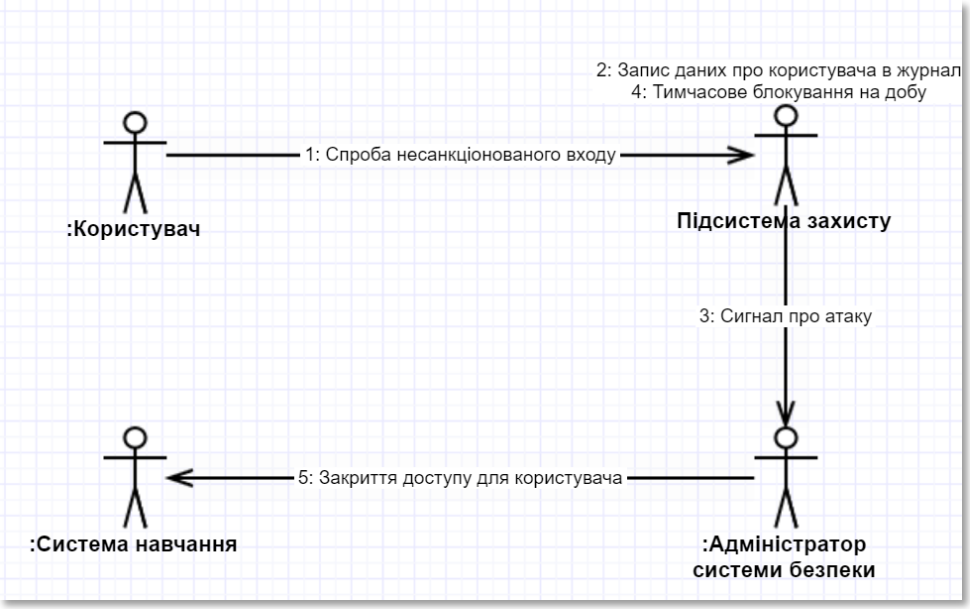


Рис. 2.41. Зразок діаграми «Несанкціонованого входу в систему»

Одне повідомлення відображає усі існуючі функції, які передаються від об`єкта до об`єкта. Є відповідні повідомлення зв`язку. На даній діаграмі представлено реакцію системи на спробу користувача отримати несанкціонований доступ до системи навчання.

Завдання 3. Побудувати кооперативну діаграму авторизації користувача в системі.

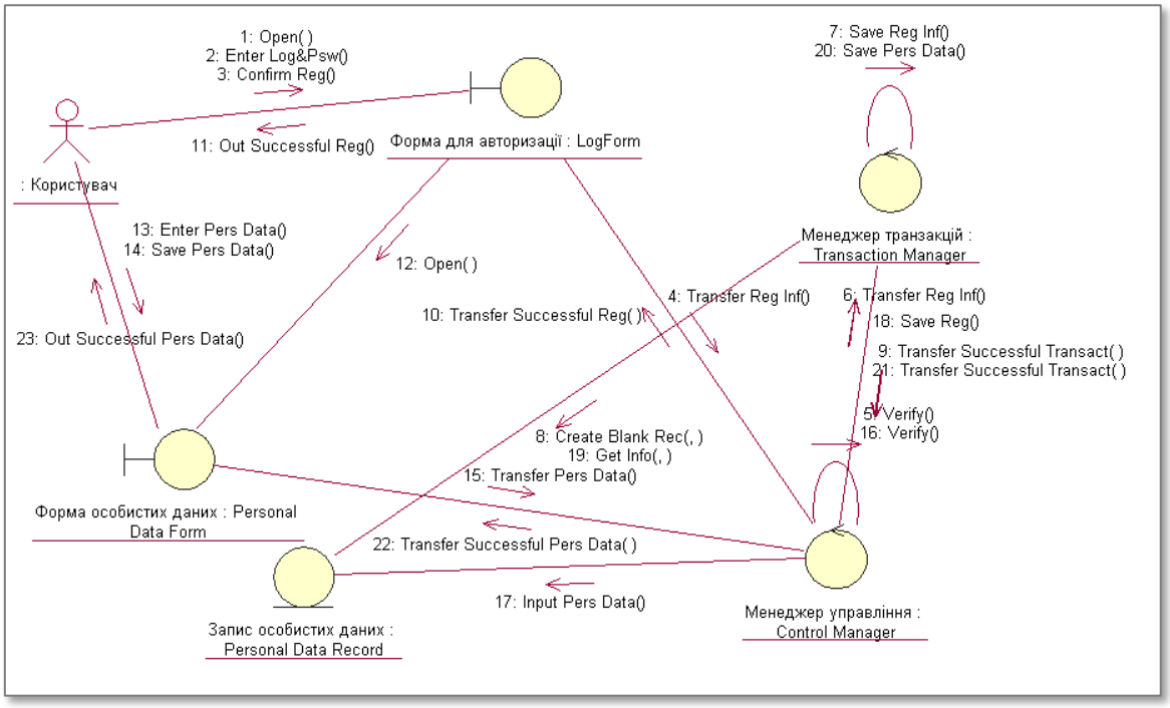


Рис. 2.42. Зразок діаграми «Авторизації користувача в системі»

Контрольні запитання

1. Для чого використовуються діаграми кооперації?
2. Що таке кооперативна діаграма?
3. З яких елементів складається діаграма кооперації?
4. Яка відмінність діаграм послідовності від діаграм кооперації?
5. На яких рівнях може бути представлена кооперація?
6. Якщо потрібно відобразити зв'язок об'єктів в реальному часі, яку діаграму варто використовувати?

Тема 2.6. Проєктування діаграм зміни станів об'єктів

Практична робота №6

Мета: сформувати навички проєктування діаграм стану для класів за допомогою програмного засобу ArgoUML.

Теоретичні відомості

Діаграма станів (від англ. Statechart Diagram) – діаграма, що визначає зміну станів об'єкту у часі, одна з діаграм моделювання

поведінки в UML. Подає об’єкт як автомат з теорії автоматів зі стандартизованими умовними позначеннями.

Наведемо приклад найпростішої діаграми запису та очистки даних.



Рис. 2.43. Зразок діаграми запису та очистки даних

Найпростіші позначення на діаграмі – це округлені прямокутники, які представляють собою стани, через які проходить об’єкт протягом свого ЖЦ. Стрілками показуються переходи між станами, які викликані виконанням методів об’єкту. Розглянемо детальніше елементи діаграми:

- Коло – позначає початковий стан.
- Коло з маленьким колом усередині – позначає кінцевий стан (якщо є).
- Округлений прямокутник – позначає окремий стан. Верхівка прямокутника містить назву стану, в середині може бути горизонтальна лінія, під якою записуються види активностей, що відбуваються в даному стані.
- Стрілка – позначає перехід. Назва події (якщо є), що викликає перехід, відзначається над/під стрілкою. Вартовий вираз може бути доданий перед «/» і укладений у квадратні дужки (назва_події), він означає, що перехід відбувається лише за умови істинності виразу. Якщо при переході відбувається певна активність, то вона додається після «/» (назва_події).
- Товста горизонтальна лінія, яка є точкою об’єднання або розгалуження переходів.

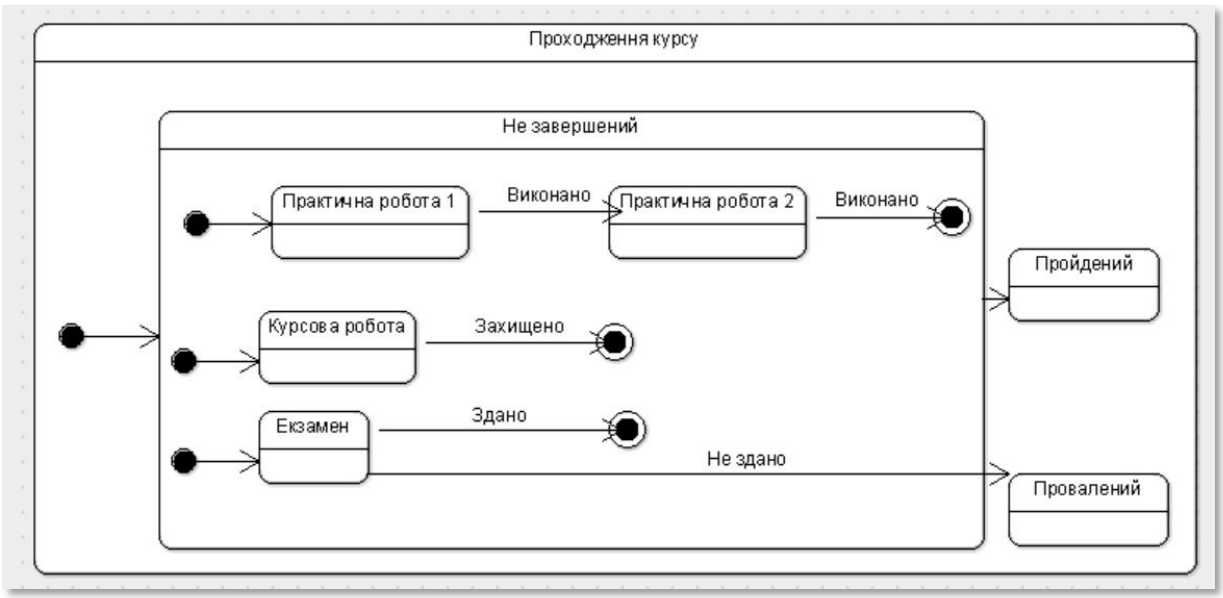


Рис. 2.44. Позначення на діаграмі стану

Розглянемо діаграму проходження академічного курсу студентом. На даній діаграми існує стан (суперстан) «Проходження курсу», що включає інші стани, один з яких містить також паралельні підстани. Для того щоб пройти курс, студент повинен виконати практичні роботи, захистити курсову роботу і скласти екзамен.

Практичні завдання

Завдання 1. Виконати завдання за інструкцією

Інструкція

1. Обрати клас «Замовлення» і для нього створити діаграму стану (обрати «CREATE DIAGRAM \ NEW STATECHART DIAGRAM»). Ввести ім'я діаграми «ЗАМОВЛЕННЯ – StatechartDiagram»).

2. Потрібно додати початковий та кінцевий стан. Для цього необхідно виконати наступні кроки:

– На панелі інструментів обрати пункт «New Initial» і помістити цей стан на діаграму.

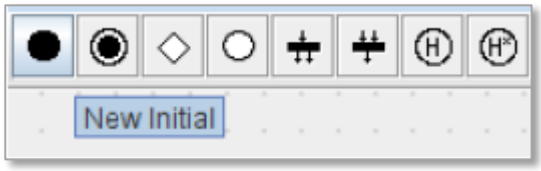


Рис. 2.45. Інструменти станів об'єктів

– На панелі інструментів обрати пункт «New Final State» і помістити цей стан на діаграму.

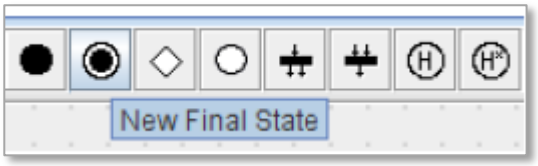


Рис. 2.46. Інструменти станів об'єктів

3. Для додання стану необхідно на панелі інструментів вибрати пункт «New Composite state» і помістити на діаграму. Назвати цей стан «Замовлення»

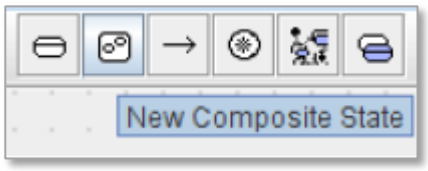


Рис. 2.47. Інструменти станів об'єктів New FComposite State

4. Для додання стану необхідно на панелі інструментів вибрати пункт «New Simple state» і помістити на діаграму.



Рис. 2.48. Інструменти станів об'єктів New Simple State

5. Аналогічно пункту 4 додати на діаграму наступні стани:
   - Відхилено.
   - Виконано.
   - Ініціалізація (розмістити всередині суперстану «Замовлення»).
   - Виконання замовлення призупинено (розмістити всередині суперстану «Замовлення»).

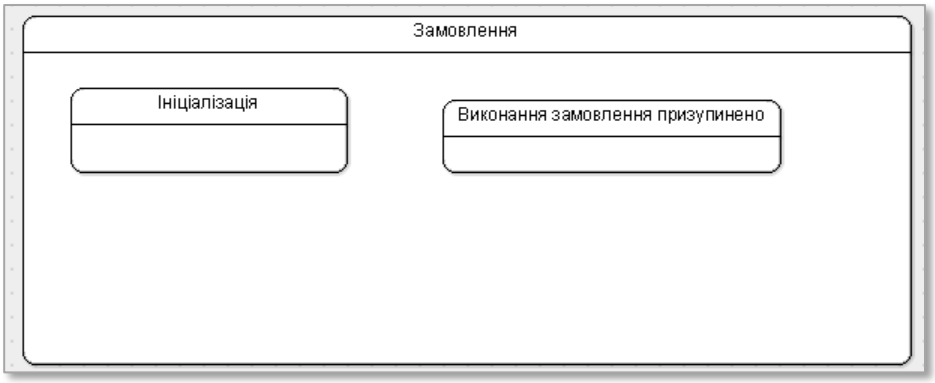


Рис. 2.49. Діаграма різних станів об'єкту «Замовлення»

6. Опис необхідно виконати наступним чином:
   - Виділити стан «Ініціалізація».

