**РЕФЕРАТ**

Пояснювальна записка до дипломної роботи містить ?? листів, ?? ілюстрацій, ?? таблиць, 1 додаток. Було використано ?? джерел.

Дана робота присвячена плануванню регресійного тестування під час процесу розробки програмного забезпечення. Були розглянуті методи визначення необхідної множини тестів, умови керованості процесу регресійного тестування, класифікація тестів при відборі, існуючі методи вибору тестів. Розроблений новий метод вибору необхідної множини тестів оснований на побудові когнітивної карти змін у програмному продукті з використанням систем контролю версій.

На основі розробленого методу і за допомоги технології .Net і мови програмування С# був реалізований програмний продукт «Regression Viewer». Програма повністю реалізує розроблений метод, підтримує використання будь-якої системи контролю версій і готова до використання під час процесу тестування програмного забезпечення.

Як результат, були зроблені висновки щодо актуальності застосування розробленого програмного забезпечення.

Ключові слова: ТЕСТУВАННЯ, РЕГРЕССІЙНЕ ТЕСТУВАННЯ, МЕТОДИ ВИБОРУ ТЕСТІВ, КОГНІТИВНА КАРТА, СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ.

**ABSTRACT**

Explanatory note to the diploma project contains?? Sheets, ?? illustrations, ?? tables, 1 appendix. There were used ?? sources.

This work was devoted to planning regression testing during the software development. In project were reviewed: methods of identifying the necessary set of tests, the conditions of controllability of the process of regression testing, classification of the tests during the selection, the existing methods of the tests selection. A new method of selecting the required set of tests based on the construction of the cognitive maps of changes in the software with use of version control systems was proposed.

On the basis of the proposed method and using .Net technology and C # programming language “Regression Viewer” software was implemented. The program fully implements the developed method, supports the use of any version control system and is ready for use during the process of software testing.

As a result, the conclusions were made on the actual application of the developed software.

Keywords: TESTING, REGRESSION TESTING, METHODS OF TEST SELECTION, COGNITIVE MAP, VERSION CONTROL SYSTEM.

**Зміст**

[ВСТУП 7](#_Toc295080331)

[ПОСТАНОВКА ЗАДАЧІ 8](#_Toc295080332)

[РОЗДІЛ 1 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА СИСТЕМИ КЕРУВАННЯ ВЕРСІЯМИ 9](#_Toc295080333)

[1.1 Види тестування 10](#_Toc295080334)

[1.2 Регресійне тестування 12](#_Toc295080335)

[1.2.1.1 Визначення необхідної множини тестів 13](#_Toc295080336)

[1.2.1.2 Умови керованості і особливості реалізації регресійного тестування 15](#_Toc295080337)

[1.2.1.3 Класифікація тестів при відборі 18](#_Toc295080338)

[1.2.1.4 Класифікація методів вибору теcтів 21](#_Toc295080339)

[1.2.1.1 Методи вибору тестів 24](#_Toc295080340)

[1.3 Системи керування версіями 27](#_Toc295080341)

[1.3.1 Традиційні системи контролю версій 29](#_Toc295080342)

[1.3.1.1 Атомарні операції 29](#_Toc295080343)

[1.3.1.2 Метод блокування файлів 29](#_Toc295080344)

[1.3.1.3 Метод злиття 30](#_Toc295080345)

[1.3.1.4 Базові точки, етикетки і бирки 30](#_Toc295080346)

[1.3.2 Розподілені системи контролю версій 32](#_Toc295080347)

[РОЗДІЛ 2 ПОБУДОВА МЕТОДУ ВИБОРУ ТЕСТІВ ДЛЯ РЕГРЕСІЙНОГО ТЕСТУВАННЯ 33](#_Toc295080348)

[РОЗДІЛ 3 ОПИС ПРОГРАМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 34](#_Toc295080349)

[3.1 Посібник користувача 35](#_Toc295080350)

[3.2 Карта переходів програмного забезпечення 40](#_Toc295080351)

[3.3 ER-модель бази даних 41](#_Toc295080352)

[3.4 Діаграма залежності класів програми 42](#_Toc295080353)

[РОЗДІЛ 4 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 43](#_Toc295080354)

[4.1 Вступна частина. 43](#_Toc295080355)

[4.2 Постановка задачі техніко-економічного аналізу 45](#_Toc295080356)

[4.3 Обґрунтування функцій програмного продукту 46](#_Toc295080357)

[4.3.1 Формування варіантів функцій 46](#_Toc295080358)

[4.3.2 Варіанти реалізації основних функцій 47](#_Toc295080359)

[4.4 Обґрунтування системи параметрів ПП 49](#_Toc295080360)

[4.4.1 Опис параметрів 49](#_Toc295080361)

[4.4.2 Кількісна оцінка параметрів 49](#_Toc295080362)

[4.4.3 Аналіз експертного оцінювання параметрів 51](#_Toc295080363)

[4.5 Аналіз рівня якості варіантів реалізації функцій 54](#_Toc295080364)

[4.6 Економічний аналіз варіантів розробки ПП 55](#_Toc295080365)

[4.7 Вибір кращого варіанта ПП техніко-економічного рівня 58](#_Toc295080366)

[4.8 Висновки 59](#_Toc295080367)

[РОЗДІЛ 5 РОЗДІЛ З ОХОРОНИ ПРАЦІ 60](#_Toc295080368)

[5.1 Вступ 60](#_Toc295080369)

[5.2 Аналіз шкідливих та небезпечних факторів 61](#_Toc295080370)

[5.3 Параметри мікроклімату 61](#_Toc295080371)

[5.3.1 Освітлення 62](#_Toc295080372)

[5.3.2 Шум і вібрація 63](#_Toc295080373)

[5.3.3 Електромагнітне і іонізуюче випромінювання 63](#_Toc295080374)

[5.3.4 Ергономічні вимоги до робочого місця 64](#_Toc295080375)

[5.4 Правила пожежної безпеки 66](#_Toc295080376)

[5.4.1 Категорії приміщень за вибухопожежною та пожежною безпекою 67](#_Toc295080377)

[5.4.2 Класи пожежі, пожежна сигналізація 67](#_Toc295080378)

[ДОДАТКИ 69](#_Toc295080379)

[ДОДАТОК А. КОД ПРОГРАМИ 69](#_Toc295080380)

##### ВСТУП

За останні роки спостерігається тенденція зростання технічної і функціональної складності прикладного забезпечення. Рівень якості програм має бути відповідно високим, щоб знайти своє місце на ринку, проте технології більшості процесів тестування не встигають за новими технологіями програмування. Невідповідність, що виникає при цьому, піддає ризику якість програм та бюджет проекту, отже процес тестування потребує перегляду.

##### ПОСТАНОВКА ЗАДАЧІ

В даній роботі необхідно:

- Проаналізувати класичні підходи та методи регресійного тестування

- Проаналізувати основні принципи роботи систем керування версіями

- Розробити метод регресійного тестування оснований на системах контролю версій

- Розробити програмне забезпечення для визначення обсягу регресійного тестування враховуючи:

* Список модулів програмного забезпечення і вихідних файлів цих модулів
* Когнітивну карту перехресного впливу модулів програми
* Список змін в програмному забезпеченні експортований з системи контроля версій

# ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА СИСТЕМИ КЕРУВАННЯ ВЕРСІЯМИ

Тестування програмного забезпечення — це процес, що використовується для виміру якості розроблюваного [програмного забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F). Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126[?]. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись.[?] До цього процесу входить виконання [програми](http://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) з метою знайдення помилок.

Існуючі на сьогоднішній день методи тестування програмного забезпечення не дозволяють однозначно і повністю виявити всі дефекти і встановити коректність функціонування програми що аналізується, тому всі існуючі методи тестування діють в рамках формального процесу перевірки досліджуваного або розроблюваного програмного забезпечення.

Такий процес формальної перевірки або верифікації може довести, що дефекти відсутні з точки зору використовуваного методу. (Тобто немає ніякої можливості точно встановити або гарантувати відсутність дефектів у програмному продукті з урахуванням людського фактора, присутнього на всіх етапах життєвого циклу ПЗ).

## Види тестування

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування. Зазвичай виділяють наступні[?]:

* За об'єктом тестування:
  + Функціональне тестування (functional testing)
  + Тестування продуктивності (performance testing)
  + Тестування навантаження (load testing)
  + Стрес-тестування (stress testing)
  + Тестування стабільності (stability / endurance / soak testing)
  + Тестування зручності використання (usability testing)
  + Тестування інтерфейсу користувача (UI testing)
  + Тестування безпеки (security testing)
  + Тестування локалізації (localization testing)
  + Тестування сумісності (compatibility testing)
* За знанням системи:
  + Тестування чорного ящика (black box)
  + Тестування білого ящика (white box)
  + Тестування сірого ящика (grey box)
* За ступенем автоматизації:
  + Ручне тестування (manual testing)
  + Автоматизоване тестування (automated testing)
  + Напівавтоматизоване тестування (semiautomated testing)
* За ступенем ізольованості компонентів:
  + Компонентне (модульне) тестування (component / unit testing)
  + Інтеграційне тестування (integration testing)
  + Системне тестування (system / end-to-end testing)
* За часом проведення тестування:
  + Тестування нової функціональності (new feature testing)
  + Регресійне тестування (regression testing)
  + Альфа-тестування (alpha testing)
  + Бета-тестування (beta testing)
  + Тестування при здачі (acceptance testing)
  + Тестування під час приймання (smoke testing)
* За ознакою позитивності сценаріїв:
  + Позитивне тестування (positive testing)
  + Негативне тестування (negative testing)
* За ступенем підготовленості до тестування:
  + Тестування з документації (formal testing)
  + Інтуїтивне тестування (ad hoc testing)

## Регресійне тестування

Регресійне тестування (regression testing) – збірна назва для всіх видів тестування програмного забезпечення, спрямованих на виявлення помилок у вже протестованих ділянках вихідного коду[?]. Такі помилки, що виникають коли після внесення змін до програми перестає працювати те, що повинно було продовжувати працювати, - називають регресійними помилками (regression defects).

