Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделі тестування застосунків для мобільних пристроїв

(тема)

Виконав:

студент II курсу, групи СПзм-20-1

Браїла І.В.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Міхаль О.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Браїлі Ірині Вадимівні
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі тестування застосунків для мобільних пристроїв

затверджена наказом по університету від " 25 " березня 2022 р. № 33 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи

розробка моделі

мобільні застосунки

тестовий набір даних

4. Перелік питань, що потрібно опрацювати у роботі

Аналіз предметної області

Методи тестування протитипів застосунків

Автоматична генерація тестів

Розробка моделі тестування застосунків для мобільних пристроїв

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання.Аналіз предметної області	25.03.2022–10.04.2022	
2	Аналіз існуючих моделей та методів	11.04.2022–26.04.2022	
3	Розробка моделі	27.04.2022–29.04.2022	
4	Розробка автоматизованих тестів	30.04.2022–02.05.2022	
5	Отримання та аналіз результатів	03.05.2022–06.05.2022	
6	Оформлення пояснювальної записки	07.05.2022–13.05.2022	

Дата видачі завдання 25 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Міхаль О.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 76 с., 27 рис., 6 табл., 1 дод., 8 джерел.

ЗАСТОСУНОК, ТЕСТУВАННЯ, МОДЕЛЬ, ГЕНЕРАЦІЯ ТЕСТІВ, СЕРВЕР ВІРТУАЛІЗАЦІЇ.

Метою кваліфікаційної роботи є дослідження моделей та алгоритмів тестування застосунків для мобільних пристроїв

У ході виконання кваліфікаційної роботи досліджені моделі та алгоритми тестування застосунків для мобільних пристроїв. Було використано прототип застосунків для мобільних пристроїв; проведено аналіз метрик тестування застосунків для мобільних пристроїв, що враховують їх особливості та відповідні критерії оцінки ефективності методів тестування. Проведено аналіз аналітичних та програмних моделей генерації автоматизованих тестів з прототипів застосунків для мобільних пристроїв.

ABSTRACT

Master’s thesis: 76 pages, 27 figures, 6 tables, 1 appendices, 8 sources.

APPLICATION, TESTING, MODEL, TEST GENERATION, VIRTUALIZATION SERVER.

The major goal of this thesis is to research models and algorithms for testing applications for mobile devices

In order to models and algorithms of testing applications for mobile devices were studied. Prototype applications for mobile devices were used; the analysis of the metrics of testing applications for mobile devices, taking into account their features and the relevant criteria for evaluating the effectiveness of testing methods, was carried out. Analytical and software models for generating automated tests from prototype applications for mobile devices were analyzed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Program Interface (інтерфейс програми застосунків)

MVC – Mode-View-Controller (технологія розробки застосунків)

MV – Mode-View (технологія розробки застосунків).

UI – призначений для користувача інтерфейс

ЗМП – застосунок для мобільних пристроїв

КА – кінцевий автомат

ПЗ – програмне забезпечення

РКА – розширений кінцевий автомат

ВСТУП

Сьогодні розробку якісного програмного забезпечення важко уявити без використання методів статичного аналізу коду. Статичний аналіз програмного коду може бути вбудований в середу розробки (стандартними методами або за допомогою додаткових модулів), може виконуватися спеціалізованим програмним забезпеченням перед запуском коду в промислову експлуатацію або «вручну» штатним або зовнішнім експертом.

Бувають ситуації, коли при розробці або тестуванні мобільного застосунку виникає необхідність переглянути мережевий трафік застосунку. Мобільний користувач очікує, що встановлені їм програми прості, інтуїтивно зрозумілі, працюють завжди і всюди без збоїв. Якщо очікування не виправдовуються, то користувач просто-напросто встановлює аналогічний додаток від іншого учасника, яких в сфері мобільних розробок завжди достатньо. Тому якість програми є одним з головних факторів його популярності.

Щоб зрозуміти особливості тестування мобільних застосунків слід брати до уваги моменти, що принципово відрізняють мобільні додатки від десктопних: специфічність ОС для мобільних платформ, різні компанії-виробники пристроїв і конфігурації комплектуючих, функціональність пристроїв як комунікаторів і т.п. У зв'язку з цими особливостями, підхід до розробки застосунків і, зокрема, тестування на мобільних пристроях досить сильно відрізняється від десктопного. Виникає безліч додаткових важливих нюансів і вимог, які необхідно протестувати. Все це робить дану роботу дуже актуальною.

Метою роботи є дослідження моделей тестування застосунків для мобільних пристроїв.

Основні розв'язувані завдання полягають у наступному:

- використовувати прототип застосунків для мобільних пристроїв;

- для генерації тестів представити прототип у вигляді кінцевого автомата;

- провести аналіз метрик тестування застосунків для мобільних пристроїв, що враховують їх особливості та відповідні критерії оцінки ефективності методів тестування;

- провести аналіз аналітичних та програмних моделей генерації автоматизованих тестів з прототипів застосунків для мобільних пристроїв.

- інтерпретація отриманих результатів в контексті дослідження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Метрика тестового покриття застосунків для мобільних пристроїв

У ітеративній схемі, в процесі розробки застосунку для мобільних пристроїв (ЗМП), додаток надходить до тестування на системній стадії, минаючи процеси модульного і інтеграційного тестування. Таким чином, процес функціонального тестування зводиться до перевірки функціональності програми на рівні користувача інтерфейсу.

У загальному випадку взаємодія користувача з застосунком відбувається за такою схемою (рисунок 1.1):

- користувач бачить на екрані мобільного пристрою деякий «вид» застосунку. Цей вид містить елементи призначеного для користувача інтерфейсу (кнопки, поля введення і ін.), які дозволяють здійснювати різні дії (запити користувача);
- запит, що генерується користувачем, приходить в «логічну» частину програми, яка обробляє запит і, можливо, звертається до бази даних по необхідними даними;
- отримавши дані, «логічна» частина генерує наступний «вид», Який побачить користувач - як результат свого запиту;
- далі процес повторюється.
-

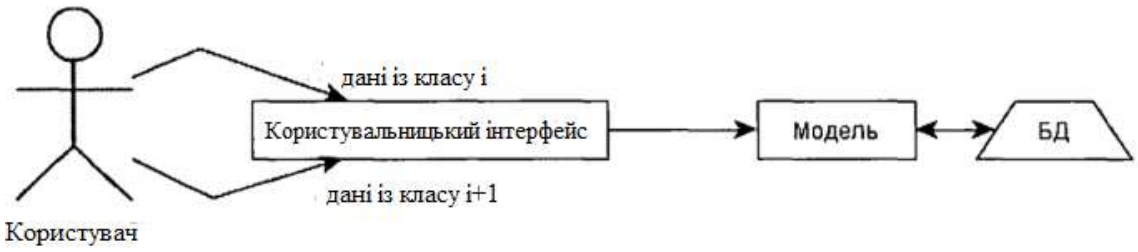

Рисунок 1.1 – Схема взаємодії користувача з застосунком

В умовах прискореного процесу розробки ЗМП, мета тестування в кінці кожного циклу ітераційної схеми виявити не всі помилки програми, а тільки ті помилки, які можуть виникнути при взаємодії застосунку з кінцевим користувачем. Переслідуючи цю мету, розглядається лише частина запитів користувача і відповідних вхідних даних. Тестування проводиться тільки на даній частині даних. На рисунку 1.2 зображено розбиття вхідних даних на класи еквівалентності.

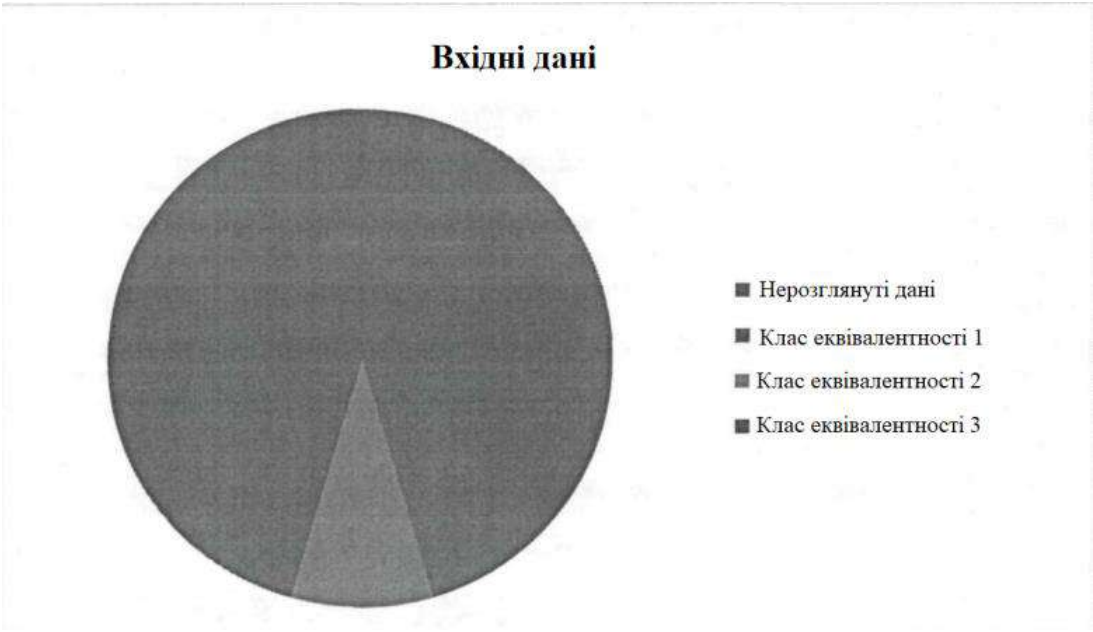

Рисунок 1.2 – Розбиття вхідних даних на класи еквівалентності

Такий підхід до тестування не виключає всі помилки в застосунку, але мінімізує ймовірність знаходження залишившись помилок кінцевим користувачем. Рисунок 1.3 ілюструє момент припинення тестування ЗМП. Таким чином, в результаті існування вищеописаних особливостей в розробці ЗМП, можна сформулювати такі тези, що визначають метрику тестування ЗМП і критерій повноти тестування:

- мета тестування ЗМП - виявити не всі помилки програми, а тільки ті помилки, які виникають при роботі застосунку з кінцевим користувачем, з відповідним обмеженням значень вхідних даних;

- при тестуванні повинні бути враховані всі запити, якими оперує кінцевий користувач при взаємодії з застосунком;

- перевірка функціоналу ЗМП проводиться на рівні користувача інтерфейсу.

