**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра інженерії програмного забезпечення



Домашня робота

з дисципліни «[Програмне забезпечення інформаційних систем](https://classroom.google.com/c/MjU5NDQ4Nzk0MjI1) »

«Вибір моделі специфікації вимог в задачі архітектурного

проєктування ІС з використанням відомих методологій»

Виконав студент:

групи ПІ-321Б

Іванюк Н.О.

Перевірив викладач:

Талалаєв В.О.

Київ 2021

**Мета роботи:** Застосування набутих знань передпроєктного обстеження прикладних системдля вирішення задачі узгодження моделі специфікації вимог до ІС з відомимиметодологіями архітектурного проєктування ІС

**Підготовка до виконання домашнього завдання**

Вивчити і уяснити зміст основних процесів передпроєктного обстеження і

побудови моделі ІТ-потреб ПрС. Ознайомитися з поширеними методологіями архітектурного проєктування ІС.

**Завдання на виконання домашнього завдання**

1). Здійснити пошук інформаційних джерел з описами відомих поширених

методологій архітектурного проєктування.

**Варіант 1**

1-2). Обрати методологію, яка в найбільшій мірі відповідає контексту

архітектурного проєктування ІС відповідно до індивідуального варіанту ПрС.

1-3). Здійснити реферативний опис обраної методології.

1-4). Провести оцінку відповідності вихідних даних для архітектурного проєктування за обраною методологією розробленій в ЛР1-4 моделі ІТ-потреб ПрС.

1-5). Розробити пропозиції щодо складу, форми і змісту специфікацій моделі ІТ-потреб, які в найбільшій мірі відповідають обраній методології. При наявності у вибраній методології патернів архітектурного проєктування обрати відповідний пеатерн і налаштувати його на умови виконання завдання.

**Хід роботи**

1. **Структурно-функціональна методологія.**
2. **Мутодологія компонентного проектування.**
3. **Методологія об'єктно-орієнтованого проектування.**

**Наочна область (domain)** - частина реального світу, яка має істотне значення або безпосереднє відношення до процесу функціонування програми. Іншими словами, наочна область включає лише ті об'єкти і взаємозв'язки між ними, які необхідні для опису вимог і умов рішення конкретної задачі.

Виділення вихідних або базових компонентів наочної області, потрібних для вирішення того або іншого завдання, представляє, в загальному випадку, нетривіальну проблему. Складність даної проблеми виявляється в неформальному характері процедур або правил, які можна застосовувати для цієї мети. Більш того, ця робота повинна виконуватися спільно з фахівцями або експертами, обізнаними наочну область. Наприклад, якщо розробляється база даних для обслуговування пасажирів крупного аеропорту, то в проектуванні концептуальної схеми бази даних повинні брати участь штатні співробітники аеропорту. Ці співробітники добре знають весь процес обслуговування пасажирів або дану наочну область. Складність моделювання наочної області і розробки корпоративних інформаційних систем привело до появи нової методології об'єктно-орієнтований аналіз і проектування.

Об'єктно-орієнтований аналіз і проектування (ООАП, Object-Oriented Analysis/Design) - технологія розробки програмних систем, в основу яких покладена об'єктно-орієнтована методологія представлення наочної області у вигляді об'єктів, відповідних класів, що є екземплярами.

Методологія ООАП тісно пов'язана з концепцією автоматизованої розробки програмного забезпечення (Computer Aided Software Engineering, CASE). До перших CASE-средствам віднеслися з певною настороженістю. З часом з'явилися як захоплені відгуки про їх вживання, так і критичні оцінки їх можливостей. Причин для настільки суперечливих думок було декілька. Перша з них полягає в тому, що ранні CASE-средства були простою надбудовою над системою управління базами даних (СУБД). Візуалізація процесу розробки концептуальної схеми БД має важливе значення, проте, вона не вирішує проблем створення програмних засобів інших типів.

Друга причина пов'язана з графічною нотацією, реалізованою в CASE-засобах. Якщо мови програмування мають строгий синтаксис, то спроби запропонувати відповідний синтаксис для візуального представлення концептуальних схем БД, були сприйняті далеко не однозначно. На цьому фоні розробка і стандартизація уніфікованої мови моделювання UML викликала натхнення у всього співтовариства корпоративних програмістів.

В рамках ООАП історично розглядалися три графічні нотації:

· діаграми "сутність-зв'язок" (Entity-Relationship Diagrams, ERD)

· діаграми функціонального моделювання (Structured Analysis and Design Technique, SADT)

· діаграми потоків даних (Data Flow Diagrams, DFD).