Регресійне тестування включає[?]: перевірку виправленого знайденого раніше дефекту, перевірку, що виправлений раніше і верифікований дефект не відтворюється в системі знову, а також перевірку того, що не порушилася працездатність працюючої раніше функціональності, якщо її код міг бути задітий при виправленні деяких дефектів в іншій функціональності. Звичайно використовувані методи регресійного тестування включають повторні прогони попередніх тестів, а також перевірки, чи не потрапили регресійні помилки в чергову версію в результаті злиття коду.

Регресійне тестування – дорогий[?], але необхідний етап тестування програмного забезпечення, спрямований на повторну перевірку коректності зміненої програми. Тому важливим завданням регресійного тестування є також зменшення вартості і скорочення часу виконання тестів.

#### Визначення необхідної множини тестів

Нехай - множина з ***N*** тестів, що використовується при первинній розробці програми ***P***, а - підмножина регресійних тестів для тестування нової версії програми ***P'***. Інформація про покриття коду, яке забезпечується , дозволяє вказати блоки ***P'***, що потребують додаткового тестування, для чого може знадобитися повторний запуск деяких тестів з множини , або навіть створення - набору нових тестів для ***P'*** - і оновлення . Правильне виявлення множини є дуже важкою задачею.

Завдання відбору тестів з набору для заданої програми ***P*** і зміненої версії цієї програми полягає у виборі підмножини для повторного запуску на зміненій програмі , де . Так як вихідні дані ***P*** і для тестів з множини завідомо однакові, немає необхідності виконувати жоден з тестів на ***P'***. У загальному випадку, за відсутності динамічної інформації про виконання ***P*** і не існує методу обчислення множини для довільних множин ***P*** , і . Це випливає з відсутності загального вирішення проблеми зупинки[?], що складається в неможливості створення в загальному випадку алгоритму, що дає відповідь на питання, чи завершується коли-небудь довільна програма ***P*** для заданих значень вхідних даних. На практиці створення можливо тільки шляхом виконання на кожного регресійного тесту, чого і хочеться уникнути.

Реалістичний варіант вирішення завдання вибіркового регресійного тестування полягає в отриманні корисної інформації за результатами виконання ***P*** і об'єднання цієї інформації з даними статичного аналізу для отримання множини у вигляді апроксимації . Цей підхід застосовується у всіх відомих вибіркових методах регресійного тестування, заснованих на аналізі коду. Множина повинно включати всі тести з , що активують змінений код, і не включати ніяких інших тестів, тобто тест входить до тоді і тільки тоді, коли задіє код ***P*** в точці, де в код був видалений або змінений, або де був доданий новий код.

Якщо певний тест задіює в ***P*** той самий код, що і в , вихідні дані ***P*** і для відрізнятися не будуть. З цього випливає, що якщо , повинен задіяти певний код, змінений в по відношенню до , тобто повинно виконуватися відношення . З іншого боку, оскільки не кожне виконання зміненого коду відбивається на вихідних значеннях тесту, можуть існувати деякі такі , що . Таким чином, містить цілком і може використовуватися в якості його альтернативи без шкоди для якості програмного продукту.

#### Умови керованості і особливості реалізації регресійного тестування

Протягом життєвого циклу програми період супроводу триває найдовше. Коли змінена програма тестується набором тестів , ми зберігаємо без змін по відношенню до тестування вихідної програми всі фактори, які могли б впливати на вихід програми. Тому атрибути конфігурації, в якій програма тестувалася останній раз (наприклад, план тестування, тести і елементи що покриваються ), підлягають управлінню конфігурацією. Практика тестування зміненої версії програми в тих же умовах, в яких тестувалася вихідна програма , називається керованим регресійним тестуванням. При некерованому регресійному тестуванні деякі властивості методів регресійного тестування можуть змінюватися, наприклад, безпечний метод відбору тестів може перестати бути безпечним. У свою чергу, для забезпечення керованості регресійного тестування необхідно виконання низки умов:

* Як при модульному, так і при інтеграційному регресійному тестуванні в якості модулів, що викликаються модулем, що тестується безпосередньо чи опосередковано, мають використовуватися реальні модулі системи. Це легко здійснити, оскільки на етапі регресійного тестування всі модулі доступні в завершеному вигляді.
* Інформація про зміни коректна. Інформація про зміни вказує на змінені модулі та розділи специфікації вимог, не маючи на увазі при цьому коректність самих змін. Крім того, при зміні специфікації вимог необхідне посилене регресійне тестування змінених функцій цієї специфікації, а також всіх функцій, які могли бути порушені з необережності. Єдиним випадком коли ми змушені покластися на правильність зміненого технічного завдання, є зміна технічного завдання для всієї системи або для модуля верхнього (у графі викликів) рівня, за умови, що крім технічного завдання, не існує ніякої додаткової документації і / або будь-якої інший інформації, за якою можна було б судити про помилку в технічному завданні.
* У програмі немає помилок, крім тих, які могли виникнути через її зміни.
* Тести, що застосовувалися для тестування попередніх версій програмного продукту, доступні, при цьому протокол прогону тестів складається з вхідних даних, вихідних даних і траєкторії. Траєкторія являє собою шлях у керуючому графові програми, проходження якого викликається використанням деякого набору вхідних даних. Її можна застосовувати для оцінки структурного покриття, забезпечуваного набором тестів.

Для проведення регресійного тестування з використанням існуючого набору тестів необхідно зберігати інформацію про результати виконання тестів на попередніх етапах тестування.

Перерахуємо деякі особливості реалізації регресійного тестування.

* Деякі ділянки коду програми не одержують керування при виконанні деяких тестів.
* Якщо ділянка коду реалізує вимогу, але змінений фрагмент коду не отримує управління при виконанні тесту, то він і не може впливати на значення вихідних даних програми при виконанні даного тесту.
* Навіть якщо ділянка коду, що реалізує вимоги, отримує управління при виконанні тесту, це далеко не завжди відбивається на вихідних даних програми при виконанні даного тесту. Дійсно, якщо змінюється перший блок програми, наприклад, шляхом додавання ініціалізації змінної, всі шляхи у програмі також змінюються, і, як наслідок, вимагають повторного тестування. Проте може так трапитися, що тільки на невеликому підмножині шляхів дійсно використовується ця ініціалізована змінна.
* Не кожен тест, перевіряючий код, що знаходиться на одному путі зі зміненим кодом, обов'язково покриває цей змінений код.
* Код, що знаходиться на одному путі зі зміненим кодом, може не впливати на значення вихідних даних змінених модулів програми.
* Не завжди кожен оператор програми впливає на кожен елемент її вихідних даних.

Припустимо, що зміни у програмі обмежуються одним оператором. Якщо при виконанні якого-небудь тесту на вихідній програмі цей оператор ніколи не отримує управління, можна з упевненістю сказати, що він не отримає управління і в ході виконання тесту на новій програмі, а результати тестування нової та старої програм будуть збігатися. Отже, немає необхідності виконувати цей тест на новій програмі. Зазначений метод легко можна узагальнити для випадку декількох змін: якщо тест не задіє жодного зміненого оператора, і його вхідні дані не змінилися, код, що виконується їм у зміненій програмі, буде в точності таким же, як в первинній версії. Такий тест не виявляє відмінностей між двома версіями системи, отже, немає необхідності проганяти його повторно. Якщо тест не зачіпає жодного оператора виведення, поведінка якого залежить від змінених операторів, це означає, що, незважаючи на зміни в програмі, всі оператори, які одержують керування при виконанні цього тесту, не змінять висновок системи по відношенню до попередньої версії. Таким чином, немає необхідності повторно проганяти і тести такого роду.

Отже, необхідно орієнтуватися на вибір тільки тих тестів, які покривають змінений код, що впливає, у свою чергу, на вихідні дані програми. Такий підхід гарантує, що будуть обрані тільки тести, які виявляють зміни, і метод буде точним.