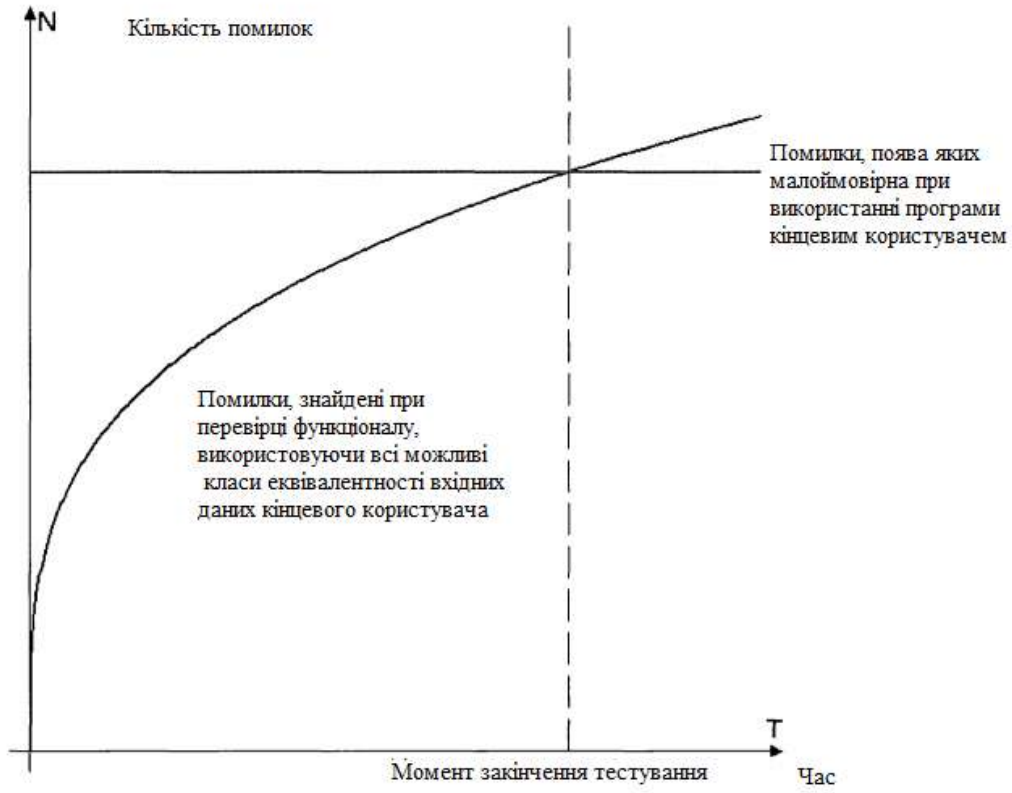

Рисунок 1.3 – Момент закінчення тестування мобільного застосунку

Беручи до уваги дані тези, можна визначити метрику тестування ЗМП і критерій повноти тестування в наступний спосіб, представлений далі.

Метрика тестування ЗМП - відсоток перевірених відгуків застосунку при впливі користувача на елементи призначеного для користувача інтерфейсу з урахуванням розбиття вхідних даних на класи еквівалентності, відповідні призначеним для користувача сценаріями використання застосунку.

Критерій повноти тестування - для забезпечення повного покриття функціоналу досить перевірити кожен відгук застосунку після впливу на кожен елемент UI, вводячи дані, відповідно призначеним для користувача

сценаріями використання програми.

1.2 Аналіз методів тестування застосунків для мобільних пристроїв

У схемі, зображеної на рисунку 1.4, на кожному етапі розробки можна вибрати один із способів переходу до наступного етапу. Залежно від обраного переходу змінюється час досягнення наступного етапу. Сумарний час розробки залежить від шляху проходження зі стану 1 схеми в стан 7 і від кількості ітерацій циклу 1-7. Зі стану 1 в стан 7 можна прийти різними шляхами, але в кожному стані вибір наступного маршруту залежить від попереднього стану процесу розробки. Кожен з можливих маршрутів визначає метод тестування програми. Розглянемо найбільш поширені шляхи переходу з 1 в 7.

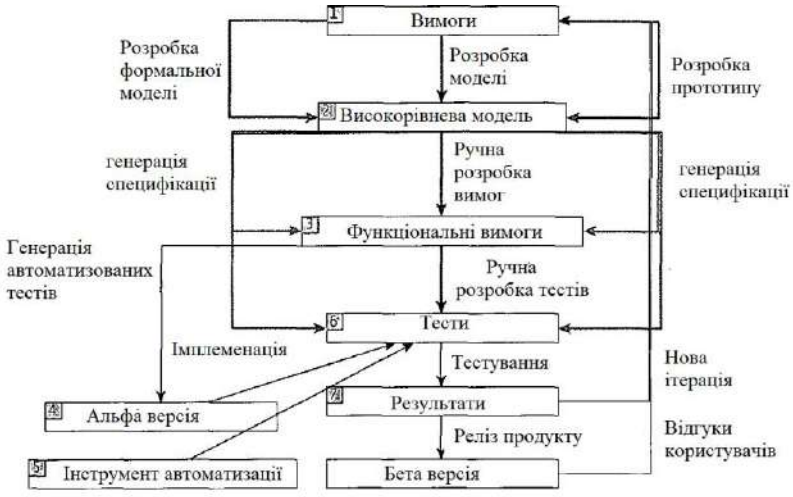

Рисунок 1.4 – Детальна ітеративна схема розробки

Ручне тестування з документації проводиться наступним чином:

а) визначення вимог;

б) розробка шаблонів;

в) ручна розробка текстової функціональної специфікації;

г) ручна розробка тестових сценаріїв;

д) ручне тестування.

Метод ручної розробки документів і тестів дуже гнучкий і дозволяє досягти будь-якого рівня тестового покриття. Однак цей метод вимагає великих тимчасових витрат. Надалі будемо називати цей метод - РР метод.

Автоматизація проводиться наступним чином:

а) визначення вимог;

б) розробка шаблонів;

в) ручна розробка текстової функціональної специфікації;

г) ручна розробка тестових сценаріїв в текстовому вигляді;

д) автоматизація тестових сценаріїв;

е) автоматизоване тестування.

Метод автоматичної розробки тестів вимагає текстового опису тестових сценаріїв. По суті, створюються тестові сценарії в текстовому вигляді, як і в випадку РР методу, а на основі даних тестових сценаріїв розробляються автоматизовані тести. Такий підхід вимагає значно більше часу на розробку тестів, але значно скорочує час проведення тестування. Отже, такий підхід ефективний при збільшенні кількості циклів розробки. Надалі будемо називати даний метод - РА метод.

Тестування на основі формальної моделі проводиться наступним чином:

а) визначення вимог;

б) розробка формальної моделі;

в) генерація специфікації;

г) генерація тестових сценаріїв;

д) автоматизоване тестування.

Даний метод добре автоматизуємо, але не досить гнучкий в області розробки ЗМП, оскільки за допомогою формальних моделей важко описувати призначені для користувача інтерфейси, отже, важко отримати необхідну тестове покриття. Надалі будемо називати цей метод - ФМ метод.

Тестування на основі прототипу проводиться таким чином:

а) визначення вимог;

б) розробка прототипу;

в) генерація специфікації;

г) генерація тестових сценаріїв;

д) ручне та автоматизоване тестування.

Метод розробки на основі прототипу дозволяє добре описувати призначені для користувача інтерфейси, отже, отримати необхідне покриття. Метод також дозволяє генерувати різні типи тестових сценаріїв при наявності відповідних інструментів генерації і особливі правила побудови прототипів, що надає додаткову гнучкість в проведенні тестування. Надалі будемо називати такий метод - ПР метод.

1.3 Аналіз методів автоматизації тестування

Для ЗМП існують різні інструменти автоматизації тестування. Зокрема в [4,5] розглянуті деякі з них. У загальному випадку можна виділити два основні підходи до автоматизації тестів.

1.3.1 Програмний метод автоматизації

Метод, який передбачає використання певних бібліотек деякого мови програмування. Автоматизація тестів зводиться до написання допоміжних модулів тестування і написання тестових скриптів на мові програмування. Основний мінус цього підходу - значні часові витрати на реалізацію тестових скриптів.

1.3.2 Метод автоматизації з використанням playback інструментів

Такий підхід вимагає мінімальних навичок в програмуванні. Автоматизація тесту відбувається за допомогою проведення тестового

сценарію на цільовому пристрої з пропущенням впливів через спеціальну програму (проксі-сервер), що записує впливу і дані. Після записи ця програма являє отриману інформацію у вигляді автоматичного тесту. При такому підході автоматизація тесту зводиться до його виконання. Основний мінус цього підходу: будь-яка зміна в тестованому застосунку вимагає перезапису тестових сценаріїв, пов'язаних зі зміненим функціоналом.