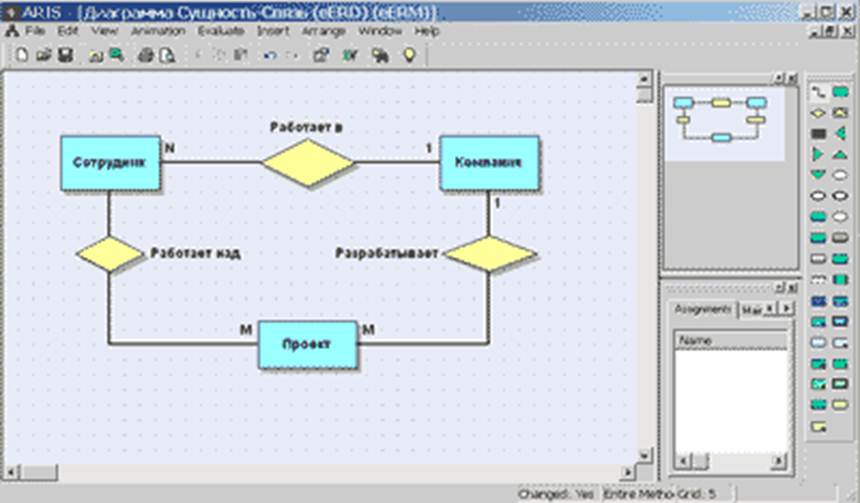
Діаграми "сутність-зв'язок" (ERD) призначені для графічного представлення моделей даних програмної системи, що розробляється, і пропонують набір стандартних позначень для визначення даних і стосунків між ними. За допомогою цього вигляду діаграм можна описати окремі компоненти концептуальної моделі даних і сукупність взаємозв'язків між ними.

Основними поняттями даної нотації є поняття сутності і зв'язку. При цьому під сутність (entity) розуміється довільна безліч реальних або абстрактних об'єктів, кожен з яких володіє однаковими властивостями і характеристиками. В цьому випадку будь-який даний об'єкт може бути екземпляром однієї і лише однієї сутності, повинен мати унікальне ім'я або ідентифікатор, а також відрізнятися від інших екземплярів даної сутності.

Зв'язок (relationship) визначається як відношення або асоціація між окремими сутностями. Прикладами зв'язків можуть бути родинні стосунки, зокрема "батько-син" або виробничі - "начальник-підлеглий". Інший тип зв'язків задається стосунками "мати у власності" або "володіти властивістю". Різні типи зв'язків графічно зображаються у формі ромба з відповідним ім'ям даного зв'язку.

Графічна модель даних будується так, щоб зв'язки між окремими сутностями відображали не лише семантичний характер відповідного відношення, але і додаткові аспекти обов'язковості зв'язків, а також кратність сутність, що беруть участь в даних стосунках екземплярів. Нотація діаграм (ERD) реалізована в різних програмних засобах. Приклад діаграми ERD, розробленої за допомогою засобу моделювання бізнес-процесів ARIS®, змальований на рис.1.2.

Обмеженість діаграм ERD виявляється при конкретизації концептуальної моделі в детальніше представлення модельованої програмної системи, яке окрім статичних зв'язків повинне містити інформацію про поведінку або функціонування окремих її компонентів.

  
Рис. 1.2. Діаграма "сутність-зв'язок" для прикладу співробітників компанії, що працюють над різними проектами

В рамках діаграм функціонального моделювання було розроблено декілька графічних мов моделювання, які отримали наступні назви:

· Нотація IDEF0 - для документування процесів виробництва і відображення інформації про використання ресурсів на кожному з етапів проектування систем

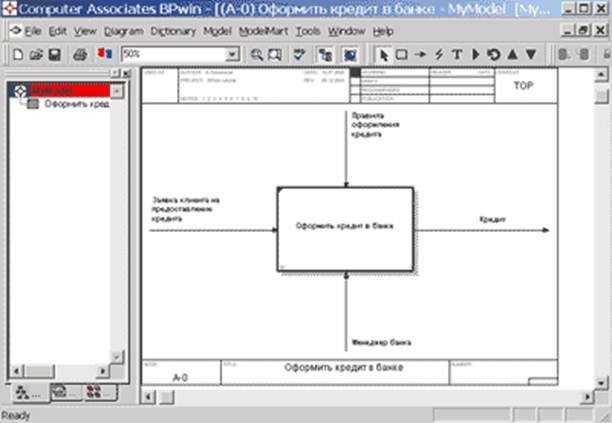
· Нотація IDEF1 - для документування інформації про виробниче оточення систем