#### Класифікація тестів при відборі

Створення наборів регресійних тестів рекомендується починати з множини вихідних тестів. При заданому критерії регресійного тестування всі вихідні тести ***t*** поділяються на чотири підмножини:

* Множина тестів, придатних для повторного використання. Це тести, які вже запускалися і придатні до використання, але зачіпають тільки елементи програми, що не зазнали змін. При повторному виконанні вихідні дані таких тестів співпадуть з вихідними даними, отриманими на вихідній програмі. Отже, такі тести не вимагають перезапуску.
* Множина тестів, які потребують повторного запуску. До них відносяться тести, які вже запускали, але вимагають перезапуску, оскільки зачіпають, принаймні, один змінений елемент, який підлягає повторному тестуванню. При повторному виконанні такі тести можуть давати результат, відмінний від результату, показаного на вихідній програмі. Множина тестів, які потребують повторного запуску, забезпечує хороше покриття структурних елементів навіть за наявності нових функціональних можливостей.
* Множина застарілих тестів. Ці тести, більше не можуть бути застосовні до зміненої програмі та непридатні для подальшого тестування, оскільки вони зачіпають тільки елементи, які були видалені при зміні програми. Їх можна видалити з набору регресійних тестів.
* Нові тести, які ще не запускалися і можуть бути використані для тестування.

Новий

Придатний для повторного використання

Потребує повторного запуску

Застарілий

Рисунок . – Життєвий цикл тесту

Рисунок 1.1 дає уявлення про життєвий цикл тесту. Відразу після створення тест вводиться в структуру бази даних як новий. Після виконання новий тест переходить у категорію тестів, придатних для повторного використання або застарілих. Якщо виконання тесту сприяло збільшенню поточної ступеня покриття коду, тест позначається як придатний для повторного використання. В іншому випадку він позначається як застарілий і відкидається. Існуючі тести, повторно запущені після внесення зміни в код, також класифікуються заново як придатні для повторного використання або застарілі в залежності від тестових траєкторій і використовуваного критерію тестування.

Класифікація тестів по відношенню до змін в коді вимагає аналізу наслідків змін. Тести, що активують код, що був змінений, можуть вимагати повторного запуску або виявитися застарілими. Щоб тест був включений у клас тестів, які потребують повторного запуску, він повинен покривати зміни в коді, а також має сприяти збільшенню ступеня покриття зміненого коду по використовуваному критерію. Покритим тестом елементом, що був змінений, може бути траєкторія, вихідні значення, або і те, й інше. Щоб тест був включений у клас тестів, придатних для повторного використання, він має вносити внесок у збільшення ступеня покриття коду і не вимагати повторного запуску.

Ступінь покриття коду визначається для тестів, придатних для повторного використання, оскільки до цього класу відносяться тести, які не потребують повторного запуску, які сприяють збільшенню ступеня покриття до бажаної величини. Якщо є компонент програми, не задіяний придатними для повторного використання тестами, то замість них вибираються і виконуються з метою збільшення ступеня покриття тести, що вимагають повторного запуску. Після запуску такий тест стає придатним для повторного використання або застарілим. Якщо тестів, які потребують повторного запуску, більше не залишилося, а необхідна ступінь покриття коду ще не досягнута, породжуються додаткові тести і тестування повторюється.

Остаточний набір тестів збирається з тестів, придатних для повторного використання, тестів, які потребують повторного запуску, і нових тестів. Нарешті, застарілі й надлишкові тести видаляються з набору тестів, оскільки надлишкові тести не перевіряють нові функціональні можливості і не збільшують покриття.

#### Класифікація методів вибору теcтів

Для перевірки коректності різних підходів до регресійного тестування використовується модель оцінки методів регресійного тестування. Основними об'єктами розгляду стали повнота, точність, ефективність та універсальність.

Повнота відображає міру відбору тестів з , на яких результат виконання зміненої програми відмінний від результату виконання вихідної програми, внаслідок чого можуть бути виявлені помилки в . Метод, повний на 100%, називається безпечним.

Точність - міра здатності методу уникати вибору тестів з , на яких результат виконання зміненої програми не буде відрізнятися від результату її первинною версією, тобто тестів, нездатних виявляти помилки в . Припустимо, що набір містить регресійних тестів. З них для тестів поведінка та результати виконання старої програми відрізняються від поведінки і результатів виконання нової програми . Набір тестів містить тестів, отриманих з використанням методу відбору регресійних тестів . З цих тестів для тестів поведінка і розрізняється. Точність щодо , , і , виражена у відсотках, визначається виразом , тоді як відповідний відсоток вибраних тестів визначається виразом , якщо або дорівнює ***100%*** , якщо .

Виходячи з наведеного визначення, точність множини тестів - це відношення числа тестів даної множини, на яких результати виконання нової та старої програм розрізняються, до загального числа тестів множини. Точність є важливим атрибутом методу регресійного тестування. Неточний метод має тенденцію відбирати тести, які не повинні були бути обрані. Чим менш точний метод, тим ближче обсяг обраного набору тестів до обсягу вихідного набору тестів.

Ефективність - оцінка обчислювальної вартості стратегії вибіркового регресійного тестування, тобто вартості реалізації її вимог щодо часу і пам'яті, а також можливості автоматизації. Відносною ефективністю називається ефективність методу тестування за умови наявності не більше однієї помилки в програмі, що тестується. Абсолютною ефективністю називається ефективність методу в реальних умовах, коли оцінка кількості помилок в програмі не обмежена.

Універсальність відображає міру здатності методу до застосування в досить широкому діапазоні ситуацій, що зустрічаються на практиці.

Для програми , її зміненої версії і набору тестів для потрібно, щоб методика вибіркового повторного тестування задовольняла наступним критеріям оцінки:

Критерій 1. Безпека. Методика вибіркового повторного тестування повинна бути безпечною, тобто повинна вибирати всі тести з , які потенційно можуть виявляти помилки (усі тести, чия поведінка на і може бути різним). Безпечна методика повинна розглядати наслідки додавання, видалення і зміни коду. При додаванні нового коду в у можуть вже міститися тести, що покривають цей новий код. Такі тести необхідно виявляти і враховувати при відборі.

Критерій 2. Точність. Стратегія повторного прогону всіх тестів є безпечною, але неточною. На додаток до вибору всіх тестів, потенційно здатних виявляти помилки, вона також вибирає тести, які в жодному разі не можуть демонструвати змінену поведінку. В ідеалі, методика вибіркового повторного тестування повинна бути точною, тобто повинна вибирати тільки тести зі зміненим поведінкою. Однак для довільно взятого тесту, не запускаючи його, неможливо визначити, чи зміниться його поведінку. Отже, в кращому випадку ми можемо розраховувати лише на деяке збільшення точності.

Всілякі існуючі вибіркові методи регресійного тестування розрізняються не в останню чергу вибором об'єкта або об'єктів, для яких виконується аналіз покриття та аналіз змін. Наприклад, при аналізі на рівні функції при зміні будь-якого оператора функції вся функція вважається зміненою; при аналізі на рівні окремих операторів ми можемо виключити частину тестів, що містять виклик функції, але не активують змінений оператор. Вибір об'єктів для аналізу покриття відбивається на рівні подробиці аналізу, а значить, і на його точності та ефективності. Абсолютні величини точності та кількості обраних тестів для заданих набору тестів і безлічі змін повинні розглядатися тільки разом зі зменшенням розміру набору тестів. Невеликий відсоток вибраних тестів може бути прийнятним, тільки якщо рівень точності залишається досить високим.

Критерій 3. Ефективність. Методика вибіркового повторного тестування повинна бути ефективною, тобто повинна допускати автоматизацію і виконуватися достатньо швидко для практичного застосування в умовах обмеженого часу регресійного тестування. Методика повинна також передбачати зберігання інформації про хід виконання тестів в мінімально можливому обсязі.

Критерій 4. Універсальність. Методика вибіркового повторного тестування повинна бути універсальною, тобто застосовної до всіх мов і мовних конструкцій, ефективною для реальних програм і здатною до обробки як завгодно складних змін коду.

У загальному випадку існує певний компроміс між безпекою, точністю і ефективністю. При відборі тестів аналіз необхідно провести за час, менший, ніж потрібно для виконання та перевірки результатів тестів з **T**, що не увійшли до . З урахуванням цього обмеження рішенням задачі регресійного тестування буде безпечний метод з хорошим балансом дешевизни і високої точності.

#### Методи вибору тестів

**1)Випадкові методи**

Коли через обмеження часу використання методу повторного прогону всіх тестів неможливо, а програмні методи відбору тестів недоступні, інженери, відповідальні за тестування, можуть вибирати тести випадковим чином або на підставі "здогадок", тобто можливого співвіднесення тестів з функціональними можливостями на підставі попередніх знань або досвіду.

**2)Безпечні методи**

Метод вибіркового регресійного тестування називається безпечним, якщо при деяких чітко визначених умов він не виключає тестів, які виявили б помилки у зміненій програмі, тобто забезпечує вибір всіх тестів, що виявляють зміни. Тест називається таким, що виявляє зміни, якщо його вихідні дані при прогоні на відрізняються від вихідних даних при прогоні на **.**

При деяких умовах безпечні методи в силу визначення "безпеки" гарантують, що всі помилки, що можуть бути виявлені, будуть знайдені. Тому відносна ефективність усіх безпечних методів дорівнює ефективності методу повторного прогону всіх тестів і становить 100%. Проте їх абсолютна ефективність падає зі збільшенням інтервалу тестування.

**3)Методи мінімізації**

Процедура мінімізації набору тестів ставить за мету відбір мінімальної (в термінах кількості тестів) підмножини T, необхідної для покриття кожного елемента програми, що зазнав змін. Для перевірки коректності програми використовуються тільки тести з мінімальної підмножини.