1.4 Вибір і обгрунтування критеріїв оцінки і оптимізації методів тестування

1.4.1 Інтегральний критерій оцінки методів тестування

Позначимо обраний шлях переходу по ітеративній схемі (рисунок 1.4) $p_{12} - p_{23} - p_{34} - \ldots = p_{ij} (ij \in [12,23,34,45,56,67])$. Позначимо сумарний час $T(p_{ij})$, а $N(p_{ij})$ – покриття, відповідно N – перевіреним відгуками, відповідно до розробленої метрикою тестування. Тоді оптимізація процесу тестування за двома параметрами задається наступним виразом:

$$\begin{cases} T(p_{ij}) \to max \\ N(p_{ij}) \to min \end{cases}, (ij \in [12,23,34,45,56,67]). \quad (1.1)$$

Так як N – деяке натуральне число, то задану умову можна записати в наступному виді:

$$T = \frac{\sum_{(ij \in [12,23,34,45,56,67])} t_{ij}}{N} \to min$$

Де t_{ij} – час, витрачений на даному етапі, в залежності від обраного способу проходження етапу.

N – покриття відгуків додатки, яке потрібно забезпечити.

Цей вираз визначає інтегральний критерій ефективності підходу до

тестування ЗМП в контексті всього процесу розробки і тестування.

Розглянемо кожний доданок в чисельнику дробі окремо. Попередньо введемо деякий абстрактний час t, відповідно часу опису одного елемента користувальницького інтерфейсу застосунку. Очевидно, що всі складові часи у формулі (1), крім t_{56}, лінійно залежать від t, а часом t_{45}, яке відповідає передачі збірки в тестування, можна знехтувати. Оскільки інтеграція кошти автоматизованого тестування в процес є одиничною операцією, цей час можна не враховувати при оцінці ефективності тестування. Введемо наступні коефіцієнти, що враховують залежності часів на кожному етапі розробки від часу t:

$C_{\text{рмод}}$ – коефіцієнт розробки прототипу, представляє відносний час розробки прототипу в розрахунку на один елемент UI.

$C_{\text{п спец}}$ –коефіцієнт отримання специфікації, являє відносний час розробки функціональної специфікації в розрахунку на один елемент UI.

$C_{\text{імп}}$ – коефіцієнт імплементації, представляє відносний час розробки одного елемента UI. застосунку.

$C_{\text{п тест}}$ – коефіцієнт отримання тестових сценаріїв, відносний час розробки тестових сценаріїв для перевірки одного елемента UI.

$C_{\text{тест}}$- – коефіцієнт проведення тестування, відносний час проведення тестових сценаріїв для перевірки одного елемента UI.

Нехай в ході розробки було пройдено К циклів розробки, на кожному з яких додавалося $\partial N_i (i = 1..K)$ нових відгуків, які потрібно перевірити для забезпечення повного покриття. Тоді, формула (1.1) прийме вид:

$$T = \frac{t\left(C_{\text{рмод}} N + C_{\text{п спец}}\ N + C_{\text{імп}} + C_{\text{п тест}} N + C_{\text{тест}} N\right)}{N} + \sum_{i-1\ K} \frac{t\left(C_{\text{рмод}} \partial N_i + C_{\text{п спец}}\ \partial N_i + C_{\text{імп}} \frac{\partial N_i}{N} + C_{\text{п тест}} \partial N_i + C_{\text{тест}} N\right)}{N} \quad (1.2)$$

Тут слід зазначити, що при проходженні і - ой ітерації циклу розробки:

- час імплементації доданих вимог в раз менше $\frac{N}{\partial N_i}$ часу імплементації

повного функціоналу;

- застосовується регресійне тестування всього функціоналу, а не тільки доданих вимог.

Нехай $\overline{\partial N}$ - середня кількість доданих відгуків за K циклів, тоді перетворимо формулу (1):

$$T = \frac{t\left(C_{\text{рмод}} N + C_{\text{п спец}}\ N + C_{\text{імп}} + C_{\text{п тест}} N + C_{\text{тест}} N\right)}{N} + \sum_{i-1\,K} \frac{t\left(C_{\text{рмод}} \partial N_i + C_{\text{п спец}}\ \partial N_i + C_{\text{імп}} \frac{\partial N_i}{N} + C_{\text{п тест}} \partial N_i + C_{\text{тест}} N\right)}{N + \partial N_i} =$$

$$t\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + C_{\text{тест}}\right) + \frac{C_{\text{імп}}}{N} + \frac{Kt\overline{\partial N}}{N + \overline{\partial N}}\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + \frac{C_{\text{імп}}}{N}\right) +$$
$$+ KtC_{\text{тест}} \frac{N}{N + + \overline{\partial N}}. \tag{1.3}$$

Для оцінки можна вважати, що середньою кількістю доданих вимог можна знехтувати в порівнянні із загальною кількістю $\frac{\overline{\partial N}}{N} \approx$, тоді:

$$T = t\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + C_{\text{тест}}\right) + \frac{C_{\text{імп}}}{N} +$$
$$+ \frac{Kt\overline{\partial N}}{N + \overline{\partial N}}\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + \frac{C_{\text{імп}}}{N}\right) + KtC_{\text{тест}} \frac{N}{N + \overline{\partial N}} =$$
$$= t\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + C_{\text{тест}}\right) + \frac{C_{\text{імп}}}{N}$$
$$+ \frac{Kt\overline{\partial N}}{N}\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}} + \frac{C_{\text{імп}}}{N}\right) + KtC_{\text{тест}} =$$
$$= t\left[\left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}}\right)\left(1 + \frac{K\overline{\partial N}}{N}\right) + C_{\text{тест}}(K+1) + C_{\text{імп}} \frac{N + K\overline{\partial N}}{N^2}\right] \tag{1.4}$$

Отже, задача оптимізації зводиться к мінімізації наступного виразу:

$$T = t\left[(C_{\text{прог}} + C_{\text{оцен}})(K+1) + \left(C_{\text{рмод}} + C_{\text{п спец}}\ + C_{\text{п тест}}\right)\left(1 + \frac{K\overline{\partial N}}{N}\right) + C_{\text{імп}} \frac{N + K\overline{\partial N}}{N^2}\right] \to min \tag{1.5}$$

Якщо розглянути кожний доданок окремо, видно, що для зменшення

загального часу розробки, в першу чергу потрібно спробувати зменшити коефіцієнт виконання тестування $C_{\text{тест}}$, оскільки навпроти нього множник $(K+1)$. У другу чергу потрібно спробувати зменшити суму коефіцієнтів $(C_{\text{рмод}} + C_{\text{п спец}} + C_{\text{п тест}})$. Також слід зазначити, що в разі автоматизації тестових сценаріїв коефіцієнт $C_{\text{п тест}}$ повинен враховувати час розробки автоматизованих тестів, необхідних для тестування одного відгуку програми, а $C_{\text{тест}}$, розбивається на час прогону тестів і час оцінки результатів тестування: $C_{\text{тест}} = C_{\text{прог}} + C_{\text{оцен}}$.

Формула (1.2) дає кількісну оцінку часу, витраченого на К циклів розробкиі тестування ЗМП з покриттям N функціональних вимог. При розрахунку ефективності процесів тестування застосунків $C_{\text{імп}} \frac{N+K\overline{\partial N}}{N^2}$ формули (1.2) можна не враховувати, оскільки воно залежить тільки від процесу розробки програмного коду і ніяк не пов'язане з проведенням тестування. Введемо таке поняття ефективності методу тестування:

$$Eff = \frac{T_{pp}}{T_x}, \tag{1.6}$$

де T_{pp} – час метода РР, T_x – час тестування по методу х. Слід зазначити, що при такому розрахунку, ефективність РР методу дорівнює 1.

З урахуванням прийнятих припущень отримуємо такі часи для процесів:

$$\begin{cases} T_{\text{РР}} = (2.3K + 5)C \\ T_{\text{РА(програмний)}} = \frac{1}{K}(0.8K^2 + 4.5K + 10) \\ T_{\text{РА}(playback)} = C(K + 5.5) \\ T_{\text{ПР}} = C(0.7K + 2.5) \\ T_{\text{ФМ}} = C(1.6K + 11.5) \end{cases} \tag{1.8}$$

Ефективність розглянутих методів зводиться до наступних формул з урахуванням припущень:

$$\begin{cases} Eff_{\text{РА(програмний)}}(K) = \dfrac{2.3K^2 + 5K}{0.8K^2 + 4.5K + 10} \\ Eff_{\text{РА(playback)}}(K) = \dfrac{2.3K + 5}{K + 5.5} \\ Eff_{\text{ПР}}(K) = \dfrac{2.3K + 5}{0.8K + 3.5} \\ Eff_{\text{ФМ}}(K) = \dfrac{2.3K + 5}{1.6K + 11.5} \end{cases} \quad (1.7)$$

Виходячи з результатів формалізації ітеративной схеми розробки, можна зробити припущення про ефективність прототипного підходу в розробці / тестуванні ЗМП. Дане припущення також грунтується на критерії ефективності процесів тестування. Однак в розглянутих критеріях оптимізації процесів тестування без подання їх математичної моделі.

1.4.2 Приватний критерій ефективності моделі генерації

Основою застосування прототипного підходу в тестуванні є ефективний алгоритм генерації тестових сценаріїв. Згенеровані тестові сценарії можуть застосовуватися як для ручного тестування, так і для автоматизованого тестування з використанням програмного підходу автоматизації. Пропонується ввести такі критерії ефективності моделі генерації тестів в рамках загального процесу розробки ПЗ:

- середня ефективність алгоритму генерації;
- середня швидкість роботи алгоритму генерації;
- середнє забезпечується тестове покриття;
- можливість генерації кроків на перевірку результатів тестування;
- розширюваний формат вихідних даних.