· Нотація IDEF2 - для документування поведінки системи в часі

Нотація IDEF2 ніколи не була повністю реалізована. Нотація IDEF1 в 1985 році була розширена і перейменована в IDEF1X. Методологія IDEF знайшла вживання в урядових і комерційних організаціях, оскільки в 1993 році з'явився стандарт FIPS уряду США для двох технологій IDEF0 і IDEF1X. Протягом подальших років цей стандарт продовжував активно розвиватися і послужив основою для реалізації в деяких CASE-средствах, найбільш відомим з яких є AllFusion Process Modeler® (нова назва BPwin® ) компанії Computer Associates.

Процес моделювання IDEF є сукупністю методів, правив і процедур, призначених для побудови функціональної моделі системи якої-небудь наочної області. Функціональна модель IDEF відображує структуру процесів функціонування системи і її окремих підсистем, тобто, виконувані ними дії і зв'язки між цими діями. Для цієї мети будуються спеціальні моделі, які дозволяють в наочній формі представити послідовність певних дій. Вихідними будівельними блоками будь-якої моделі нотації IDEF0 процесу є діяльність (activity) і стрілки (arrows).

Одна з найбільш важливих особливостей нотації IDEF0 - поступове введення усе більш детальних представлень моделі системи у міру розробки окремих діаграм. Побудова моделі IDEF0 починається з представлення всієї системи у вигляді простої діаграми, що складається з одного блоку процесу і стрілок ICOM, службовців для зображення основних видів взаємодії з об'єктами поза системою. Оскільки вихідний процес представляє всю систему як єдина ціла, дана вистава є найбільш загальною і підлягає подальшій декомпозиції. Приклад представлення загальній моделі процесу оформлення кредиту в банці, розробленій за допомогою CASE-средства AllFusion Process Modeler®, змальований на мал.



Зрештою модель IDEF0 є набором ієрархічно взаємозв'язаних діаграм з супровідною документацією, як

а розбиває вихідне представлення складної системи на окремі складові частини. Деталі кожного основного процесу представляються у вигляді детальніших процесів на інших діаграмах. В цьому випадку кожна діаграма нижнього рівня є декомпозицією процесу із загальнішої діаграми. Тому на кожному кроці декомпозиції загальніша діаграма конкретизується на ряд детальних діаграм.

Основний недолік даної методології пов'язаний з відсутністю явних засобів для об'єктно-орієнтованого представлення моделей складних систем. Деякі аналітики відзначають важливість знання і вживання нотації IDEF0, проте відсутність можливості реалізації відповідних графічних моделей в об'єктно-орієнтованому програмному коді істотно звужують діапазон вирішуваних з її допомогою завдань.

У основі графічного моделювання інформаційних систем за допомогою діаграм потоків даних лежить спеціальна технологія побудови діаграм потоків даних DFD. У розробці методології DFD взяли участь багато аналітиків, серед яких слід зазначити Э. Йордона. Він автор однієї з перших графічних нотацій DFD.

Недолік розглянутих нотацій пов'язаний з відсутністю явних засобів для об'єктно-орієнтованого представлення моделей складних систем, а також складних алгоритмів обробки даних. Оскільки на розглянутих типах діаграм не вказуються характеристики часу виконання окремих процесів і передачі даних між процесами, то моделі систем, що реалізовують синхронну обробку даних, не можуть бути адекватно представлені в цих нотаціях. Всі ці особливості методів структурного системного аналізу обмежили можливості широкого вживання відповідних нотацій і послужили основою для розробки уніфікованої мови моделювання UML.

1. **Методологія MSF.**

Модель команди проекту MSF (MSF Team Model) описує підхід Microsoft до організації персоналу проекту та заходів для максимального успіху проекту.

Ця модель визначає рольові кластери, їх сфери компетенції та сфери відповідальності, а також рекомендації членам команди проектів, які дозволяють їм успішно реалізувати свою місію щодо втілення проекту в життя.

Модель команди проектів MSF розроблялася протягом декількох років і виникла в результаті розуміння недоліків пірамідальної, ієрархічної структури традиційних проектних команд.

Відповідно до моделі MSF, проектні команди будуються як невеликі, багатопрофільні команди, члени яких поділяють обов'язки та доповнюють сфери знань один одного. Це дає можливість чітко зосередитися на потребах проекту. Команду проекту об'єднує спільне бачення проекту, бажання його реалізувати, високі вимоги до якості роботи та бажання вдосконалюватися.