Мінімізація набору тестів вимагає певних витрат на аналіз. Якщо вартість цього аналізу більше витрат на виконання деякого порогового числа тестів, існує більш дешевий випадковий метод, що забезпечує таку ж ефективність виявлення помилок.

Хоча мінімальні набори тестів можуть забезпечувати структурне покриття зміненого коду, найчастіше вони не є безпечними, оскільки очевидно, що деякі тести, потенційно здатні виявляти помилки, можуть залишитися за межею відбору.

**4)Методи, засновані на покритті коду**

Значення методів, заснованих на покритті коду, полягає в тому, що вони гарантують збереження обраним набором тестів необхідного ступеня покриття елементів щодо деякого критерію структурного покриття , що використовувався при створенні первинного набору тестів. Це не означає, що якщо атрибут програми, визначений , покривається початковою множиною тестів, він буде також покритим і вибраною множиною; гарантується тільки збереження відсотка покриття коду. Методи, засновані на покритті, зменшують розкид по покриттю, вимагаючи відбору тестів, що активують важкодоступний код, і виключення тестів, які тільки дублюють покриття. Оскільки на практиці критерії покриття коду зазвичай застосовуються для відбору єдиного тесту для кожного елемента, що покривається, підходи, засновані на покритті коду, можна розглядати як специфічний вид методів мінімізації. Метод стовідсоткового покриття зміненого коду аналогічний методу мінімізації.

## Системи керування версіями

Система управління ревізіями, також відома як система контролю версій або система управління – є система управління змінами в документах, програмах та іншій інформації що зберігається у вигляді файлів на комп'ютері[?]. Найбільш часто використовується в розробці програмного забезпечення, де група людей може змінювати одні і ті ж файли. Зміни, як правило, позначаються певною кодовою цифрою або літерою, що називається "номер ревізії". Наприклад, початковий набір файлів називають "ревізія 1". Коли перша зміна зроблена, результуюча множина називається "ревізія 2", і так далі. Кожна ревізія пов'язана з відміткою про час зміни та особу, що вносила зміну. Ревізії можна порівнювати, відновлювати більш стару ревізію, і з деякими типами файлів – зливати дві ревізії створені різними людьми у одну – об’єднану.

Системи контролю версій найбільш часто використовуються як самостійні програми, але контроль версій також вбудований у різні види програмного забезпечення, такі як текстові процесори (наприклад, Microsoft Word, OpenOffice.org Writer, KWord, Pages), процесори електронних таблиць (наприклад, Microsoft Excel, OpenOffice.org Calc, KSpread, Numbers), а також у різні системи управління контентом (наприклад, Drupal, Joomla, WordPress). Вбудований контроль ревізій є ключовою особливістю wiki пакетів програмного забезпечення, таких як MediaWiki, DokuWiki, TWiki т.д. У wiki, контроль версій дає здатність повернутися до сторінки попередньої ревізії, яка має вирішальне значення для редакторів і дозволяє відстежувати правки один одного, виправляти помилки, і захищати суспільні wiki від вандалізму і спаму.

У галузі розробки програмного забезпечення, контроль версій є єдиною можливістю відстежувати і забезпечувати контроль за змінами у вихідному коді. Розробники іноді використовують програмне забезпечення контролю версій для збереження документації і конфігураційних файлів так само як вихідного коду.

Команди розробників одночасно проектують, розробляють та впроваджують декілька версій одного і того ж програмного забезпечення, які будуть розгорнуті в різних місцях, і одночасно працюють над оновленнями(patches). Певні помилки або функції програмного забезпечення часто присутні тільки в деяких версіях (через фіксацію деяких проблем і впровадження інших, програма розвивається). Таким чином, з метою пошуку та виправлення помилок, життєво важливо мати можливість контролювати і запускати різні версії програмного забезпечення, щоб визначити в якій версії виникає проблема.

На найпростішому рівні, розробники могли просто зберегти кілька копій різних версій програми, і позначити їх відповідним чином. Цей простий підхід був використаний на багатьох великих програмних проектах. Хоча цей метод може працювати, це неефективно,так як багато майже ідентичних копії програми повинні бути збережені. Це вимагає гарної дисципліни з боку розробників, і часто призводить до помилок. Отже, були розроблені системи для автоматизації деяких або всіх процесів контролю версій.

Крім того, в розробці програмного забезпечення, в правовій та бізнесовії сферах все частіше однин документ або фрагмент коду редагує вся команда, члени якої можуть бути географічно розділені і можуть переслідувати різні і навіть протилежні інтереси . Складна система контролю версій, яка відстежує і встановлює хто змінював певну частину документу може бути дуже корисною і навіть необхідною.

### Традиційні системи контролю версій

Традиційні системи контролю версій використовують централізовану модель, в якій усі функції контролю версій відбуваються на сервері. Якщо два розробники намагаються змінити той же файл в той же час, без будь-якого методу управління доступом розробники можуть в кінцевому підсумку перезаписати роботу один одного. Централізовані системи контролю версій вирішують цю проблему однією з двох можливих моделей управління : блокування файлів і злиття версій.

#### Атомарні операції

Вчені з комп'ютерних наук говорять о атомарних операціях, якщо система залишається в узгодженому стані, навіть якщо операція перервана. Як правило, найбільш важливим є щоб команда фіксування змін була атомарною у цьому сенсі. Операція фіксування говорить системі контролю версій що ви хочете, щоб група змін, що ви робили стали доступним для всіх користувачів. Не всі системи контролю версій мають можливість атомарних змін; зокрема, широко використовувана система CVS не має цієї функції.

#### Метод блокування файлів

Найпростіший спосіб запобігання "одночасного доступ" передбачає блокування файлів, так що тільки один розробник в той час, має доступ на запис до центрального "сховища" копії певних файлів. Після того як один розробник заблокував файл, інші можуть прочитати цей файл, але ніхто не може змінити цей файл, поки розробник не зафіксує оновлену версію (чи скасує блокування).

Блокування файлів має як переваги і недоліки. Воно може забезпечити деякий захист від важких конфліктів злиття, коли користувач вносить радикальні зміни в багато розділів великого файлу (або групи файлів). Однак, якщо файли залишаються заблокованими занадто довго, інші розробники можуть бути змушені обійти програмне забезпечення контролю версій і змінювати файли локально, що призводить до більш серйозних проблем.

#### Метод злиття

Більшість систем керування версіями дозволяє декільком розробникам редагувати той же файл в той же час. Перший розробник фіксує зміни в центральному репозиторії завжди успішно, при цьому система повинна забезпечити можливості для подальшого злиття змін в центральне сховище, і зберегти зміни першого розробника, коли інші розробники будуть фіксувати свої зміни.

Злиття двох файлів – дуже тонка операція, і, як правило можливе тільки тоді, коли структура файлів, проста, як в текстових файлах. Злиття двох файлів зображень не може призвести до результуючого файлу зображення. Також розробник повинен перевірити результат злиття, щоб переконатися, що зміни сумісні і що операція злиття не вносить свої власні помилки логіки у файли. Метод злиття може передбачати блокування файлів для ексклюзивного доступу на запис, навіть якщо можливість злиття існує.

#### Базові точки, етикетки і бирки

Більшість інструментів контролю версій використовує тільки один з цих подібних методів(базова точка, етикетка, бирка) для позначення дій виявлення знімку проекту в певний час. Ці три назви можна вважати синонімами.

У більшості проектів деякі знімки є більш значущими, ніж інші, такі, як ті, які використовуються для позначення опублікованих релізів, версій, або оновлень.

### Розподілені системи контролю версій

Розподілені системи контролю версій використовують так званий Peer -To-Peer підхід, на відміну від клієнт-серверного підходу централізованих систем. Замість одного, центрального сховища, з яким клієнти синхронізуються, кожний розробник має свою робочу копію у своєму локальному сховищі. Розподілені системи контролю версій проводять синхронізацію шляхом обміну патчами між собою. У результаті вони мають деякі важливі відмінності від централізованої системи:

* Найчастіше використовувані операції (наприклад, фіксація, перегляд історії, і скасування змін) відбуваються швидко, тому що немає необхідності спілкуватися з центральним сервером.
* Мережевий зв'язок необхідний тільки при завантаженні або вивантаженні змін іншим розробникам.
* Кожна робоча копія ефективно функціонує в якості віддаленого резервного серверу збереження коду, історії, і забезпечуює захист від втрати даних.

# ПОБУДОВА МЕТОДУ ВИБОРУ ТЕСТІВ ДЛЯ РЕГРЕСІЙНОГО ТЕСТУВАННЯ

# ОПИС ПРОГРАМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Створений програмний продукт призначений для оцінки потрібних розмірів регресійного тестування для будь-яких проектів що зберігають інформацію в системах контролю версій.

Концепція оцінки потрібних розмірів регресійного тестування полягає в тому що б максимально зменшити кількість тестів, що треба запустити, за рахунок виключення з тестування тих модулів що не були змінені в даній версії/патчі.

## Посібник користувача

Програма була реалізована за допомогою мови програмування С#. Для зберігання даних використовується база даних SQLite3. Для побудови когнітивних карт зв’язків використовувалася бібліотека MSAGL(Microsoft Automatic Graph Layout)[?].