Усереднення береться за різними вхідним застосунків, що піддаються тестування:

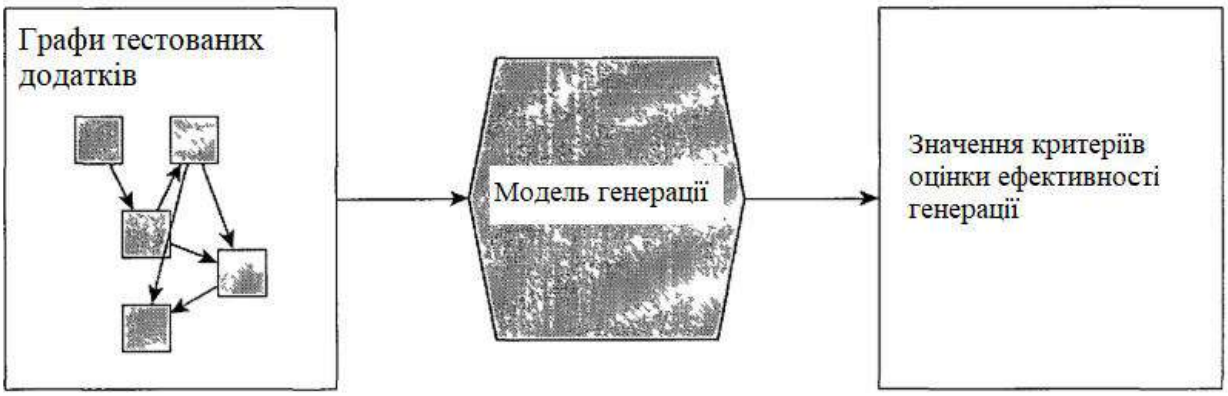

Рисунок 1.5 – Процес оцінки ефективності алгоритму генерації тестів

Середня ефективність алгоритму генерації.

Щоб оцінити ефективність алгоритму генерації, пропонується використовувати методику, описану нижче:

Нехай N - кількість тестових сценаріїв, згенерованих алгоритмом, L - середня довжина тестового сценарію. Процес оцінки ефективності можна представить наступною схемою: Обидва параметра залежать від подаваного графа РКА. Відповідно до теореми Ейлера, в зв'язковому направленому графі існує Ейлером цикл тоді і тільки тоді, коли полустепені результату дорівнює полустепені заходу. Тобто кількість вхідних в вузол гілок дорівнює кількості вихідних. Використовуючи цю теорему можна зробити оцінку нижньої межі кількості переходів, необхідних для покриття всіх переходів РКА. Ця оцінка буде:

$$N_e = \sum_{s_l \in S} \max(s_l^{in}, s_l^{out}) \tag{1.8}$$

Дійсно, відповідно до визначення РКА прототипу, відповідний граф РКА є зв'язковим, оскільки в РКА додається новий стан, тільки якщо існує перехід з поточного стану в нове. Для «ейлерізації» графа необхідно обійти всі його вузли і «доповнити» стан мінімальною кількістю вихідних або

вхідних гілок, таким чином, щоб кількість вхідних гілок в даний вузол дорівнювала кількості вихідних гілок. Очевидно, після подібного доповнення кількість вихідних гілок графа стану s_l, дорівнюватиме кількість вхідних гілок і дорівнюватиме $\max(s_l^{in}, s_l^{out})$. Таким чином, загальна кількість переходів, яке потрібно зробити для обходу всього «Ейлерізованного» графа буде

$$\sum_{s_l \in s} \max\left(s_l^{in}, s_l^{out}\right) \tag{1.9}$$

Тоді показник ефективності числа згенерованих кроків:

$$Eff(G) = \frac{N_e}{N * L} 100\% \tag{1.10}$$

G – граф прототипу ЗМП;

N*L - загальна кількість переходів, здійснених алгоритмом в внаслідок обходу.

$N_e = \sum_{s_l \epsilon s} \max(s_l^{in}, s_l^{out})$ - мінімальне і достатня кількість переходів, які потрібно здійснити для покриття всіх переходів РКА (відповідно до теореми Ейлера).

Середня швидкість роботи.

Пропонується оцінювати швидкість роботи в секундах в залежності від подаються на вхід графів застосунків, що піддаються тестуванню.

Середнє забезпечується тестове покриття.

У загальному випадку можлива ситуація, коли пропонований алгоритм генерації не в змозі забезпечити повне тестове покриття через попадання в «нескінченний» цикл, або при попаданні в тупиковий стан. У цьому випадку даний алгоритм забезпечить покриття менше 100%. Пропонований критерій

усереднює забезпечуваємо алгоритмом покриття по різним РКА додаток, що подаються на вхід.

Можливість генерації кроків на перевірку результатів тестування.

Цей критерій дозволяє оцінити можливість задавати очікувані параметри станів РКА, що подаються на вхід алгоритму генерації. За заданим значенням алгоритм зможе згенерувати кроки, призначені для перевірки відповідності поточного стану програми очікуваному станом, відповідному заданим параметрам.

Цей критерій дозволяє оцінити можливість отримувати в результаті генерації тестові сценарії в необхідному форматі.

1.5 Висновки по першому розділу

Розглянуті техніки збільшення ефективності процесів генерації тестів. Однак розглянуті техніки відносяться до тестування складних інтегральних схем і використовують їх специфіку. Розглянуті формальні методи генерації тестів, але не представлені критерії оптимізації процесу генерації. Ефективність методу тестування ПМУ безпосередньо залежить від кількості ітерацій розробки, тому оптимізація методів тестування за часом зводиться до автоматизації процесів створення тестових сценаріїв і автоматизації їх виконання. Найбільшу ефективність імовірно надає метод тестування на основі прототипів, при цьому тестові сценарії для тестування ПМУ генеруються з побудованого прототипу. Необхідно мати можливість генерувати тестові сценарії таким чином, щоб використовувати їх в програмному методі автоматизації тестування з найменшими доопрацюваннями

2 МЕТОДИ ТЕСТУВАННЯ ПРОТОТИПІВ ЗАСТОСУНКІВ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

2.1 Взаємодія користувача з застосунком

В контексті тестування ЗМП на системному рівні, взаємодія з кінцевим користувачем і ЗМП можна розбити на наступні типи:

- взаємодія з елементом UI, який не передбачає введення даних. Надалі будемо називати такий тип - «детерміновану дію» (ДД). Безліч різних ДД безпосередньо залежить від різноманітності типів елементів інтерфейсу ЗМП і способів дії на них і є кінцевим числом;

- взаємодія з елементом UI, який передбачає введення даних (поля введення, форми та ін.). У загальному випадку користувач може ввести в додаток будь-який набір даних, це передбачає нескінченне число подібних дій введення даних. Будь-яка дія, пов'язане з введенням даних будемо позначати типом «варіаційне дію» (ВД);

- визначивши кінцевий набір детермінованих дій, і обмеживши нескінченний набір варіаційних дій кінцевим набором, можна звести взаємодія користувача з ЗМП до детермінованого, кінцевого набору операцій.

2.1.1 Визначення кінцевого набору детермінованих дій

У всіх мобільних операційних системах (Android, iOS, Symbian, Blackberry і ін.) Існує певний, кінцевий набір елементів призначеного для користувача інтерфейсу. Більшість типів елементів однаково для всіх ОС, тому можна задати загальний для всіх ОС, кінцевий набір детермінованих дій. Для кожної конкретної ОС даний набір може бути розширений з урахуванням елементів, що не увійшли в перетин елементів інтерфейсу всіх

ОС.

Такий набір дозволяє формально описувати взаємодії типу ДД в уніфікованому вигляді (в незалежності від типу ОС).

Очевидно, що існує деяке відображення елементів UI на дії, які можна зробити з даними елементами. Позначимо його action (ed). Це відображення повертає набір дій, які застосовні для даного елемента UI ed, який передбачає кінцевий, детермінований набір дій з собою.

Введене відображення буде використано пізніше при визначенні РКА для ЗМП.

2.1.2 Обмеження набору варіаційних дій

У загальному випадку користувач може ввести в додаток будь-який набір даних. Для обмеження даного числа наборів пропонується скористатися принципом еквівалентного роздроблення. Відповідно до цього принципу необхідно всі дані, які може ввести користувач в даний набір елементів UI, які передбачають введення даних розділити на кілька класів. На всіх даних із заданого класу еквівалентності додаток поводиться однаково. Позначимо $V = (e_{v1}, e_{v2}, \dots, e_{vl})$ - набір всіх варіаційних елементів UI, які бачить користувач в поточному стані додатки, ev позначає варіаційний елемент UI, який передбачає введення даних.

У загальному випадку число класів еквівалентності кінцеве число. Розглянемо приклад: припустимо, в деякому стані додатки існує набір варіаційних елементів інтерфейсу. Залежно від даних, які можна ввести отримуємо наступний набір класів еквівалентності:

Таблиця 2.1 – Набір класів еквівалентності

Тип елемента UI	Передбачуваний тип даних, що вводяться	Класи еквівалентності	Число класів еквівалентності	Контрольні значення
область введення	$N \in [1,2,\dots,10] \subset Z$	$N \in [1,\dots,10]$ $N \in Z - [1,\dots,10]$ $N \notin Z$ $N \neq$ число	5	5 0 5.5 “abcde”
область введення	S – будь-який рядок	S – любая_строка S≠ строка	2	“abcde” 0
список	Випадаючий список значень	в залежності від логіки роботи списку - кожен елемент списку або один (будь-який) елемент списку	1 або n - число елементів списку	один елемент або кожен елемент списку
флаг	відзначений / не відзначений	0 | 1	2	відзначений не відзначений
перемикач	відзначений / не відзначений	0 | 1	2	відзначений не відзначений

В даному випадку в ході тестування необхідно буде розглянути всі можливі комбінації контрольних значень, але в не залежності від набору

даних, що вводяться, користувач робить одну дію типу ВД. Задамо відображення, яке визначає набір контрольних значень для елемента e як eq(e) тоді всі набори даних, які можна ввести в даному стані застосунків, задаються добутком $\prod_{i=1}^{l} eq(e_1)$, де добуток береться за всіма варіаційним елементам (що передбачає ввід даних).