Нижче описано основні принципи, ключові ідеї та перевірені методи MSF, що застосовуються до моделі команди проекту.

MSF включає в себе ряд основних принципів. Ось ті, які мають відношення до успішної роботи команди:

* Розподіл відповідальності за фіксацію звітів
* Розширення прав і можливостей членів команди
* Зосередьтеся на пріоритетах бізнесу
* Єдине бачення проекту
* Будьте гнучкими – будьте готові до змін
* Заохочуйте вільне спілкування

Успішне використання моделі команди проекту MSF базується на ряді ключових концепцій:

* Команда однодумців
* Зосередьтеся на потребах клієнтів
* Зосередьтеся на кінцевому результаті
* Установка при відсутності дефектів
* Прагнення до самовдосконалення
* Зацікавлені команди працюють ефективно

MSF базується на постулаті шести якісних цілей, досягнення яких визначає успішність проекту. Ці цілі визначають модель команди проекту. Хоча вся команда несе відповідальність за успіх проекту, кожен з його рольових кластерів, визначених моделлю, пов'язаний з однією з шести згаданих цілей і працює над його досягненням.

Команда проекту включає в себе такі кластери ролей:

* управління програмами
* управління продуктами
* розробка
* тестування
* керування випуском
* задоволеності

Вони відповідають за різні функціональні області і пов'язані з ними цілі і завдання. Іноді рольові кластери просто називаються ролями. Але в будь-якому випадку суть концепції залишається незмінною - побудувати основу виробничих відносин і пов'язану з ними модель колективу, щоб вони були адаптовані (масштабовані) для задоволення потреб будь-якого проекту.

Як уже згадувалося вище, команда проекту MSF складається з шести рольових кластерів, кожен з яких відповідає за:

програмний менеджер — розробка архітектури рішень, адміністративних послуг;

розробник — розробка додатків та інфраструктури, технологічний консалтинг;

Тестування (QAE) - Планування, розробка та звітність.

менеджер з випуску — інфраструктура, технічне обслуговування, бізнес-процеси, випуск готового продукту;

задоволеність клієнтів (користувацький досвід) – навчання, ергономіка, графічний дизайн, технічна підтримка;

управління продуктами – пріоритети бізнесу, маркетинг, представництво інтересів клієнтів.

Наявність шести рольових кластерів не означає, що кількість членів команди повинна бути кратна шести - одна людина може об'єднати кілька ролей і навпаки, рольовий кластер може складатися з декількох осіб залежно від розміру проекту, його складності та навичок, необхідних для реалізації всіх компетенцій кластера. Мінімальна команда MSF може складатися всього з трьох осіб. Модель не вимагає призначення окремого співробітника для кожного кластера ролей. Мається на увазі, що всі шість якісних цілей повинні бути представлені в команді. Зазвичай виділення хоча б однієї людини на кожну рольову кластер надає повну увагу інтересам кожної з ролей, але це економічно не виправдано для всіх проектів. Часто члени команди проекту можуть поєднувати ролі.

У невеликих проектних командах необхідно об'єднати ролі. Необхідно дотримуватися двох принципів:

Роль команди розробників не може поєднуватися з будь-якою іншою роллю.

Уникайте комбінації ролей, які попередньо визначили конфлікт інтересів.

Як і в будь-якій іншій командній діяльності, відповідне поєднання ролей залежить від самих членів команди, їх досвіду і професійних навичок. На практиці поєднання ролей не рідкість. І якщо команда проекту виробляє його навмисно і управляє ризиками, пов'язаними з таким об'єднанням, проблеми, які виникають, будуть мінімальними.

MSF не надає конкретних рецептів управління проектами і не дає пояснень щодо різних методів роботи, які використовують досвідчені менеджери. Принципи MSF формують підхід до управління проектами,в якому:

відповідальність за управління проектами розподіляється між лідерами рольових кластерів всередині команди – кожен член команди проекту несе відповідальність за загальний успіх проекту та якість створюваного продукту.

професійні менеджери виступають в якості консультантів і наставників команди, і не виконують функцій контролю над ним - в ефективному колективі кожен з його членів має необхідні повноваження для виконання своїх обов'язків і впевнений, що отримає все необхідне від колег.

Як випливає з вищесказаного, однією з характерних особливостей MSF є відсутність посади менеджера проекту!

Модель команди проекту MSF пропонує поділ великих команд (більше 10 осіб) на невеликі мультидисциплінарні групи областей (функціональні команди). Ці маленькі команди працюють паралельно, регулярно синхронізуючи свої зусилля. Крім того, коли рольовий кластер вимагає багато ресурсів, формуються так звані функціональні команди, які потім об'єднуються в рольові кластери.

