

Зміст

ВСТУП	11
ПОСТАНОВКА ЗАДАЧІ	12
РОЗДІЛ 1 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА СИСТЕМИ УПРАВЛІННЯ ВЕРСІЯМИ.....	13
1.1 Регресійне тестування	14
1.1.1 Визначення необхідної множини тестів.....	15
1.1.2 Умови керованості і особливості реалізації регресійного тестування.....	17
1.1.3 Класифікація тестів при відборі	20
1.1.4 Класифікація методів вибору тестів	23
1.1.5 Методи вибору тестів	26
1.2 Системи управління версіями.....	28
1.2.1 Традиційні системи управління версіями	30
1.2.2 Розподілені системи управління версіями	32
Висновки до розділу 1	33
РОЗДІЛ 2 ПОБУДОВА МЕТОДУ ВИБОРУ ТЕСТІВ ДЛЯ РЕГРЕСІЙНОГО ТЕСТУВАННЯ.....	34
2.1 Побудова методу вибору тестів на основі системи управління версіями.....	34
2.2 Аналіз переваг та недоліків побудованого методу.....	38
Висновки до розділу 2	39
РОЗДІЛ 3 ОПИС ПРОГРАМИ «REGRESSION VIEWER»	40

3.1 Посібник користувача.....	41
3.2 Карта переходів програмного забезпечення	45
3.3 ER-модель бази даних	46
3.4 Діаграма залежності класів програми.....	47
3.5 Блок-схема алгоритму програми	48
Висновки до розділу 3	49
РОЗДІЛ 4 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ.	
ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	50
4.1 Постановка задачі техніко-економічного аналізу	52
4.2 Обґрунтування функцій програмного продукту.....	53
4.3 Обґрунтування системи параметрів ПП	56
4.4 Аналіз рівня якості варіантів реалізації функцій.....	62
4.5 Економічний аналіз варіантів розробки ПП.....	63
4.6 Вибір кращого варіанта ПП техніко-економічного рівня.....	66
Висновки до розділу 4	67
РОЗДІЛ 5 РОЗДІЛ 3 ОХОРОНИ ПРАЦІ	68
5.1 Аналіз шкідливих та небезпечних факторів	69
5.2 Параметри мікроклімату	69
5.3 Правила пожежної безпеки	75
Висновки до розділу 5	77
ВИСНОВКИ.....	78
ПЕРЕЛІК ПОСИЛАНЬ.....	79
ДОДАТОК А.....	81
ДОДАТОК Б	102

ВСТУП

На сьогоднішній день темпи розвитку програмного забезпечення набирають таких обертів, що виробники змушені випускати нові версії програмного забезпечення все частіше і частіше додаючи нові функції та виправляючи існуючі проблеми. Не зважаючи на швидкий темп якості програмного забезпечення повинна залишатися на високому рівні. Регресійне тестування (спрямоване на виявлення помилок у вже протестованих раніше ділянках вихідного коду) повинно виконуватися при випуску кожної нової версії програмного забезпечення. Але регресійне тестування це досить дорогий етап і метою планування регресійного тестування є побудова оптимальної множини тестів, що зможуть виявити максимальну кількість потенційних помилок.

Метою даної роботи є розроблення нового методу регресійного тестування з використанням систем контролю версій, який розвиватиме ідеї класичних методів та буде оснований на врахуванні змін у програмному забезпеченні.

Перший розділ присвячений існуючим методам регресійного тестування, та огляду основних функцій систем управління ревізіями.

У другому розділі будується новий метод регресійного тестування.

У третьому розділі описується програмне забезпечення «Regression Viewer», яке реалізує побудований метод.

Четвертий розділ – функціонально-вартісний аналіз програмного продукту.

П'ятий розділ присвячений питанням охорони праці.

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 контролю версій

РОЗДІЛ 1 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА СИСТЕМИ УПРАВЛІННЯ ВЕРСІЯМИ

Тестування програмного забезпечення — це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандартах ISO 9126, ISO 25010 [1,2]. 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[3]. До цього процесу входить виконання програми 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).

1.1 Регресійне тестування

Регресійне тестування (regression testing) – збірна назва для всіх видів тестування програмного забезпечення, спрямованих на виявлення помилок у вже протестованих ділянках вихідного коду[4]. 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[5]: 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 задієтий 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[6], 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.

1.1.1 Визначення необхідної множини тестів

Нехай $T = \{t_1, t_2, \dots, t_N\}$ - множина з N тестів, що використовується при первинній розробці програми P , а $T' \in T$ - підмножина регресійних тестів для тестування нової версії програми P' . Інформація про покриття коду, яке забезпечується T' , дозволяє вказати блоки P' , що потребують додаткового тестування, для чого може знадобитися повторний запуск деяких тестів з множини T' , або навіть створення T'' - набору нових тестів для P' - і оновлення T . Правильне виявлення множини T' є дуже важкою задачею[7,8].

Завдання відбору тестів з набору T для заданої програми P і зміненої версії цієї програми P' полягає у виборі підмножини $T' \in T$ для повторного запуску на зміненій програмі P' , де $T' = \{t \in T \mid P'(t) \neq P(t)\}$ [9]. Так як вихідні дані P і P' для тестів з множини T' завідомо однакові, немає необхідності виконувати жоден з тестів $t \notin T'$ на P' . У загальному випадку, за відсутності динамічної інформації про виконання P і P' не існує методу обчислення множини T' для довільних множин P , P' і T . Це впливає 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 T' можливо тільки шляхом виконання на P' 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 T'_p у вигляді апроксимації T' . Цей підхід застосовується у всіх відомих вибіркового методах регресійного тестування, заснованих на аналізі коду[10]. Множина T'_p повинно включати всі тести з T , що активують змінений код, і не включати ніяких інших тестів, тобто тест $t \in T$ входить до T'_p тоді і тільки тоді,

коли t задіє код P в точці, де в P' код був видалений або змінений, або де був доданий новий код.

Якщо певний тест t задіює в P той самий код, що і в P' , вихідні дані P і P' для t відрізнятися не будуть. З цього випливає, що якщо $P'(t) \neq P(t)$, t повинен задіяти певний код, змінений в P' по відношенню до P , тобто повинно виконуватися відношення $t \in T'_p$. З іншого боку, оскільки не кожне виконання зміненого коду відбивається на вихідних значеннях тесту, можуть існувати деякі такі $t \in T'_p$, що $P'(t) = P(t)$. Таким чином, T'_p містить T' цілком і може використовуватися в якості його альтернативи без шкоди для якості програмного продукту[11].

1.1.2 Умови керованості і особливості реалізації регресійного тестування

Протягом життєвого циклу програми період супроводу триває найдовше. Коли змінена програма тестується набором тестів T , ми зберігаємо без змін по відношенню до тестування вихідної програми P всі фактори, які могли б впливати на вихід програми. Тому атрибути конфігурації, в якій програма тестувалася останній раз (наприклад, план тестування, тести t_j і елементи що покриваються $MT(P, C, t_j)$), підлягають управлінню конфігурацією. Практика тестування зміненої версії програми P' в тих же умовах, в яких тестувалася вихідна програма P , 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[5]:

- 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 поклатися 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 ні-

якої додаткової документації і / або будь-якої іншої 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, забезпеченого 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 $t_k \in T$, 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.

1.1.3 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 співпадут 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 програми 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.

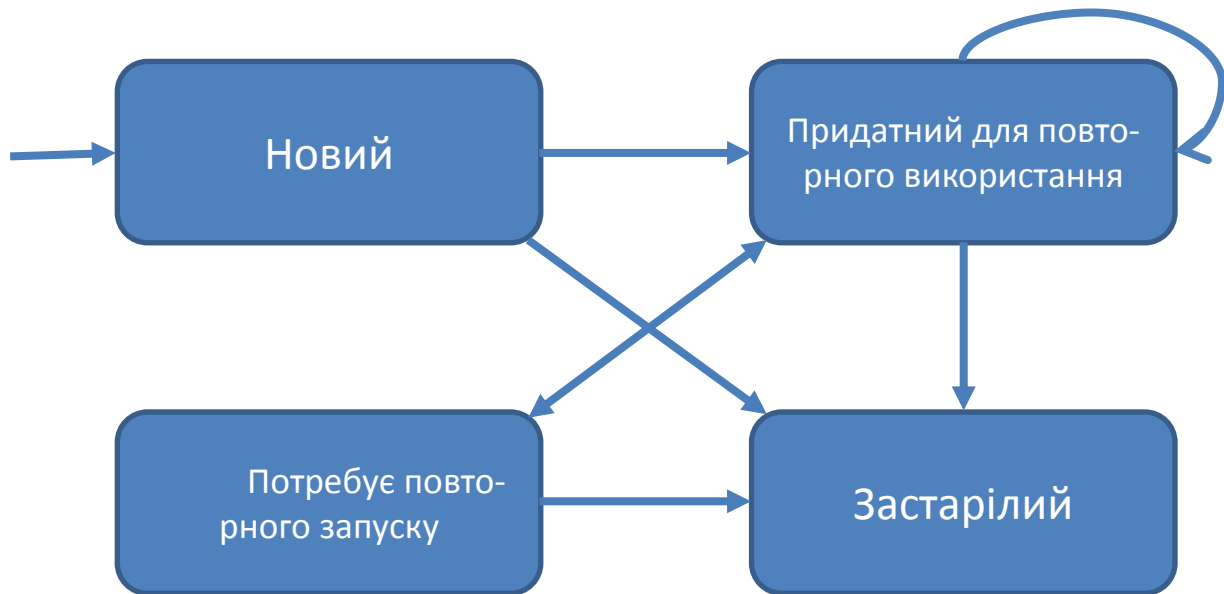


Рисунок 1.1 – Життєвий цикл тесту

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 запуснені 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.

1.1.4 Класифікація методів вибору тестів

Для перевірки коректності різних підходів до регресійного тестування використовується модель оцінки методів регресійного тестування. Основними об'єктами розгляду стали повнота, точність, ефективність та універсальність[5].

Повнота відображає міру відбору тестів з T , на яких результат виконання зміненої програми відмінний від результату виконання вихідної програми, внаслідок чого можуть бути виявлені помилки в P' . Метод, повний на 100%, називається безпечним.

Точність - міра здатності методу уникати вибору тестів з T , на яких результат виконання зміненої програми не буде відрізнятися від результату її первинною версією, тобто тестів, нездатних виявляти помилки в P' . Припустимо, що набір T містить r регресійних тестів. З них для n тестів поведінка та результати виконання старої програми P відрізняються від поведінки і результатів виконання нової програми P' . Набір тестів T' містить m тестів, отриманих з використанням методу відбору регресійних тестів M . З цих m тестів для l тестів поведінка P' і P розрізняється. Точність T' щодо P , P' , T і M , виражена у відсотках, визначається виразом $100 * \frac{l}{m}$, тоді як відповідний відсоток вибраних тестів визначається виразом $100 * \frac{l}{n}$, якщо $n \neq 0$ або дорівнює 100% , якщо $n = 0$.

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 ефекти-

вність 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 P , її зміненої версії P' і набору тестів T для P 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 T , які потенційно можуть виявляти помилки (усі тести, чия поведінка на P' і P може бути різним). Безпечна методика повинна розглядати наслідки додавання, видалення і зміни коду. При додаванні нового коду в P у T 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 застосовною 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 T' . З урахуванням цього обмеження рішенням задачі регресійного тестування буде безпечний метод з хорошим балансом дешевизни і високої точності.

1.1.5 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 змінній програмі, тобто забезпечує вибір всіх тестів, що виявляють зміни. Тест називається таким, що виявляє зміни, якщо його вихідні дані при прогоні на P' відрізняються від вихідних даних при прогоні на P .

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 P' щодо деякого критерію структурного покриття C , що використовувався при створенні первинного набору тестів. Це не означає, що якщо атрибут програми, визначений C , 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.

1.2 Системи управління 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[12]. 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 проєктують, 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 проєктах. Хоча цей метод може працювати, це неефективно, так як багато майже ідентичних копій 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 бізнесовій сферах все частіше одним 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.

1.2.1 Традиційні системи управління версіями

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 радика-

льні 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 можливість 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.

1.2.2 Розподілені системи управління версіями

Розподілені системи контролю версій використовують так званий Peer - To-Peer підхід[13], 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 забезпечує захист від втрати даних.

Висновки до розділу 1

У цьому розділі були розглянуті основні методи регресійного тестування а також проаналізовані основні можливості систем управління ревізіями. Проведений аналіз дає можливість вказати на недоліки існуючих методів регресійного тестування та складає підґрунтя для побудови нового метода регресійного тестування.

РОЗДІЛ 2 ПОБУДОВА МЕТОДУ ВИБОРУ ТЕСТІВ ДЛЯ РЕГРЕСІЙНОГО ТЕСТУВАННЯ

У цьому розділі ставиться за мету побудувати новий метод вибору тестів, що буде вдосконаленням вже існуючих методів розглянутих у розділі 1 але під його аналіз будуть підпадати дані о змінах у програмному забезпеченні отримані із систем контролю версій.

2.1 Побудова методу вибору тестів на основі системи управління версіями

Нагадаємо позначення введені у розділі 1:

P – певна версія програмного забезпечення

$T = \{t_1, t_2, \dots, t_N\}$ – множина з N тестів, що використовується при первинному тестуванні програми P

P' – нова версія програми P

$T' \subset T$ – невідома ідеальна підмножина регресійних тестів для тестування нової версії програми P' , тобто $T' = \{t \in T \mid P'(t) \neq P(t)\}$

$T_p \supset T'$ – шукана множина тестів що є ціллю метода вибору тестів. Множина T_p повинно включати всі тести з T , що активують змінений код, і не включати ніяких інших тестів, тобто тест $t \in T$ входить до T_p тоді і тільки тоді, коли t задіє код P в точці, де в P' код був видалений або змінений, або де був доданий новий код.

Введемо деякі нові позначення:

$P = \{M_i \mid i = 1..m\}$, де M_i – модуль (функціонально закінчений фрагмент) програми

$M_i = \{F_{il} \mid l = 1..p_l\}$, де F_{il} – файл вихідного коду що входить до складу модулю.