При відкритті програми доступним є тільки пункт меню «Project». За його допомогою можна створити новий проект або відкрити вже існуючий:

* Для створення нового проекту потрібно вибрати місце де буде зберігатися база даних і вибрати ім’я проекту, після цього потрібно вибрати директорію у яку попередньо вивантажили вихідні коди проекту з системи контролю версій
* Щоб відкрити проект треба вказати розміщення файлу бази даних

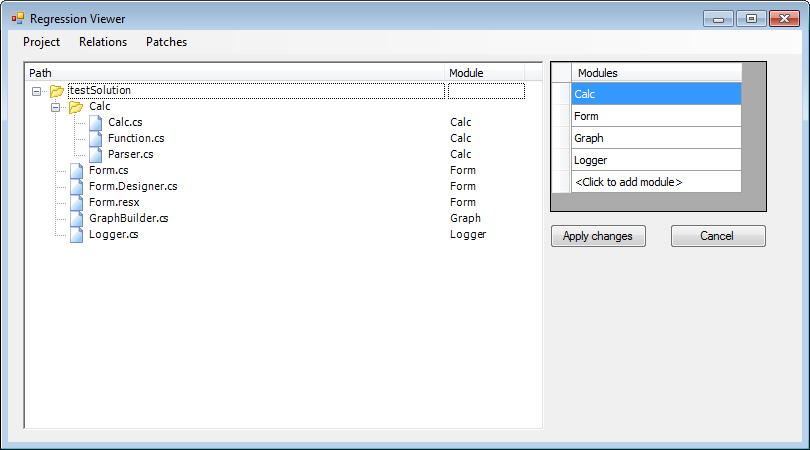


Рисунок . – Основна форма

Після створення або відкриття проекту користувач може(рисунок 3.1): переглянути які файли входять до проекту через дерево файлів і папок, що розміщено на основній формі; додати, видалити або змінити ім’я модуля програми; установлювати, видалити або змінити зв'язок між файлом та модулем.

Після того як необхідні модулі були додані, зв’язки установлені, користувач повинен установити зв’язки між файлами проекту. Для цього потрібно викликати меню «Relations->Files Relations». Відкриється форма «Relations View» (рисунок 3.2). Форма показує який файл з якого модулю використовує інший файл, та силу зв’язку. Форма дає можливість додати(для цього використовується форма «Add Relation» (рисунок 3.3)) та видалити зв'язок між файлами.

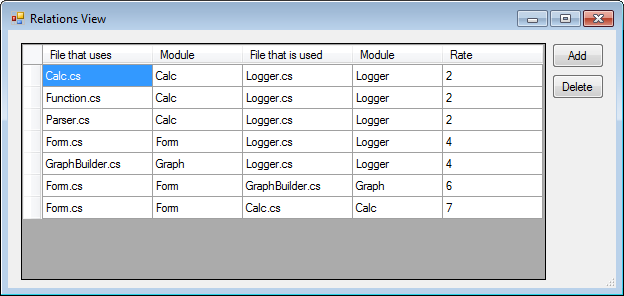


Рисунок . – Форма зв’язків

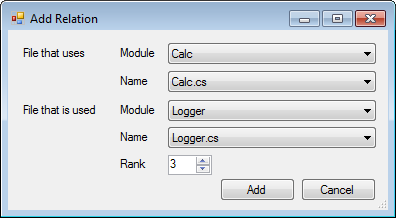


Рисунок . – Форма додавання звязків

Після того як були визначені зв’язки між файлами, можна подивитися на когнітивну карту зв’язків між модулями(рисунок 3.4). Для цього потрібно вибрати пункт меню «Relations->Modules Graph».

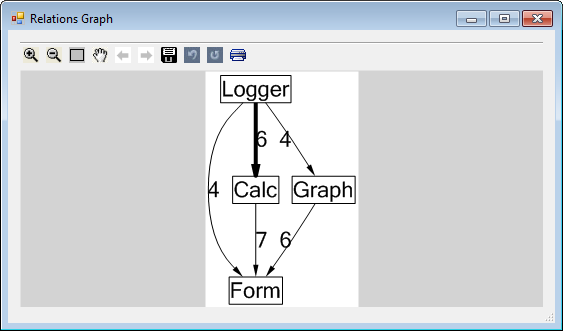


Рисунок . – Когнітивна карта зв’язків між модулями

Меню «Patches» дозволяю створювати, дивитися когнітивну карту патчу, та видаляти патч

Щоб додати патч потрібно вибрати пункт меню «Patches->Add Patch», відкриється форма з такою ж назвою (рисунок 3.5). В ній потрібно: вибрати систему контролю версій з якою працює користувач(підтримуються найпоширеніші системи керування версіями SVN і GIT), або написати .Net Regexp[?] для файлу довільного формату; вибрати файл що містить вихід певної команди історії системи керування версіями, вибрати назву для патчу. Після цього у таблиці «Preview» можна переглянути список файлів на які вплинув обраний патч. Щоб зберегти патч потрібно натиснути кнопку «Add».

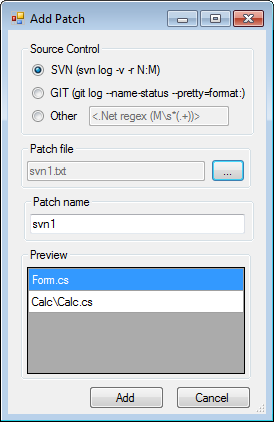


Рисунок . – Форма додавання патчу

Після додавання патчу можна переглянути його когнітивну карту через меню «Patches->Show Patch» (рисунок 3.6). На карті товщина ребер вказує на кількість зв’язків між модулями які порушує певний патч, цифра біля ребра вказує на сумарну вагу всіх зв’язків. Червоним кольором позначено безпосередні зв’язки між модулями, жовтим – зв’язки що могли бути порушені (зв’язки через один модуль), для таких зв’язків цифра при ребрі поділена на 10.