– У вікні Properties цього стану знайти пункт «EntryActions».
– Навпроти цього пункту натиснути на кнопку New call action:

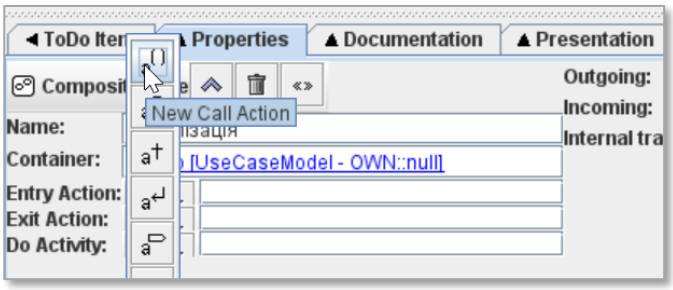


Рис. 2.50. Створення опису стану «Ініціалізація»

– Ввести ім'я дії «Зберегти дату замовлення» на діаграмі:

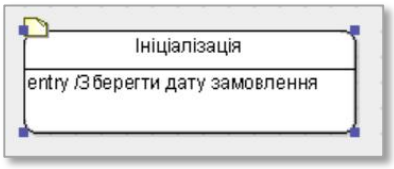


Рис. 2.51. Результат створення опису стану «Ініціалізація»

7. Виконавши аналогічні дії пункту 6 додати опис стану «Ініціалізація»:

– «Зібрати інформацію про клієнтів» для Do activity (Виконувати між входом та виходом).
– Додати до замовлення нові позиції для Do activity.

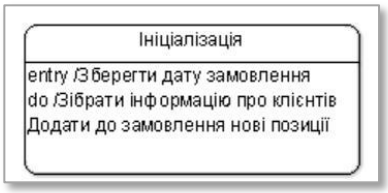


Рис. 2.52. Результат створення інших описів стану «Ініціалізація»

8. Виконавши аналогічні дії пункту 5 додати наступний опис станів:

– Стани «Відхилено» – «Зберегти дату відхилення», для «Exit Action»(На виході).
– Стани «Виконано» – «Виписати рахунок», вказавши дію «Exit Action»(На виході).



Рис. 2.53. Результат створення описів для стану «Відхилено» та «Виконано»

9. Додавання переходів виконується наступним чином:
   - На панелі інструментів вибрати пункт «New Transition».
   - Клацнути лівою кнопкою миші на початковому стані і провести лінію переходу до стану «Ініціалізація»:

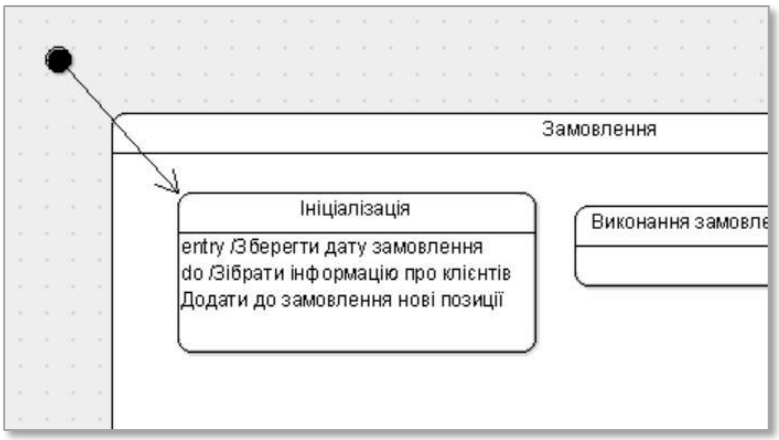


Рис. 2.54. Додавання переходів між станами

10. Виконавши аналогічні дії пункту 9, створити наступні переходи:
   - Від стану «Ініціалізація» до стану «Виконання замовлення призупинено».
   - Від стану «Виконання замовлення призупинено» до стану «Виконано».
   - Від суперстану «Замовлення» до стану «Відхилено».
   - Від стану «Відхилено» до кінцевого стану.
   - Від стану «Виконано» до кінцевого стану.

11. З’єднати стан «Виконання замовлення призупинено» з самим собою:



Рис. 2.55. Рефлексивний перехід «Виконання замовлення призупинено»

12. Додати опис переходів наступним чином:
   - Натиснути на перехід від стану «Ініціалізація» до стану «Виконання замовлення призупинено», відкриваючи вікно його властивостей.
   - В полі ім’я введіть «Виконати замовлення».

13. Виконавши дії аналогічні пункту 12 додати наступні описи:
   - «Відмінити замовлення» до переходу між суперстаном і станом «Відхилено».
   - «Додати до замовлення нову позицію» від стану «Виконання замовлення призупинено» до стану «Виконано». Виділити перехід мишкою і в властивостях переходу, в полі «Guard» натиснути кнопку «New Guard» і ввести «Не залишилось незаповнених позицій».

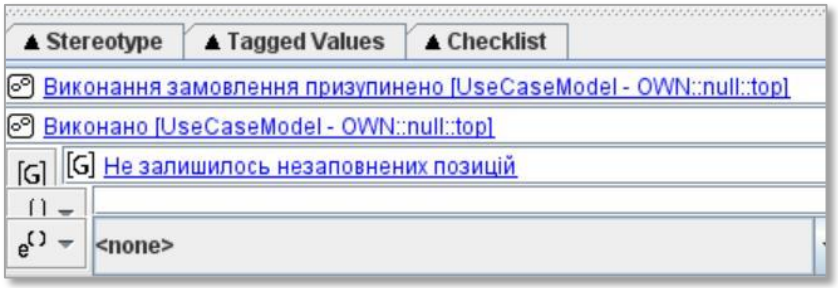


Рис. 2.56. Додавання наступних описів

14. Обрати рефлексивний перехід стану «Виконання замовлення призупинено» і у властивостях переходу, в полі «Guard» натиснути кнопку «New Guard» і ввести «Залишаються незаповнені позиції».

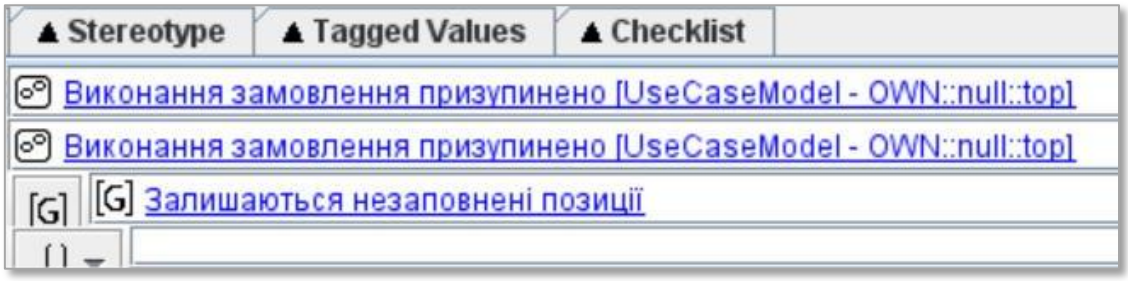


Рис. 2.57. Повідомлення «Залишаються незаповнені позиції»

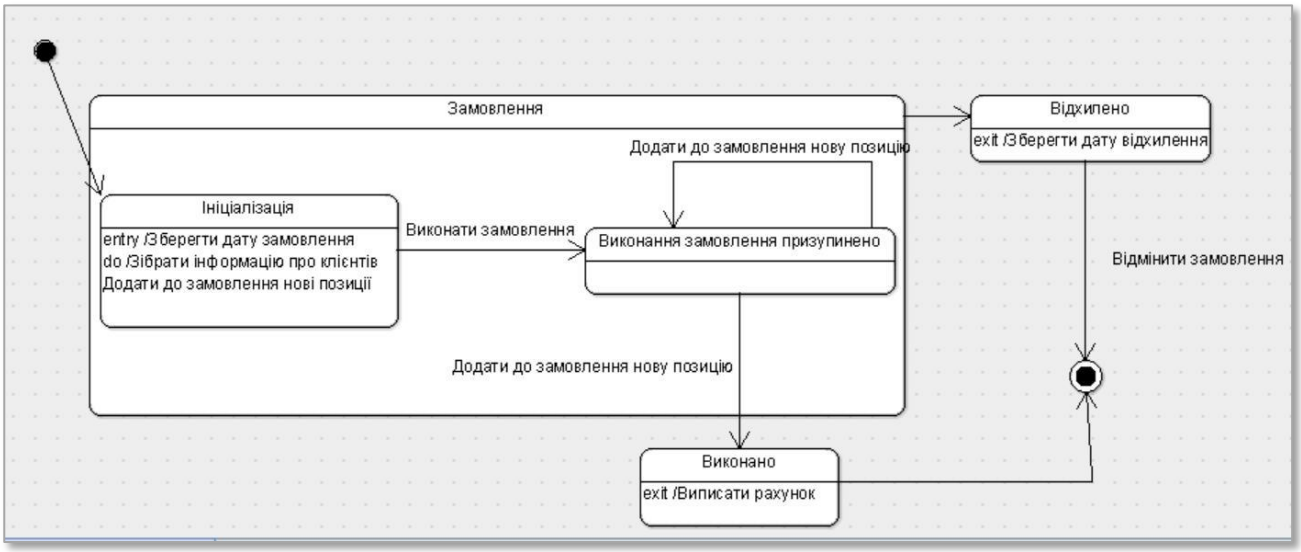


Рис. 2.58. Кінцевий вигляд діаграми станів для класу «Замовлення»

Завдання 2. Проаналізувати діаграму на рис. 2.59. Виконати дану діаграму в середовищі Argo UML.

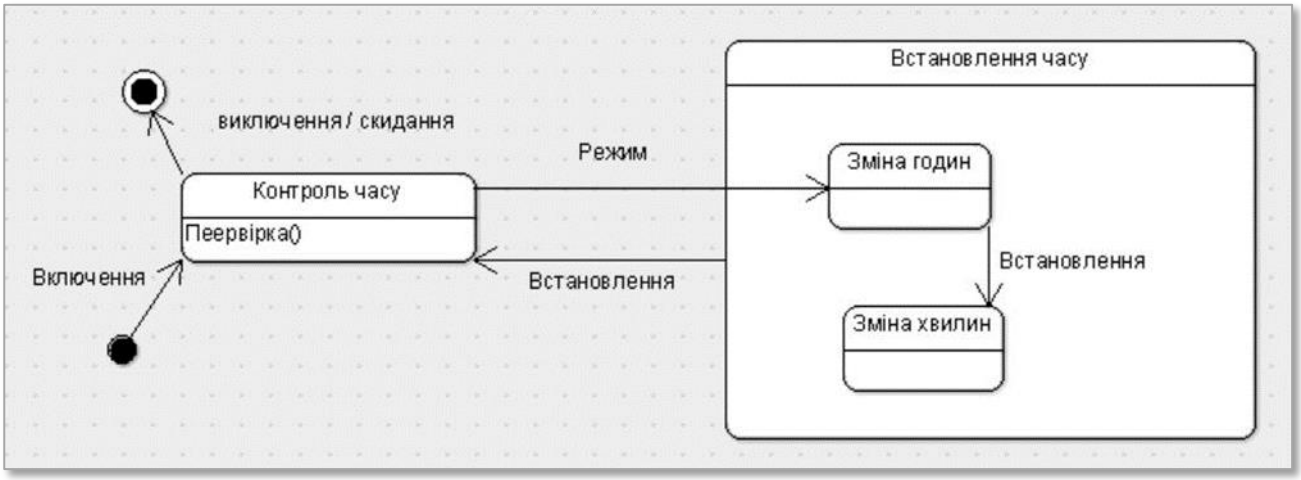


Рис. 2.59. Діаграма за зразком

Завдання 3. Переглянути відеопояснення https://www.youtube.com/watch?v=DWnbtodOVmM&ab_channel=PratikMatkar. Розробити діаграму станів за наданим відеопоясненням.

Контрольні запитання

1. Для чого використовуються діаграми станів?
2. З яких елементів складається діаграма станів?
3. Як позначається початок і кінець дії?
4. Що таке суперстан?
5. Як позначається окремий стан на діаграмі станів?
6. Що означає стрілка на діаграмі станів?

Тема 2.7. Створення діаграм UML в онлайновому середовищі Gliffy.

Практична робота №7

Мета: сформувати навички проєктування діаграм UML в онлайновому середовищі Gliffy.

Теоретичні відомості

У попередніх практичних роботах для створення діаграм використовувався програмний засбі ArgoUML. Проте існують інші середовища для проєктування діаграм UML.

Розглянемо онлайнову програму ***Gliffy***, яка використовується для створення схем, діаграм (BPMN, UML, UI Design, Venn diagrams, SWOT), графіків, планів приміщень тощо.

Gliffy має ряд ***переваги*** використання:

– Можливість створити власний аккаунт, всередині якого можна зберігати діаграми, які будуть зберігатися на сервері Gliffy. Також можна зберігати діаграми на жорсткий диск користувача, не створюючи аккаунт.

– Можливість працювати з будь-якого комп’ютера, що має доступ в мережу Інтернет на сайті http://www.gliffy.com.

– Можливість скористатися Gliffy API, щоб додавати на будь-який сайт діаграми з Gliffy шляхом простої вставки спеціального js-коду або ж вставки посилання на картинку на сервері. При зміні діаграми, картинка (або об'єкт, у разі використання javascript) автоматично зміниться на сайті.

– Можливість працювати над діаграмою спільно зі своїми колегами за допомогою функції collaborate.

– Можливість відкрити діаграми для загального доступу, тільки для деяких людей або тільки для себе.

– Можливість встановити Gliffy на власний сервер як плагін, тоді всі дані будуть зберігається на сервері користувача.

Gliffy містить ***шаблони***, які забезпечують розробку різних видів діаграм і схем. При створенні нової діаграми Gliffy пропонує вибрати шаблон. Шаблон представляє собою вже створену діаграму. Можна створювати і зберігати власні шаблони (фактично це майже теж саме, що відкрити свій файл).

Огляд шаблонів дозволяє швидко зрозуміти, які основні діаграми можна робити в цій програмі.