$R_{ij_k} = R(F_{i_l}, F_{j_k})$ – зв'язок між файлами вихідного коду, виражається цілим числом від 0 до 10: 0 – зв'язку між файлами не існує (за замовчуванням); 5 – F_{i_l} використовує F_{j_k} ; 10 – F_{i_l} повністю залежить від F_{j_k} . Ступінь залежності визначається особою, що приймає рішення (експерт у програмному забезпеченні)

Тоді маємо формулу R_{ij} – зв'язку між двома модулями:

$$R_{ij} = \sum_{l=1}^{p_l} \sum_{k=1}^{p_k} R_{ij_k}$$

Тепер можемо ввести когнітивну карту залежностей у програмному забезпеченні: $(\{M_i\}, \{R_{ij_k}\})$ – модель подання знань експерта у вигляді знакового орграфа, де $\{M_i\}$ – множина вершин, $\{R_{ij_k}\}$ – множина зважених ребер (Приклад показаний на рисунку 2.1).

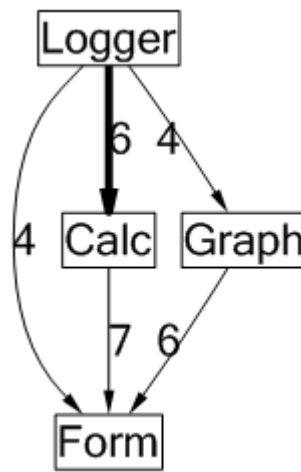


Рисунок 2.1 - Когнітивна карта залежностей у програмному забезпеченні

Зробимо ще декілька означень:

$C^{P'}(F_{i_l})$ – функція-індикатор що вказує чи змінився файл F_{i_l} , у версії P' у порівнянні з P . У даному методі пропонується отримати $C^{P'}$ з системи управління версіями, наприклад найпоширеніші безкоштовні системи управління версіями git та svn мають функцію що називається log і показує які файли мінялися у зазначений проміжок часу або у певному інтервалі ревізій системи контролю версій.

Тоді маємо формулу $R_{ij}^{P'}$ - міри зміни модулю j у зв'язку зі змінами у модулі i у версії P' :

$$R_{ij}^{P'} = \sum_{l=1}^{p_l} \sum_{k=1}^{p_k} (R_{ij_k} \cdot C^{P'}(F_{i_l}))$$

$R_{ij}^{P'} = 0$, тоді і тільки тоді коли або файл F_{j_k} не залежить від F_{i_l} , або F_{i_l} не змінювався в версії P' .

Також можемо побудувати когнітивну карту змін у новій версії P' програмного забезпечення: $(\{M_i\}, \{R_{ij}^{P'}\})$, де $\{M_i\}$ – множина вершин, $\{R_{ij}^{P'}\}$ – множина зважених ребер (Приклад показаний на рисунку 2.2).

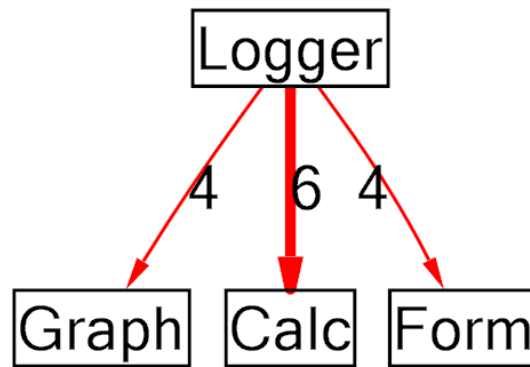


Рисунок 2.2 - Когнітивна карта змін у новій версії P' програмного забезпечення

Як вдосконалення даної карти можемо ввести:

$C_2^{P'}(F_{i_l})$ – функція-індикатор що вказує чи зазнав файл F_{i_l} прямого впливу від змін у версії P' у порівнянні з P (на рисунку 2.2 прямих змін від зазнали модулі Graph, Calc, Form)

Тоді маємо формулу $IR_{ij}^{P'}$ - міри непрямих змін модулю j у зв'язку зі змінами у модулі i у версії P' :

$$IR_{ij}^{P'} = \sum_{l=1}^{p_l} \sum_{k=1}^{p_k} (R_{ij_k} \cdot C_2^{P'}(F_{i_l}))$$

Тепер можемо побудувати когнітивну карту змін у новій версії P' програмного забезпечення, враховуючи непрямі зв'язки: $\left(\{M_i\}, \{R_{ij}^{P'}\} \cup \{IR_{ij}^{P'}\}\right)$. (Приклад показаний на рисунку 2.3).

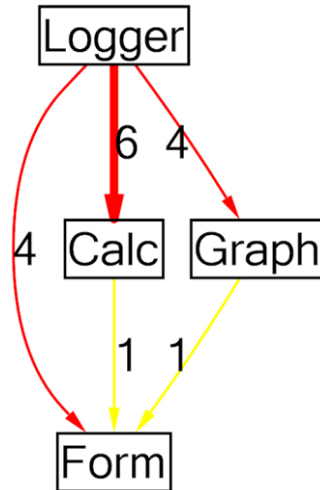


Рисунок 2.3 - Когнітивна карта змін у новій версії P' програмного забезпечення, враховуючи непрямий вплив

Таким чином за розробленим методом до тестового набору регресійного тестування слід включати тести що покривають модулі що змінювались у цій версії, модулі, що зазнали прямих змін у зв'язку зі змінами у тих модулях від яких вони залежать, а також модулі, що зазнали непрямих змін.

2.2 Аналіз переваг та недоліків побудованого методу

Розглянемо порівняння класичних методів регресійного тестування[5]:

Метод	Випадкові	Безпечні	Мінімізації	Покриття
Повнота	0% - 100%	100%	<100%	<100%
Розмір набору тестів	Настроюється	Великий	Невеликий	Залежить від параметрів методу
Час виконання аналізу	Дуже малий	Значний	Середній	Значний
Перспективні властивості	Відсутність засобів підтримки регресійного тестування	Високі вимоги щодо якості	Вартість пропуску помилки невелика	Набір вихідних тестів створюється за критерієм покриття

Розглянемо ті ж самі характеристики для побудованого методу:

Метод	Оснований на системах управління версіями
Повнота	<100%
Розмір набору тестів	Невеликий
Час виконання аналізу	Малий
Перспективні властивості	Має графічну інтерпретацію, не потребує людського втручання під час аналізу(повністю піддається автоматизації), може використовуватися з будь-якою системою контролю версій

Висновки до розділу 2

Зробивши аналіз класичних методів і методу основаного на системах управління версіями, можна зробити висновок, що розроблений метод регресійного тестування за розглянутими характеристиками має значні переваги над класичними методами регресійного тестування. Також розроблений метод регресійного тестування має перспективні властивості: має графічну інтерпретацію, не потребує людського втручання під час аналізу(повністю піддається автоматизації), може використовуватися з будь-якою системою контролю версій.

РОЗДІЛ 3 ОПИС ПРОГРАМИ «REGRESSION VIEWER»

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.

3.1 Посібник користувача

Програма була реалізована за допомогою мови програмування С#. Для зберігання даних використовується база даних SQLite3[14]. Для побудови когнітивних карт зв'язків використовувалася бібліотека MSAGL(Microsoft Automatic Graph Layout)[15].

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

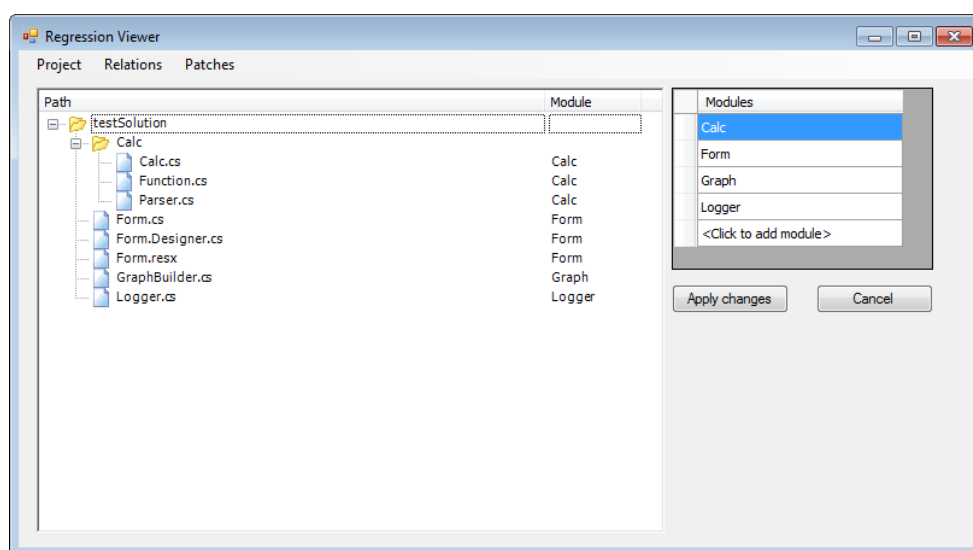
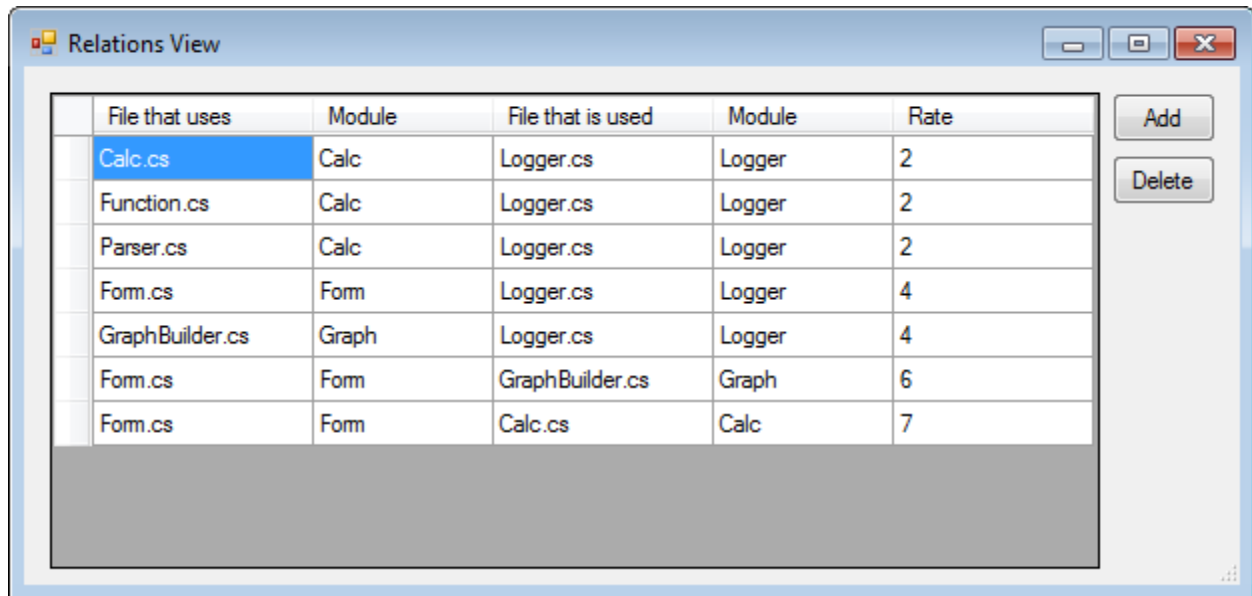


Рисунок 3.1 – Основна форма

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.

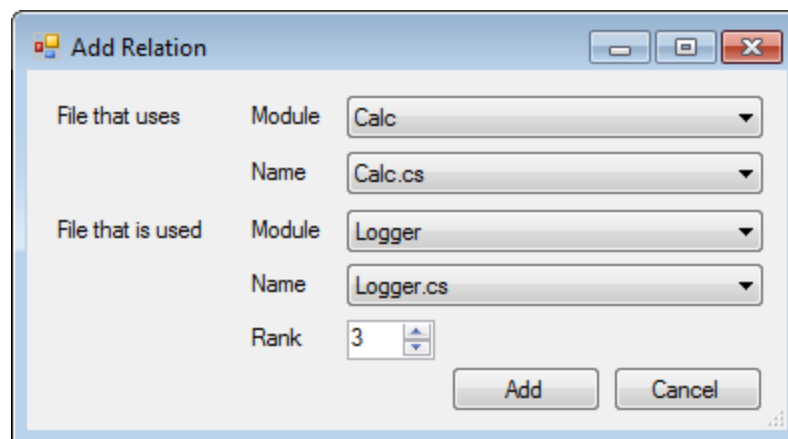
Після того як необхідні модулі були додані, зв'язки установлені, користувач повинен установити зв'язки між файлами проекту. Для цього потрібно викликати меню «Relations->Files Relations». Відкриється форма «Relations View»

(рисунк 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» (рисунк 3.3)) та видалити зв'язок між файлами.



File that uses	Module	File that is used	Module	Rate
Calc.cs	Calc	Logger.cs	Logger	2
Function.cs	Calc	Logger.cs	Logger	2
Parser.cs	Calc	Logger.cs	Logger	2
Form.cs	Form	Logger.cs	Logger	4
GraphBuilder.cs	Graph	Logger.cs	Logger	4
Form.cs	Form	GraphBuilder.cs	Graph	6
Form.cs	Form	Calc.cs	Calc	7

Рисунок 3.2 – Форма зв'язків



File that uses Module: Calc

 Name: Calc.cs

File that is used Module: Logger

 Name: Logger.cs

Rank: 3

Add Cancel

Рисунок 3.3 – Форма додавання зв'язків

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

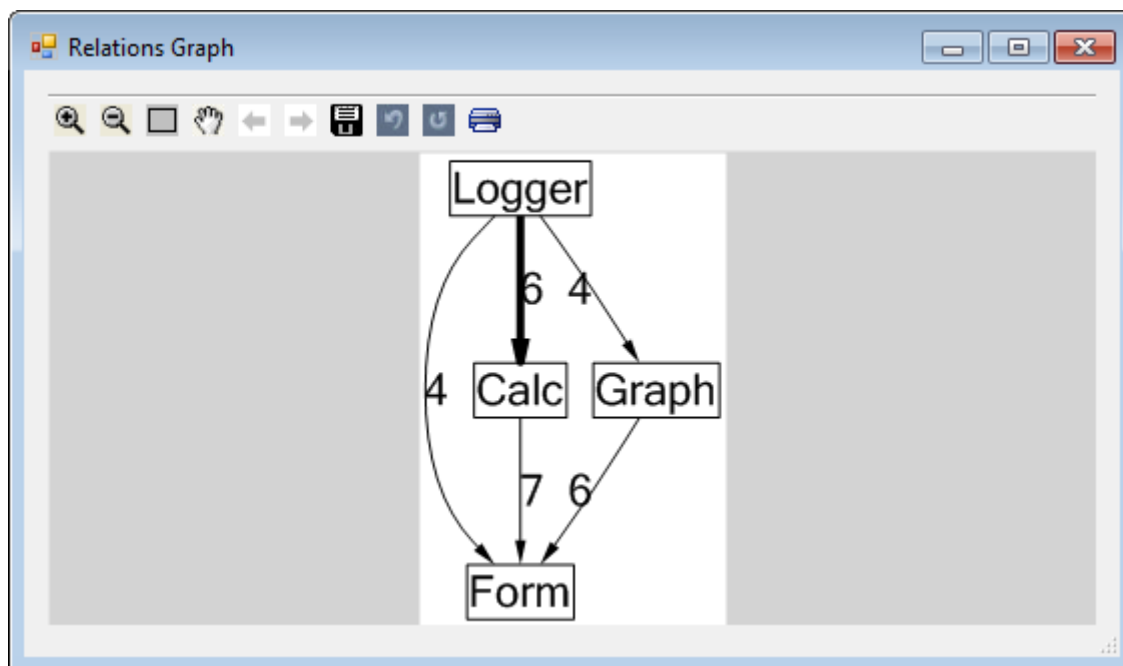


Рисунок 3.4 – Когнітивна карта зв’язків між модулями

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[16] 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».