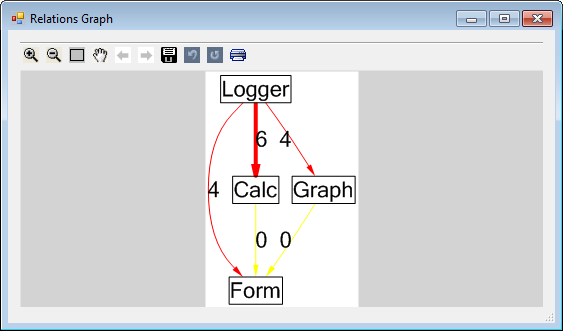


Рисунок . – Когнітивна карта патчу

## Карта переходів програмного забезпечення

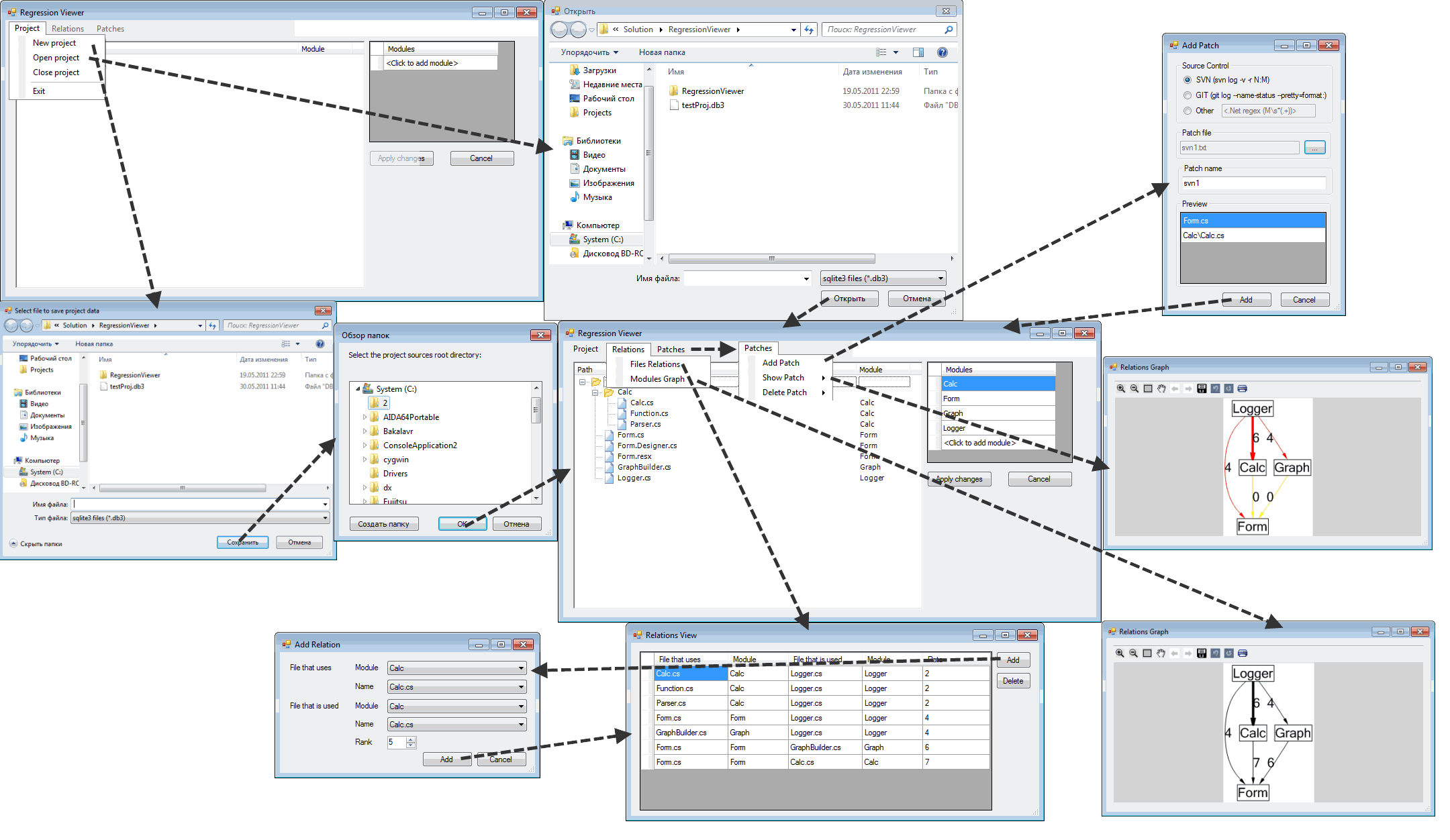


Рисунок 3.7 – Карта переходів програмного забезпечення

## ER-модель бази даних

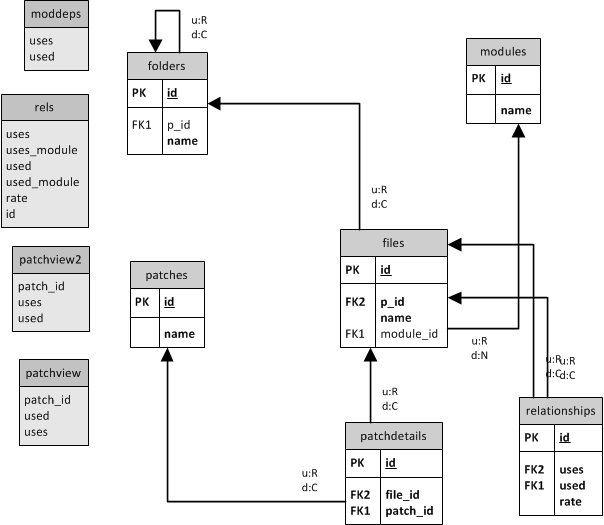


Рисунок 3.8 – ER-модель бази даних, що використовується

## Діаграма залежності класів програми

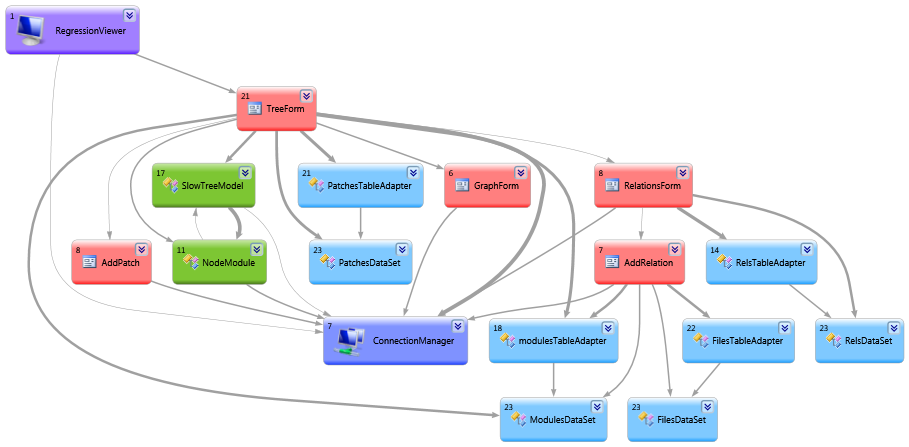


Рисунок 3.9 – Діаграма залежності класів програми

# ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

## Вступна частина.

У даній роботі проводиться оцінка програми що здійснює розрахунок оцінки обсягу потрібного регресійного тестування при процесі розроблення програмного забезпечення.

Програмний продукт призначений для використання на персональних комп’ютерах під управлінням операційної системи Windows, але також може бути використаний під іншими операційними системами, для яких розроблена підтримка платформи .Net.

Нижче наведено аналіз різних варіантів роботи програми з метою вибору оптимального, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

* визначається послідовність функцій, необхідних для виробництва продукту (в даному випадку математичного апарату). Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
* для кожної функції визначаються повні річні витрати й кількість робочих часів.
* для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
* після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на математичний апарат.

## Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки математичного апарату. Оскільки основні такі роботи стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмних продуктів, призначених для збору, обробки проміжних математичних результатів.

Відповідно цьому варто обирати і систему показників якості програмних продуктів.

Технічні вимоги до продукту наступні:

* програмні продукти повинні функціонувати на персональних комп’ютерах із стандартним набором компонент;
* забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;
* забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля.

## Обґрунтування функцій програмного продукту

### Формування варіантів функцій

Функції:

*F0*– Введення користувацьких даних про склад програмного продукту

*F1* – Збереження користувацьких даних для подальшого користування

*F2* – Побудова когнітивної карти змін

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція *F0*:

1)Розробка діалогів у програмі

2)Введення даних через сторонні програми(текстові редактори або інтегроване середовище розробки баз даних)

Функція *F1*:

1)Використання xml формату

2)Використання бази даних

3)Використання ini формату

Функція *F*2:

1)Використання модуля для побудови графів MSAGL

2)Створення власного модуля для побудови графів

### Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рисунок 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

Рис 4.3.2.1 Морфологічна карта

Рисунок . – Морфологічна карта

F11

F12

F21

F22

F23

F31

F32

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця . – Позитивно-негативна матриця

|  |  |  |  |
| --- | --- | --- | --- |
| **Основні функції** | **Варіанти реалізації** | **Переваги** | **Недоліки** |
| *F1* | *а* | Більш інформативний інтерфейс | Великі витрати часу |
|  | *б* | Займає менше часу при написанні коду | Займає більше часу при написанні коду |
| *F2* | *а* | Легкий для розуміння формат | Складна реалізація створення і читання файлу |
|  | *б* | Висока швидкість роботи, можлива легка зміна даних, побудова звітів | Великі витрати часу | *F2* | *а* | Найоптимальніший для паралельних розрахунків | Дорожче |
|  | *в* | Дуже проста реалізація | Низька швидкодія |  | *б* | Більша сумісність з іншими програмами | Менша швидкодія |
| *F3* | *а* | Гарний інтерфейс | Платний компонент |
|  | *б* | Можливість кастомізувати інтерфейс | Займає більше часу при написанні коду |

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим нами цілям. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки інтерфейс програми повинен бути зрозумілим користувачеві, то варіант 2 має бути відкинутий.

Функція *F2:*

Оскільки час виконання програмного коду є дуже важливим то варіант 3 можна відкинути.

Функція *F3:*

Оскільки розробка власного модуля побудови графів може зайняти забагато часу то варіант 2 можна відкинути.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F11 – F21– F31;
2. F11 – F22– F31;

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

## Обґрунтування системи параметрів ПП

### Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати необхідні складові даної роботи, будемо використовувати наступні параметри:

* *X1* – максимальна швидкодія програми;
* *X2* – мінімальний час обробки даних;
* *X3* – розмір файлу збережених даних;

*X1*: Відображає швидкодію операцій залежно від обраного продукту.

*X2:* Відображає час, необхідний для збереження та обробки даних під час виконання програми.

*X3:* Відображає розмір отриманого файлу даних після введення і збереження даних

### Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця . – Основні параметри ПП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Назва** п**араметра** | **Умовні позначення** | **Одиниці виміру** | **Значення параметра** | | |
|  |  |  | **гірші** | **середні** | **кращі** |
| Швидкодія продукту | X1 | Оп/мс | 100000 | 500000 | 800000 |
| Час обробки даних алгоритмом | X2 | Мс | 8000 | 4120 | 600 |
| Розмір файлу | X3 | Mb | 10 | 1 | 0,1 |

За даними таблиці 4.2 будуються графічні характеристики параметрів Гіршому значенню відповідає оцінка 0, середньому одержуваному значенню - 5, кращому значенню параметра – 10. Отже,

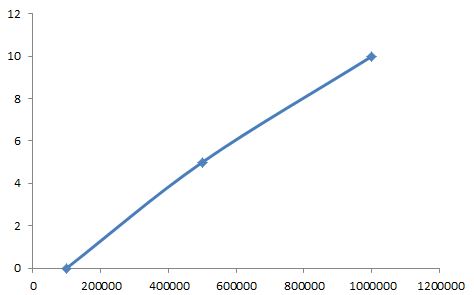


Рисунок . – Графічні характеристики для Х1

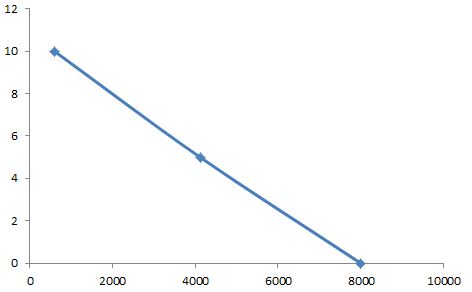


Рисунок . – Графічні характеристики для Х2

Рисунок . – Графічні характеристики для Х3

### Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі - розробка програмного продукту, який дає найбільш точні результати при знаходженні власних чисел розріджених матриць великої розмірності з найбільшою гнучкістю для подальшого розвитку продукту. Результати експертного ранжування наведені у таблиці 4.3.