Перелік доступних шаблонів в Gliffy:

– Blank: порожній документ.
– Web site, software UI Design: вебсайт, дизайн інтерфейсу користувача ПЗ.
– Contact form wireframe: форма зворотного зв'язку.
– Wireframe – Product Page: сторінка продукту на сайті.
– Basic Sitemap і Gliffy Sitemap: класична і модифікована карти сайту.
– Venn Diagrams: діаграми Ейлера-Вена. На такий діаграмі об'єкти представляються у вигляді кіл; таким чином одні об'єкти можуть бути самостійними або належати іншим об'єктам, або перетинатися, причому на перетині можуть утворювати інші об'єкти. Розміри кіл можуть візуально виступати як кількісний фактор.
– Org Charts: для проектування організаційної структури підприємства (кількісний і якісний склад підрозділів, а також схематичне відображення порядку їх взаємодії між собою). За допомогою таких діаграм зручно представляти та ієрархії об'єктів, дерева.
– SWOT analysis: діаграми такого типу використовуються для оцінки факторів і явищ, що впливають на проект. Всі фактори поділяються на чотири категорії: strengths (сильні сторони), weaknesses (слабкі сторони), opportunities (можливості) і threats (загрози). Метод включає визначення мети проєкту та виявлення

внутрішніх і зовнішніх факторів, що сприяють її досягненню або ускладнюють його.

- Flowchart: класичні блок-схеми (в першу чергу використовуються для графічного представлення алгоритмів).
- Network Diagrams: для побудови діаграм, що описує топології мереж, а також для проектування полиць з обладнанням (network rack).

Також Gliffy підтримує інструментарій для розробки ***UML-діаграм***:

- Діаграма класів (Class diagram).
- Діаграма послідовності (Sequence Diagram).
- Діаграма використання (Use Case Diagram).
- Діаграма станів (UML State Machine).
- Deployment diagram (Діаграма розміщення) тощо.

Також існує інструментарій ***Floor Plan***, який використовується для проектування планів приміщень та інтер'єрів. Можна оперувати як ескізними чорно-білими елементами, так і кольоровими високо-деталізованими. Є велика бібліотека стандартних елементів (наприклад, можна працювати і з офісами і зі звичайними квартирами).

Також існує шаблон Business process для діаграм будь-яких бізнес-процесів.

Практичні завдання

Завдання 1. Проаналізувати середовище Gliffy у порівнянні з ArgoUML. Заповнити таблицю.

Таблиця 2.2

Критерії / Назва середовища	Інструменти створення діаграм							
	Класів	Станів	Використання	Діяльності	Кооперації	Розміщення	Послідовності	Інші особливості
ArgoUML								
Gliffy								

Завдання 2. Використовуючи інструментарій онлайнового середовища Gliffy створити діаграми стану та кооперації (можна використовувати приклади з попередніх практичних робіт).

Контрольні запитання

1. Яка особливість програмного засобу Gliffy у порівнянні з ArgoUML?
2. Які переваги використання засобу Gliffy?
3. Для чого призначено Gliffy API?
4. Що означає функція collaborate у програмі Gliffy?
5. Які шаблони можна використати у програмі Gliffy?
6. Які шаблони для створення UML-діаграм доступні в Gliffy?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bauer F. L. Software Engineering. Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971 / Editors : C. V. Freiman, J. E. Griffith, J. L. Rosenfeld. Amsterdam : North-Holland Publishing Company, 1972. Vol. 1. Foundations and Systems. P. 530-538.
2. Beck K., Andres C. Extreme Programming Explained: Embrace Change. 2nd edition. Addison-Wesley, 2004. 224 p.
3. Dorenskyi O. P. The Methodology of Evaluating the Test Cases Quality for Simple IT Monoprojects Software Testing . Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : тези доп. VIII Міжнар. наук.-практ. конф. (21–23 вер. 2016 р., м. Запоріжжя). Запоріжжя : ЗНТУ, 2016. С. 111–112.
4. History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 – 30, 1996 organized by William Aspray, Reinhard Keil-Slawik and David L. Parnas / Editors : Andreas Brennecke, Reinhard Keil-Slawik. – Paderborn, July 1997. III+57 p. URL: https://www.dagstuhl.de/Reports/96/9635.pdf.
5. ISO/IEC 15288 Systems and software engineering - System life cycle processes. [Чинний від 2008-03-18]. 70 с.
6. ISO/IEC/IEEE 12207:2017. Systems and software engineering. Software life cycle processes. [Valid from 2017-11-01]. ISO/IEC/IEEE, 2017. 145 p.
7. ISO/IEC/IEEE 15288: 2015. Systems and software engineering. System life cycle processes. [Valid from 2015-05-01]. ISO/IEC/IEEE, 2015. 108 p.
8. ISO/IEC/IEEE 24748-1:2018. Systems and software engineering. Life cycle management. Part 1: Guidelines for life cycle management. [Valid from 2018-11-01]. ISO/IEC/IEEE, 2018. 72 p.
9. Oettinger A. A. Letter to the ACM membership. Communications of the ACM. 1966. Vol. 9. No. 8. P. 545-546.
10. Software Engineering : Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968 / Editors : Peter Naur and Brian Randell. Brussels : Scientific Affairs Division, NATO, January 1969. 231 p.
11. Standard Glossary of terms used in Software Testing. ISTQB. URL: www.istqb.org/downloads/glossary.

12. SWEBOK. Guide to the Software Engineering Body of Knowledge. Version 3.0. IEEE Computer Society, 2014. 335 p.

13. Theory Z. How American Business Can Meet the Japanese Challenge. Perseus Publishing, 1981.

14. Velychko O., Gordiyenko T., Hrabovskyi O. Testing of measurement instrument software on the national level. Eastern-European Journal of Enterprise Technologies. Information and controlling systems. 2018. № 2/9 (92). P. 13–20.

15. Velychko O., Hrabovskyi O., Gordiyenko T. Peculiarities of Testing of Measurement Instrument Software in Ukraine. Sensors and Electronics Instrumentation Advances: Proceedings of the 5th International Conference on Sensors Engineering and Electronics Instrumentation Advances (SEIA' 2019), 25-27 September 2019, Adaje, Canary Islands (Tenerife), Spain. P. 241–243.

16. WELMEC 7.2. Software Guide (Measuring Instruments Directive 2014/32/EC). WELMEC, 2019. 132 p.

17. Величко О. М., Грабовський О. В., Гордієнко Т. Б. Особливості застосування V-моделі при розробленні та оцінюванні якості програмного забезпечення засобів вимірювальної техніки. Збірник наукових праць ОДАТРЯ. 2019. Вип. 1 (14). С. 6–11.

18. Величко О. М., Грабовський О. В., Гордієнко Т. Б. Особливості застосування спіральної моделі для випробувань програмного забезпечення засобів вимірювальної техніки. Збірник наукових праць ОДАТРЯ. 2019. Вип. 2(15). С. 42–49.

19. Говорущенко Т. О., Павлова О. О. Сучасні проблеми оцінювання початкових етапів життєвого циклу програмного забезпечення. Електротехнічні та комп'ютерні системи. 2018. №27 (103). С.165-175. URL: http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/etks_2018_27_22.pdf.

20. Голян В.В., Кравченко О.К. Порівняння моделей життєвих циклів програмного забезпечення з метою виявлення найефективнішого. Системи обробки інформації. 2019. Вип.2 (157). DOI: 10.30748/soi.2019.157.08.

21. Дідковська М.В. Y-подібна модель життєвого циклу програмного забезпечення. Доповіді Національної академії наук України. 2007. № 6. С. 32-35.

22. ДСТУ 9000:2015 (ISO 9000:2015, IDT). Системи управління якістю. Основні положення та словник термінів. Вид. офіц. Київ, 2016. 45 с. URL: https://khoda.gov.ua/image/catalog/files/%209000.pdf

23. ДСТУ ISO/IEC 14764:2014. Інженерія програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Технічне обслуговування (ISO/IEC 14764:2006, IDT) [Чинний від 2016-01-01]. Вид. офіц. Київ, 2016. 45 с.

24. ДСТУ ISO/IEC TR 24748-3:2016. Інженерія систем і програмного забезпечення. Керування життєвим циклом. Частина 3. Настанова щодо застосування ISO/IEC 12207 (Процеси життєвого циклу програмного забезпечення). (ISO/IEC TR 24748-3:2011, IDT). [Чинний від 2018-01-01]. Вид. офіц. Київ, 2016. 91 с.

25. ДСТУ ISO/IEC TS 24748-1:2018. Інженерія систем і програмних засобів. Керування життєвим циклом. Частина 1. Настанови щодо керування життєвим циклом. (ISO/IEC TS 24748-1:2018, IDT). [Чинний від 2018-08-15]. Вид. офіц. Київ, 2018. 72 с.

26. ДСТУ ISO/IEC/IEEE 12207:2018. Інженерія систем і програмних засобів. Процеси життєвого циклу програмних засобів. (ISO/IEC/IEEE 12207:2017, IDT). [Чинний від 2018-08-15]. Вид.офіц. Київ, 2018. 145 с.

27. Колеснікова К. В., Негрі А. О., Барчанова Ю. С., Олех Г. С. Підходи до управління програмними проектами у SWEBOK V3. III міжнародна науково-практична конференція «Управління розвитком технологій». URL: https://www.researchgate.net/ /publication/316493834.

28. Курлянський С. С., Шаров С. В. Проєктування як необхідний етап життєвого циклу програмного забезпечення. Матеріали І Всеукраїнської науково-практичної інтернет-конференції «Сучасні комп'ютерні та інформаційні системи і технології». 2020. С.89-92. URL: http://elar.tsatu.edu.ua/bitstream/ /123456789/12911/1/2_sharov_2020_89.pdf.

29. Левус Є., Мельник Н. Вступ до інженерії програмного забезпечення. Львів: Видавництво Львівської політехніки, 2017. 280 с.

30. Моделі життєвого циклу, принципи і методології розробки програмного забезпечення. URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html.

31. Нечай О. С., Сидоров М. О. Методи та засоби виявлення дефектів проєктування об'єктно-орієнтованого програмного забезпечення. Вісник Національного Авіаційного Університету. 2009. Т.40. №3. С. 200–205.

32. Офіційна сторінка організації ISO – розділ «Про нас». URL: https://www.iso.org/about-us.html.

33. Підтримка життєвого циклу програмного забезпечення поддержка жизненного цикла программного обеспечения SOFTWARE LIFE CYCLE SUPPORT. URL: https://cutt.ly/rT94qlC.

34. Стрюк А.М. Інженерія програмного забезпечення:перші 50 років становлення та розвитку. URL: http://ceur-ws.org/Vol-2292/paper01.pdf.

35. Технології програмування та створення програмних продуктів: конспект лекцій / укладач О. В. Алексенко. Суми: Сумський державний університет. 2013. 133 с. URL: https://cutt.ly/dT94yfW.

36. Томашевський О. М., Цегелик Г. Г., Вітер М. Б., Дудук В. І. Інформаційні технології та моделювання бізнес-процесів. К.: «Видавництво «Центр учбової літератури», 2012. 296 с.

37. Хлистун В.В. Порівняльний аналіз v- та y-подібної моделей життєвого циклу програмного забезпечення. Збірник тез IV Всеукраїнської науково-практичної конференції молодих науковців, 18.05.2017, м. Київ. С. 401-403. URL: http://dspace.kntu.kr.ua//jspui/bitstream/123456789/6536/3/ITconference_2017_p401_403.pdf.

ДОДАТКИ

Додаток 1

Питання для підсумкового контролю

1. Програмна інженерія: поняття та специфіка.
2. Становлення і розвиток інженерії ПЗ.
3. Головні умови успішної інженерії ПЗ.
4. Класифікація ПЗ.
5. Поняття стандарту проєктування ПЗ.
6. Міжнародні організації в галузі стандартизації.
7. Базовий стандарт розроблення ПЗ ISO 12207.
8. Характеристика областей знань інженерії ПЗ – SWEBOK.
9. Стандарт зрілості компанії-розробника ПЗ.
10. Поняття життєвого циклу програмного забезпечення.
11. Класичний життєвий цикл ПЗ – каскадна модель.
12. Нотації проєктування програмного забезпечення.
13. Процеси ЖЦ ПЗ за стандартом ISO/IEC 12207.
14. Спіральна модель.
15. Ітераційна та V-подібна моделі.
16. Інкрементна модель.
17. Промислові технології створення ПЗ.
18. Поняття та класифікація CASE-засобів.
19. Вимоги до ПЗ.
20. Професійні і етичні вимоги.
21. Кодекс етики IEEE-CS / ACM.
22. Проєкт і управління проєктом.
23. Менеджер ІТ-проєкту та його компетентності.
24. Команда ІТ-проєкту та рольова модель.
25. Моделі організації команд.
26. Спілкування в команді.
27. Поняття технічного завдання проєкту.
28. Функціональні і нефункціональні вимоги до ПЗ.
29. Моделювання в середовищі ArgoUML.
30. Поняття діаграм варіантів використання (прецедентів).
31. Проєктування діаграм послідовності в середовищі ArgoUML.
32. Особливості розробки коопертивних діаграм в середовищі ArgoUML.
33. Проєктування діаграм зміни станів об’єктів в системі.

Додаток 2

Тести для поточного та підсумкового контролю і самоконтролю

1. Як називається високорівневе подання структури системи і специфікація її компонентів?

□ Архітектура проекту
□ Стандарт проекту
□ Деталі проекту
□ Реалізація проекту

2. До якої категорії відносяться системні вимоги до програмних інтерфейсів, надійності, обладнанню?

□ Нефункціональні вимоги
□ Функціональні вимоги

3. Як називається заходи щодо аналізу сформульованих у вимогах атрибутів якості, функцій, структури ПЗ, з перевірки якості результатів проектування за допомогою метрик (функціональних, структурних тощо) і методів моделювання і прототипування?

□ Аналіз і оцінка якості проектування ПЗ
□ Нотації проектування
□ Базова концепція проектування ПЗ
□ Стратегія і методи проектування

4. Який процес ініціює ЖЦ програмної системи і визначає дії організації-покупця (або замовника) що отримує автоматизовану систему або сервіс?