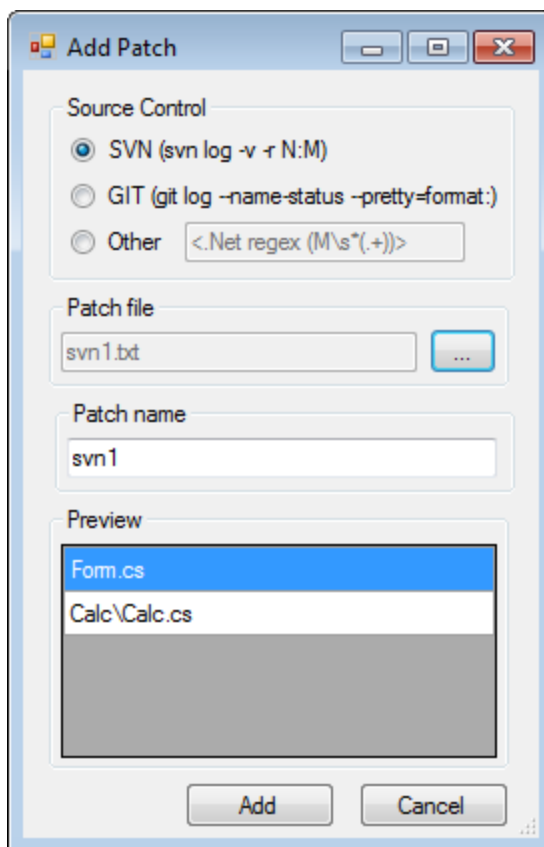


Рисунок 3.5 – Форма додавання патчу

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.