Використання рольових кластерів не передбачає і не нав'язує ніякої спеціальної організаційної структури або обов'язкових посад. Адміністративний склад ролей може сильно відрізнятися в різних організаціях і командах проектів. Найчастіше ролі розподіляються між різними підрозділами однієї організації, але іноді частина з них присвоюється споживчій спільноті або консультантам і партнерам, зовнішнім для організації. Ключ полягає в тому, щоб чітко визначити співробітників, відповідальних за кожен кластер ролей, їх функції, обов'язки та очікуваний внесок у кінцевий результат.

Модель команди проекту MSF не забезпечує успіх самостійно. Є багато інших факторів, які визначають успіх або провал проекту, але структура команди проекту, безумовно, робить значний внесок.

Відповідна структура команди є основою успіху, а впровадження моделі MSF з використанням основних принципів допоможе зробити проектні команди більш ефективними і, як наслідок, більш успішними.

1. **Методологія RUP.**

*Основні принципи:*

1 – рання ідентифікація та постійна (до кінця проекту) усунення основних ризиків

2 — концентрація на виконанні вимог замовника до виконуваної програми

3 – очікування змін вимог, проектнихрішень та впровадження в процесі розробки

4 — компонентна архітектура,реалізована і випробувана на ранніх стадіях проекту

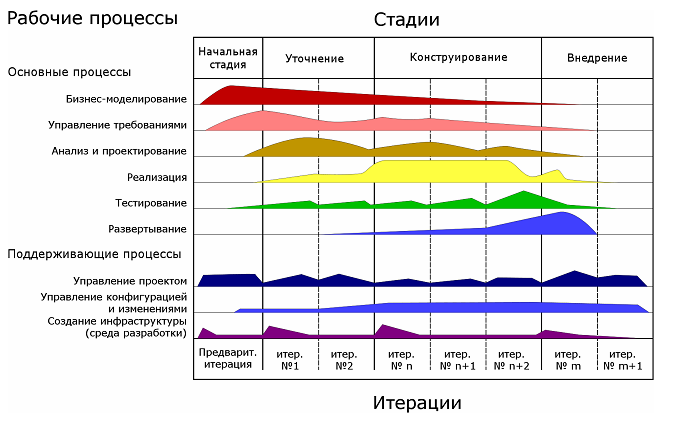
5 – постійне забезпечення якості на всіх етапах розробки проекту (продукту)

6 – робота над проектом в згуртованій команді,ключова роль в якій належить архітекторам

*Процеси та етапи РУП*

RUP використовує ітераційну модель розробки. В кінці кожної ітерації (в ідеалі тривалістю від 2 до 6 тижнів) команда проекту повинна досягти цілей, запланованих для цієї ітерації, створити або уточнити артефакти дизайну, а також отримати проміжну, але функціональну версію кінцевого продукту.

Повний життєвий цикл розробки програмного забезпечення складається з 4 етапів:



*Методологія РУП*

1.Початковий етап (початок)

На початковому етапі:

*Формуються бачення і межі проекту.*

*Створюється бізнес-кейс.*

Визначено основні вимоги, обмеження та ключовий функціонал продукту.

Створено базову версію моделі випадку використання.

*Оцінюються ризики.*

В кінці початкового етапу оцінюється досягнення етапу життєвого циклу поставленої мети, що передбачає згоду зацікавлених сторін на продовження проекту.

*Розробка*

На етапі уточнення ви аналізуєте домен і будуєте виконувану архітектуру,якавключає в себе:

Документація вимог (включаючи детальний опис для більшості випадків використання).

Розроблена, реалізована та перевірена виконувана архітектура.

Оновлений бізнес-кейс і більш точні оцінки термінів і вартості.

Зниження основних ризиків.

Успішне завершення етапу розробки означає досягнення стадії життєвого циклу архітектури

*Будівництво*

На етапі Build реалізована велика частина функціональності продукту.Фаза build закінчується першим зовнішнім випуском системи і віхою початкової функціональної готовності.

*Перехід*

на етапі «реалізації» остаточна версія продукту створюється і передається від розробника замовнику. етапи готового продукту і завершення повного циклу розробки.

**Висновок:** застосував набуті знаня передпроєктного обстеження прикладних системдля вирішення задачі узгодження моделі специфікації вимог до ІС з відомимиметодологіями архітектурного проєктування ІС.