□ Процес придбання
□ Процес постачання
□ Процес розроблення
□ Процес супроводження

5. Який процес визначає дії з передачі покупцю програмного про-дукту або сервісу і містить у собі такі види діяльності: підготовку пропозицій (від-повідей на запити), оформлення контракту, планування, виконання і контроль, про що повідомляє, аналіз і оцінку продукту; постачання і завершення робіт і постачання?

□ Процес постачання
□ Процес придбання

□ Процес експлуатації
□ Процес розробки

6. Як називається безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації?

□ Життєвий цикл ПЗ
□ Розробка ПЗ
□ Експлуатація ПЗ
□ Утилізація ПЗ

7. Як називається один з інструментів проектування архітектури ПЗ та типовий конструктивний елемент ПЗ, що задає взаємодію об'єктів (компонентів) проектованої системи, а також ролі і відповідальності виконавців?

□ Патерн
□ Каркас
□ Нотація
□ Інструкція

8. Як називається галузь інформаційних технологій, завданням якої є створення методів розроблення якісного програмного забезпечення в умовах обмежених часових та фінансових ресурсів?

□ Програмна інженерія
□ Комп'ютерні технології
□ Програмування
□ Інформатика

9. Як називається документ, який розробляється для загального уявлення про проект, а також певною мірою регламентує всі відносини замовника і виконавця?

□ Технічне завдання
□ Стандарт розробки ПЗ
□ Кодекс честі
□ Функціональні вимоги

10.Як називається засіб специфікації дій (функцій) програмної системи з точки зору отримання основними акторами деяких суттєвих для них результатів?

□ Прецедент
□ Інцидент
□ Актор

□ Діаграма

11.Як називається зовнішня до системи сутність (об'єкт), яка взаємодіє з нею для досягнення певних цілей (при UML-моделюванні)?

□ Актор
□ Прецедент
□ Асоціація
□ Діаграма

12.Як називається модель проектування ПЗ, яка забезпечує полегшений (рухомий) процес проектування та орієнтований на групи малого і середнього розміру, які проектують ПЗ в умовах невизначеності, або швидкої зміни вимог?

□ Інкрементна модель
□ Спіральна модель
□ Модель швидкої розробки ПЗ
□ Екстремальне програмування

13.Як називається модель розробки ПЗ, що є розвитком спіральної моделі і також базується на еволюційній стратегії конструювання?

□ Компонентно-орієнтована модель
□ Інкрементна модель
□ Еволюційна модель
□ Спіральна модель

14.Як називається модель, що будується у вигляді послідовності версій, але на початку визначені не всі вимоги?

□ Інкрементна модель
□ Еволюційна модель
□ Спіральна модель
□ Компонентно-орієнтована модель

15.Як називається модель розробки ПЗ, що забезпечує дуже короткий цикл розробки?

□ Компонентно-орієнтована модель
□ Спіральна модель
□ Інкрементна модель
□ RAD-модель

16.Як називається модель розробки ПЗ, що передбачає розробку продукту ітераціями, і кожна ітерація закінчується випуском працездатної версії?

□ Інкрементна модель
□ Еволюційна модель
□ Спіральна модель
□ Компонентно-орієнтована модель

17.Як називається модель, що представляється у вигляді чотирьох квадрантів, які визначають чотири види діяльності (планування, аналіз ризику, конструювання, оцінювання), що виконуються послідовно?

□ Інкрементна модель
□ Еволюційна модель
□ Спіральна модель
□ Компонентно-орієнтована модель

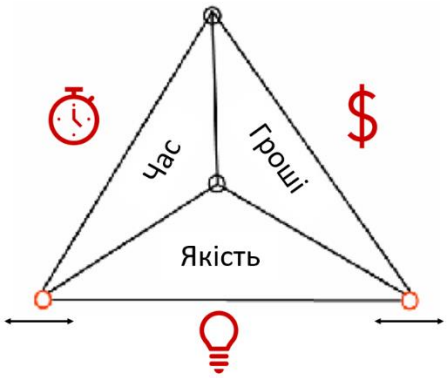
18.Як називається набір інструментів і методів програмної інженерії для проектування програмного забезпечення?

□ CASE-засоби
□ Кодекс етики IEEE-CS / ACM
□ Стандарти ПЗ ISO
□ BASE-технології

19.Як називається процес виконання програми для виявлення дефектів в функціях, логіці і формі реалізації програмного продукту?

□ Тестування
□ Супровід
□ Кодування

20.Як називається трикутник, зображений на рисунку?



□ Трикутник обмежень проєкту
□ Трикутник можливостей проєкту
□ Трикутник потреб проєкту
□ Трикутник бажань проєкту

21.Як називають множину комп'ютерних програм, процедур і пов'язаних із ними документації та даних?

□ Продукти інженерії програмного забезпечення
□ Правила інженерії програмного забезпечення
□ Об'єкти інженерії програмного забезпечення
□ Інструменти інженерії програмного забезпечення

22.Як називаються вимоги до програмного забезпечення, що описують цілі (завдання) користувачів системи, які повинні досягатися (виконуватися) користувачами за допомогою створюваної програмної системи?

□ Бізнес-вимоги
□ Вимоги користувача
□ Функціональні вимоги
□ Системні вимоги

23.Як називається група властивостей програмного забезпечення, які повинні бути належним чином представлені в ньому для вирішення конкретних практичних завдань?

□ Програмні вимоги
□ Інтерфейс користувача
□ Функціональність
□ Результативність

24.Які CASE-засоби класифікують за рівнем застосування?

□ Upper CASE-засоби аналізу вимог
□ Middle CASE-засоби проектування
□ Low CASE-засоби розробки додатків
□ Допоміжні засоби тестування
□ Спеціалізовані засоби для проектування баз даних

25.Які є професійні та етичні вимоги до розробника ПЗ?

□ Конфіденційність
□ Компетентність
□ Захист інтелектуальної власності
□ Адекватна документація
□ Час виклику
□ Безпечність
□ Відповідність проектної документації вихідному коду
□ Відмовостійкість

26.Які особливості встановлення і функціонування програмного забезпечення ArgoUML?

□ Наявність середовища виконання Java
□ Робота в командному рядку
□ Обмеження на кількість проєктів
□ Робота з емулятором Java

27.Які формальні мови специфікацій і проектування відносять до нотацій?

□ UML (Unified Modeling Language)
□ HTML (HyperText Markup Language)
□ DML (Data Manipulation Language)
□ SQL (Structured query language)

28.*Якій моделі життєвого циклу відповідає технологія швидкої розробки застосунків (Rapid Application Development (RAD))?*

□ Спіральна модель
□ Каскадна модель
□ Ітераційна модель
□ V-подібна модель
□ Інкрементна модель

29.Яка діаграма також називається діаграмою прецедентів?

□ Діаграма варіантів використання
□ Діаграма класів
□ Діаграма послідовності
□ Діаграма станів
□ Діаграма діяльності

30.Яка міжнародна організація розробляє стандарти практично в усіх галузях людської діяльності, в тому числі і в галузі інформаційних технологій?

□ ISO
□ ACM
□ SEI
□ PMI

31.Яка модель використовує так званий метод "живої" розробки вебзастосунків?

□ Модель екстремального програмування
□ Каскадна модель
□ Спіральна модель
□ V-подібна модель

32.Який вид тестування передбачає перевірку відповідності прикладної програми до вимог графічного інтерфейсу?

□ Функціональне тестування
□ Тестування інтерфейсу користувача
□ Тестування швидкості й надійності
□ Тестування безпеки

33.Який вид тестування передбачає перевірку поводження системи (або окремих її частин) при одночасному використанні великої кількості користувачів?

□ Тестування швидкості й надійності
□ Тестування безпеки
□ Тестування інтерфейсу користувача
□ Функціональне тестування

34.Який тип UML-діаграми відображає часові особливості передачі і прийому повідомлень об'єктами?

□ Діаграма послідовності
□ Діаграма діяльності
□ Діаграма варіантів використання
□ Діаграма класів
□ Діаграма станів

35.Який тип UML-діаграми визначає зміну станів об'єкту у часі і є однією з діаграм моделювання поведінки системи?

□ Діаграма станів
□ Діаграма варіантів використання
□ Діаграма послідовностей
□ Кооперативна діаграма

36.Який тип UML-діаграми дає змогу представити структурні відношення між об’єктами, а також виявити завантаженість об’єкта з метою оптимального розподілу?

□ Кооперативна діаграма
□ Діаграма варіантів використання
□ Діаграма класів
□ Діаграма послідовності
□ Діаграма розміщення

Додаток 3

Критерії оцінювання результатів навчання

Критерій	Результат навчання
90–100	Здатність аналізувати складні об'єкти та системи; обирати і застосовувати основні методи проєктування ПЗ; проводити аналіз дефектів, помилок та ризиків у ЖЦ ПЗ; вміння визначати вимоги до ПЗ; складати ТЗ на основі визначених вимог; знання мов моделювання ПЗ та середовищ розробки моделей даних.
82–89	Обізнаність в галузі основних стандартів інженерії ПЗ; здатність використовувати методи проєктування ПЗ та мови моделювання ПЗ.
75–81	Розуміння базових принципів організації та функціонування складних систем; вміння аналізу вимог до ПЗ та їх опису; вміння розробляти стандартну модель ПЗ.
69–74	Здатність відтворювати знання основних методів дослідження та аналізу складних об'єктів та систем; володіння основним синтаксисом мови моделювання ПЗ та застосування їх для проєктування нескладних систем.
60–68	Відтворення знань на початковому рівні щодо проектування ПЗ; вміння спроєктувати розробити елементарний фрагмент нескладної програмної системи.
35–59	Володіння навчальним матеріалом на рівні елементарного розпізнавання і відтворення окремих фактів в галузі програмної інженерії, що позначаються окремими словами чи реченнями.
1-34	Володіння матеріалом на рівні окремих фрагментів в галузі програмної інженерії, що становлять незначну частину навчального матеріалу.

Додаток 4

Зразок технічного завдання

Технічне завдання для розробки вебсайту

У рамках проекту Фонду демократії ООН (UNDEF) «Підлітки змінюють громади в Україні через громадянське суспільство», що виконується Центром «Ейдос» спільно з МАН України, БФ «Агенти змін», МІ «Серце в долоньках», оголошується тендер на розробку сайту для проекту «Агенти змін. Діти».

Мета: розробка вебсайту для проекту «Агенти змін. Діти».

Цільова аудиторія, на яку розрахований сайт:

1) майбутні та діючі учасники проекту "діти-агенти" – матимуть швидкий доступ до необхідної інформації про проект, етапи, дедлайни, вимоги тощо;

2) вчителі / батьки – інструменти неформальної громадянської освіти та ідеї;

3) представники громадських організацій, шкільних та молодіжних рад, інші – дізнатися про топ-проекти по всій Україні та історії успіху «агентів».

Ключовий момент: простий та зручний у користуванні, привабливий для підлітків інтерфейс.

Потрібна технологічна підтримка сайту після його створення.

Центр «Ейдос» буде самостійно наповнювати сайт контентом.

Логотип: в наявності. Попередньо визначені корпоративні стандарти (побажання до кольорів та шрифтів), однак, відкриті до пропозицій.

Тип сайту: сайт-візитівка з елементами новинного сайту.

Мова сайту: українська та англійська (спрощена версія: про проект, контакти).

Опис сторінок

Головна сторінка

Запропонована назва сайту (доменне ім’я):

agenty.org.ua / agentyzmindity.org.ua / teenagents.org.ua

Шапка (header)

1. Елемент фірмового стилю сайту (логотип Школа «Агенти змін») – зліва.

2. Назва порталу.

3. Пошукове поле.

5. Логотип донора UNDEF (bilingual) – справа.

Слайдер (фото з основними новинами)

Структура контектних блоків

Про проект

- Мета, цілі та очікувані результати.
- Історія створення.
- Як проходить навчання.
- Партнери.

Кожен підрозділ – в межах однієї сторінки, яка може рухатись вниз після натискання.

Проекти учасників

- Опис успішних проектів.
- Інтерв'ю.
- Про них говорять (посилання на інші медів-ресурси).
- Всі проекти (тут будуть зібрані за стандартизованим профайлом всі проекти Агентів. Має бути функція пошуку: за роком навчання, прізвищем, регіоном, тощо. Сюди ж буде переносити користувача, коли він на інтерактивній мапі натисне на овал – невелике вікно із короткою інформацією: Учасник, Назва проекту, Короткий опис, Детальніше).

Контакти

Навчальні матеріали

- Статті.
- Посібники та дослідження.
- Методологія Школи.
- Відео.

Новини (на фото-слайдері та в меню).

Блок із 4-х основних розділів, що супроводжується окремою ілюстрацією (змінюється при натисканні кожного з чотирьох блоків):

1. Інфо для агентів (як стати учасником програми, конкурс есе, етапи школи, тощо).

2. Інфо для вчителів.

3. Топ історії успіху.

4. Цікаві факти.

Інтерактивна мапа проектів

Пошук за класифікатором: рік, регіон, прізвище, напрям проекту (культура, спорт, освіта, волонтерство).

Зображення карти України із позначеними «точками» – проектами, що були реалізовані в населених пунктах. Коли користувач натискає кружечок, з'являється віконечко із короткою інформацією: Учасник, Назва проекту, Короткий опис, Детальніше...). При виборі опції «Детальніше» – відкривається сторінка з детальним описом проекту з фотографіями.

Нижнє поле сторінки:

6. Контакти (меню та footer).

7. Піктограми соцмереж (Facebook, Instagram, Telegram).

8. Донор (блок на головній и footer).

9. Політика використання ресурсу (footer).

10. Про нас (footer).

11. RSS (footer).

12. Пошук по сайту (верхнє поле).

13. Лого партнерів.

14. Кнопка підтримати проект.