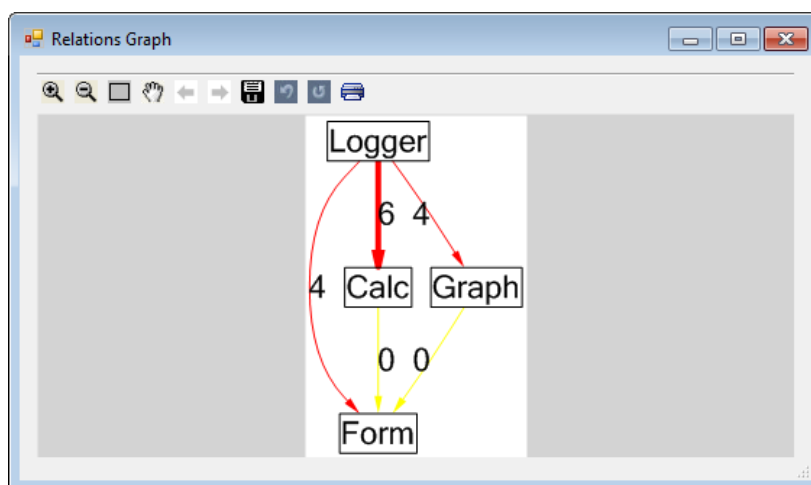


Рисунок 3.6 – Когнітивна карта патчу

3.2 Карта переходів програмного забезпечення

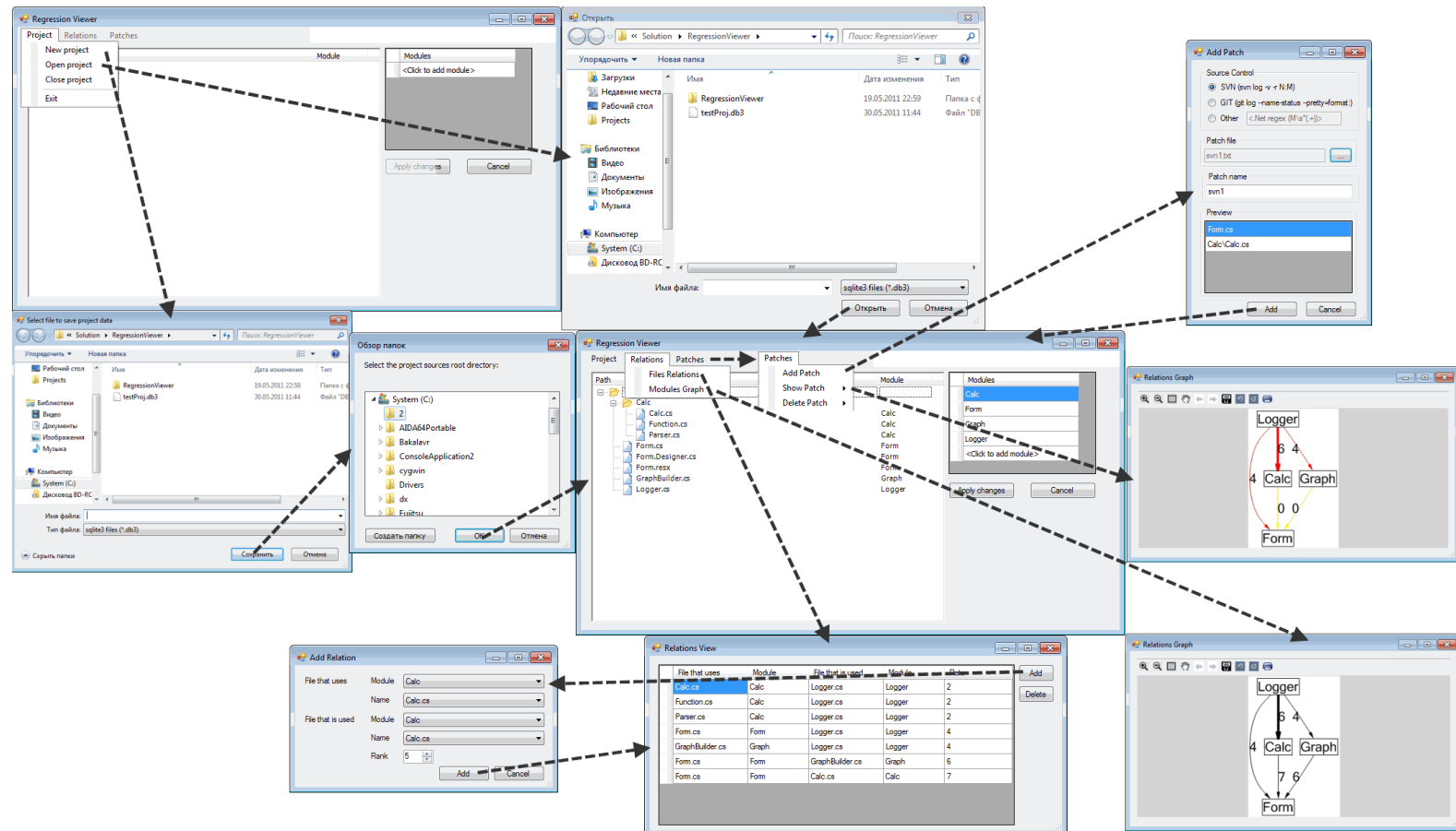


Рисунок 3.7 – Карта переходів програмного забезпечення

3.3 ER-модель бази даних

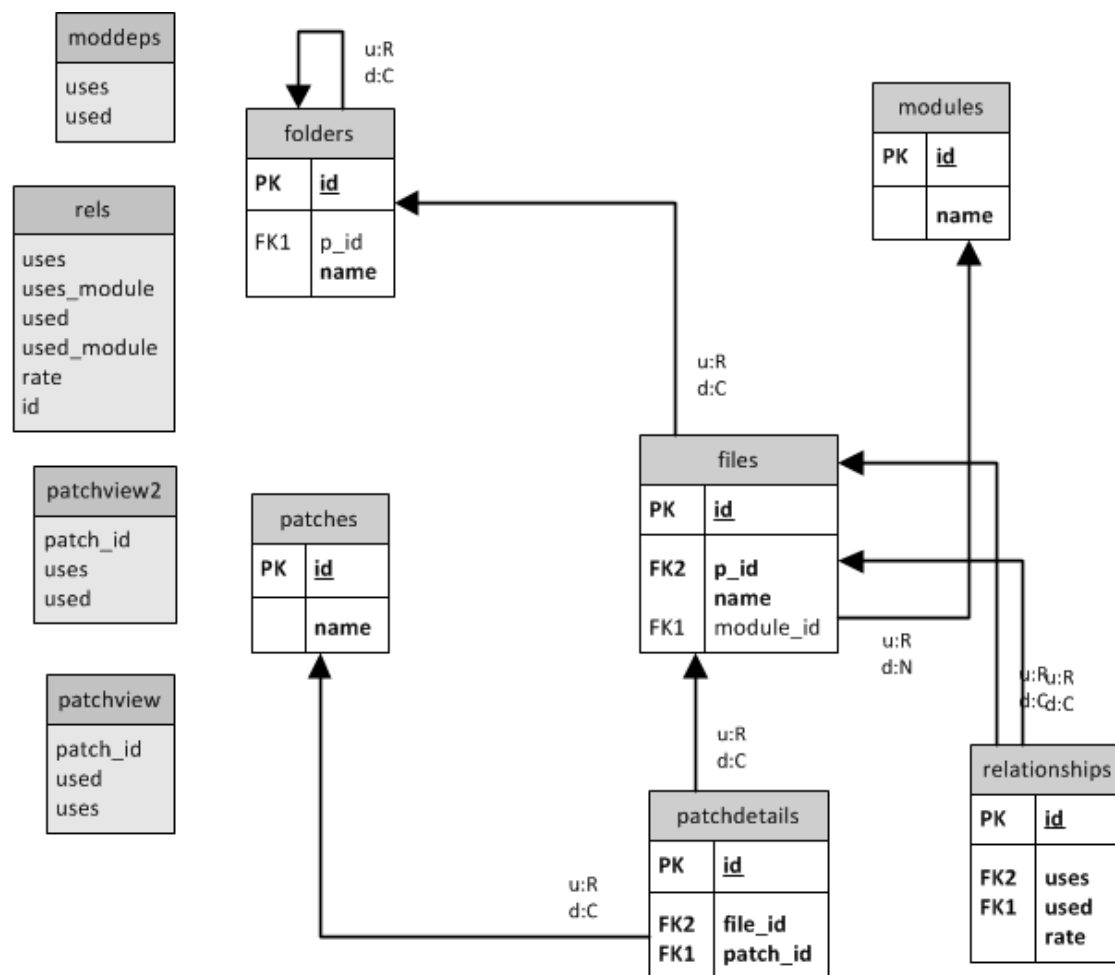


Рисунок 3.8 – ER-модель бази даних, що використовується

3.4 Діаграма залежності класів програми

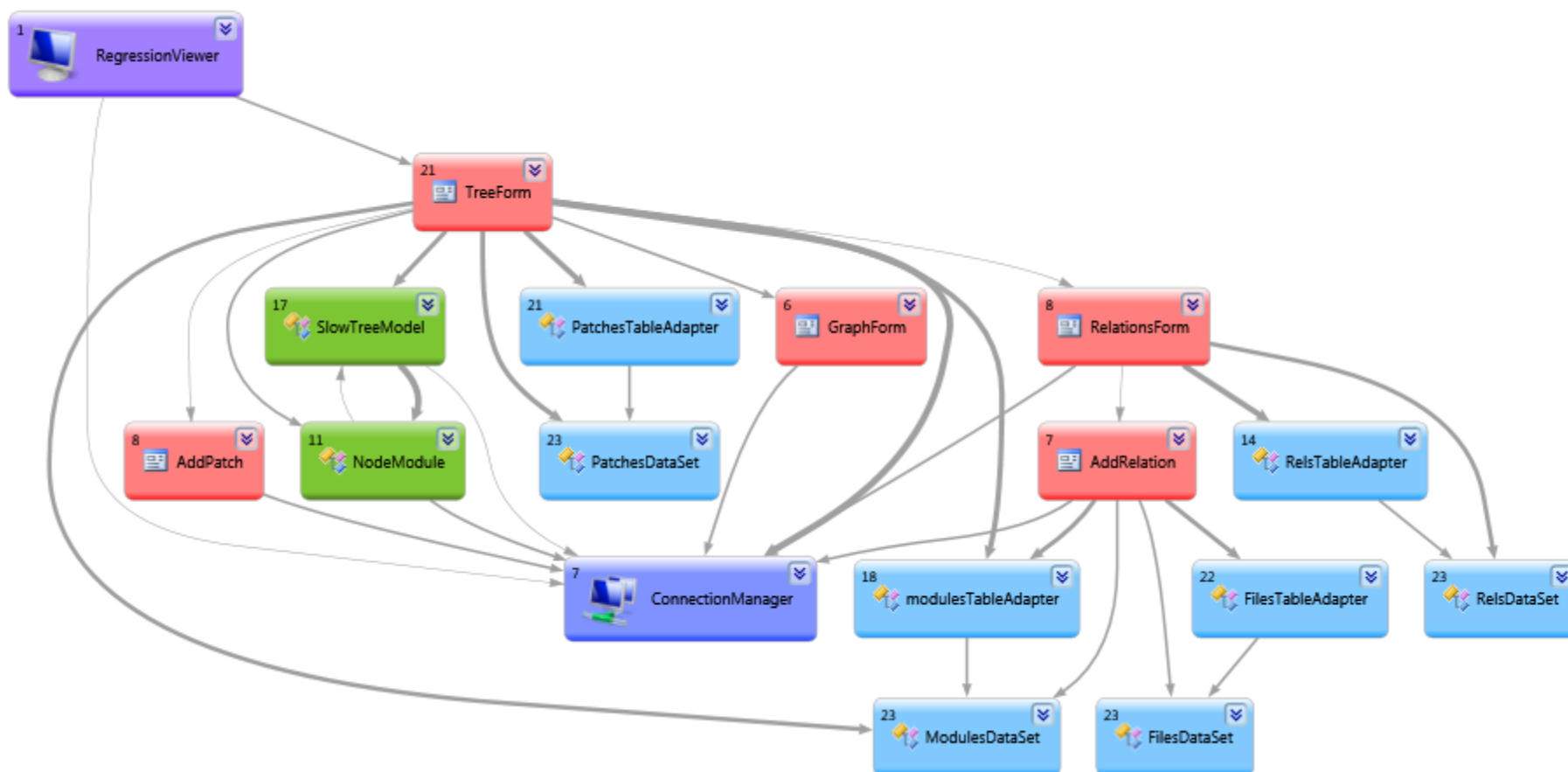


Рисунок 3.9 – Діаграма залежності класів програми

3.5 Блок-схема алгоритму програми

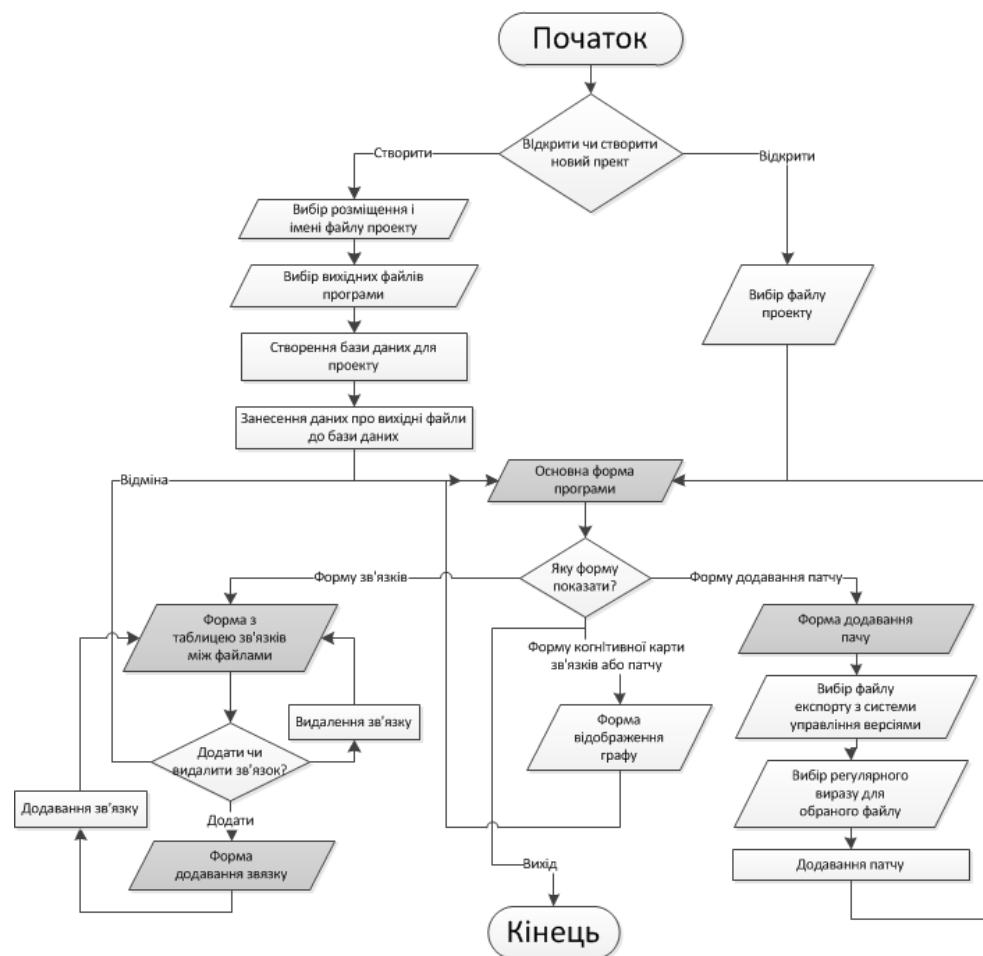


Рисунок 3.10 - Блок-схема алгоритму програми

Висновки до розділу 3

У сучасному світі темпи розробки програмного забезпечення кидають виклик можливостям процесів тестування. Мінімізація кількості тестів, що треба зробити, без втрати якості тестування є основною задачею регресійного тестування. Класичні методи регресійного тестування не дають гарних результатів, оскільки або пропонують запустити всі тести, або випадкову їх кількість, або робити аналіз коду, але не вказують як саме.

У даній роботі, виходячи з аналізу класичних методів, був запропонований новий метод оснований на аналізі коду за допомогою систем контролю версій. Метод дозволяє побудувати когнітивну карту змін програмного забезпечення, використовуючи дані з системи контролю версій.

На основі запропонованого методу було побудоване програмне забезпечення що його реалізовує. Розроблений програмний продукт візуалізує когнітивні карти зв'язків між модулями програмного забезпечення та когнітивні карти змін між різними версіями програмного забезпечення.

Таким чином розроблений теоретичний метод був підтверджений розробленою програмою. Аналізуючи когнітивні карти особа що планує регресійне тестування може бути впевненою, що включає тільки ті модулі, що зазнали змін у новій версії програми, тим самим зменшуючи кількість тестів, що треба виконати.

РОЗДІЛ 4 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ

РОЗДІЛ. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ

АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

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 виробницт-

ва 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.

4.1 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 використання його як модуля.

4.2 Обґрунтування функцій програмного продукту

Формування варіантів функцій

Функції:

F_0 – Введення користувацьких даних про склад програмного продукту

F_1 – Збереження користувацьких даних для подальшого користування

F_2 – Побудова когнітивної карти змін

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_0 :

1)Розробка діалогів у програмі

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

Функція F_1 :

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).

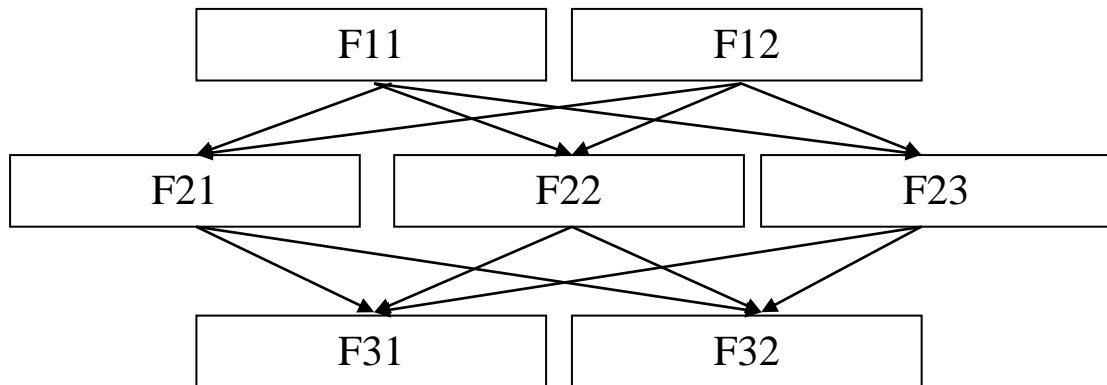


Рисунок 4.1 – Морфологічна карта

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

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
<i>F1</i>	<i>a</i>	Більш інформативний інтерфейс	Великі витрати часу
	<i>б</i>	Займає менше часу при написанні коду	Займає більше часу при написанні коду
<i>F2</i>	<i>a</i>	Легкий для розуміння формат	Складна реалізація створення і читання файлу
	<i>б</i>	Висока швидкість роботи, можлива легка зміна даних, побудова звітів	Великі витрати часу
	<i>в</i>	Дуже проста реалізація	Низька швидкодія
<i>F3</i>	<i>a</i>	Гарний інтерфейс	Платний компонент
	<i>б</i>	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 відкинтий.

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 багато 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.

4.3 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 4.2 – Основні параметри ПП

Назва параметра	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	$X3$	Мб	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,

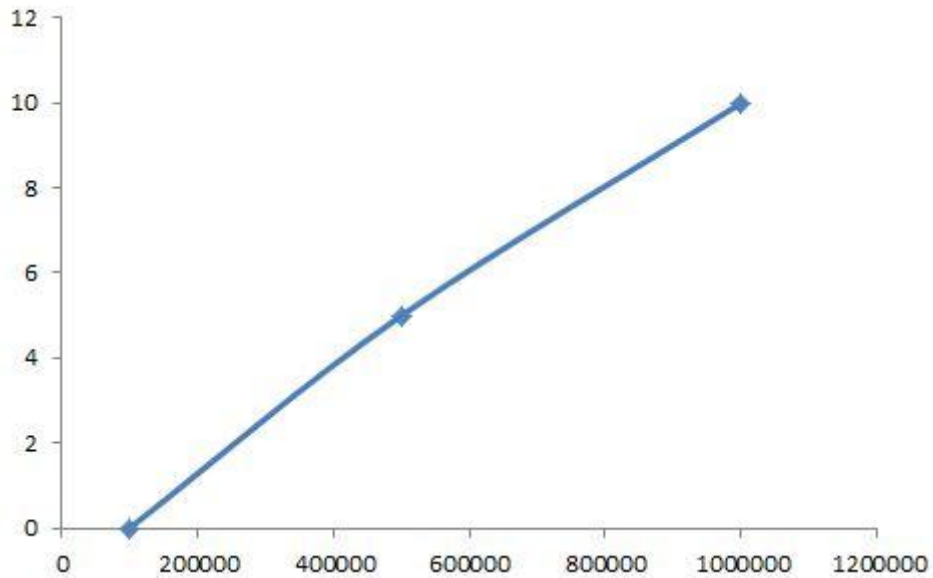


Рисунок 4.2 – Графічні характеристики для X1

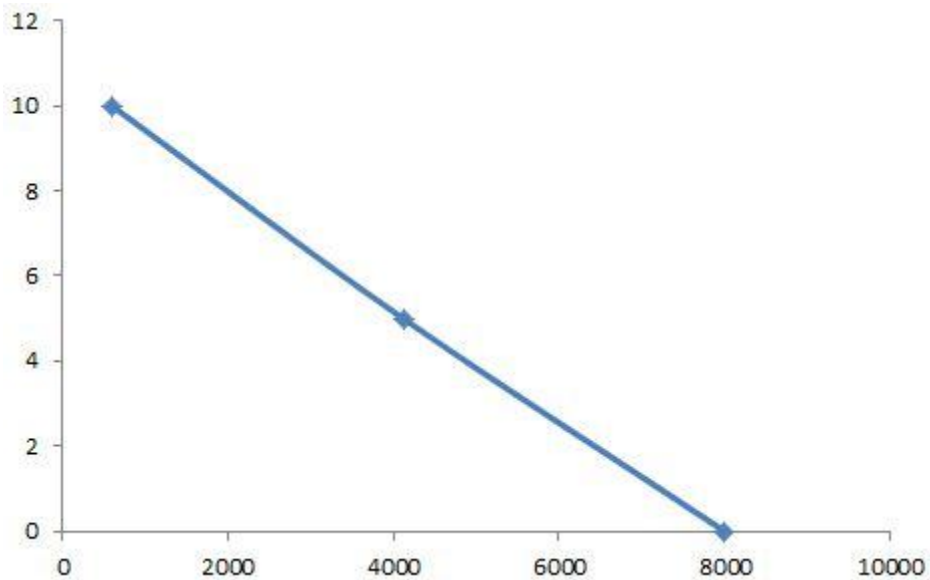


Рисунок 4.3 – Графічні характеристики для X2

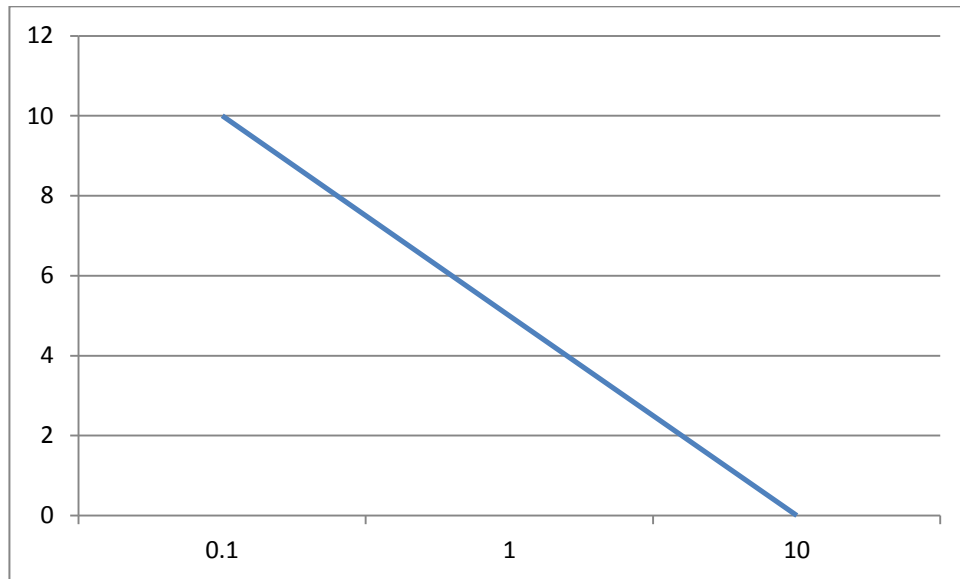


Рисунок 4.4 – Графічні характеристики для X3

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 4.3 – Результати ранжування параметрів

Позн. параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	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	4

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_j^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 42 \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 14 \quad (4.2)$$

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

$$\Delta_i = R_i - T \quad (4.3)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^n \Delta_i^2 = 14 \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)}, \quad (4.5)$$

$$W = \frac{12 * 42}{(7^2 * (3^3 - 3))} = 1.7 > W_k = 0.67$$

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, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1,5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості K_{bi} за наступними формулами:

$$K_{bi} = \frac{b_i}{\sum_{i=1}^n b_i}, \text{ де } b_i = \sum_{i=1}^n a_{ij} \quad (4.7)$$

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:

$$K_{bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \text{ де } b'_i = \sum_{j=1}^n a_{ij} b_j \quad (4.8)$$

Таблиця 4.4 – Розрахунок вагомості параметрів

Параметри	Параметри			Перша ітер.		Друга ітер.	
	X1	X2	X3	b_i	K_{bi}	b_i^1	$\hat{E}_{a^3}^1$
X1	1,0	0,5	1,5	3	0,33	9	0,32
X2	1,5	1,0	1,5	4	0,44	16	0,451
X3	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.