Таблиця . – Результати ранжування параметрів

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Позн. параметра** | **Ранг параметра за оцінкою експерта** | | | | | | | **Сума рангів *Ri*** | **Відхилення *Δi*** | ***Δi2*** |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |  |  |  |
| *X1* | 3 | 1 | 2 | 1 | 1 | 3 | 2 | 13 | -1 | 1 |
| *X2* | 1 | 2 | 3 | 2 | 3 | 1 | 3 | 17 | +3 | 9 |
| *X3* | 2 | 3 | 1 | 3 | 2 | 2 | 1 | 12 | -2 | 4 |
| Разом | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 42 | 0 | 14 |

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:



де *N* – число експертів, *n* – кількість параметрів;

б) середня сума рангів:



в) відхилення суми рангів кожного параметра від середньої суми рангів:



Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:



Порахуємо коефіцієнт узгодженості:



Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю.

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, *aij* визначається по формулі:

1,5 при Хі>Xj



1.0 при Хі= Хj

0.5 при Хі < Xj

З отриманих числових оцінок переваги складемо матрицю A=║ *aij* ║.

Для кожного параметра зробимо розрахунок вагомості Kві за наступними формулами:



Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

Таблиця . – Розрахунок вагомості параметрів

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Параметри | Параметри | | | Перша ітер. |  | Друга ітер. |  |
|  | Х1 | Х2 | Х3 | *bi* | Кві |  |  |
| Х1 | 1,0 | 0,5 | 1,5 | 3 | 0,33 | 9 | 0,32 |
| Х2 | 1,5 | 1,0 | 1,5 | 4 | 0.44 | 16 | 0,451 |
| Х3 | 0,5 | 0,5 | 1,0 | 2 | 0,23 | 4 | 0,239 |
| Всього: |  |  |  | 9 | 1 | 29 | 1 |

Як видно з таблиці, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

## Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *Х2*(мінімальний час обробки даних алгоритмом) та *X1* (максимальна швидкодія продуктів) відповідають технічним вимогам умов функціонування.

Абсолютне значення параметра *Х3 (*точність розв’язку) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту 1Mb або варіанту 0.1Mb

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так:



де *n* – кількість параметрів;

*Квi* – коефіцієнт вагомості *i*–го параметра;

*Вi* – оцінка *i*–го параметра в балах.

Таблиця 4.5 – Розрахунок показників рівня якості варіантів реалізації основних функцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Основні функції** | **Варіант реалізації функції** | **Абсолютне значення параметра** | **Бальна оцінка параметра** | **Коефіцієнт вагомості параметра** | **Коефіцієнт рівня якості** |
| F1 | А | 800000 | 7 | 0,333 | 2,331 |
|  | В | 500000 | 5 | 0,238 | 1,19 |
| F2 | А | 700 | 9 | 0,428 | 3,852 |

За цими даними визначаємо рівень якості кожного з варіантів:

*КТЕР1* = 2,331 + 3,852= 6,183

*КТЕР*2 = 1,19 + 3,852= 5,042

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

## Економічний аналіз варіантів розробки ПП

Визначимо деякі параметри, необхідні для проведення вартісного аналізу розробки програмного продукту.

Всі варіанти включають в себе два окремих завдання:

1. Розробка ділогів введення даних

2.Створення модулю побудови графа.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: Тр=90 людино - днів. Поправочний коефіцієнт, який враховує вид довідкової інформації для першого завдання: Кп = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: Кск = 1. Оскільки при розробці першого завдання використовуються деякі стандартні підходи, врахуємо це за допомогою коефіцієнта Кст = 0.8. Таким чином, загальна трудомісткість програмування першого завдання дорівнює:

Т1 = 90•1.7•0.8 = 122.4 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто Тр=27 людино-днів, Кп=0.9, Кст=0.8:

Т2 = 27 • 0.9 • 0.8 = 19.44 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з двох обраних варіантів реалізації, щоб отримати їх трудомісткість:

ТI = (122.4 + 19.44 + 4.8 + 19.44) • 8 = 1328.55 людино-годин;

ТII = (122.4 + 19.44 + 6.91 + 19.44) • 8 = 1345.55 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один аналітик з окладом 5650 грн., один науковий діяч з окладом 9250грн. Визначимо зарплату за годину:

грн.

Тоді зарплата розробників за варіантами становить

I. Сзп = 29,42 • 1328.55 • 1.2 = 46903,12 грн.

II. Сзп = 29,42 • 1345.55 • 1.2 = 47503,29 грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 36,77%:

I. Сот = 46903,12 • 0.3677 = 17246 грн.

II. Сот = 47503,29 • 0.3677 = 17467 грн.

Тепер визначимо витрати на оплату однієї машино-години.

Так як одна ЕОМ обслуговує одного аналітика з окладом 5650 грн, з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

СГ = 12 • 5650• 0,2 = 10360 грн.

З урахуванням додаткової заробітної плати

СЗП = 10360 • (1 + 0.2)=12630 грн.

Відрахування на єдиний соціальний внесок:

СОТ = СЗП • 0.3677 = 12630 • 0,362 = 4572,06 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

СА = 1.15 • 0.25 • 8000 = 2300 грн.

Витрати на ремонт та профілактику

СР = 1.15 • 8000 • 0.05 = 460 грн.

Ефективний годинний фонд часу ПК за рік

ТЕФ = (365 – 104 – 8 – 16) • 8 • 0.9 = 1706.4 годин

Витрати на оплату електроенергії:

СЕЛ = 1706,4 • 0,156 • 0,2436 • 2 = 129,695 грн.

Накладні витрати:

СН = 6360 • 0,67 =4261,2 грн.

Тоді, річні експлуатаційні витрати будуть:

СЕКС = 7632 + 2762,78 + 2300 + 460 + 129,69 + 4261,2 = 17545,67 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

СМ-Г = Секс / Теф = 17545,67 /1706,4 = 10,28 грн/час.

Оскільки в даному випадку всі роботи, які пов′язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації пакету, складає:

І. СМ = 10,28 • 1328,55 = 13657,50 грн.;

ІІ. СМ = 10,28 • 1345.55 = 13832,25 грн.;

Накладні витрати складають 67% від заробітної плати:

І. СН = 27118,37 • 0,67 = 18169,31 грн.;

ІІ. СН = 27422 • 0,67 = 18372,74 грн.;

Отже, вартість розробки програмного продукту за варіантами становить:

І. СПП = 46903,12 + 29549,4 + 13657,50 + 18169,31 = 108279,33 грн.;

ІІ. СПП = 47503,29 + 29880 + 13832,25 + 18372,74 = 109588,28 грн.;

## Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня:

Ктер1 = 5,214 / 144927,72 = 0,481 • 10-6;

Ктер2 = 3,569 / 146550,14 = 0,243 • 10-6;

Як бачимо, найбільш ефективним є перший варіант реалізації з коефіцієнтом техніко-економічного рівня Ктер3 = 0,481 *• 10-6.*

## Висновки

В результаті виконання економічного розділу були систематизовані і закріплені теоретичні знання в галузі економіки та організації виробництва використанням їх для техніко-економічного обґрунтування розробки методом ФСА.

На основі даних про зміст основних функцій, які мають бути реалізовані, були визначені два найбільш перспективні варіанти реалізації математичного апарату. Найбільш ефективним виявився перший варіант як простіший та не менш досконалий.

У зв’язку з чим можна зробити висновки, що даний аналіз не потребуватиме значного часу на навчання аналітиків і зможе слугувати хорошою опорою для подальшої наукової діяльності.

# РОЗДІЛ З ОХОРОНИ ПРАЦІ

## Вступ

З розвитком науково-технічного прогресу важливу роль відіграє можливість безпечного виконання людьми своїх трудових обов'язків. У зв'язку з цим була створена і розвивається наука про безпеку праці і життєдіяльності людини.

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності працівників розумової праці. Їх праця стала більш інтенсивною, напруженою, що вимагає значних витрат розумової, емоційної і фізичної енергії. Це зажадало комплексне рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці і відпочинку.

В даний час комп'ютерна техніка широко застосовується у всіх областях діяльності людини. При роботі з комп'ютером людина піддається дії ряду небезпечних і шкідливих виробничих чинників: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і ін.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташування елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора.

В процесі роботи з комп'ютером необхідно дотримувати правильний режим праці і відпочинку. Інакше у персоналу [з'являється](http://www.kavkazcenter.com/ukr/content/2011/05/09/18767.shtml) значна напруга зорового апарату з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, утомленість і хворобливі відчуття в очах, в поясниці, в області шиї і руках.

## Аналіз шкідливих та небезпечних факторів

Передбачається, що в приміщенні, в якому працює людина, присутня обчислювальна техніка. До фізичних небезпечних чинників, що діють на людину, що працює в приміщенні, відносяться:

1. Недостатня вологість повітря
2. Недостатній рівень освітленості робочого місця
3. Наявність шуму від працюючої техніки (принтер, факс)
4. Іонізуючі електромагнітні випромінювання (в приміщенні використовуються монітори з електронно-променевою трубкою, рівень електромагнітних випромінювань якого досить високий)
5. Небезпека ураження електричним струмом.  
   Біологічні, хімічно шкідливі виробничі фактори в даному приміщенні відсутні.