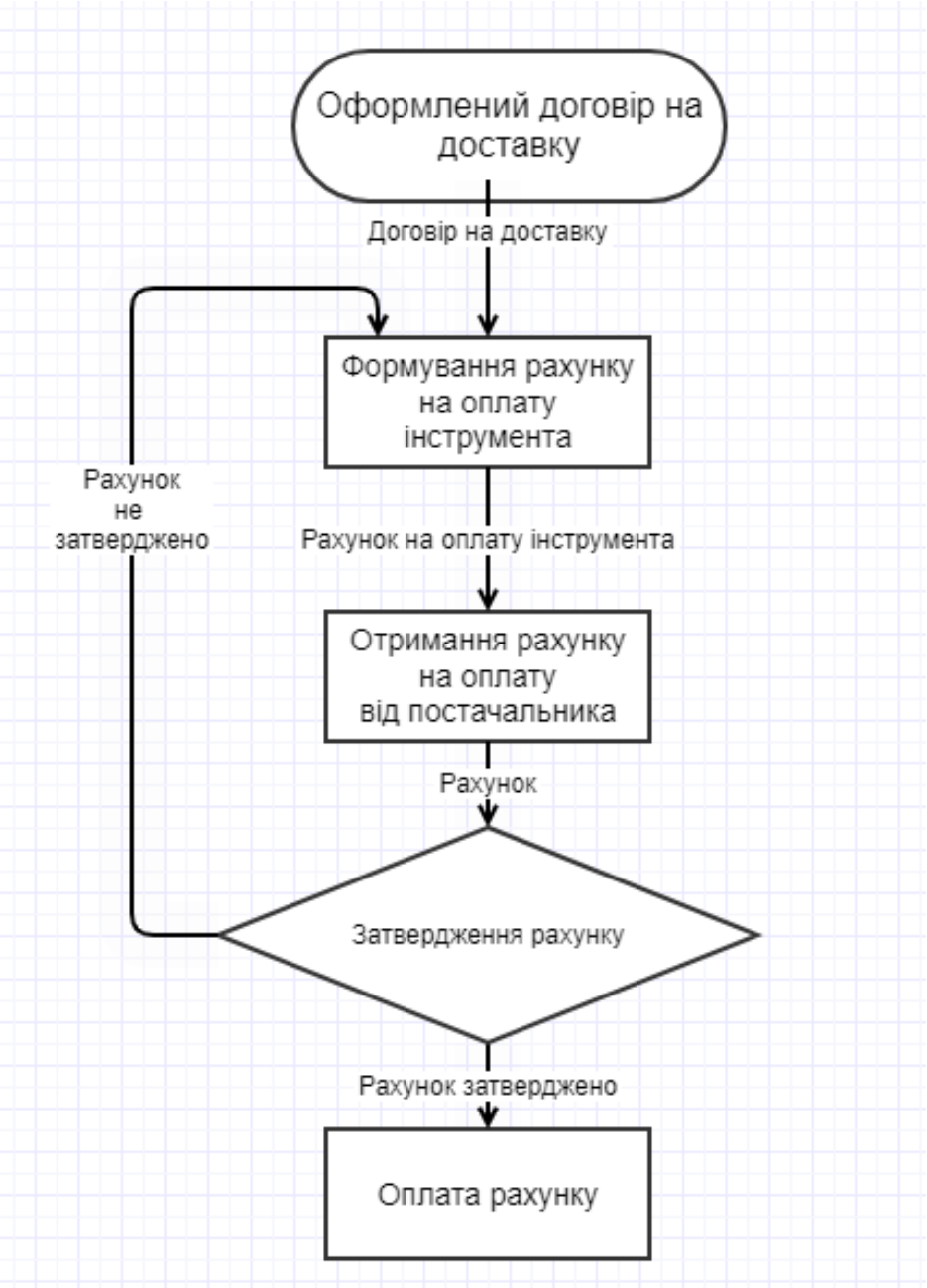
Додаток 5

Зміст основних процесів ЖЦ ПЗ відповідно до ISO/IEC 12207

Процес (виконавець)	Дії	Вхід	Результат
Купівля (замовник ПЗ)	Ініціювання Підготовка вимог заявки Підготовка угоди Контроль діяльності постачальника Приймання ПЗ	Рішення про початок впровадження ПЗ Результати дослідження діяльності замовника Результати аналізу ринку ПЗ План постачання/ розроблення Комплексний тест ПЗ	Техніко-економічне обгрунтування впровадження ПЗ Технічне завдання на ПЗ Угода на постачання/ розроблення Акти приймання етапів роботи Акт приймально-передавальних випробувань
Постачання (розробник ПЗ)	Ініціювання Відповідь на замовлення Підготовка угоди Планування виконання Постачання ІС	Технічне завдання на ПЗ Рішення про участь у розробленні Результати тендера Технічне завдання на ПЗ План керування проєктом Створена ПЗ та документація	Рішення про участь в розробленні Комерційна пропозиція/ конкурсна заявка Угода про постачання/ розроблення План керування проєктом Реалізація/ коригування Акт приймально-передавальних випробувань
Розроблення (розробник ПЗ)	Підготовка Аналіз вимог до ПЗ Проєктування архітектури ПЗ Розроблення вимог до ПЗ Проєктування архітектури ПЗ Детальне проєктування ПЗ Кодування та тестування ПЗ Інтеграція ПЗ та кваліфікаційне тестування ПЗ Інтеграція ПЗ та кваліфікаційне тестування ПЗ	Технічне завдання на ПЗ Технічне завдання, модель ЖЦ Технічне завдання на ПЗ Підсистеми ПЗ Специфікації вимог до компонентів ПЗ Архітектура ПЗ Інформація про деталі проєктування ПЗ План інтеграції ПЗ, тести Архітектура ПЗ, документація на ПЗ, тести	Модель ЖЦ, стандарти розроблення План робіт Склад підсистем, компоненти обладнання Специфікації вимог до компонентів ПЗ Склад компонентів ПЗ, інтерфейси з БД, план інтеграції ПЗ Проєкт БД, специфікації інтерфейсів між компонентами ПЗ, вимоги до тестів Тексти модулів ПЗ, акти автономного тестування Оцінка відповідності ПЗ вимогам ТЗ Оцінка відповідності ПЗ, БД, технічного комплексу та комплекту документації вимогам ТЗ

Додаток 6

Приклад фрагменту поведінкової нотації



Додаток 7

SQI: 34 компетенції IT менеджера

1. Методика розробки продукту:

- Процеси оцінювання – визначення критеріїв для відбору.
- Знання стандартів процесу.
- Визначення продукту – ідентифікація клієнтського середовища і вимог, що висуваються до продукту.
- Оцінка альтернативних процесів.
- Управління вимогами – моніторинг зміни вимог.
- Управління субпідрядниками – планування, управління і здійснення контролю.
- Початкова оцінки – оцінка ступеня складності, ризиків, витрат і створення графіків.
- Відбір методів та інструментів – визначення процесів відбору.
- Упорядкування процесів – модифікація стандартних процесів з метою задоволення вимог проєктів.
- Відстеження якості продукту – контроль якості в процесі розробки продукту.
- Розуміння дій щодо розробки продукту – вивчення циклу розробки ПЗ.

2. Навички управління проєктами:

- Створення структури покрокового переліку робіт.
- Документування планів – ідентифікація ключових компонент.
- Оцінка вартості – вартості завершення проєкту.
- Оцінка трудовитрат – необхідних для завершення проєкту.
- Менеджмент ризиків – ідентифікація, визначення впливу, обробка ризиків.
- Відстеження процесу розробки – контроль процесу розробки.
- Складання графіка – розробка графіка і ключових стадій проєкту.
- Вибір метричних показників.
- Відбір інструментів менеджменту проєкту – вибір методик та інструментів.
- Відстеження процесів – моніторинг сумісності членів команди.
- Відстеження ходу розробки продукту – моніторинг ходу розробки за обраними метричних показниками.

3. Навички управління персоналом:

- Оцінка продуктивності – оцінка дій команди, спрямованих на підвищення її продуктивності.

- Питання інтелектуальної власності – розуміння ступеня впливу критичних проблем.
- Організація ефективних зустрічей – планування і проведення.
- Взаємодія та спілкування – з розробниками, керівництвом і іншими командами.
- Лідерство – навчання проєктних команд для досягнення оптимального результату.
- Управління змінами – забезпечення ефективного управління змінами.
- Успішне ведення переговорів – вирішення конфліктів та ведення переговорів.
- Планування кар'єрного росту – структурування та управління ходом реалізації кар'єри.
- Ефективне уявлення – використання письмових і усних навичок.
- Набір персоналу – вербування і співбесіда з членами команди.
- Відбір команди – висококваліфікованих фахівців.
- Створення команди – формування, керівництво і підтримка ефективної команди.

Додаток 8

Приклад аналізу сутностей сайту туристичної бази

З розповідей замовника стає зрозуміло, що є наступні інформаційні сутності:

1) Будинки.
2) Номери в будинках.
3) Типи (категорії) номерів (простий, зручний, тощо).
4) Періоди часу (зима-літо).
5) Ціни, які залежать від пори року і номера.
6) Бронь.

Ці сутності доцільно описати в 5-му розділі технічного завдання «Дані і списки».

Інформаційна сутність «Номер»

Номер – це кімната в будинку, з атрибутів має «номер» (номер, в сенсі назва), ідентифікатор (наприклад, «номер 205») і категорію.

Інформаційна сутність «Будинок»

За великим рахунком, це просто сукупність номерів. Тобто він має кімнати, і все. Немає фотографій, поверхів, координат розташування. Можливо лише назва типу «Вітерець» або «Сонечко» або просто «Будинок 2». В даному випадку номер однозначно визначає будинок і кімнату, наприклад, номер «205» означає будинок 2, кімната 5, тоді і назви будинку не потрібно. Все зрозуміло з назви номера. З бесіди з замовником ми з'ясували, що назва у будинку буде, і номера у номерів повинні бути текстовим рядком, «про всяк випадок».

Будинки і номера – це сутності, пов'язані ставленням один до багатьох: один будинок має кілька номерів.

Змінна «Номери» з типом «номери» підкреслює наявність відношення один-до-багатьох.

Якщо на якомусь етапі є змінна, але невідомий її сенс, її тип можна відзначити як «Невизначений». Також це може стати в нагоді, якщо ще не створено тип «Номер», і його ще немає в списку типів даних.

Інформаційна сутність «Категорія»

Має тільки назву. Замовник повідомив, що категорії на сайті повинні йти в такому порядку: «Люкс», «Напівлюкс» і «Стандарт». Інтуїція підказує, що нам потрібно додати можливість в адмінпанелі встановлювати порядок слідування категорій. Це можна відмітити, додавши в опис структури змінної «Порядок слідування».

Інформаційна сутність «Період»

Сутність, яка визначає періоди з різними цінами, залежно від попиту. Наприклад, влітку, коли користуються найбільшим попитом, ціна вище, взимку, коли попит низький, щоб стимулювати клієнтів, ціна знижується. Період повинен мати початок і кінець, однак, якщо ввести для кожного періоду дві дати «початок» і «кінець», доведеться стежити за тим, щоб

кінець періоду завжди збігався з початком наступного, інакше між періодами випадково може виникнути прогалина, коли зима вже закінчилася, а літо ще не почалося або, навпаки, накладання періодів один на одного.

Якщо потрібно стежити за тим, щоб два значення завжди збігалися, то краще за все одне значення прибрати. Тоді період матиме тільки початок і кінцем одного періоду буде початок наступного.

Інформаційна сутність «Ціна»

Ціна (сутність) має ціну (атрибут) і, з одного боку, залежить від категорії номери, з іншого боку, від періоду часу. Можна зробити висновок, що ціна – це зв'язна таблиця, яка пов'язує всі категорії з усім періодами року, на цих зв'язках і знаходяться ціни.

Інформаційна сутність «Броня»

Мабуть, найбільша структура. Броня повинна мати:

- Початок і кінець дії (дата).
- Бажана категорія номера (категорія).
- Кількість номерів (ціле число).
- Ім'я замовника, його телефон, email (рядки).
- Побажання «всі номери в одному будинку» (логічне).
- Примітка – текст, який залишає клієнт при оформленні броні (текст).
- Статус заявки (ціле число).
- Останній термін підтвердження заявки (дата).
- Коментар – примітка менеджера сайту (текст).
- Номери – список номерів входять в заявку (список номерів)

Інформаційна сутність «Списки»

Списки можуть знадобитись в адмінпанелі. Списки – це перелік усіх описаних вище сутностей, крім ціни. У даному прикладі списки прості, без будь-яких деталізацій, і ніяких особливих описів не вимагають.

Додаток 9

Функціональні характеристики сайту туристичної бази

Наприклад, замовник повідомив, що бронювання повинно відбуватися таким чином: «Є певна кількість номерів. Номери мають різні типи (дешевше, з мінімальними зручностями і з максимальними зручностями), в будиночках по 2 або 4 номери, не обов'язково одного типу. Клієнт може вибрати період, коли він хоче відпочити, тип номера і кількість номерів і зробити заявку. Заявка повинна бути перевірена, для цього менеджер зв'язується з клієнтом, якщо клієнт підтверджує замовлення, номери бронюються. Далі деякий час очікується оплата від клієнта, якщо оплата не прийшла через N днів, менеджер знімає броню. Якщо оплата прийшла, броня позначається як оплачена. При бронюванні у клієнта повинна бути можливість вказати, що номери хотілося б зняти в одному будиночку. Ціни на відпочинок залежать від типу номера і періоду. Влітку одна ціна, взимку інша.»

Отже, які функціональні характеристики повинна мати система:

1) Система бронювання повинна надавати можливість користувачеві ознайомиться з наявністю вільних місць на потрібний для нього період часу і зробити заявку на бронювання.

2) Передбачається, що на турбазі існує багато однотипних будинків. Кімнати в будинках розділені на невелику кількість категорій, типу стандартний і люксовий.

3) При бронюванні користувач вказує бажаний тип номера, але не вказує який саме номер він хоче. Конкретний номер, в який буде заселений гість, визначається менеджером.

4) Заявка на бронювання має 3 стани: не підтверджено; підтверджено; заброньовано.