Варто зазначити, що:

- кожен елемент такого твору задає набір вхідних даних для застосування в даному стані;

- довжина кожного елемента твори дорівнює числу елементів ev в даному стані додатки;

- добуток покриває всі класи еквівалентності вхідних даних.

Позначимо дію, пов'язане з введенням одного елемента добутка як $input(\prod_{i=1}^{l} eq(e_1))$

Введені позначення будуть використані пізніше при визначенні РКА для ЗМП.

2.2 Розширений кінцевий автомат для тестування застосунків для мобільних пристроїв

ЗМП розробляються з використанням принципу відділення логіки роботи додатки та подання (призначеного для користувача інтерфейсу). Особливості даного принципу і розглянута метрика тестування ЗМП дозволяють використовувати кінцеві автомати для формального опису прототипів ЗМП. При аналізі існуючих ЗМП було виявлено, що кількість основних видів застосунків - кінцеве невелике число (не більше 100), але перехід в кожен вид може залежати від вводяться кінцевим користувачем даних. Виникає проблема «розмноження» однакових видів при введенні «Однотипних» даних. Дійсно, припустимо в даному виді існує деяка форма введення, яка приймає будь-які строкові значення на вхід і після відправки даних «в додаток», відбувається перехід до наступного вигляду, в якому

відображаються введені дані. При побудові відповідного КА отримуємо нескінченне число станів.

Розмноження числа станів представлено на рисунку 2.2.

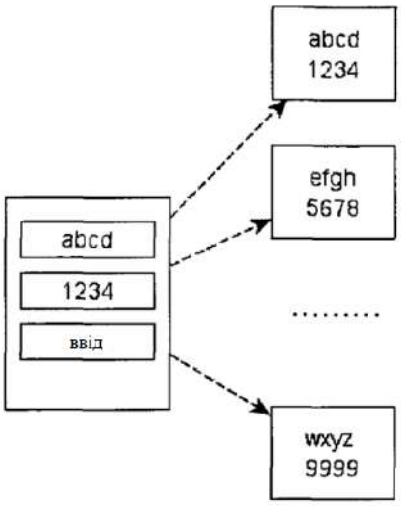

Рисунок 2.2 – Розмноження числа станів КА

Вирішення цієї проблеми полягає в розбитті всіх даних, що вводяться на класи еквівалентності, це обмежить число «кінцевих» станів. Друга проблема, яка виникає при побудові КА для ЗМП - залежність «майбутніх» переходів від даних, що вводяться в «поточному» переході. Наприклад, в разі існування рольової моделі в застосунку, перехід до деяких видів може бути «закритий» для користувачів одних ролей і відкритий для користувачів інших ролей (рисунок 2.3).

Рисунок 2.3 – Приклад умови на переході КА

При побудові КА такого застосунку виникає необхідність «запам'ятовувати» під яким користувачем ми увійшли в систему і «перевіряти» запам'ятоване значення при кожній спробі перейти до стану, де потрібно мати визначені права. Вирішення цієї проблеми полягає у використанні розширеного кінцевого автомата для побудови прототипу застосунку. такий спосіб завдання КА дозволить «запам'ятовувати» необхідні дані у вигляді параметрів РКА і перевіряти значення цих параметрів при переході до «закритих» сторінок, попередньо поставивши умови на переходах. «Розширена звичайно-автоматна модель - це вдосконалена модель кінцевого автомата. У традиційному кінцевому автоматі перехід з стану в стан пов'язаний з набором вхідних булевих умов і набором вихідних булевих функцій. У розширеній моделі перехід може бути виражений "IF" виразом. Якщо всі умови переходу дотримані, то відбувається перехід, переводячи автомат в наступний стан, при цьому виробляються необхідні операції з даними. формально: «Розширений кінцевий автомат -набір (S, V, P, sO, PB, I, nl, X, O, nO, Y, T), де

S – кінцева множина станів автомата;

W – множина, можливо нескінченна, значень внутрішніх даних автомата;

P – відображення кінцевого набору [1..n] індексів в W, P: [1..n] -> W; значення P на індексі і називається значенням і-ой змінної автомата, яке також позначається pi.

So – елемент S, званий початковим станом;

P0 – відображення [1..n] індексів в W, зване початковими значеннями змінних;

I – кінцева множина, елементи якої називаються операціями або стимулами, саме I називають вхідним алфавітом автомата;

ni – відображення I в невід'ємних числах, визначає число параметрів для кожного стимулу;

X – множина, можливо нескінченна, значень параметрів стимулів;

O – кінцева множина, елементи якої називаються реакціями, саме O називають вихідним алфавітом автомата;

nO – відображення O в невід'ємних числах, визначає число параметрів або даних кожної реакції;

Y – множина, можливо нескінченна, значень даних реакцій;

T – множина переходів автомата; кожен перехід t включає початковий керуючий стан Si, стимул I, умова переходу gt (guard condition) - предикат на множині Vn ×Xni в множині Vn, визначаючи нові значення змінних.

Виконання розширеного автомата відрізняється від виконання звичайного тим, що крім поточного стану є поточні значення змінних, при подачі стимулу з набором аргументів охоронна умова визначає, чи може бути виконаний даний перехід при поточному наборі значень змінних і заданих значеннях параметрів стимулу. Здійснюємий перехід обирається детерміновано з усіх, позначених даними стимулом, що починаються в даному керуючому стані і мають виконане охоронну умову. При виконанні деякого переходу новий керуючий стан автомата дорівнює кінцевому керуючому стану переходу, нові значення змінних визначаються за допомогою його дії – нове pi=at(p1 ,…,pn, xl,…,xnl), значення параметрів реакції - за відповідним відображенням в переході. Розширений кінцевий автомат зображений на рисунку 2.4

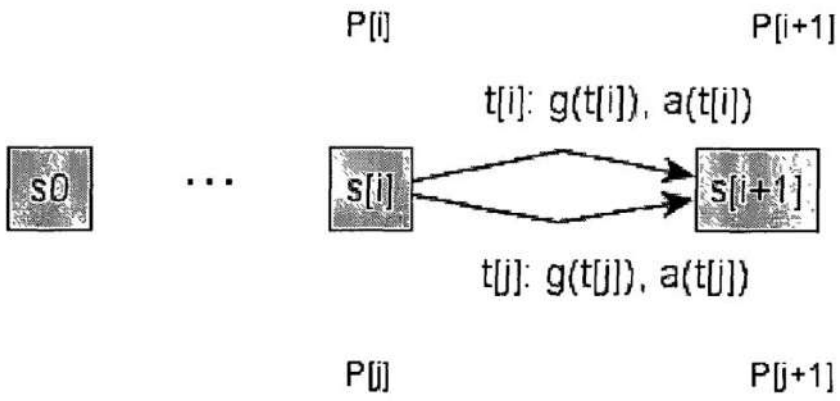

Рисунок 2.4 – Розширений кінцевий автомат

2.2.1 Визначення станів РКА

Щоб побудувати РКА для даного ЗМП, потрібно ввести правила побудови прототипу майбутньої програми, які полягають в декомпозиції призначеного для користувача інтерфейсу майбутнього програми, як показано на рисунку 2.5

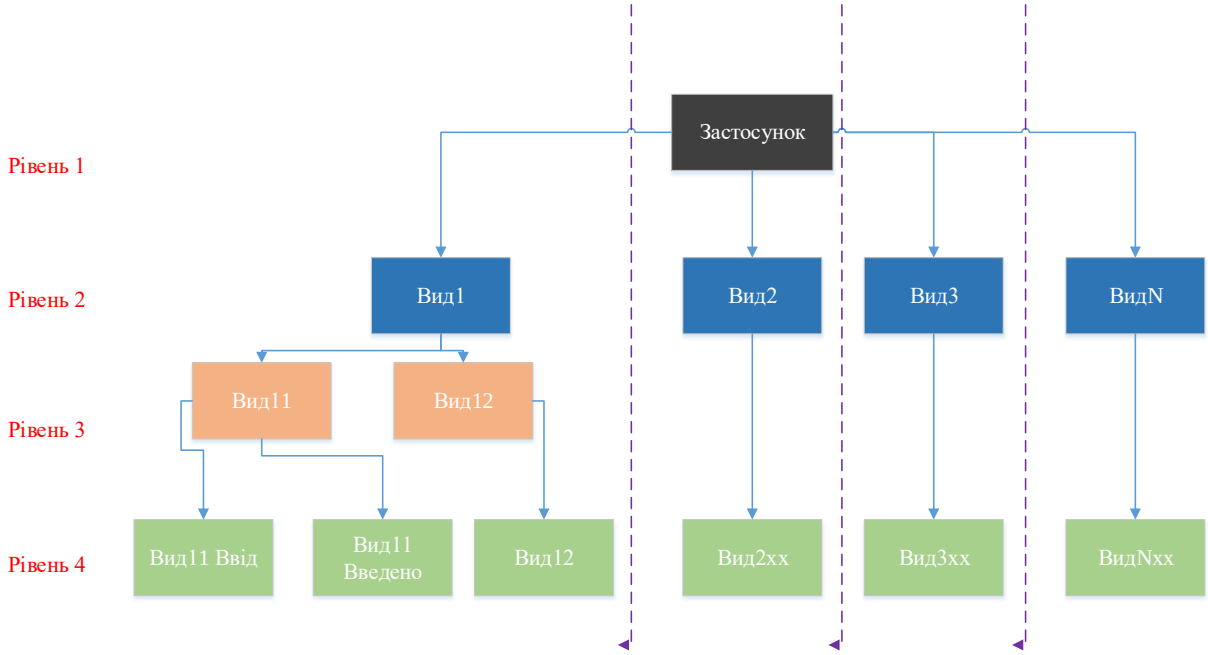

Рисунок 2.5 – Декомпозиція призначеного для користувача інтерфейсу застосунку

Рівні:

а) перший рівень - рівень програми;

б) другий рівень - рівень видів. Додаток розбивається на види, які відповідають його різним станам призначеного для користувача інтерфейсу. Перехід між видами здійснюється за допомогою команд користувача;

в) третій рівень - рівень підвидів. У кожного виду можуть бути підвиди. Підвид відповідає будь-якому спливаючому діалогу застосунку в цьому ж вигляді;

г) четвертий рівень - рівень РКА. На цьому рівні види або підвиди

попередніх рівнів, в яких можливе введення даних розбиваються на 2 стану - ВидВвод і ВіидВведено. ВидВвод відповідає стану автомата, коли введення даних ще не був здійсненим. ВідВведено відповідає стану автомата, в якому користувач уже ввів деякі дані. Переходи між ВидВвод і ВидВведено відповідають всім діям типу ДВ.