4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_2 (мінімальний час обробки даних алгоритмом) та X_1 (максимальна швидкодія продуктів) відповідають технічним вимогам умов функціонування.

Абсолютне значення параметра X_3 (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:

$$K_{TP} = \sum_i K_{ei} \cdot B_i, i = 1 \dots n, \quad (4.9)$$

де n – кількість параметрів;

K_{ei} – коефіцієнт вагомості i -го параметра;

B_i – 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	A	800000	7	0,333	2,331
	B	500000	5	0,238	1,19
F2	A	700	9	0,428	3,852

За цими даними визначаємо рівень якості кожного з варіантів:

$$K_{TEP1} = 2,331 + 3,852 = 6,183$$

$$K_{TEP2} = 1,19 + 3,852 = 5,042$$

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

4.5 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: $T_p=90$ людино - днів. Поправочний коефіцієнт, який враховує вид довідкової інформації для першого завдання: $K_n = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{ск} = 1$. Оскільки при розробці першого завдання використовуються деякі стандартні підходи, врахуємо це за допомогою коефіцієнта $K_{ст} = 0.8$. Таким чином, загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p=27$ людино-днів, $K_n=0.9$, $K_{ст}=0.8$:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

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

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328.55 \text{ людино-годин};$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.55 \text{ людино-годин};$$

Найбільш високу трудомісткість має варіант II.

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

$$C_q = \frac{5650 + 9250}{3 \cdot 21.1 \cdot 8} = 29.42 \text{ грн.}$$

Тоді зарплата розробників за варіантами становить

$$I. \quad C_{зп} = 29,42 \cdot 1328.55 \cdot 1.2 = 46903,12 \text{ грн.}$$

$$II. \quad C_{зп} = 29,42 \cdot 1345.55 \cdot 1.2 = 47503,29 \text{ грн.}$$

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

$$I. \quad C_{от} = 46903,12 \cdot 0.3677 = 17246 \text{ грн.}$$

$$II. \quad C_{от} = 47503,29 \cdot 0.3677 = 17467 \text{ грн.}$$

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:

$$C_{г} = 12 \cdot 5650 \cdot 0,2 = 10360 \text{ грн.}$$

З урахуванням додаткової заробітної плати

$$C_{зп} = 10360 \cdot (1 + 0.2) = 12630 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$C_{от} = C_{зп} \cdot 0.3677 = 12630 \cdot 0,362 = 4572,06 \text{ грн.}$$

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

$$C_A = 1.15 \cdot 0.25 \cdot 8000 = 2300 \text{ грн.}$$

Витрати на ремонт та профілактику

$$C_p = 1.15 \cdot 8000 \cdot 0.05 = 460 \text{ грн.}$$

Ефективний годинний фонд часу ПК за рік

$$T_{\text{ЕФ}} = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин}$$

Витрати на оплату електроенергії:

$$C_{\text{ЕЛ}} = 1706,4 \cdot 0,156 \cdot 0,2436 \cdot 2 = 129,695 \text{ грн.}$$

Накладні витрати:

$$C_{\text{Н}} = 6360 \cdot 0,67 = 4261,2 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = 7632 + 2762,78 + 2300 + 460 + 129,69 + 4261,2 = 17545,67 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{екс}} / T_{\text{еф}} = 17545,67 / 1706,4 = 10,28 \text{ грн/час.}$$

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:

I. $C_{\text{М}} = 10,28 \cdot 1328,55 = 13657,50 \text{ грн.};$

II. $C_{\text{М}} = 10,28 \cdot 1345,55 = 13832,25 \text{ грн.};$

Накладні витрати складають 67% від заробітної плати:

I. $C_{\text{Н}} = 27118,37 \cdot 0,67 = 18169,31 \text{ грн.};$

II. $C_{\text{Н}} = 27422 \cdot 0,67 = 18372,74 \text{ грн.};$

Отже, вартість розробки програмного продукту за варіантами становить:

I. $C_{\text{ПП}} = 46903,12 + 29549,4 + 13657,50 + 18169,31 = 108279,33 \text{ грн.};$

II. $C_{\text{ПП}} = 47503,29 + 29880 + 13832,25 + 18372,74 = 109588,28 \text{ грн.};$

4.6 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня:

$$K_{тер1} = 5,214 / 144927,72 = 0,481 \cdot 10^{-6};$$

$$K_{тер2} = 3,569 / 146550,14 = 0,243 \cdot 10^{-6};$$

Як бачимо, найбільш ефективним є перший варіант реалізації з коефіцієнтом техніко-економічного рівня $K_{тер3} = 0,481 \cdot 10^{-6}$.

Висновки до розділу 4

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 5 РОЗДІЛ 3 ОХОРОНИ ПРАЦІ

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 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.

5.1 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.

5.2 Параметри мікроклімату

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості 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 $19,5\text{м}^3/\text{людина}$ з урахуванням максимального числа одночасно працюючих в змiну.

Таблиця 5.1 – Параметри мікроклімату для приміщень, де встановлені комп'ютери

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

сця програміста є стіл і крісло. Основним робочим положенням є положення сидючи.

Робоча поза сидючи 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 напрямку.

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.

5.3 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 пробією 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 реа-

гують 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.

Висновки до розділу 5

У даному розділі був розглянутий аналіз шкідливих та небезпечних факторів на робочому місці інженера-програміста, були приведені рекомендовані значення основних параметрів мікроклімату та правила пожежної безпеки на робочому місці. Були зроблені рекомендації щодо розміщення пожежної сигналізації в робочому приміщенні.

ВИСНОВКИ

У сучасному світі темпи розробки програмного забезпечення кидають виклик можливостям процесів тестування. Мінімізація кількості тестів, що треба зробити, без втрати якості тестування є основною задачею регресійного тестування. Класичні методи регресійного тестування не дають гарних результатів, оскільки або пропонують запустити всі тести, або випадкову їх кількість, або робити аналіз коду, але не вказують як саме.

В даній роботі розв'язувалась задача побудови нового методу регресійного тестування. В роботі отримані наступні результати:

1. Виходячи з аналізу класичних методів, був запропонований новий метод оснований на аналізі коду за допомогою систем контролю версій. Метод дозволяє побудувати когнітивну карту змін програмного забезпечення, використовуючи дані з системи контролю версій.
2. На основі запропонованого методу було побудоване програмне забезпечення що його реалізовує. Розроблений програмний продукт візуалізує когнітивні карти зв'язків між модулями програмного забезпечення та когнітивні карти змін між різними версіями програмного забезпечення.

Таким чином розроблений теоретичний метод був підтверджений розробленою програмою. Аналізуючи когнітивні карти особа що планує регресійне тестування може бути впевненою, що включає тільки ті модулі, що зазнали змін у новій версії програми, тим самим зменшуючи кількість тестів, що треба виконати.

ПЕРЕЛІК ПОСИЛАНЬ

1. Software engineering — Product quality : ISO/IEC 9126-1:2001 [Text]. — International Standard. — 2001. — 34 p.
2. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) : ISO/IEC 25010:2011 [Text]. — International Standard. — 2011. — 34 p.
3. Тестування програмного забезпечення[Електронний ресурс]. Режим доступу:
http://www.uk.wikipedia.org/wiki/Тестування_програмного_забезпечення
4. Регрессионное тестирование [Електронний ресурс]. Режим доступу:
http://www.ru.wikipedia.org/wiki/Регрессионное_тестирование
5. Интернет-Университет Информационных Технологий: Регрессионное тестирование: цели и задачи, условия применения, классификация тестов и методов отбора [Електронний ресурс]. Режим доступу:
http://www.intuit.ru/departments/se/testing/11/testing_11.html
6. Mary Jean Harrold Testing Evolving Software [Text]/ Mary Jean Harrold. — Journal of Systems and Software, 1999. — vol. 47, number 2-3. — pp. 173-181.
7. D. Rosenblum An empirical comparison of regression test selection techniques [Text] / D. Rosenblum, G. Rothermel. — In Proceedings of the International Workshop for Empirical Studies of Software Maintenance. — October 1997.
8. G. Rothermel Empirical studies of a safe regression test selection technique. [Text] / G. Rothermel, M. J. Harrold. — IEEE Trans.on Softw. Eng., 24(6). — June 1998.
9. G. Rothermel Selecting tests and identifying test coverage requirements for modified software [Text] / G. Rothermel, M.J. Harrold. — In Proceedings of

- the 1994 International Symposium on Software Testing and Analysis. — pages 169-184.
10. J. Stafford Chaining: A dependence analysis technique for software architecture [Text] / J. Stafford, D. J. Richardson, A. L. Wolf. — In Proceedings of the 1994 International Symposium on Software Testing and Analysis, pages 312-325.
 11. An empirical study of regression test selection techniques [Text] / [T. Graves, M. J. Harrold, J.-M. Kim, A. Porter, G. Rothermel]. — In Proceedings of the International Conference on Software Engineering, Apr. 1998. — pages 188-197.
 12. Система управления версіями [Електронний ресурс]. Режим доступу: http://ru.wikipedia.org/wiki/Система_управления_версиями
 13. Peer-to-peer [Електронний ресурс]. Режим доступу: <http://en.wikipedia.org/wiki/Peer-to-peer>
 14. SQLite Home Page[Електронний ресурс]. Режим доступу: <http://www.sqlite.org/>
 15. Microsoft Automatic Graph Layout - Microsoft Research [Електронний ресурс]. Режим доступу: <http://research.microsoft.com/en-us/projects/msagl/>
 16. .NET Framework Regular Expressions [Електронний ресурс]. Режим доступу: <http://msdn.microsoft.com/en-us/library/hs600312.aspx>

ДОДАТОК А

Код програми:

RegressionViewer.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using RegressionViewer.Forms;

namespace RegressionViewer
{
    static class RegressionViewer
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            TreeForm f1 = new TreeForm();
            Application.Run(f1);
            ConnectionManager.CloseConnection();
        }
    }
}
```

ConnectionManager.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Common;
using System.Data.SQLite;

namespace RegressionViewer
{
    static class ConnectionManager
    {
        private static string _filename;
        public static string ConnectionString
        {
            get
            {
                return "data source=" + _filename;
            }
        }
        private static SQLiteConnection _connection;
        public static SQLiteConnection connection
        {
            get
            {
                if (_connection==null)
                {
                    _connection = new SQLiteConnection("Data Source = " + filename);
                }
            }
        }
    }
}
```

```

        _connection.Open();
    }
    return _connection;
}
}
public static string filename
{
    get
    {
        return _filename;
    }
    set
    {
        _filename = value;
        if (_connection!=null) _connection.Close();
        _connection = null;
    }
}
public static void CloseConnection()
{
    if (_connection!=null)
        _connection.Close();
    _connection = null;
}
public static void ExecuteNonQuery(string sql)
{
    using (SQLiteCommand command = new SQLiteCommand(sql,connection))
        command.ExecuteNonQuery();
}
public static void CreateDB()
{
    ExecuteNonQuery(@"CREATE TABLE [folders] (
        [id] integer PRIMARY KEY AUTOINCREMENT NOT NULL,
        [p_id] integer,
        [name] char(255) NOT NULL
    );");
    // CONSTRAINT FK_folders_folders FOREIGN KEY (p_id) REFERENCES folders (id)
on delete cascade

    ExecuteNonQuery(@"CREATE TABLE [modules] (
        [id] integer PRIMARY KEY AUTOINCREMENT NOT NULL,
        [name] char(255) NOT NULL
    );");
    ExecuteNonQuery(@"CREATE TABLE [files] (
        [id] integer PRIMARY KEY AUTOINCREMENT NOT NULL,
        [p_id] integer NOT NULL,
        [name] char(255) NOT NULL,
        [module_id] integer,
        CONSTRAINT FK_files_folders FOREIGN KEY (p_id) REFERENCES folders
(id) on delete cascade
        CONSTRAINT FK_files_modules FOREIGN KEY (module_id) REFERENCES mod-
ules (id) on delete set NULL
    );");
    ExecuteNonQuery(@"CREATE TABLE [relationships] (
        [id] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        [uses] INTEGER NOT NULL,
        [used] INTEGER NOT NULL,
        [rate] INTEGER NOT NULL,
        CONSTRAINT FK_relationships_files FOREIGN KEY (uses,used) REFERENCES
files (id,id) ON DELETE CASCADE
    );");
    ExecuteNonQuery(@"CREATE VIEW [rels] AS
    SELECT

```

```

        f1.name AS uses,
        m1.name AS uses_module,
        f2.name AS used,
        m2.name AS used_module,
        relationships.rate AS rate,
        relationships.id AS id
    FROM
    modules AS m1
    INNER JOIN files AS f1 ON m1.id = f1.module_id
    INNER JOIN relationships ON f1.id = relationships.uses
    INNER JOIN files AS f2 ON relationships.used = f2.id
    INNER JOIN modules AS m2 ON f2.module_id = m2.id");
ExecuteNonQuery(@"CREATE VIEW moddeps AS
    SELECT
        m1.name AS uses,
        m2.name AS used,
        Count(relationships.rate) AS count,
        Sum(relationships.rate) AS rate
    FROM
    files AS f1
    INNER JOIN modules AS m1 ON m1.id = f1.module_id
    INNER JOIN relationships ON f1.id = relationships.uses
    INNER JOIN files AS f2 ON relationships.used = f2.id
    INNER JOIN modules AS m2 ON f2.module_id = m2.id
    WHERE
        m1.id <> m2.id
    GROUP BY
        m1.name,
        m2.name;");
ExecuteNonQuery(@"CREATE TABLE patches (
    id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    name TEXT NOT NULL);");
ExecuteNonQuery(@"CREATE TABLE patchdetails (
    id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    file_id INTEGER NOT NULL,
    patch_id INTEGER NOT NULL,
    CONSTRAINT FK_patchdetails_files FOREIGN KEY (file_id) REFERENCES
files (id) ON DELETE CASCADE,
    CONSTRAINT FK_patchdetails_patches FOREIGN KEY (patch_id) REFERENCES
patches (id) ON DELETE CASCADE);");
ExecuteNonQuery(@"CREATE VIEW patchview AS
    SELECT
        patchdetails.patch_id,
        m2.name AS used,
        m1.name AS uses,
        Count(relationships.rate) AS count,
        Sum(relationships.rate) AS rate
    FROM
    modules AS m1
    INNER JOIN files AS f1 ON m1.id = f1.module_id
    INNER JOIN relationships ON f1.id = relationships.uses
    INNER JOIN files AS f2 ON relationships.used = f2.id
    INNER JOIN modules AS m2 ON f2.module_id = m2.id
    INNER JOIN patchdetails ON patchdetails.file_id = relationships.used
    GROUP BY
        patchdetails.patch_id,
        m2.name,
        m1.name;");
ExecuteNonQuery(@"CREATE VIEW patchview2 AS
    SELECT
        patchview.patch_id,
        m1.name AS uses,
        m2.name AS used,