5) Коли користувач оформляє заявку, вона автоматично стає непідтвердженою. Після того, як менеджер зв'яжеться з клієнтом і той підтвердить замовлення, менеджер змінить стан замовлення на "Підтверджено" і зробить прив'язку заявки з конкретним номером в конкретному будинку. Після надходження оплати менеджер змінить статус заявки на "заброньовано".

Непідтверджена заявка має період життя, який визначається полем "Термін підтвердження заявки". Якщо статус броні не був змінений зі статусу "не підтверджено", то заявка повинна бути автоматично видалена при досягненні терміну підтвердження заявки.

6) Ціни на номер залежать від категорії номера (стандартний, люксовий тощо) і від сезону (наприклад, влітку дорожче, взимку дешевше).

7) При оформленні бронювання, користувачеві повинна бути доступна галочка для позначки того, що користувач хотів би забронювати номер в одному будиночку. Однак ця галочка є тільки лише побажанням і служить для інформування менеджера, кінцеве рішення про розселення по номерам приймає менеджер в залежності від обставин.

Додаток 10

Структура компоненту і опис сторінок

Для того, щоб уявити, які сторінки потрібні на сайті, потрібно розробити схему-структуру сайту.

Розглянемо на прикладі публічної сторони бронювання туристичного сайту. Ця сторона буде невелика і включає сторінку з відображенням цін і сторінку з бронюванням, а також декількома дочірніми.

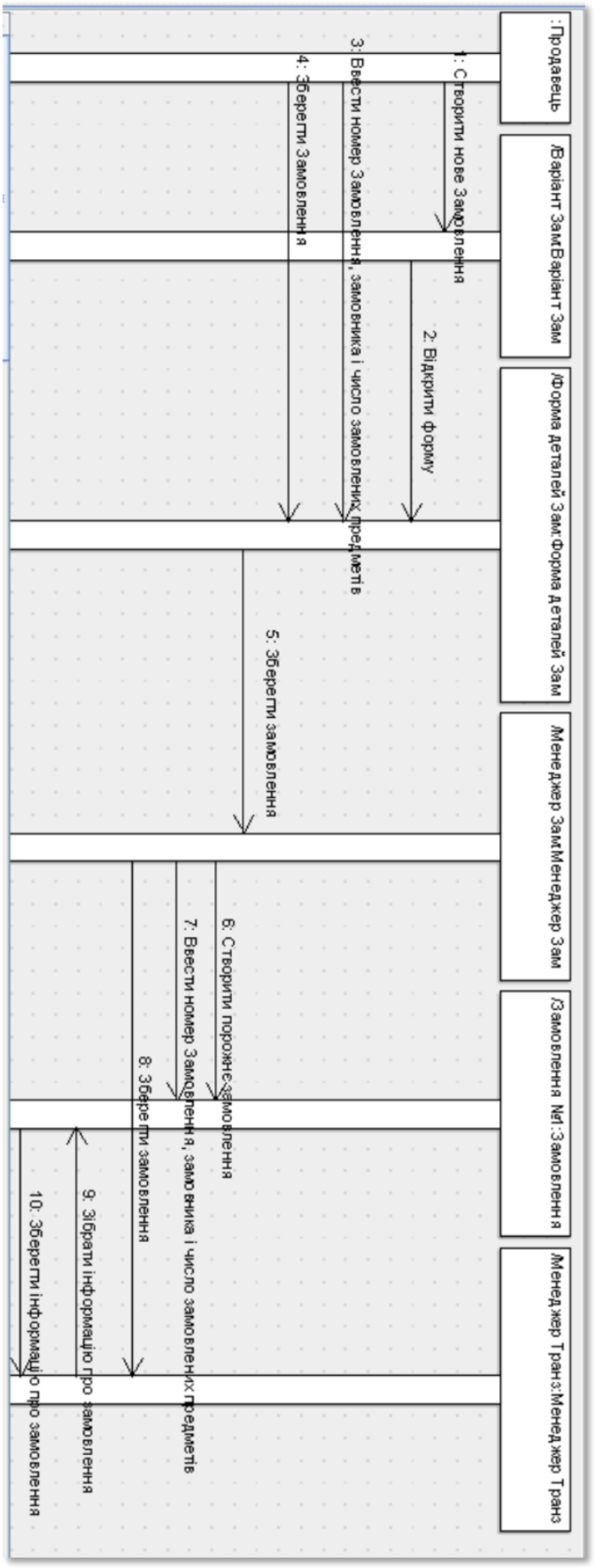
В адмінпанелі потрібно зробити ряд розділів для редагування описаних раніше сутностей. Редагування таких «довідників» має типовий вигляд: список всіх елементів інформаційної сутності, перехід до редагування існуючої сутності, створення нової і видалення існуючої. Також на кожну сутність потрібно виділити 3 сторінки: список з можливістю видалити елемент(и), сторінка для створення і редагування записів.

Для ціни доцільно зробити окрему сторінку, оскільки вона вимагає окремого розгляду.

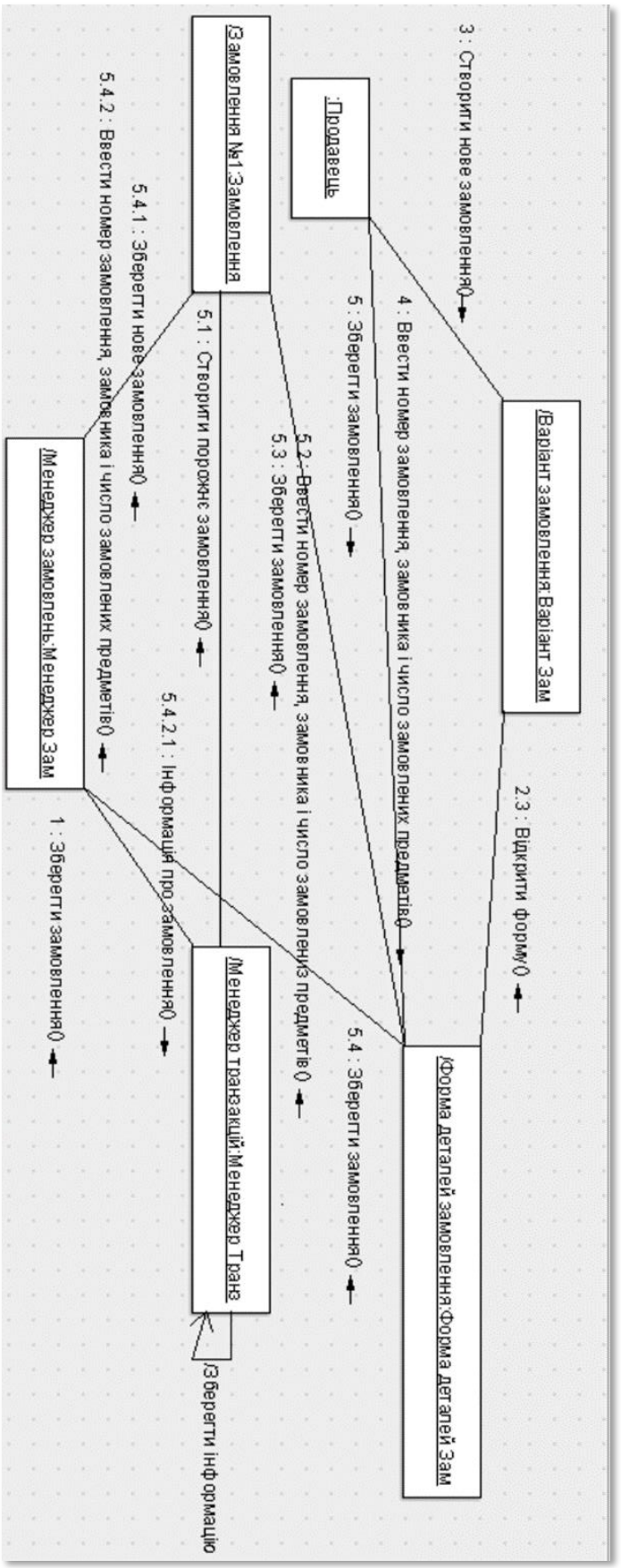
Допрацьована схема може мати такий вигляд:



Додаток 11

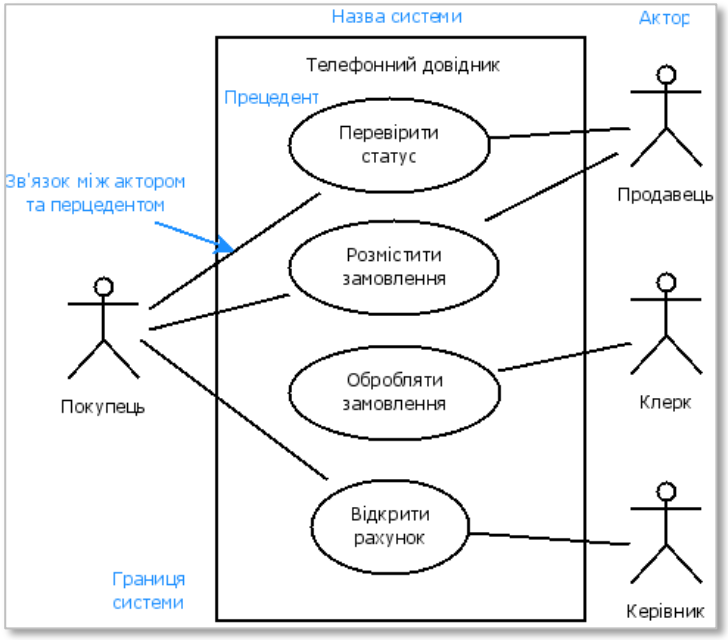

:Продавець
/ВаріантЗамВаріантЗам
/ФормаДеталейЗам:ФормаДеталейЗам
/МенеджерЗамМенеджерЗам
/ЗамовленняNe1:Замовлення
/МенеджерТранз:МенеджерТранз
1: Створити нове замовлення
2: Відкрити форму
3: Ввести номер замовлення, замовника і число замовлених предметів
4: Зберегти замовлення
5: Зберегти замовлення
6: Створити порожнє замовлення
7: Ввести номер замовлення, замовника і число замовлених предметів
8: Зберегти замовлення
9: Зібрати інформацію про замовлення
10: Зберегти інформацію про замовлення

Додаток 12

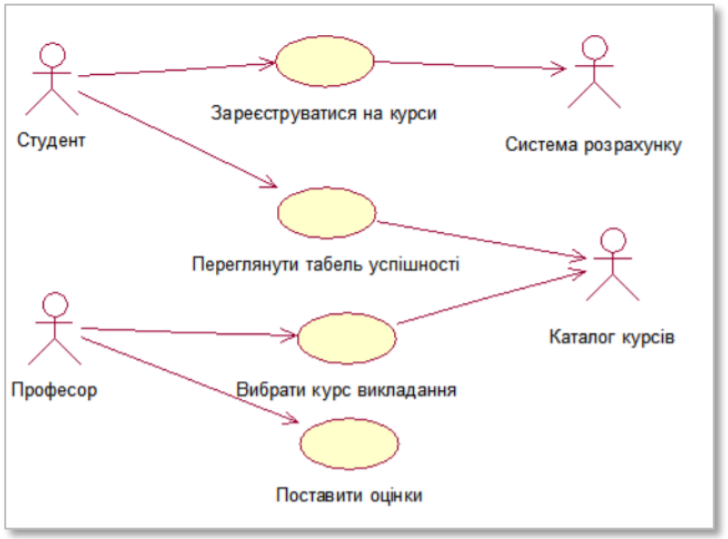

/Варіант замовлення:Варіант Зам
/Форма деталей замовлення:Форма деталей Зам
:Продавець
/Замовлення №1:Замовлення
/Менеджер замовлень:Менеджер Зам
/Менеджер транзакцій:Менеджер Транз
3 : Створити нове замовлення()
2.3 : Відкрити форму()
4 : Ввести номер замовлення, замовника і число замовлених предметів()
5 : Зберегти замовлення()
5.2 : Ввести номер замовлення, замовника і число замовлених предметів()
5.3 : Зберегти замовлення()
5.4 : Зберегти замовлення()
5.1 : Створити порожнє замовлення()
5.4.2.1 : Інформація про замовлення()
5.4.1 : Зберегти нове замовлення()
5.4.2 : Ввести номер замовлення, замовника і число замовлених предметів()
1 : Зберегти замовлення()
/Зберегти інформацію