Таким чином, множина S РКА - це множина елементів 4-го рівня застосунків. $S = \bigcup_{i=1...N} v_i^4$, де v_i^4 - деякий вид 4-го рівня. Об'єднання береться за всіма видами 4-го рівня, отриманим в результаті декомпозиції. Надалі будуть розглядатися тільки види 4-го рівня, і позначення рівня буде опускатися $v_i^4 = v_i$. Введемо позначення для набору детермінованих елементів інтерфейсу даного виду Di, а для набору варіаційних елементів даного виду v_i .

2.2.2 Визначення безлічі стимулів і значень параметрів стимулів РКА

Для того щоб автомат перейшов в наступний стан, необхідно подати йому на вхід стимул у вигляді дії користувача. Нехай d_{ij} – набір контрольних значень для варіаційного елемента $e_i \in v_i$ тоді набір можливих дій для даного стану визначиться наступним виразом:

$$I_i = \bigcup_{e \in D_i} action(e) \cup \bigcup input(\prod_{e \in v_i} eq(e))$$

Дана множина визначає набір можливих дій, які може зробити користувач, перебуваючи в даному виді програми. Будь-яка дія користувача є або впливом на варіаційний елемент UI (введенням даних відповідно до класів еквівалентності), або впливом на детермінований елемент UI одним з відповідних даному елементу дій.

Загальний набір можливих дій визначається об'єднанням таких множин за всіма видами прототипу:

$I_i = \cup_{i=1}^{N}[\cup_{e \in D_i} action(e) \cup \cup input(\prod_{e \in v_i} eq(e))]$,

де і задає уссі види 4-го рівня

ni=1 кількість параметрів для кожної дії типа ДД або ДВ.

X – Множина значень параметрів кожної дії типа ДД або ДВ.

2.2.3 Визначення безлічі реакцій і параметрів реакцій РКА

Реакція на кожне вплив визначає зміни в застосунку в результаті вчинення дій. Можливо 2 типу реакції:

В результаті дії типу ВД автомат переходить з зі стану ВідВвод в стан ВідВведено. Розглянутий тип дій не перевіряється на коректність в генеруються тестах, оскільки відповідає процесу введення даних в додаток.

В результаті дії типу ДД автомат переходить в наступний стан, відповідне наступного вигляду. Цей вид має набір елементів призначеного для користувача інтерфейсу. За змістом набору елементів вид може бути однозначно ідентифікований щодо інших видів (не втрачаючи спільність вважаємо, що в застосунку не існує двох різних видів з однаковим набором елементів UI). Використовуючи ці набори елементів можна перевірити коректність переходу, тобто задати перевірочні умови в генерованому тесті. Таким чином, вихідним алфавітом РКА в даному виді є деякий набір елементів UI, однозначно визначають «наступний» вигляд програми (наступне стан РКА):

$O_i = D_i \cup V_i$- повний набір елементів UI «наступного» виду. тоді весь вихідний алфавіт задається всіма елементами UI програми:

$O = \cup_{i=1}^{N}(D_i \cup V_i)$, де і задає усі види 4-го рівня.

no=2 для кожного елемента UI слід задати тип і його зміст

(Наприклад, текст на елементі). Це дозволить задавати перевірки коректності скоєних переходів під час генерації тестів. Y - безліч значень задається типами всіх елементів UI і їх змістом.

2.2.4 Визначення параметрів РКА

Для кожного елемента, який передбачає введення даних, слід задати відповідний параметр РКА, який дозволить «запам'ятати» введене значення для дальнейшік перевірок умов на переходах. Кожному вариационному елементу ставимо у відповідність параметр: $\forall e_i \in \longrightarrow p_{ij}$• Безліч значень параметрів визначається об'єднанням наборів контрольних значень за всіма видами програми:

$W = \mathrm{U}_{i-1\ldots N}\left[\mathrm{U}_{e\in V_i}\, eq(e)\right]$, де і задає всі види 4-го рівня.

$\forall e_i \in [1..N], \forall e_i \in V_i \langle p_{ij} \longrightarrow eq(e) \rangle$ - задає значення параметра з даного набору контрольних значень.

2.2.5 Визначення початкового стану РКА

Введемо фіктивне початковий стан s0. Початкові значення параметрівв цьому стані встановлюються нульовими: $p_{Xj} = 0$.

2.2.6 Визначення безлічі переходів РКА

Безліч переходів автомата задається структурою користувальницького інтерфейсу застосунку. Перехід з одного стану в інший відповідає деякому одиничного взаємодії користувача з застосунком.

2.2.7 Формальне визначення РКА

В результаті побудови отримуємо наступне формальне визначення РКА для тестування ДМ

$$PKA = \left\langle \begin{array}{l} S = \bigcup_{i=1..N} V_i^4, W = \bigcup_{i=1..N} \left\lceil \bigcup_{e \in V_i} eq(e) \right\rceil, \\ \forall i \in [1..N], \forall e \in V_i \left\langle p_{ij} \xrightarrow{P} eq(e) \right\rangle, \\ S_0, \forall j : p_{1j} = \varnothing, \\ I = \bigcup_{i=1}^{N} \left\lceil \bigcup_{e \in D_i} action(e) \cup \bigcup input(\prod_{e \in V_i} eq(e)) \right\rceil, \\ n_I \in [1,2], O = \bigcup_{i=1}^{N} (D_i \cup V_i), n_O = 2 \end{array} \right\rangle$$

Рисунок 2.5 – Приклад РКА частини програми із заданими параметрами

2.3 Метод тестування застосунків для мобільних пристроїв

Запропонований спосіб представлення програми у вигляді РКА лежить в основі запропонованого методу тестування ПМУ.

Крок 1. Побудувати прототип застосунку на етапі проектування за такими правилами:

а) провести декомпозицію призначеного для користувача інтерфейсу майбутнього програми;

б) внести в прототип інформацію про детермінованих і варіаційних діях користувача, використовуючи функції інструменту прототипування.

Крок 2. Автоматично отримати РКА прототипу, використовуючи інструмент конвертації прототипу в відповідний РКА.

Крок 3. Реалізувати вхідний і вихідний алфавіти РКА на програмному рівні за допомогою інструменту автотестування.

Крок 4. Згенерувати тестові сценарії, з використанням програмних засобів генерації шляхом обходу графа отриманого РКА.

Крок 5. Провести тестування з використанням програмної реалізації

вхідного і вихідного алфавітів РКА на тестованому ЗМП по автоматично згенерували тестів.

Особливість даного методу полягає в використанні коштів генерації тестових сценаріїв на базі РКА прототипу. Слід зазначити, що РКА будуватися по прототипу, тому що згенерували тестові сценарії дозволяють перевіряти функціонал розроблюваного продукту, який розробляється також як і сценарії, на основі побудованого прототипу.

2.4 Висновки по другому розділу

Запропонований метод дозволяє звести процес написання ручних і автоматичних тестів для ЗМП до їх генерації, мінімізувати трудовитрати тестування.

.

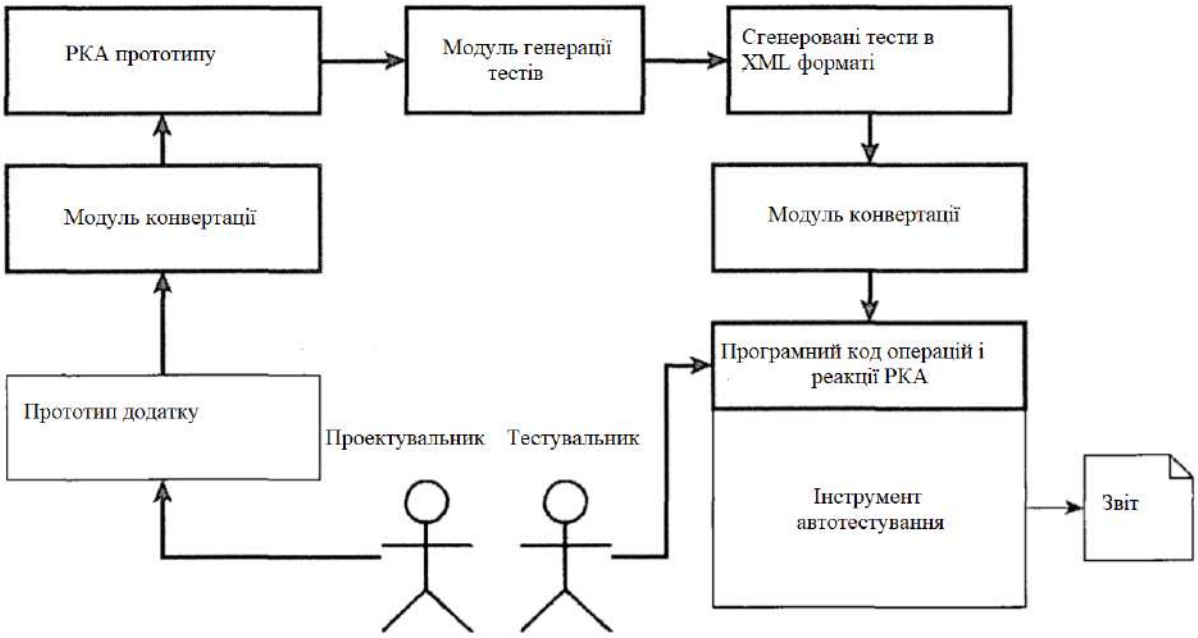

Рисунок 2.6 – Загальна схема запропонованого методу тестування

Акцент роботи відділу тестування, таким чином, зміщується в область підтримки та оптимізації інструменту генерації тестів і програмного інтерфейсу букв алфавіту тестування. У 4 розділі буде показано, що подібний

підхід дозволяє значно оптимізувати процес розробки і тестування ЗМП по тимчасовому критерію. Розроблений же в 3 алгоритм генерації дозволяє досягти повного тестового покриття ЗМП.