```

```

        Count(relationships.rate) AS count,
        Sum(relationships.rate) AS rate
    FROM
        modules AS m1
    INNER JOIN files AS f1 ON m1.id = f1.module_id
    INNER JOIN relationships ON f1.id = relationships.uses
    INNER JOIN files AS f2 ON relationships.used = f2.id
    INNER JOIN modules AS m2 ON f2.module_id = m2.id
    INNER JOIN patchview ON m2.name = patchview.uses
    WHERE
        m1.id <> m2.id AND
        patchview.used <> m1.name
    GROUP BY
        m1.name,
        m2.name,
        patchview.patch_id;");
    }
}
}

```

AddPatch.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Text.RegularExpressions;
using System.Data.SQLite;

namespace RegressionViewer.Forms
{
    public partial class AddPatch : Form
    {
        public AddPatch()
        {
            InitializeComponent();
        }

        private void OtherRadioButton_CheckedChanged(object sender, EventArgs e)
        {
            if (OtherRadioButton.Checked == true)
                regexTextBox.Enabled = true;
            else
                regexTextBox.Enabled = false;
        }

        private void selectFileButton_Click(object sender, EventArgs e)
        {
            string pattern;
            if (GITRadioButton.Checked == true)
                pattern = @"M\s*(.+)";
            else if (SVNRadioButton.Checked == true)
                pattern = @"\s\s\sM\s(.+)";
            else pattern = regexTextBox.Text;
            OpenFileDialog ofp = new OpenFileDialog();

```

```

ofp.Multiselect = false;
ofp.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
ofp.FilterIndex = 2;
ofp.RestoreDirectory = true;

if (ofp.ShowDialog() == DialogResult.OK)
{
    FileInfo input = new FileInfo(ofp.FileName);
    if (input.Exists)
    {
        StreamReader SR;
        try { SR = input.OpenText(); }
        catch { MessageBox.Show("Cannot open File for reading"); return; };
        string s;
        changesDataGridView.Rows.Clear();
        while (!SR.EndOfStream)
        {
            s = SR.ReadLine();
            foreach (Match match in Regex.Matches(s, pattern, Regex-
Options.IgnoreCase))
            {
                changesDataGridView.Rows.Add();
                bool found = FindFile(match.Groups[1].Value)!=0;
                changesDataGridView.Rows[changesDataGridView.Rows.Count -
1].Cells[0].Value = match.Groups[1].Value;
                if (!found)
                    changesDataGridView.Rows[changesDataGridView.Rows.Count -
1].Cells[0].Style.ForeColor = Color.Purple;
            }
        }
        patchNameTextBox.Text =
Path.GetFileNameWithoutExtension(ofp.FileName);
        patchFileTextBox.Text = input.Name;
        OKButton.Enabled = true;
    }
    else MessageBox.Show("File doesn't exist");
}

}

private void OKButton_Click(object sender, EventArgs e)
{
    string patchname = patchNameTextBox.Text;
    int patch_id;
    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"insert into patches (name) values (@name); SELECT
last_insert_rowid() AS [ID]";
        command.Parameters.Add(new SQLiteParameter("@name", patchname));
        patch_id = int.Parse(command.ExecuteScalar().ToString());
    }

    foreach (DataGridViewRow dgvr in changesDataGridView.Rows)
    {
        string file = dgvr.Cells[0].Value.ToString();
        int file_id = FindFile(file);
        if (file_id!=0)
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"insert into patchdetails
(patch_id,file_id) values (@patch_id,@file_id)";

```

```

        patch_id));

        command.Parameters.Add(new SQLiteParameter("@patch_id",
        command.Parameters.Add(new SQLiteParameter("@file_id", file_id));
        command.ExecuteNonQuery();
    }
    }
    this.Close();
}
private int FindFile(string file)
{
    if (file.StartsWith("/") || file.StartsWith("\\")) file = file.Substring(1);
    string[] parts = file.Split(new char[2] { '/', '\\' });
    int id=0;
    int index = 0;
    if (parts.Length > 1)
    {
        using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
        {
            command.CommandText = @"select id from folders where name=@name and
p_id='null'";
            command.Parameters.Add(new SQLiteParameter("@name", parts[index]));
            id = Convert.ToInt32(command.ExecuteScalar());
            if (id == 0)
            {
                if (parts.Length > 2)
                {
                    index++;
                    command.Parameters.Add(new SQLiteParameter("@name",
parts[index]));
                    id = Convert.ToInt32(command.ExecuteScalar());
                }
            }
        }
    }
    if (id == 0)
    {
        using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
        {
            command.CommandText = @"select id from folders where name=@name and
p_id=(select id from folders where p_id='null')";
            command.Parameters.Add(new SQLiteParameter("@name", parts[index]));
            id = Convert.ToInt32(command.ExecuteScalar());
        }
    }
    if (id == 0)
    {
        if (index == parts.Length - 1)
        {
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"select id from files where name=@name and
p_id=(select id from folders where p_id='null')";
                command.Parameters.Add(new SQLiteParameter("@name",
parts[index]));
                id = Convert.ToInt32(command.ExecuteScalar());
                if (id != 0) return id;
            }
        }
        else return 0;
    }
}

```

```

        //Parent folder found
        for (; index < parts.Length - 2; )
        {
            index++;
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"select id from folders where name=@name and
p_id=@id";
                command.Parameters.Add(new SQLiteParameter("@name", parts[index]));
                command.Parameters.Add(new SQLiteParameter("@id", id));
                id = Convert.ToInt32(command.ExecuteScalar());
                if (id == 0) return 0;
            }
        }
        index++;
        using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
        {
            command.CommandText = @"select id from files where name=@name and
p_id=@id";
            command.Parameters.Add(new SQLiteParameter("@name", parts[index]));
            command.Parameters.Add(new SQLiteParameter("@id", id));
            id = Convert.ToInt32(command.ExecuteScalar());
        }
        return id;
    }

    private void cancelButton_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
}

```

AddRelationForm.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Common;
using System.Data.SQLite;

namespace RegressionViewer.Forms
{
    public partial class AddRelation : Form
    {
        public AddRelation()
        {
            InitializeComponent();
            modulesTableAdapter.Connection.ConnectionString = ConnectionManag-
er.ConnectionString;
            filesTableAdapter.Connection.ConnectionString = ConnectionManag-
er.ConnectionString;
        }
    }
}

```

```

    }

    private void AddRelation_Load(object sender, EventArgs e)
    {
        // TODO: This line of code loads data into the 'filesDa-
        taSet.filesGridViewColumn' table. You can move, or remove it, as needed.
        this.modulesTableAdapter.Fill(this.usesModulesDataSet.modules);
        fillFilesUsingParametrizedQuery(usesModuleCombo, null);
        this.modulesTableAdapter.Fill(this.usedModulesDataSet.modules);
        fillFilesUsingParametrizedQuery(usedModuleCombo, null);
    }

    private void fillFilesUsingParametrizedQuery(object sender, EventArgs e)
    {
        try
        {
            if (sender.Equals(usesModuleCombo))
                this.filesTableAdapter.FillBy1(this.usesFilesDataSet.files, new Sys-
                tem.Nullable<long>(((long)(System.Convert.ChangeType(usesModuleCombo.SelectedValue,
                typeof(long))))));
            else if (sender.Equals(usedModuleCombo))
                this.filesTableAdapter.FillBy1(this.usedFilesDataSet.files, new Sys-
                tem.Nullable<long>(((long)(System.Convert.ChangeType(usedModuleCombo.SelectedValue,
                typeof(long))))));
        }
        catch (System.Exception ex)
        {
            System.Windows.Forms.MessageBox.Show(ex.Message);
        }
    }

    private void AddButton_Click(object sender, EventArgs e)
    {
        using (SQLiteCommand command = new SQLiteCom-
        mand(ConnectionManager.connection))
        {
            command.CommandText = @"Insert into [relationships] (uses,used,rate) val-
            ues (@uses,@used,@rate)";
            command.Parameters.Add(new SQLiteParameter("@uses", usesNameCom-
            bo.SelectedValue));
            command.Parameters.Add(new SQLiteParameter("@used", usedNameCom-
            bo.SelectedValue));
            command.Parameters.Add(new SQLiteParameter("@rate", rateUpDown.Value));
            command.ExecuteNonQuery();
        }

        this.Close();
    }

    private void cancelButton_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
}

```

GraphForm.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;

```



```

using System.Drawing;
using System.Linq;
using System.Text;
using System.Data.Common;
using System.Data.SQLite;
using System.Windows.Forms;
using Microsoft.Msagl.Drawing;

namespace RegressionViewer.Forms
{
    public partial class GraphForm : Form
    {
        public GraphForm()
        {
            InitializeComponent();
            ModulesGraph();
        }

        public GraphForm(int patchid)
        {
            InitializeComponent();
            PatchGraph(patchid);
        }

        private void PatchGraph(int id)
        {
            Graph g = new Graph();

            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"select name from modules";

                SQLiteDataReader DataReader = command.ExecuteReader();
                while (DataReader.Read())
                    g.AddNode(DataReader["name"].ToString());
            }

            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"select * from patchview where
patch_id=@patch_id";
                command.Parameters.Add(new SQLiteParameter("@patch_id", id));
                SQLiteDataReader DataReader = command.ExecuteReader();
                while (DataReader.Read())
                {
                    Microsoft.Msagl.Drawing.Edge e =
g.AddEdge(DataReader["used"].ToString(), "", DataReader["uses"].ToString());
                    e.Attr.LineWidth = int.Parse(DataReader["count"].ToString());
                    e.Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
                    e.LabelText = DataReader["rate"].ToString();
                }
            }

            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"select * from patchview2 where
patch_id=@patch_id";
                command.Parameters.Add(new SQLiteParameter("@patch_id", id));
                SQLiteDataReader DataReader = command.ExecuteReader();
                while (DataReader.Read())
                {

```

```

        Microsoft.Msagl.Drawing.Edge e =
g.AddEdge(DataReader["used"].ToString(), "", DataReader["uses"].ToString());
        e.Attr.LineWidth = int.Parse(DataReader["count"].ToString());
        e.Attr.Color = Microsoft.Msagl.Drawing.Color.Yellow;
        e.LabelText =
(int.Parse(DataReader["rate"].ToString())/10).ToString();
    }
    }
    gViewer.Graph = g;
}

private void ModulesGraph()
{
    Graph g = new Graph();

    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"select name from modules";

        SQLiteDataReader DataReader = command.ExecuteReader();
        while (DataReader.Read())
            g.AddNode(DataReader["name"].ToString());
    }

    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"select * from moddeps";
        SQLiteDataReader DataReader = command.ExecuteReader();
        while (DataReader.Read())
        {
            Microsoft.Msagl.Drawing.Edge e =
g.AddEdge(DataReader["used"].ToString(), "", DataReader["uses"].ToString());
            e.Attr.LineWidth = int.Parse(DataReader["count"].ToString());
            //e.Attr.Color = Microsoft.Msagl.Drawing.Color.Yellow;
            e.LabelText = DataReader["rate"].ToString();
        }
    }
    gViewer.Graph = g;
}
}
}
}

```

RelationsForm.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Common;
using System.Data.SQLite;

namespace RegressionViewer.Forms
{
    public partial class RelationsForm : Form
    {
        public RelationsForm()
    }
}

```

```

    {
        InitializeComponent();
        relsTableAdapter.Connection.ConnectionString = ConnectionManag-
er.ConnectionString;
    }

    private void RelationsForm_Load(object sender, EventArgs e)
    {
        // TODO: This line of code loads data into the 'relsDataSet.rels' table. You
        can move, or remove it, as needed.
        this.relsTableAdapter.Fill(this.relsDataSet.rels);
    }

    private void addToolStripMenuItem_Click(object sender, EventArgs e)
    {
        (new AddRelation()).ShowDialog();
        this.relsTableAdapter.Fill(this.relsDataSet.rels);
    }

    private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
    {
        SQLiteDataAdapter mDA = relsTableAdapter.Adapter;
        DataSet mDS = relsDataSet;

        int ri = relsGrid.SelectedCells[0].RowIndex;
        string id = mDS.Tables["rels"].Rows[ri]["id"].ToString();

        using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
        {
            command.CommandText = @"delete from [relationships] where id=@id";
            command.Parameters.Add(new SQLiteParameter("@id", id));
            command.ExecuteNonQuery();
        }
        this.relsTableAdapter.Fill(this.relsDataSet.rels);
    }

    private void addButton_Click(object sender, EventArgs e)
    {
        addToolStripMenuItem_Click(null, null);
    }

    private void deleteButton_Click(object sender, EventArgs e)
    {
        deleteToolStripMenuItem_Click(null, null);
    }
}

```

TreeForm.cs:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Common;
using System.Data.SQLite;

```

```

using System.IO;
using Aga.Controls.Tree;
using Aga.Controls.Tree.NodeControls;
using Aga.Controls;

namespace RegressionViewer.Forms
{
    public partial class TreeForm : Form
    {
        public TreeForm()
        {
            InitializeComponent();
            this.patchesStripComboBox.ComboBox.BindingContext = this.BindingContext;
            this.patchesStripComboBox.ComboBox.DataSource = patchesBindingSource;
            this.patchesStripComboBox.ComboBox.ValueMember = "id";
            this.patchesStripComboBox.ComboBox.DisplayMember = "name";

            this.DeletePatchToolStripMenuCombo.ComboBox.BindingContext =
this.BindingContext;
            this.DeletePatchToolStripMenuCombo.ComboBox.DataSource = patchesDelBind-
ingSource;
            this.DeletePatchToolStripMenuCombo.ComboBox.ValueMember = "id";
            this.DeletePatchToolStripMenuCombo.ComboBox.DisplayMember = "name";
        }

        public void AddDirectoriesRecursively(string folder, int? p_id)
        {
            DirectoryInfo di = new DirectoryInfo(folder);