## Параметри мікроклімату

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вогкості в приміщенні. В приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. Норми мікроклімату встановлюються залежно від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 5.1).

Об'єм приміщень, в яких розміщені працівники обчислювальних центрів, не повинен бути меншим 19,5м3/людина з урахуванням максимального числа одночасно працюючих в зміну.

Таблиця . – Параметри мікроклімату для приміщень, де встановлені комп'ютери

|  |  |  |
| --- | --- | --- |
| Період року | Параметр мікроклімату | Величина |
| Холодний | Температура повітря в приміщенні  Відносна вологість  Швидкість руху повітря | 22-24°С  40-60%  до 0,1м/с |
| Теплий | Температура повітря в приміщенні  Відносна вологість  Швидкість руху повітря | 23-25°С  40-60%  0,1-0,2м/с |
|  |  |  |

Для забезпечення комфортних умов потрібно:

* раціонально організовувати проведення робіт залежно від пори року і доби
* чергувати працю і відпочинок
* встановити вентиляцію
* кондиціонувати повітря
* регулювати режими опалення

### Освітлення

Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах.

Згідно НПАОП 0.00-1.28-10 в приміщеннях обчислювальних центрів необхідно застосувати систему комбінованого освітлення. При виконанні робіт категорії високої зорової точності (найменший розмір об'єкту розрізнення 0,3 - 0,5мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижчою 1,5%, а при зоровій роботі середньої точності (якнайменший розмір об'єкту розрізнення 0,5 - 1,0 мм) КЕО повинен бути не нижчим 1,0%. У якості джерела штучного освітлення використовуються люмінесцентні лампи типа ЛБ, або ДРЛ, які попарно об'єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями. Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні:

* при виконанні зорових робіт високої точності загальна освітленість повинна складати 300лк,
* комбінована - 750лк;
* аналогічні вимоги при виконанні робіт середньої точності - 200 і 300лк відповідно.

Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими, оскільки яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, приводить до їх швидкої стомлюваності.

### Шум і вібрація

Рівень шуму на робочому місці математиків-програмістів і операторів відеоматеріалів не повинен перевищувати 50дБ, а в залах обробки інформації на обчислювальних машинах - 65дБ. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути фанеровані звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

### Електромагнітне і іонізуюче випромінювання

На електронно-променевій трубці є потенціал близько 20 000 вольт (в 100 разів вище напруги в мережі). Цей потенціал створюється між екраном дисплея і обличчям оператора, і розганяє порошинки, що осіли на екран, до величезних швидкостей. І ці порошинки, як кулі, врізаються в шкіру того, хто сидить перед екраном.

Способом боротьби з цим явищем є зниження кількості пилу в приміщенні.

Максимальний рівень рентгенівського випромінювання на робочому місці оператора комп'ютера звичайно не перевищує 10мкбэр/ч, а інтенсивність ультрафіолетового і інфрачервоного випромінювань від екрану монітора лежить в межах 10-100мВт/м2.

### Ергономічні вимоги до робочого місця

Проектування робочих місць, забезпечених відеотерміналами, відноситься до числа важливих проблем ергономічного проектування в області обчислювальної техніки.

Робоче місце і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця програміста повинні бути дотримані наступні основні умови: оптимальне розміщення устаткування, що входить до складу робочого місця і достатній робочий простір, що дозволяє здійснювати всі необхідні рухи і переміщення.

Ергономічними аспектами проектування відеотермінальних робочих місць, зокрема, є: висота робочої поверхні, розміри простору для ніг, вимоги до того, як розташовані документи на робочому місці (наявність і розміри підставки для документів, можливість різного розміщення документів, відстань від очей користувача до екрану, документа, клавіатури і т.д.), характеристики робочого крісла, вимоги до поверхні робочого столу, можливість регулювання елементів робочого місця. Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи.

Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і постійність розміщення предметів, засобів праці і документації. Те, що потрібне для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Для комфортної роботи стіл повинен задовольняти наступним умовам:

* висота столу повинна бути вибрана з урахуванням можливості сидіти вільно, в зручній позі, при необхідності спираючись на підлокітники;
* нижня частина столу повинна бути сконструйована так, щоб програміст міг зручно сидіти, не був вимушений підтискати ноги;
* поверхня столу повинна володіти властивостями, що виключають появу відблисків в полі зору програміста;
* конструкція столу повинна передбачати наявність висувних ящиків (не менше 3 для зберігання документації, лістингів, канцелярських обладнань);
* висота робочої поверхні рекомендується в межах 680-760мм. Висота поверхні, на яку встановлюється клавіатура, повинна бути біля 650мм.

Велике значення надається характеристикам робочого крісла. Так, висота сидіння над рівнем підлоги, що рекомендується, знаходиться в межах 420-550мм. Поверхня сидіння м'яка, передній край закруглений, а кут нахилу спинки - регульований.

Положення екрану визначається:

* відстанню прочитування (0,6-0,7м);
* кутом прочитування, напрямом погляду на 20 нижче горизонталі до центру екрану, причому екран перпендикулярний цьому напряму.

Велике значення також надається правильній робочій позі користувача. При незручній робочій позі можуть з'явитися болі в м'язах, суглобах і сухожиллях. Вимоги до робочої пози користувача відеотерміналу наступні:

* голова не повинна бути нахилена більш ніж на 20,
* плечі повинні бути розслаблені
* лікті - під кутом 80-100,
* передпліччя і долоні рук - в горизонтальному положенні.

Під час користування комп'ютером медики радять встановлювати монітор на відстані 50-60 см від очей. Фахівці також вважають, що верхня частина відеодисплея повинна бути на рівні очей або трохи нижче. Коли людина дивиться прямо перед собою, її очі відкриваються ширше, ніж коли вона дивиться вниз. За рахунок цього площа огляду значно збільшується, викликаючи обезводнення очей. До того ж якщо екран встановлений високо, а очі широко відкриті, порушується функція моргання. Це значить, що очі не закриваються повністю, не омиваються слізною рідиною, не одержують достатнього зволоження, що приводить до їх швидкої стомлюваності.

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення її привабливості, позитивно впливаючи на продуктивність праці.

## Правила пожежної безпеки

У робочому приміщенні причинами виникнення пожежі можуть бути:

* Несправність електропроводки, розеток вимикачів, а також техніки які можуть призвести до короткого замикання або пробою ізоляції;
* Використання пошкоджених (несправних) електроприладів;
* Використання в приміщенні електронагрівальних приладів з відкритими нагрівальними елементами.

### Категорії приміщень за вибухопожежною та пожежною безпекою

Будівля, в якій знаходиться лабораторія з вибухопожежної небезпеки можна віднести до категорії Д (згідно ДНАОП 0.00-1.31-99), оскільки тут присутні горючі (книги, документи, меблі, оргтехніка і т.д.) і важкозгораємі речовини (сейфи, різне обладнання і т.д.), які при взаємодії з вогнем можуть горіти без вибуху.

### Класи пожежі, пожежна сигналізація

У лабораторії з електронно-обчислювальною технікою можливе виникнення пожеж класів А (підкласів А1 і А2), Е, тому що в приміщенні лабораторії знаходяться системні блоки, монітори, периферійних пристроїв, проектори та безліч інших типів техніки, що підключені до електричної мережі і знаходяться під напругою. Також у приміщенні знаходяться книги, предмети меблів, виконані з пластику, текстилю, деревини. Отже, для гасіння пожежі в приміщенні лабораторії я пропоную розмістити вуглекислі ручні вогнегасники ОУ-5 в кількості 2 штук.

Я пропоную встановити в приміщенні пожежну сигналізацію, що використовує адресні димові сповіщувачі СП 212-45 в кількості 5 штук, що реагують на аерозольні продукти горіння в місці їх встановлення і постійно або періодично активно формують сигнал про стан пожежонебезпечності в приміщенні та власну працездатність з зазначенням свого номера (адреси), тому що даний вид пожежної сигналізації дозволяє виявляти пожежу, що почалася на ранній стадії.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. ISO/IEC 9126 Software engineering — Product quality. — International Standard . — 2001 . — 34 p
2. ISO/IEC 25010 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. — International Standard . — 2011 . — 34 p
3. Тестування програмного забезпечення // Вікіпедія — вільна енциклопедія. – http://www.uk.wikipedia.org/wiki/Тестування\_програмного\_забезпечення
4. Регрессионное тестирование // Википедия — свободная энциклопедия . – http://www.ru.wikipedia.org/wiki/Регрессионное\_тестирование
5. Регрессионное тестирование: цели и задачи, условия применения, классификация тестов и методов отбора // Интернет-Университет Информационных Технологий .- http://www.intuit.ru/department/se/testing/11/testing\_11.html
6. Лайза Криспин, Джанет Грегори Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2010. — 464 с.
7. Канер Кем, Фолк Джек, Нгуен Енг Кек Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. — Киев: ДиаСофт, 2001. — 544 с.
8. Калбертсон Роберт, Браун Крис, Кобб Гэри Быстрое тестирование. — М.: «Вильямс», 2002. — 374 с.

##### ДОДАТКИ

##### ДОДАТОК А. КОД ПРОГРАМИ