Ідентифікатор дії	Тип елемента користувацького інтерфейсу	Дія	Опис
1	кнопка, активна область на екрані	press (identificator)	натискання на кнопку (активну область)
2	кнопка, активна область на екрані	long_press(identificator)	довге натискання для виклику контекстного меню
3	випадаючий список елементів	select (identificator)	вибір елемента в списку, що випадає
4	прапор	check (identificator,Off)	перемикання прапора в стан 0 | 1
5	перемикач	check (identificator,Off)	перемикання перемикача в стан 0 | 1
6	елемент введення даних	input (identificator,data)	ввести в даний елемент потрібні дані
7	вигляд програми	swipe_left	перенос екрану вліво
8	вигляд програми	swipe_right	перенос екрану вправо
9	вигляд програми	scroll_up_to(identificator)	перемщення вгору з використанням скролінгу елемента - element
10	вигляд програми	scroll_down_to(identificator)	перемщення вниз, до елемента - element
11	вигляд програми	zoom (value)	зміна масштабу
12	Регулятор гучності	volume_up	збільшення гучності
13	регулятор гучності	volume_down	зменшення гучності
14	кнопка живлення	Power	натискання на кнопку відключення живлення
15	кнопка живлення	long_power	довге натискання на кнопку відключення живлення
16	роз'єм даних	cable (inout)	підключення /відключення кабелю даних або живлення
17	вигляд програми	element_exists (type, text)	перевірка наявності елемента даного типу з даним текстом на екрані (включаючи невидиму область)
18	вигляд програми	element_present (type, text)	перевірка наявності елемента даного типу з даним текстом у видимій частині екрана
19	вигляд програми	text_exists (text)	перевірка наявності тексту на екрані (включаючи невидиму область)
20	вигляд програми	text_present (text)	перевірка наявності тексту у видимій частині екрана

Рисунок 2.7 – Набір детермінованих дій користувача

На рисунку 2.6 приведена загальна схема запропонованого методу тестування. На представленій схемі відзначені запропоновані нововведення, що відображають особливості даного методу.

3 АВТОМАТИЧНА ГЕНЕРАЦІЯ ТЕСТІВ

Відповідно до обраного критерію повноти тестування ЗМП, для заданого прототипу ЗМП у вигляді розширеного кінцевого автомата, потрібно розглянути всі можливі переходи з одного стану в інший. Відомо, що кінцеві автомати зручно представляти у вигляді графів. У графові стан КА - це вузли графа, а переходи між станами - це гілки графа. Завдання генерації тестового набору зводиться до задачі обходу графа. При цьому, необхідно пройти по всіх гілках графа, вибравши найкоротший шлях. Таким чином, буде згенеровано мінімальний набір тестових сценаріїв, що забезпечує повне тестове покриття.

Завдання генерації тестових сценаріїв досліджена, але пропоновані підходи не враховують специфіки ЗМП. Описується підхід генерації тестових сценаріїв для користувача інтерфейсу. Однак згенеровані таким чином тести не відповідають критеріям повноти тестування.Пропонується підхід генерації тестових сценаріїв для WEB застосунків. Даний підхід використовує специфіку WEB застосунків, а саме, їх клієнт-серверну архітектуру і існування сесій користувач - web додаток. Даний підхід не може бути застосовний для генерації тестів ЗМП. Розглянуті формальні методи генерації тестових сценаріїв, одним з яких є «T-method», в рамках якого розглянуто задачу «Китайського листоноші». Існують різні рішення даної задачі в окремих випадках. Наприклад, пропонується підхід до вирішення даного завдання з використанням пароз’єднання на графах.

Згідно з теоремою Ейлера в графі існує ейлерів цикл тоді і тільки тоді, коли він сильно-пов'язаний і для кожної вершини графа її напівстепінь заходу дорівнює її напівстепіні результату. Тому вирішити задачу китайського листоноші для довільного графа неможливо. Необхідно якомога «ближче» підійти до її вирішення з точки зору тестування ЗМП, а саме: обійти всі гілки графа з мінімальною кількістю повторень кожної гілки.

Завдання обходу графа також добре досліджена, існує безліч алгоритмів, що реалізують обхід графа за тим чи іншим правилом. Для вирішення поставленого завдань, доцільно використовувати алгоритми пошуку мінімального шляху від початкової точки до кінцевої. Основна складність полягає в тому, що поставлена задача не визначає початку і кінця обходу. Обхід потрібно виробляти до тих пір, поки не будуть пройдені всі гілки графа. Існують різні алгоритми пошуку найкоротшого шляху. Модифікація одного з їх дозволить вирішити поставлене завдання. Алгоритм Дейкстри дозволяє знаходити найкоротші відстані від однієї з вершин графа до всіх інших. Даний алгоритм не підходить для вирішення поставленого завдання, оскільки в разі розширеного кінцевого автомата, ваги гілок можуть динамічно змінюватися в залежності від поточного контексту змінних КА. Аналогічна ситуація виникає з алгоритмом Беллмана-Форда (30).

Алгоритм Флойда-Уоршелла знаходить найкоротші відстані між усіма вершинами графа, його застосування є недоцільним, оскільки граф РКА в загальному випадку динамічно змінюється після кожного наступного переходу, і перерахунок шляхів потрібно буде робити після кожного переходу.

3.1 Аналітична модель алгоритму генерації

Алгоритм A* є алгоритмом пошуку шляху з найменшою вартістю. Алгоритм є жадібним алгоритмом, який працює за першим збігом. Крім цього алгоритм використовує евристичну функцію для визначення шляху. Існують модифікації, що дозволяють здійснювати обхід графа, до тих пір, поки не будуть покриті всі переходи. У даній роботі пропонується модифікація A*, яка містить інформацію про параметри графа і може вибирати маршрут з урахуванням умовних переходів кінцевого автомата.

Функція вартості F є основоположною для роботи алгоритму. Перебуваючи в деякому вузлі графа, ця функція визначає - який перехід

вибрати наступним. Вибирається той перехід, для якого значення F найменше.

Функція вартості F є основоположною для роботи алгоритму. Перебуваючи в деякому вузлі графа, ця функція визначає - який перехід вибрати наступним. Вибирається той перехід, для якого значення F найменше. Вивчивши задачу про знаходження Ейлерова шляху на графі, в роботі пропонується використовувати наступний спосіб для вибору «наступного» переходу, перебуваючи в «поточному» стані. Пропонується уявлення

$$F = G + H \tag{3.1}$$

де G - вартість шляху до наступного стану з початкового. H - евристична функція, яка оцінює довжину шляху до «кінцевого» стану:

G - сумарна вага всіх покритих на даний момент переходів. Ця вага збільшується під час роботи алгоритму.

H - сумарна вартість шляху до найближчого непокритого переходу. Шлях до найближчого непокритого переходу шукається за допомогою пошуку в ширину (алгоритм BFS).

При цьому перед кожним розрахунком G і H ваги всіх переходів перераховуються з урахуванням умови «наступного» переходу. Переходи, які стають недоступними через умове, отримують нескінченну вартість. Формально, нехай:

T = [t_1, t_2, ..., t_n] - масив всіх переходів РКА.

C =[t_i,| t_i $-$ покритий] - масив всіх покритих на даний момент переходів. Масив змінюється в процесі роботи алгоритму.

W = [$w_1, w_2, \ldots, w_n$] - масив динамічних ваг, що змінюються в процесі роботи алгоритму. Ваги змінюються відповідно до параметрів РКА.

$$c_1 = \begin{cases} 1, t_i \in C \\ 0, t_i \notin C \end{cases} \quad (3.2)$$

P_i – шлях з і-го стану до найближчого непокритого переходу.

N – коефіцієнт, що враховує пріоритет умов: спочатку вибираються непокриті переходи, потім переходи, найближчі до непокритим переходів.

Тоді функція вартості алгоритму A * задається наступною формулою:

$$F_i = \left(\left||C|\right| + c_i\right) * N + \sum_{P_i} w_t \quad (3.3)$$

3.1.1 Схема аналітичної моделі алгоритму

Розглянемо загальну схему роботи алгоритму пошуку шляху для покриття всіх переходів КА (рисунок 3.1).