            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"Insert into [folders] (p_id,name) values
(@p_id,@name); SELECT last_insert_rowid() AS [ID]";
                command.Parameters.Add(new SQLiteParameter("@p_id", p_id == null ? "null"
: p_id.ToString()));
                command.Parameters.Add(new SQLiteParameter("@name", di.Name));
                p_id = int.Parse(command.ExecuteScalar().ToString());
            }

            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"Insert into [files] (p_id,name) values
(@p_id,@name)";
                SQLiteParameter _name = new SQLiteParameter("@name");
                command.Parameters.Add(_name);
                command.Parameters.Add(new SQLiteParameter("@p_id", p_id));

                foreach (FileInfo cfi in di.GetFiles())
                {
                    _name.Value = cfi.Name;
                    command.ExecuteNonQuery();
                }
            }

            foreach (DirectoryInfo cdi in di.GetDirectories())
                AddDirectoriesRecursively(cdi.FullName, p_id);
        }

        private void UpdateModulesDropDown()

```

```

{
    treeNodeModule.DropDownItems.Clear();
    treeNodeModule.DropDownItems.Add("");

    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"select name from modules";

        SQLiteDataReader DataReader = command.ExecuteReader();
        while (DataReader.Read())
            treeNodeModule.DropDownItems.Add(DataReader["name"]);
    }
}

private void _nodeModule_ChangesApplied(object sender, EventArgs e)
{
    (sender as NodeComboBox).Parent.SelectedNode.Expand();
}

private void applyButton_Click(object sender, EventArgs e)
{
    SQLiteConnection mCN = ConnectionManager.connection;
    SQLiteDataAdapter mDA = modulesTableAdapter.Adapter;
    SQLiteCommandBuilder mCB = new SQLiteCommandBuilder(mDA);
    DataSet mDS = modulesDataSet;
    DataSet dsChanges = new DataSet();
    if (!mDS.HasChanges()) return;
    dsChanges = mDS.GetChanges(DataRowState.Modified);
    if (dsChanges != null)
    {
        mDA.UpdateCommand = mCB.GetUpdateCommand();
        mDA.Update(dsChanges, "modules");
    }
    dsChanges = mDS.GetChanges(DataRowState.Added);
    if (dsChanges != null)
    {
        mDA.InsertCommand = mCB.GetInsertCommand();
        mDA.Update(dsChanges, "modules");
    }
    dsChanges = mDS.GetChanges(DataRowState.Deleted);
    if (dsChanges != null)
    {
        mDA.DeleteCommand = mCB.GetDeleteCommand();
        mDA.Update(dsChanges, "modules");
    }
    mDS.AcceptChanges();
    UpdateModulesDropDown();
    (treeView.Model as SlowTreeModel).Root.UpdateModulesFromDbRec();
    treeView.Invalidate();
}

private void cancelButton_Click(object sender, EventArgs e)
{
    modulesTableAdapter.Fill(modulesDataSet.modules);
}

private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (treeView.SelectedNode != null)
    {

```

```

Node parent = treeView.SelectedNode.Parent.Tag as Node;
List<string> parents = new List<string>();
while (parent != null)
{
    parents.Add(parent.ToString());
    parent = parent.Parent;
}
parents.Reverse();

NodeModule nm = treeView.SelectedNode.Tag as NodeModule;
string id = nm.Tag.ToString();
if (id.StartsWith("f"))
{
    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"delete from files where id=@id";
        command.Parameters.Add(new SQLiteParameter("@id",
id.Substring(1)));
        command.ExecuteNonQuery();
    }
}
else
{
    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"delete from folders where id=@id";
        command.Parameters.Add(new SQLiteParameter("@id", id));
        command.ExecuteNonQuery();
    }
}

treeView.Model = new SlowTreeModel();
TreePath tp = new TreePath();
treeView.FindNode(tp).Expand();

for (int i = 0; i < parents.Count; i++)
{
    tp = new TreePath(tp, parents[i]);
    treeView.FindNode(tp).Expand();
}
}

private void treeViewAdv1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyData == Keys.Delete)
        deleteToolStripMenuItem_Click(null, null);
}

private void deleteModulesContextMenu_Click(object sender, EventArgs e)
{
    modulesView.Rows.RemoveAt(modulesView.SelectedCells[0].RowIndex);
}

private void patchesStripComboBox_Click(object sender, EventArgs e)
{
    if (patchesToolStripMenuItem.DropDown.Visible == true)
    {
        ToolStripComboBox cb = sender as ToolStripComboBox;
        if (cb.ComboBox.SelectedValue != null)
        {

```

```

        patchesToolStripMenuItem.DropDown.Hide();
        (new Graph-
Form(int.Parse(cb.ComboBox.SelectedValue.ToString()))).ShowDialog();
    }
}

private void addPatchToolStripMenuItem_Click(object sender, EventArgs e)
{
    (new AddPatch()).ShowDialog();
    this.patchesTableAdapter.Fill(this.patchesDataSet.patches);
}

private void filesRelationsToolStripMenuItem_Click(object sender, EventArgs e)
{
    (new RelationsForm()).ShowDialog();
}

private void modulesGraphToolStripMenuItem_Click(object sender, EventArgs e)
{
    (new GraphForm()).ShowDialog();
}

private void openProjectToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofp = new OpenFileDialog();
    ofp.Multiselect = false;
    ofp.Filter = "sqlite3 files|*.db3";
    ofp.FilterIndex = 1;
    ofp.RestoreDirectory = true;

    if (ofp.ShowDialog() == DialogResult.OK)
    {
        string baseName = ofp.FileName;
        ConnectionManager.filename = baseName;
        setupView();
    }
}

private void setupView()
{
    ConnectionManager.ExecuteNonQuery("PRAGMA foreign_keys = true;");
    this.modulesTableAdapter.Connection.ConnectionString = ConnectionManag-
er.ConnectionString;
    this.patchesTableAdapter.Connection.ConnectionString = ConnectionManag-
er.ConnectionString;
    mainPanel.Enabled = true;
    this.patchesTableAdapter.Fill(this.patchesDataSet.patches);
    this.modulesTableAdapter.Fill(this.modulesDataSet.modules);
    treeView.Model = new SlowTreeModel();
    UpdateModulesDropDown();
}

private void mainPanel_EnabledChanged(object sender, EventArgs e)
{
    foreach (System.Windows.Forms.Control ctrl in mainPanel.Controls)
    {
        ctrl.Enabled = mainPanel.Enabled;
    }
    patchesToolStripMenuItem.Enabled = mainPanel.Enabled;
    relationsToolStripMenuItem.Enabled = mainPanel.Enabled;
}

```

```

    }

    private void closeProjectToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.patchesDataSet.patches.Clear();
        this.modulesDataSet.modules.Clear();
        treeView.Model = null;
        mainPanel.Enabled = false;
    }

    private void newToolStripMenuItem_Click(object sender, EventArgs e)
    {
        SaveFileDialog sfd = new SaveFileDialog();
        sfd.Title = "Select file to save project data";
        sfd.Filter = "sqlite3 files|*.db3";
        sfd.FilterIndex = 1;
        sfd.RestoreDirectory = true;

        if (sfd.ShowDialog() == DialogResult.OK)
        {
            FolderBrowserDialog fbd = new FolderBrowserDialog();
            fbd.Description = "Select the project sources root directory:";
            fbd.RootFolder = Environment.SpecialFolder.MyComputer;
            DialogResult result = fbd.ShowDialog();
            if (result == DialogResult.OK)
            {
                string baseName = sfd.FileName;
                ConnectionManager.filename = baseName;
                SQLiteConnection.CreateFile(baseName);
                ConnectionManager.CreateDB();

                AddDirectoriesRecursively(fbd.SelectedPath, null);
                setupView();
            }
        }
    }

    private void DeletePatchToolStripMenuCombo_SelectedIndexChanged(object sender,
    EventArgs e)
    {
        if (patchesToolStripMenuItem.DropDown.Visible == true)
        {
            patchesToolStripMenuItem.DropDown.Hide();
            if (MessageBox.Show("Do you want to delete '" + DeletePatchToolStripMenu-
            Combo.ComboBox.Text + "'",
                                "Delete patch",
                                MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                using (SQLiteCommand command = new SQLiteCom-
            mand(ConnectionManager.connection))
                {
                    command.CommandText = @"delete from patches where id=@id";
                    command.Parameters.Add(new SQLiteParameter("@id", DeletePatch-
            ToolStripMenuCombo.ComboBox.SelectedValue));
                    command.ExecuteNonQuery();
                }
                patchesTableAdapter.Fill(patchesDataSet.patches);
            }
        }
    }
}

```


NodeModule.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SQLite;
using Aga.Controls.Tree;
using Aga.Controls.Tree.NodeControls;
using Aga.Controls;

namespace RegressionViewer
{
    public class NodeModule : Node
    {
        public NodeModule(string text) : base(text) { }
        public NodeModule(string text, string module) : base(text)
        {
            if (module!=null) _module = module;
        }
        private SlowTreeModel _model;
        public new SlowTreeModel Model
        {
            get { return _model; }
            set { _model = value; }
        }
        public new bool IsLeaf
        {
            get
            {
                if (this.Tag != null)
                    return this.Tag.ToString().StartsWith("f");
                else return false;
            }
        }

        private string _module;
        public string Module
        {
            get
            {
                return _module;
            }
            set
            {
                if (value != _module)
                {
                    if (this.IsLeaf) _module = value;
                    if (value!=null)
                    {
                        UpdateModulesInTreeRec(this, value);
                        UpdateModulesInDBRec(value, this.Tag.ToString());
                    }
                }
            }
        }

        //Fired when folder in tree is updated to update its childs
        private void UpdateModulesInTreeRec(Node N, string value)
        {
            foreach (NodeModule n in N.Nodes)

```

```

        {
            if (n.IsLeaf) n._module = value;
            UpdateModulesInTreeRec(n,value);
        }
    }

    //Fired to update changes made in GUI in database
    public void UpdateModulesInDBRec(string moduleName, string id)
    {
        if (id.StartsWith("f"))
        {
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"Update [files] set module_id=(select id from
modules where name=@module) where id=@id";
                command.Parameters.Add(new SQLiteParameter("@id", id.Substring(1)));
                command.Parameters.Add(new SQLiteParameter("@module", moduleName));
                command.ExecuteNonQuery();
            }
        }
        else
        {
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"Select id from folders where p_id=@p_id";
                command.Parameters.Add(new SQLiteParameter("@p_id", id));
                SQLiteDataReader DataReader = command.ExecuteReader();
                while (DataReader.Read())
                    UpdateModulesInDBRec(moduleName, DataReader["id"].ToString());
            }
            using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
            {
                command.CommandText = @"Update [files] set module_id=(select id from
modules where name=@module) where p_id=@p_id";
                command.Parameters.Add(new SQLiteParameter("@p_id", id));
                command.Parameters.Add(new SQLiteParameter("@module", moduleName));
                command.ExecuteNonQuery();
            }
        }
    }

    //Fired when modules where changed in database (eg deleted, renamed)
    public void UpdateModulesFromDbRec()
    {
        foreach (NodeModule n in this.Nodes)
        {
            if (!n.IsLeaf)
            {
                using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
                {
                    command.CommandText = @"select name,id,(select name from modules
where id=module_id) as module from files where p_id=@p_id";
                    command.Parameters.Add(new SQLiteParameter("@p_id", n.Tag));
                    SQLiteDataReader DataReader = command.ExecuteReader();
                    while (DataReader.Read())
                    {
                        try

```

```
        {  
            NodeModule nm = n.Nodes.First(a => a.Tag.ToString() ==  
"f" + DataReader["id"]) as NodeModule;  
            nm.Module = DataReader["module"].ToString();  
        }  
    catch { }  
    }  
}  
n.UpdateModulesFromDbRec();  
}  
}  
}  
}
```

SlowTreeModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Collections.ObjectModel;
using System.Data.Common;
using System.Data.SQLite;
using Aga.Controls.Tree;
using Aga.Controls.Tree.NodeControls;
using Aga.Controls;

namespace RegressionViewer
{
    public class SlowTreeModel : ITreeModel
    {
        private NodeModule _root;
        public NodeModule Root
        {
            get { return _root; }
        }

        public Collection<Node> Nodes
        {
            get { return _root.Nodes; }
        }

        public TreePath GetPath(NodeModule node)
        {
            if (node == _root)
                return TreePath.Empty;
            else
            {
                Stack<object> stack = new Stack<object>();
                while (node != _root)
                {
                    stack.Push(node);
                    node = node.Parent as NodeModule;
                }
                return new TreePath(stack.ToArray());
            }
        }

        public NodeModule FindNode(TreePath path)
        {

```

```

    if (path.FullPath.Length == 0)
        return null;
    else if (path.FullPath.Length==1)
        return _root;
    else
        return FindNode(_root, path, 1);
}

private NodeModule FindNode(NodeModule root, TreePath path, int level)
{
    foreach (NodeModule node in root.Nodes)
        if (node == path.FullPath[level])
        {
            if (level == path.FullPath.Length - 1)
                return node;
            else
                return FindNode(node, path, level + 1);
        }
    return null;
}

#region ITreeModel Members

public System.Collections.IEnumerable GetChildren(TreePath treePath)
{
    NodeModule nodeSelected;
    NodeModule nodeChild;

    nodeSelected = this.FindNode(treePath);

    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"select * from folders where p_id=@p_id";
        command.Parameters.Add(new SQLiteParameter("@p_id", (nodeSelected != null
? nodeSelected.Tag : "null")));
        SQLiteDataReader DataReader = command.ExecuteReader();
        while (DataReader.Read())
        {
            nodeChild = new NodeModule(DataReader["name"].ToString());
            nodeChild.Tag = DataReader["id"];

            if (nodeSelected == null)
            {
                _root = nodeChild;
                _root.Model = this;
            }
            else nodeSelected.Nodes.Add(nodeChild);
            yield return nodeChild;
        }
    }

    using (SQLiteCommand command = new SQLiteCom-
mand(ConnectionManager.connection))
    {
        command.CommandText = @"select name,id,(select name from modules where
id=module_id) as module from files where p_id=@p_id";
        command.Parameters.Add(new SQLiteParameter("@p_id", (nodeSelected != null
? nodeSelected.Tag : "null")));
        SQLiteDataReader DataReader = command.ExecuteReader();
        while (DataReader.Read())
        {

```