Додаток 13

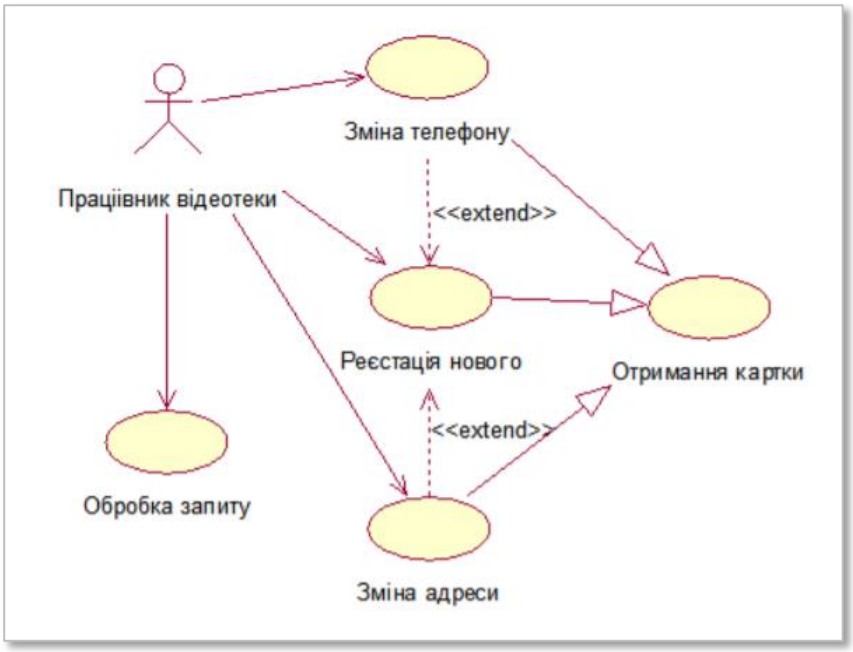
1. Телефонний довідник



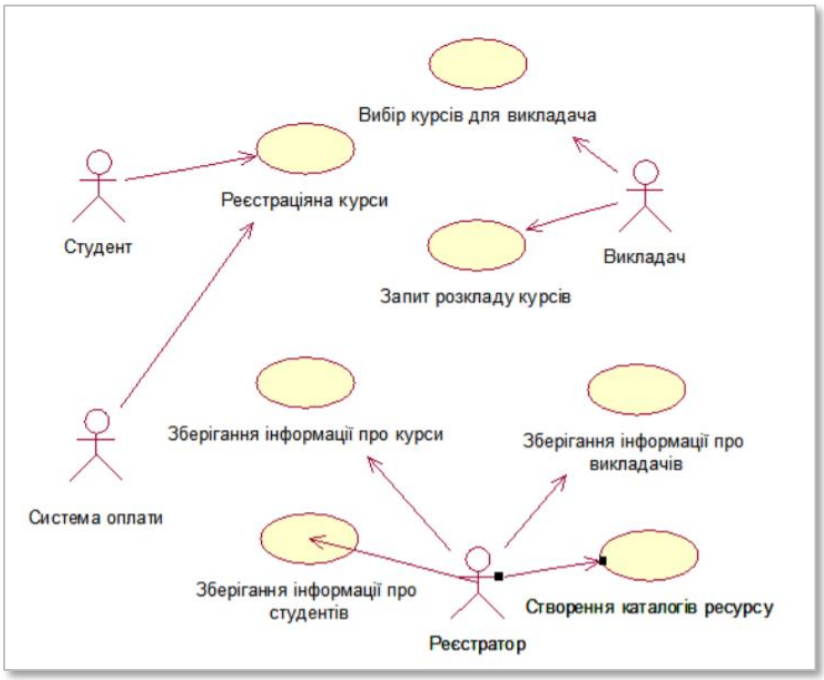
2. Система інтернет-курсів



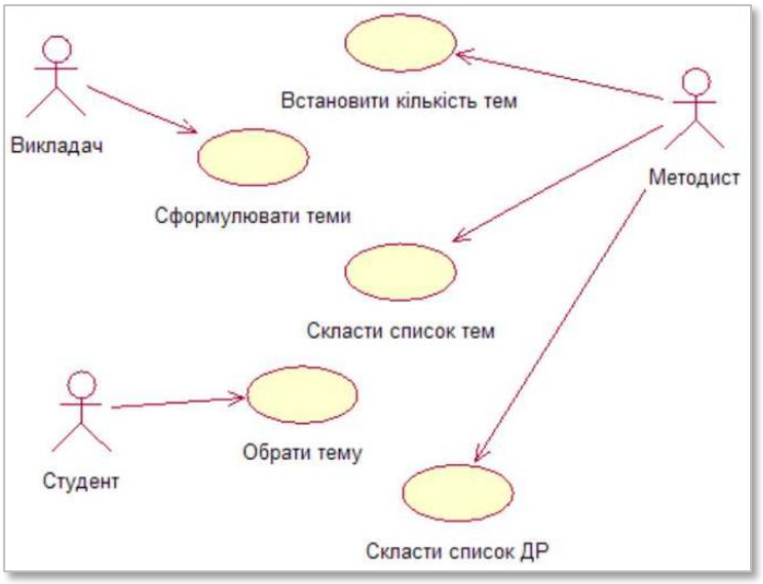
3. Система «Відеопрокат»



4. Система реєстрації навчальних курсів Університету



5. Система «Дипломні роботи»


Встановити кількість тем
Викладач
Методист
Сформулювати теми
Скласти список тем
Студент
Обрати тему
Скласти список ДР

Додаток 14

Уманський державний педагогічний університет імені Павла Тичини
Факультет _____
Кафедра _____

«ЗАТВЕРДЖУЮ»

Завідувач кафедри
_____ ПІБ
ДАТА

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

ВБ 1.8 Вибрані питання комп'ютерної інженерії

Галузь знань: 01 Освіта

Спеціальність: 014.09 Середня освіта (Інформатика).

Освітня програма: Середня освіта (Інформатика).

Умань – 20__

Робоча програма з навчальної дисципліни «Вибрані питання комп'ютерної інженерії» для студентів з галузі знань: 01 Освіта, спеціальності: 014.09 Середня освіта (Інформатика).

Розробник: Ткачук Галина Володимирівна, професор кафедри інформатики і інформаційно-комунікаційних технологій, доктор педагогічних наук, доцент.

Робочу програму схвалено на засіданні кафедри _____.

Протокол №__ від ___ _____ 20__ року.

Завідувач кафедри _____

_____ _____
(підпис) (прізвище та ініціали)

Робочу програму розглянуто та затверджено на засіданні науково-методичної комісії факультету _____.

Протокол № __ від __ _____ 20___ року.

Голова науково-методичної комісії факультету _____

_____ _____
(підпис) (прізвище та ініціали)

© Ткачук Г.В., 20__ рік

1. Опис навчальної дисципліни

Найменування показників	Характеристика дисципліни за формами навчання	
	денна	заочна
Вид дисципліни (обов’язкова чи вибіркова)	Вибіркова	Вибіркова
Мова викладання, навчання та оцінювання	Українська	Українська
Загальний обсяг у кредитах ЄКТС / годинах	3 / 90	3 / 90
Курс	IV	IV
Семестр	8	8
Кількість змістових модулів із розподілом:	1	1
Обсяг кредитів	3	3
Обсяг годин, у тому числі:		
Аудиторні:	44	10
Лекційні	14	4
Практичні	30	6
Самостійна робота	46	80
Форма семестрового контролю	Залік	

2. Мета та завдання навчальної дисципліни

Метою курсу є формування професійних навичок інженерії програмного забезпечення, дослідження та аналізу складних об'єктів та систем, розв'язання прикладних і наукових завдань, використання основних парадигм проектування програмного забезпечення комп'ютеризованих систем.

Завдання: систематизація знань щодо програмної інженерії, її специфіки при проєктуванні складних систем; стандартів в галузі проєктування ПЗ; моделей життєвого циклу програмного продукту; засобів проєктування ПЗ та вимог до нього.

У результаті вивчення навчальної дисципліни студент повинен:

знати: поняття та специфіки інженерії програмного забезпечення; стандартів в галузі проєктування програмного забезпечення; кодексу професійної етики, соціальної значимості та культурних аспектів інженерії програмного забезпечення; поняття якості програмного забезпечення; поняття життєвого циклу програмного забезпечення; різних моделей життєвого циклу програмного забезпечення; CASE-засобів розробки програмного забезпечення; вимог до програмного забезпечення відповідно до стандартів; різниці між функціональними та нефункціональними вимогами; поняття технічного завдання та його основних розділів; функціонального призначення програмного забезпечення.

вміти: брати участь у проєктування програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та

процесів функціонування; застосовувати специфікації, стандарти, правила і рекомендації в професійній галузі при реалізації процесів життєвого циклу; документування інформації про проєкт, обґрунтування експлуатаційного та функціонального призначення; ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення; створювати інформаційні моделі даних та інформаційні сутності проєкту; розробки діаграм варіантів використання (прецедентів), діаграм послідовності, розробки кооперативних діаграм, проєктування діаграм стану для класів.

3. Результати навчання за дисципліною

Відповідно до освітньої програми «Середня освіта (Інформатика)», після вивчення дисципліни студент отримає наступні результати:

ПРН 1. Знає основні історичні етапи розвитку предметної галузі.

ПРН 5. Оперує базовими категоріями та поняттями спеціальності.

ПРН 6. Використовує інструменти демократичної правової держави в професійній та громадській діяльності; застосовує міжнародні та національні стандарти і практики в професійній діяльності.

ПРН 10. Здатний цінувати різноманіття та мультикультурність, керуватися в діяльності сучасними етичними нормами, принципами толерантності, діалогу і співробітництва.

ПРН 22. Уміє створювати інформаційні моделі, реалізовувати їх засобами інформаційно-комунікаційних технологій, здійснювати дослідження, інтерпретувати, аналізувати та узагальнювати його результати.

4. Програма навчальної дисципліни

Тема 1. Вступ до інженерії програмного забезпечення

1. Програмна інженерія: поняття та специфіка.
2. Становлення і розвиток інженерії програмного забезпечення (ПЗ).
3. Головні умови успішної інженерії ПЗ.
4. Класифікація ПЗ.

Тема 2. Стандарти в галузі проєктування програмного забезпечення

1. Поняття стандарту проєктування ПЗ.
2. Міжнародні організації в галузі стандартизації.
3. Базовий стандарт розроблення ПЗ ISO 12207.
4. Характеристика областей знань інженерії ПЗ – SWEBOK
5. Стандарт зрілості компанії-розробника ПЗ CMM.

Тема 3. Життєвий цикл програмного забезпечення

1. Поняття життєвого циклу ПЗ.
2. Класичний життєвий цикл ПЗ – каскадна модель.
3. Нотації проєктування програмного забезпечення.
4. Процеси ЖЦ ПЗ за стандартом ISO/IEC 12207.

Тема 4. Моделі життєвого циклу програмного продукту

1. Спіральна модель.
2. Ітераційна та V-подібна моделі.

3. Інкрементна модель.
4. Промислові технології створення програмного забезпечення.

Тема 5. CASE-засоби та вимоги до програмного забезпечення

1. Поняття та класифікація CASE-засобів.
2. Вимоги до програмного забезпечення.
3. Професійні і етичні вимоги.
4. Кодекс етики IEEE-CS / ACM.

Тема 6. Управління програмним проектом

1. Проект і управління проектом.
2. Менеджер IT-проекту та його компетентності.
3. Команда IT-проекту та рольова модель.
4. Моделі організації команд.
5. Спілкування в команді.

5. Структура навчальної дисципліни

Назви змістових модулів і тем	**Кількість годин**							
	денна форма				**заочна форма**			
	усього	у тому числі			**усього**	у тому числі		
		лек	пр	ср		лек	пр	ср
1	2	3	4	5	6	7	8	9
Змістовий модуль 1. Теоретичні аспекти інженерії програмного забезпечення								
Тема 1. Вступ до інженерії програмного забезпечення	**8**	2	1	4	**21**		1	20
Тема 2. Стандарти в галузі проєктування програмного забезпечення	**8**	2	1	6	**11**		1	10
Тема 3. Життєвий цикл програмного забезпечення	**16**	2	4	10	**21**		1	20
Тема 4. Моделі життєвого циклу програмного забезпечення	**28**	2	16	10	**12**	1	1	10
Тема 5. CASE-засоби та вимоги до програмного забезпечення	**18**	4	4	10	**13**	2	1	10
Тема 6. Управління програмним проєктом	**12**	2	4	6	**12**	1	1	10
Усього годин	**90**	**14**	**30**	**46**	**90**	**4**	**6**	**80**

6. Теми практичних занять

№ з/п	***Назва теми***	***Кількість годин***	
		Денна	***Заочна***
1.	Розробка технічного завдання на виконання проєкту	6	3

2.	Введення до моделювання в середовищі ArgoUML	4	0,5
3.	Розробка діаграм варіантів використання (прецедентів)	4	0,5
4.	Розробка діаграм послідовності	4	0,5
5.	Розробка кооперативної діаграми	4	0,5
6.	Проектування діаграми стану для класів	4	0,5
7.	Створення діаграм UML в онлайновому середовищі Gliffy	4	0,5
	Разом:	**30**	**6**

7. Самостійна робота

№з/п	***Назва теми***	***К-ть годин***	
		Денна	*Заочна*
1.	Продукти інженерії програмного забезпечення, продукція і програмне забезпечення	4	6
2.	Продукти інженерії програмного забезпечення, продукція і програмне забезпечення	4	6
3.	Пряма інженерія програмного забезпечення	4	6
4.	Обернена інженерія програмного забезпечення	4	6
5.	Емпірична інженерія програмного забезпечення	4	6
6.	Складові елементи еталонної моделі інженерії програмного забезпечення	4	6
7.	Моделі та методи оцінювання вартості програмного забезпечення	4	6
8.	Засоби оцінювання вартості програмного забезпечення	4	6
9.	Проектування та програмна реалізація – конструкторські етапи розроблення програмного забезпечення	4	10
10.	Документування вимог до ПЗ	2	6
11.	Тестування – етап контролю якості програмного забезпечення	4	6
12.	Експлуатація і супровід – завершальні етапи життєвого циклу програмного забезпечення	4	10
	Разом	**46**	**80**

8. Індивідуальні завдання

ІНДИВІДУАЛЬНІ НАВЧАЛЬНО-ДОСЛІДНІ ЗАВДАННЯ

Під час вивчення дисципліни студенти мають виконати індивідуальні завдання. Завдання передбачає, що кожний студент повинен скласти технічне завдання (ТЗ) на програмний засіб.

Складання технічного завдання – це один з важливих етапів при створенні програмного засобу. При формулюванні завдання на розробку програмного

засобу необхідно оформити своє уявлення про засіб документально.

У ТЗ повинні бути відображені вимоги до дизайну, навігації, способам представлення інформації, термінам і об’ємам робіт кожного етапу розробки ПЗ.

ТЗ найчастіше представлено наступними розділами:

- загальні відомості;
- експлуатаційне призначення;
- функціональне призначення;
- терміни та визначення;
- дані і списки;
- функціональні характеристики;
- сторінки;
- вимоги до надійності;
- вимоги до хостингу;
- наповнення контентом;
- здача і приймання.

9. Методи навчання

Традиційні лекції та лекції-візуалізації із застосуванням мультимедійного комплексу; практичні роботи з використанням інтегрованих середовищ розробки; робота з теоретичними матеріалами у системі дистанційного навчання; робота над понятійним апаратом у вікі-енциклопедії «Вікі-клік» (http://wikifizmat.udpu.edu.ua); створення карт знань; розв’язування задач; групова робота, самостійна робота.