```

        nodeChild = new NodeModule(
            DataReader["name"].ToString(), DataReader["module"].ToString());
        nodeChild.Tag = "f" + DataReader["id"];

        nodeSelected.Nodes.Add(nodeChild);
        yield return nodeChild;
    }
}

public bool IsLeaf(TreePath treePath)
{
    return (treePath.LastNode as NodeModule).Tag.ToString().StartsWith("f");
}

#pragma warning disable 67
public event EventHandler<TreeModelEventArgs> NodesChanged;
public event EventHandler<TreeModelEventArgs> NodesInserted;
public event EventHandler<TreeModelEventArgs> NodesRemoved;
public event EventHandler<TreePathEventArgs> StructureChanged;
#pragma warning restore 67

#endregion
}

}

```

ДОДАТОК Б

Презентація до дипломної роботи:

* Дослідження методів
регресійного тестування за
допомогою систем управління
версіями

Виконав: Даниленко Артем Сергійович

Науковий керівник: Дідковська Марина Віталіївна

Рисунок Б.1 - Тема Бакалаврської роботи

* Актуальність теми

Регресійне тестування - дорогий, але необхідний етап тестування програмного забезпечення, спрямований на повторну перевірку коректності зміненої програми.

Регресійне тестування потрібно виконувати при випуску будь-якої версії ПЗ, а темпи розвитку програмного забезпечення набирають таких обертів, що виробники змушені випускати нові версії програмного забезпечення все частіше і частіше додаючи нові функції та виправляючи існуючі проблеми

План випуску нових версій

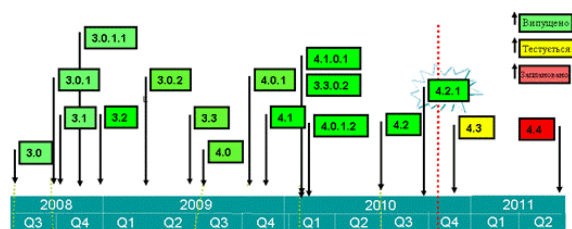


Рисунок Б.2 – Актуальність теми

* Цілі даної роботи

У даній роботі ставиться за мету:

- * Аналіз методів регресійного тестування
- * Аналіз систем управління версіями
- * Розробка методу регресійного тестування на основі систем управління версіями
- * Розробка програмного продукту для планування регресійного тестування на основі систем управління версіями

Рисунок Б.3 – Цілі даної роботи

* Постановка задачі

Планування регресійного тестування ставить такі задачі:

- мінімізація кількості виконуваних тестів
- вибирати тести, що мають максимальну можливість виявити несправність
- мінімізувати час самого планування

Рисунок Б.4 – Постановка задачі

* Постановка задачі

Дано:

P - певна версія програмного забезпечення

$T = \{t_1, t_2, \dots, t_N\}$ - множина з N тестів, що використовується при первинному тестуванні програми P

P' - нова версія програми P

Знайти:

$T'_p \supset T'$ - шукана множина тестів що є ціллю метода вибору тестів. Множина T'_p повинна включати всі тести з T , що активують змінений код, і не включати ніяких інших тестів, тобто тест $t \in T$ входить до T'_p тоді і тільки тоді, коли t задіє код P в точці, де в P' код був видалений або змінений, або де був доданий новий код,

де $T' \subset T$ - невідома, ідеальна підмножина регресійних тестів для тестування нової версії програми P' , тобто $T' = \{t \in T \mid P'(t) \neq P(t)\}$

Рисунок Б.5 – Постановка задачі

* Класифікація методів регресійного тестування

1)Випадкові методи

Тести вибираються випадковим чином або на основі "здогадок", тобто можливого співвіднесення тестів з функціональними можливостями на підставі попередніх знань або досвіду.

2)Безпечні методи

Метод вибіркового регресійного тестування називається безпечним, якщо він не виключає тестів, які виявили б помилки у змінений програмі.

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. Мінімізація набору тестів вимагає великих витрат на аналіз.

4)Методи, засновані на покритті коду

Вибираються тільки ті тести з T що активують змінений у P' код.

Рисунок Б.6 – Класифікація методів регресійного тестування

* Використання систем контролю версій для планування регресійного тестування

У галузі розробки програмного забезпечення, контроль версій є єдиною можливістю відстежувати і забезпечувати контроль за змінами у вихідному коді. 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 виправленнями(patches). Певні помилки або функції програмного забезпечення часто присутні тільки в деяких версіях. Таким чином, з метою пошуку та виправлення помилок, життєво важливо мати можливість контролювати процес регресійного тестування і запускати різні набори тестів на різних версіях.

Системи контролю версій дозволяють отримати детальну інформацію в яких саме файлах(або модулях програмного забезпечення) відбулися зміни в порівнянні з попередньою версією.

Рисунок Б.7 – Використання систем контролю версій для планування регресійного тестування

* Формалізація залежностей у програмному забезпеченні

$P = \{M_i | i = 1..m\}$, де M_i - модуль (функціонально закінчений фрагмент) програми

$M_i = \{F_{i_l} | l = 1..p_i\}$, де F_{i_l} - файл вихідного коду що входить до складу модулю.

$R_{ij_k} = R(F_{i_l}, F_{j_k})$ - зв'язок між файлами вихідного коду, виражається цілим числом від 0 до 10: 0 - зв'язку між файлами не існує(за замовчуванням); 5 - F_{i_l} використовує F_{j_k} ; 10 - F_{i_l} повністю залежить від F_{j_k} . Ступінь залежності визначається ОПР (експерт у програмному забезпеченні)

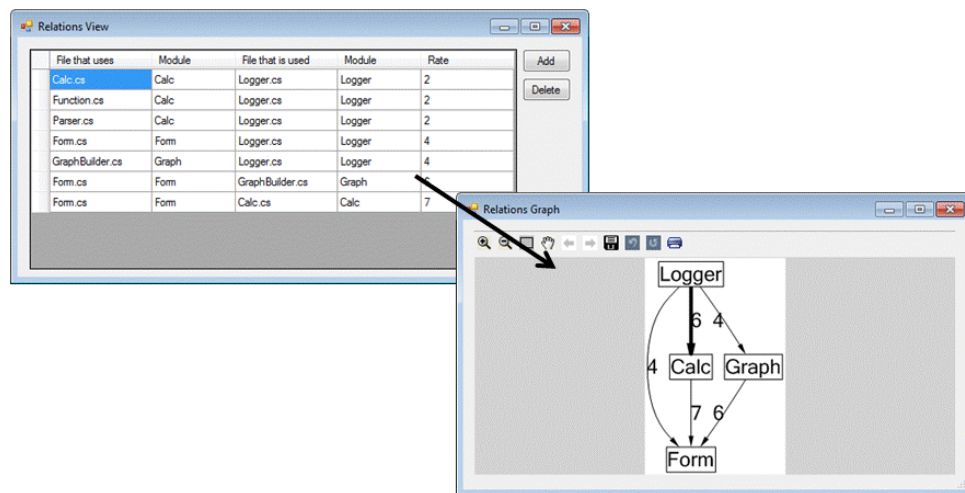
Тоді маємо формулу R_{ij} - зв'язку між двома модулями:

$$R_{ij} = \sum_{l=1}^{p_i} \sum_{k=1}^{p_j} R_{ij_k}$$

Тепер можемо ввести когнітивну карту залежностей у програмному забезпеченні: $(\{M_i\}, \{R_{ij_k}\})$ - модель подання знань експерта у вигляді знакового орграфа, де $\{M_i\}$ - множина вершин, $\{R_{ij_k}\}$ - множина зважених ребер

Рисунок Б.8 – Формалізація залежностей у програмному забезпеченні

* Когнітивна карта залежностей у програмному забезпеченні



Визначення зв'язків між файлами дає когнітивну карту ПЗ

Рисунок Б.9 – Когнітивна карта залежностей у програмному забезпеченні

* Використання систем контролю версій для планування регресійного тестування

$C^{P'}(F_{i_l})$ - функція-індикатор що вказує чи змінився файл F_{i_l} , у версії P' у порівнянні з P . У даному методі пропонується отримати $C^{P'}$ з системи управління версіями, наприклад найпоширеніші безкоштовні системи управління версіями git та svn мають функцію що називається log і показує які файли мінялися у зазначений проміжок часу або у певному інтервалі ревізій системи контролю версій.

Тоді маємо формулу $R_{ij}^{P'}$ - міри зміни модулю j у зв'язку зі змінами у модулі i у версії P' :

$$R_{ij}^{P'} = \sum_{l=1}^{p_l} \sum_{k=1}^{p_k} (R_{i_l j_k} \cdot C^{P'}(F_{i_l}))$$

$R_{ij}^{P'} = 0$, тоді і тільки тоді коли або файл F_{j_k} не залежить від F_{i_l} , або F_{i_l} не змінювався в версії P' .

Тепер можемо побудувати когнітивну карту змін у новій версії P' програмного забезпечення: $(\{M_i\}, \{R_{ij}^{P'}\})$, де $\{M_i\}$ - множина вершин, $\{R_{ij}^{P'}\}$ - множина зважених ребер

Рисунок Б.10 – Використання систем контролю версій для планування регресійного тестування

* Використання систем контролю версій для планування регресійного тестування

$C_2^{P'}(F_{i_l})$ - функція-індикатор що вказує чи зазнав файл F_{i_l} прямого впливу від змін у версії P' у порівнянні з P

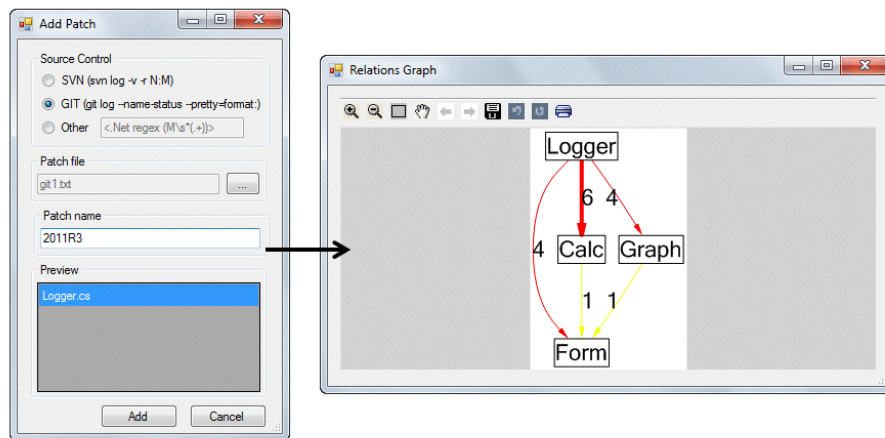
Тоді маємо формулу $IR_{ij}^{P'}$ - міри непрямих змін модулю j у зв'язку зі змінами у модулі i у версії P' :

$$IR_{ij}^{P'} = \sum_{l=1}^{p_l} \sum_{k=1}^{p_k} (R_{i_l j_k} \cdot C_2^{P'}(F_{i_l}))$$

Тепер можемо побудувати когнітивну карту змін у новій версії P' програмного забезпечення, враховуючи непрямі зв'язки: $(\{M_i\}, \{R_{ij}^{P'}\} \cup \{IR_{ij}^{P'}\})$.

Рисунок Б.11 – Використання систем контролю версій для планування регресійного тестування

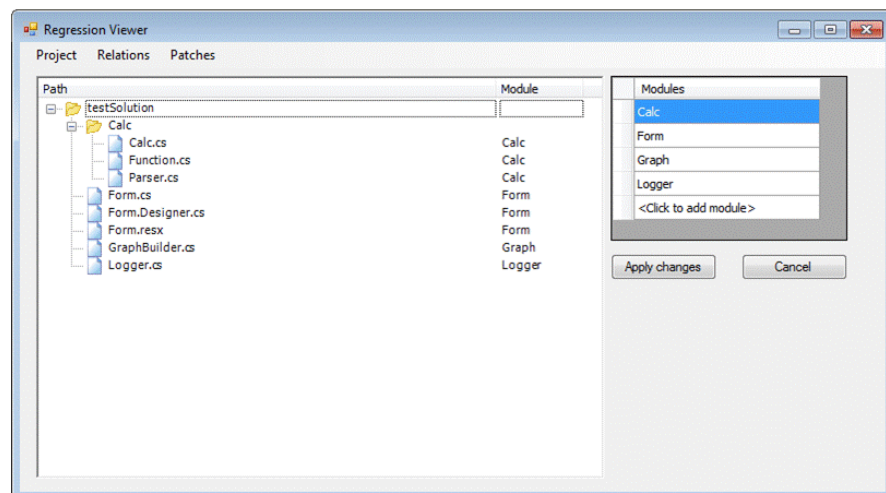
* Когнітивна карта нової версії програмного забезпечення



Визначення файлів, що увійшли у патч дає когнітивну карту патчу

Рисунок Б.12 – Когнітивна карта нової версії програмного забезпечення

* Програмне забезпечення для планування регресійного тестування Regression Viewer



Головна форма програми - Дерево вихідних файлів та модулі ПЗ

Рисунок Б.13 – Програмне забезпечення для планування регресійного тестування Regression Viewer

* Блок-схема алгоритму програмного забезпечення Regression Viewer

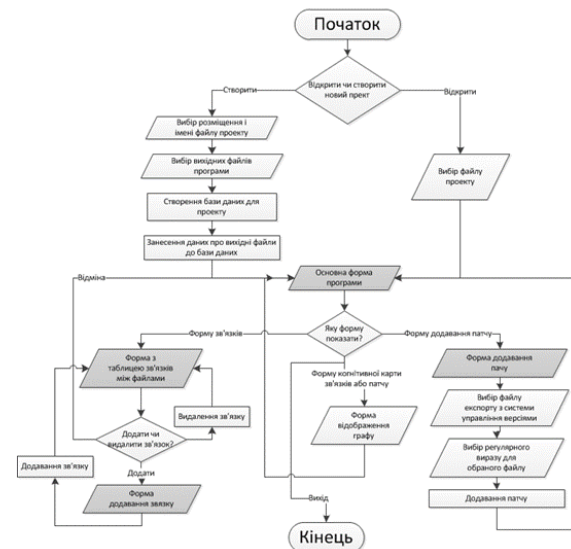


Рисунок Б.14 – Блок-схема алгоритму програмного забезпечення Regression Viewer

* ВИСНОВКИ

Проведений аналіз методів регресійного тестування та систем управління версіями дав можливість:

- * розробити метод планування регресійного тестування заснований на використанні систем управління версіями
- * розробити програмне забезпечення що реалізує дані функції за допомогою побудови когнітивних карт

Рисунок Б.15 – Висновки