10. Методи контролю

Поточне оцінювання результатів роботи, виконаної за комп’ютером; доповідь під час захисту ІНДЗ; виступ під час організації вебінару «Програмна інженерія»; поточний та підсумковий контроль – тестування.

11. Критерії оцінювання результатів навчання

Критерій	Результат навчання
90–100	Студент здатний аналізувати складні об'єкти та системи; обирати і застосовувати основні методи проектування програмного забезпечення; проводити аналіз дефектів, помилок та ризиків у життєвому циклі програмного забезпечення.
82–89	Студент добре обізнаний в галузі основних стандартів інженерії програмного забезпечення; здатний використовувати методи проектування програмного забезпечення та мови моделювання програмного забезпечення.
75–81	Студент розуміє базові принципи організації та функціонування складних систем; вміє розробляти модель програмного забезпечення; застосовує синтаксис та основні конструкції мов програмування для розробки програмних систем.
69–74	Студент може відтворювати знання основних методів дослідження та аналізу складних об'єктів та систем; володіє

	основним синтаксисом та конструкціями мов програмування та застосовує їх для розробки нескладних систем.
60–68	Студент на початковому рівні відтворює знання щодо проектування складних об'єктів та систем; може спроектувати розробити елементарний фрагмент нескладної програмної системи.
35–59	Студент володіє навчальним матеріалом на рівні елементарного розпізнавання і відтворення окремих фактів в галузі комп'ютерної інженерії, що позначаються студентом окремими словами чи реченнями.
1-34	Студент володіє матеріалом на рівні окремих фрагментів в галузі комп'ютерної інженерії, що становлять незначну частину навчального матеріалу.

12. Розподіл балів, які отримують студенти

Поточне оцінювання результатів практичної роботи та самостійна робота						ІНДЗ	Підсумковий контроль (залік)
Т1	Т2	Т3	Т4	Т5	Т6	20	20
10	10	10	10	10	10		
100							

Т1, Т2 ... Т6 – теми змістового модуля.

Шкала оцінювання: національна та ЄКТС

Сума балів за всі види навчальної діяльності	Оцінка за національною шкалою	
	для екзамену, курсової роботи, практики	для заліку
90–100	відмінно	зараховано
82–89	добре	
75–81		
69–74	задовільно	
60–68		
35–59	незадовільно з можливістю повторного складання	не зараховано з можливістю повторного складання
1–34	незадовільно з обов'язковим повторним вивченням дисципліни	не зараховано з обов'язковим повторним вивченням дисципліни

13. Рекомендована література

Основна

1. Заяць В. М. Методи і засоби комп'ютерних інформаційних технологій. Львів, 2013. 142 с.

2. Цідило І. М. Нечітке моделювання педагогічних явищ : монографія. Тернопіль, 2013. 263 с.

3. Цідило І. М. Підготовка інженера-педагога до застосування інтелектуальних технологій у професійній діяльності : монографія. Тернопіль, 2014. 421 с.

4. Технології програмування та створення програмних продуктів: конспект лекцій. Суми, 2013. 133 с.

5. Вступ до програмної інженерії. У 2 томах. Том 1: Історія розвитку. Основні поняття. навч. посіб. 2-е видання. Черкаси: ЧНУ імені Богдана Хмельницького, 2015. 500 с.

6. Вступ до програмної інженерії. У 2 томах. Том 2: Основи програмної інженерії: навч. посіб. 2-е видання. Черкаси: ЧНУ імені Богдана Хмельницького, 2015. 370 с.

7. Погорілий С. Д. Програмне конструювання: підручник серії Автоматизація наукових досліджень. 2-е вид. Київ: ВПЦ "Київський університет", 2007. 438 с.

8. Левус Є.В., Мельник Н.Б. Вступ до інженерії програмного забезпечення: навч. посібник. Львів: Вид-во Львівської політехніки, 2017. 280 с.

9. Білас О. Якість програмного забезпечення та тестування. Львів: Вид-во Львівської політехніки, 2011. 214 с.

10. Вакалюк Т. А. Технології тестування програм: навч.-метод. посібник для студентів напряму 6.040302 Інформатика*. Житомир: Вид-во ЖДУ, 2013. 96 с.

Допоміжна

1. Коротєєва Т.О. Алгоритми і структури даних: навчальний посібник. Львів : Вид-во Львівської політехніки, 2014. 280 с.

2. Левус Є., Марусенкова Т., Нитребич О. Життєвий цикл програмного забезпечення: навч. посібник. Львів: Вид-во Львівської політехніки, 2017. 240 с.

3. Життєвий цикл ПЗ: тестування. Методичні вказівки до виконання лабораторної роботи із дисципліни «Основи програмної інженерії» для студентів базового напряму «Програмна інженерія» / Укл.: Є.В. Левус, Т.А. Марусенкова. Львів: Вид-во Львівської політехніки, 2016. 26 с.

4. ISO/IEC/IEEE 12207:2017. Systems and software engineering. Software life cycle processes. ISO/IEC/IEEE, 2017. 145 p.

5. ДСТУ ISO/IEC/IEEE 12207:2018. Інженерія систем і програмних засобів. Процеси життєвого циклу програмних засобів. (ISO/IEC/IEEE 12207:2017, IDT). [Чинний від 2018-08-15]. Вид. офіц. Київ, 2018. 145 с.

6. ДСТУ ISO/IEC 14764:2014. Інженерія програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Технічне обслуговування (ISO/IEC 14764:2006, IDT) [Чинний від 2016-01-01]. Вид. офіц. Київ, 2016. 45 с.

7. Величко О. М., Грабовський О. В., Гордієнко Т. Б. Особливості застосування V-моделі при розробленні та оцінюванні якості програмного забезпечення засобів вимірювальної техніки. *Збірник наукових праць ОДАТРЯ.* 2019. Вип. 1 (14). С. 6–11.

8. Величко О. М., Грабовський О. В., Гордієнко Т. Б. Особливості застосування спіральної моделі для випробувань програмного забезпечення

засобів вимірювальної техніки. *Збірник наукових праць ОДАТРЯ*. 2019. Вип. 2(15). С. 42–49.

9. Velychko O., Gordiyenko T., Hrabovskyi O. Testing of measurement instrument software on the national level. *Eastern-European Journal of Enterprise Technologies. Information and controlling systems*. 2018. № 2/9 (92). P. 13–20.

10. WELMEC 7.2. Software Guide (Measuring Instruments Directive 2014/32/EC). WELMEC, 2019. 132 p.

11. ISO/IEC/IEEE 24748-1:2018. Systems and software engineering. Life cycle management. Part 1: Guidelines for life cycle management. [Valid from 2018-11-01]. ISO/IEC/IEEE, 2018. 72 p.

12. ДСТУ ISO/IEC TS 24748-1:2018. Інженерія систем і програмних засобів. Керування життєвим циклом. Частина 1. Настанови щодо керування життєвим циклом. (ISO/IEC TS 247481:2018, IDT). [Чинний від 2018-08-15]. Вид. офіц. Київ, 2018. 72 с.

13. ДСТУ ISO/IEC TR 24748-3:2016. Інженерія систем і програмного забезпечення. Керування життєвим циклом. Частина 3. Настанова щодо застосування ISO/IEC 12207 (Процеси життєвого циклу програмного забезпечення). (ISO/IEC TR 24748-3:2011, IDT). [Чинний від 2018-01-01]. Вид. офіц. Київ, 2016. 91 с.

14. SWEBOK. Guide to the Software Engineering Body of Knowledge. Version 3.0. IEEE Computer Society, 2014. 335 p.

15. ISO/IEC/IEEE 15288: 2015. Systems and software engineering. System life cycle processes. [Valid from 2015-05-01]. ISO/IEC/IEEE, 2015. 108 p.

16. ДСТУ ISO/IEC/IEEE 15288:2016. Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем. (ISO/IEC/IEEE 15288:2015, IDT). [Чинний від 2018-01-01]. Вид.офіц. Київ, 2018. 90 с.

14. Інформаційні ресурси

1. Дистанційні курси з комп’ютерної інженерії. URL: https://www.edx.org/learn/computer-engineering.
2. Дистанційний курс «Software Engineering Essentials» (Основи програмної інженерії). URL: https://www.edx.org/course/software-engineering-essentials.
3. Інформаційні системи і технології на підприємствах (розділ «Інформатика»). URL: https://pidruchniki.com.
4. Комп’ютерна інженерія. URL: https://www.techopedia.com/definition/25933/computer-engineering.
5. Сидоров М.О. Вступ до інженерії програмного забезпечення. URL: https://it.wikireading.ru/22993.
6. Наукова електронна бібліотека періодичних видань НАН України. URL: http://dspace.nbuv.gov.ua.
7. Проектування програмних засобів систем управління. URL: http://posibnyky.vntu.edu.ua/bevz/2.htm.
8. Міжнародний стандарт ISO/IEC 12207. URL: https://studopedia.com.ua/1_243496_mizhnarodniy-standart-ISOIEC-.html.

Предметний покажчик

ArgoUML ..... *Див.* ***UML-засіб***
CASE-засіб 60, 62, 67, 68, 70, 108
CPM.....*Див.* ***Метод критичного шляху***
Gliffy ..... *Див.* ***UML-засіб***
IEC ..... *Див.* ***Міжнародна електротехнічна комісія***
IEEE .... *Див.* ***Інститут інженерів з електротехніки та електроніки***
ISO .....*Див.* ***Міжнародна організація зі стандартизації***
Microsoft Solution Framework..... 60
MSF.....*Див.* ***Microsoft Solution Framework***
PERT..... *Див.* ***Метод аналізу і оцінки програм***
PMBOK 78, 81
PMI 6, 81
RAD..... 55
RUP..... 61
SEI..... 25
Software..... *Див.* ***Програмне забезпечення***
SQI 6, 83, 84, 169
SWEBOK..... 29
UML діаграми..... 107
UML-засіб 106, 108, 110, 112, 113, 122, 138
UML-моделювання 106
V модель ЖЦ..... 58, 59
Галузевий стандарт..... 18
Державний стандарт..... 18
Діаграма кооперації.....107, 134
Діаграма послідовності 122
Діаграма прецедентів 113
Діаграма станів..... 138
Експлуатація ПЗ 41, 42
Екстремальне програмування 64, 65
Етапи ЖЦ..... 39
Ефективність ПЗ..... 72
Життєвий цикл.....38, 39, 40
Закон Лермана 79
Зручність ПЗ 72
Інженерія ПЗ..... 39
Інженерна дисципліна..... 7
Інженер-програміст..... 8
Інкрементна модель ЖЦ..... 59
Інститут інженерів з електротехніки та електроніки.....19, 24, 29, 67, 73
Інструментальне ПЗ ... *Див.* ***Системи програмування***
Ітераційна модель ЖЦ..... 58
Каскадна модель ЖЦ..... 40, 58
Кейс..... 68
Класифікація ПЗ..... 14
Класичний ЖЦ ПЗ 40, 42
Кодування..... 41
Кооперативна діаграма.....Див. ***Діаграма кооперації***
Корпоративний стандарт..... 17
Криза ПЗ..... 9, 38
Криза програмування.....*Див.* ***Криза ПЗ***
Менеджер проєкту.....31, 75, 80, 84, 85
Метод аналізу і оцінки програм 79
Метод критичного шляху..... 79
Методологія IDEF0 44
Міжнародна електротехнічна комісія..... 22, 23
Міжнародна організація зі стандартизації.... 6, 17, 18, 20, 21, 22, 23, 25, 27, 28, 38, 46, 50
Міжнародний стандарт..... 18, 19
Мобільність ПЗ..... 14
Модель СММ..... 36
Модель ЖЦ..... 50
Модель хаосу 94
Надійність ПЗ 71
Нефункціональні вимоги 71
Нотації.....38, 44, 69
Об'єктно-орієнтований підхід..... 11
Парне програмування 66
Поведінкова нотація..... 45
Прикладне ПЗ 15
Програма 12, 77
Програмна документація..... 40
Програмна інженерія 7, 8, 9, 68
Програмне забезпечення..... 9, 15
Програмний засіб..... 14
Програмування..... 8
Проєкт 75, 76, 77
Проєктувальник 85
Проєктування..... 41
Прототип 56, 70
Прототипування..... 50, 57, 70, 88
Процеси ЖЦ ПЗ 46
Регіональний стандарт 18
Розробка 52, 56, 63, 65, 66, 89
Розробник..... 85
Системи програмування..... 16
Системна інженерія..... 8
Системне ПЗ 15
Специфікація вимог..... 41
Спіральна модель ЖЦ..... 57
Стадії створення ПЗ 28
Стандарт..... 17
Стандарт BSP..... 18
Стандарт якості ПЗ 28
Структурна нотація..... 44
Структурний підхід 11
Супровід ПЗ..... 41, 71
Творча задача..... 78
Теорія X 92
Теорія Y 93
Теорія Z..... 95
Тестувальник..... 86
Тестування 41
Технічне завдання..... 98
Технічний редактор 87
Трикутник обмежень..... 80
Управління продуктом..... 89
Управління проєктом 74
Функціональні вимоги 71
Якість ПЗ..... 13, 19
Якість програмного забезпечення..... 12

Навчальне видання

Г. В. Ткачук

ВИБРАНІ ПИТАННЯ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Видається в авторській редакції

Підписано до друку 25.10.2021 р. Формат 60х84/16.
Папір офсетний. Ум. друк. арк. 11,10
Тираж 100 прим. Замовлення № 2670

Видавничо-поліграфічний центр «Візаві»
20300, м. Умань, вул. Тищика, 18/19, вул. Садова, 2
Свідоцтво суб'єкта видавничої справи
ДК № 2521 від 08.06.2006.
тел. (04744) 4-64-88, 3-51-33, (067) 104-64-88
vizavi-print.jimdo.com
e-mail: vizavi008@gmail.com
vizavisadova@gmail.com