3.1.2 Пошук значення F

$$F_i = \left(\left||C|\right| + c_i\right) * N + \sum_{P_i} w_t \quad (3.4)$$

Це основна формула. Для пошуку значення F_i. для i переходу необхідно знайти значення суми $\sum_{P_i} w_t$ - сума ваг переходів до найближчого непокритого переходу. Для знаходження даної суми використовується окрема процедура, яка використовує жадібний алгоритм пошуку по першому збігу - «Пошук в ширину» (BFS). Схема роботи процедури представлена на рисунку 3.2.

3.1.3 Додаткові умови виходу з циклу

Існує ряд умов, при яких відбувається передчасний вихід з циклу, з поверненням помилки AppErrог: «Неможливо знайти перехід з даного стану

прототипу в наступний стан, з поточними значеннями змінних, внаслідок некоректних значень параметрів прототипу».

Для всіх послідовників даного переходу, які залишилися, знайдені шляхи рівні між собою і містять всі переходи прототипу. Це означає що для будь-якого послідовника алгоритм пройшов весь прототип з урахуванням дій і умов переходів і не знайшов жодного непокритого переходу.

Не знайдено жодного шлях для всіх послідовників поточного переходу.

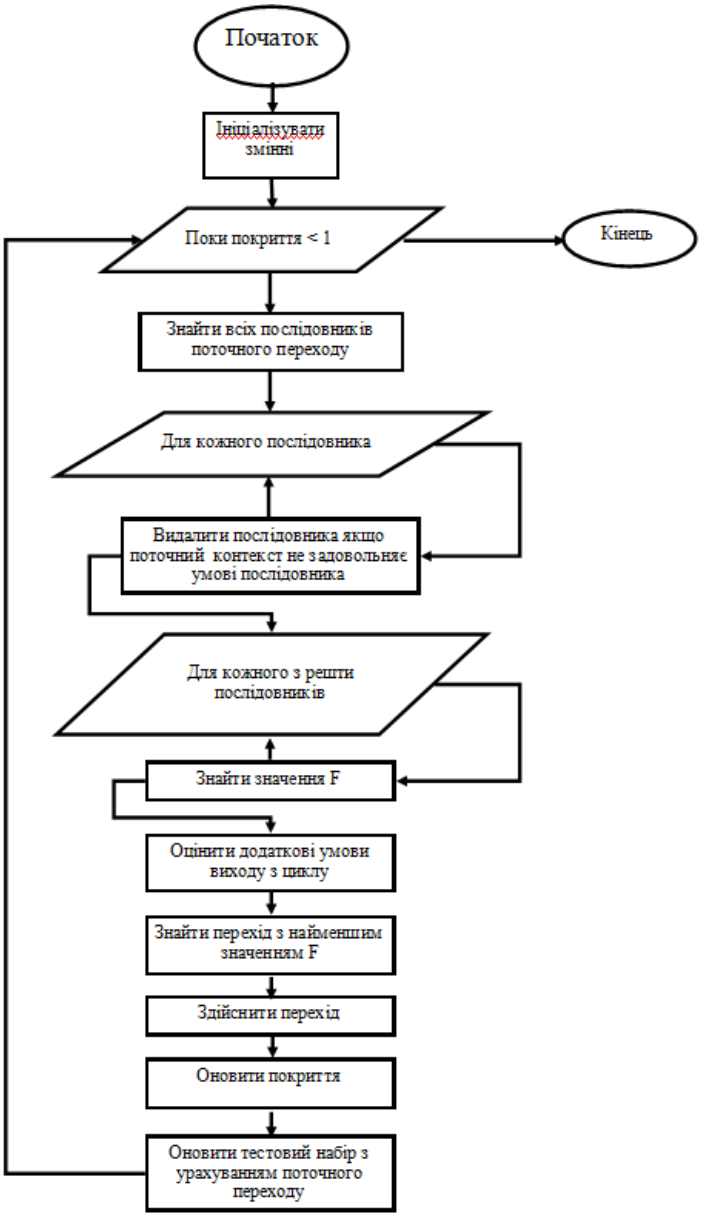

Рисунок 3.1 – Схема генерації тестових сценаріїв

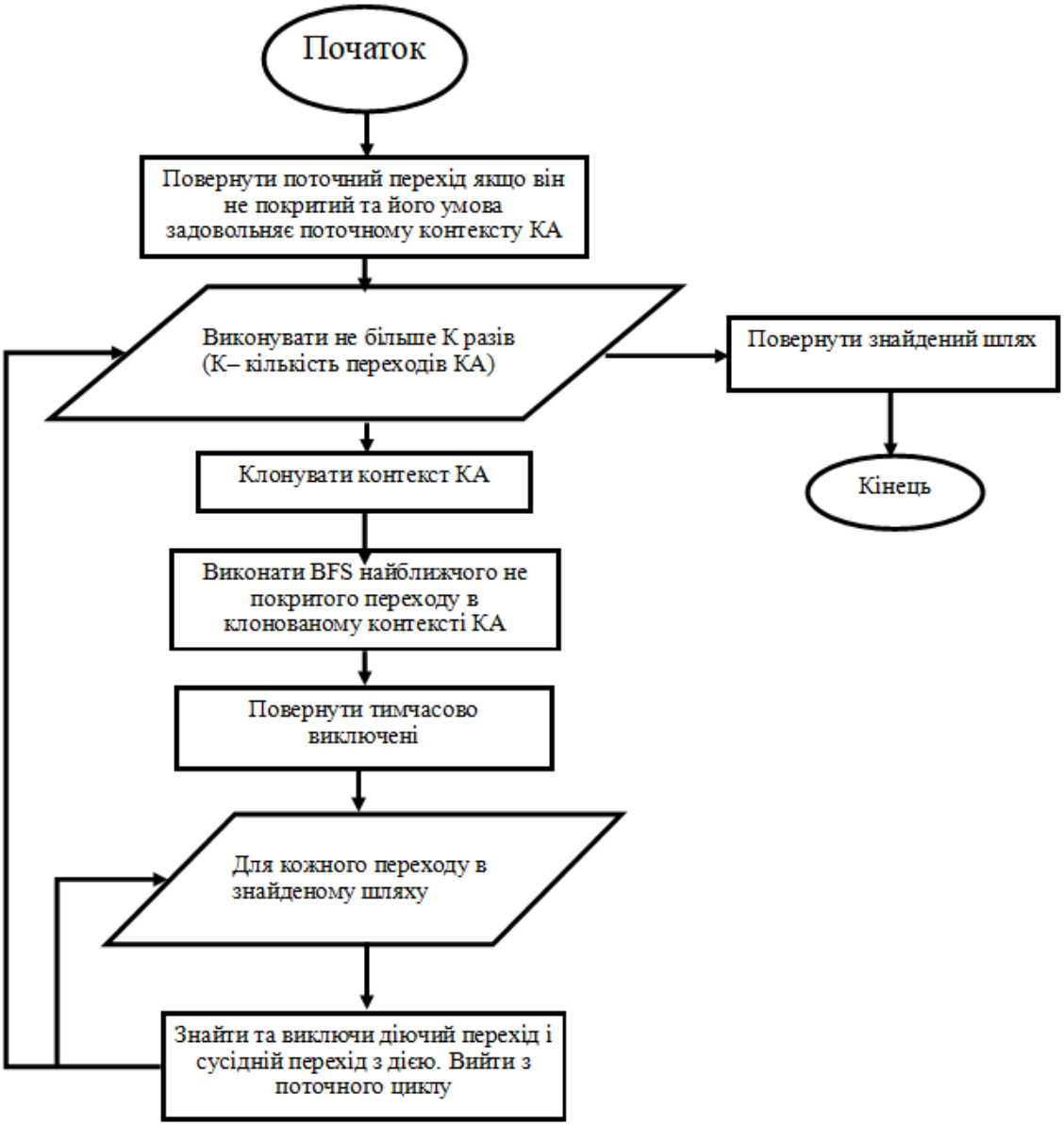

Рисунок 3.2 – Пошук найближчого не покритого переходу

3.2 Програмна модель алгоритму генерації

Пропонується розробити програмну модель алгоритму генерації тестів (ПМАГТ), в основі якої лежить запропонована аналітична модель алгоритму.

3.2.1 Загальна схема програмної моделі

ПМАГТ являє собою програмний продукт, написаний на об'єктно орієнтованої мові високого рівня. Програма запускається з командного рядка,

приймає на вхід різні параметри, є платформо-незалежною. Програма може бути інтегрована з сервером, що забезпечує візуалізацію роботи програмної моделі, яка складається з декількох модулів, пов'язаних між собою. Модульна структура ПМАГТ дозволяє розширювати його додаванням нових модулів. Також модульна структура покращує «читаність» програмного коду продукту.

Алгоритм роботи ПМАГТ:

- конвертація прототипу застосунку в XML прототип, структура якого зрозуміла ПМАГТ;
- запуск алгоритму генерації тестових сценаріїв;
- читання вхідних параметрів і завантаження прототипу в пам'ять (перетворення прототипу в РКА на програмному рівні);
- робота алгоритму обходу графа РКА і генерація тестових сценаріїв відповідно до вимог вхідних параметрів і параметрів РКА;
- вивантаження тестових сценаріїв в XML поданні з використанням алфавіту тестування мобільних застосунок.

Схема ПМАГТ із зображенням взаємодії модулів зображена на рисунку 3.3.

3.2.2 Вхідні і вихідні дані

На вхід ПМАГТ можна подавати прототипи призначених для користувача застосунків, реалізованих за допомогою спеціальних засобів прототипування. Програма приймає на вхід XML певної структури. Робота з приведення прототипів, одержуваних на виході різних інструментів прототипування, до XML прототипу «зрозуміле» для ПМАГТ, «лягає» на модулі конвертації прототипів. На момент написання даної роботи ПМАГТ володіє двома конвертерами - Yed → XML, AxureRP → XML.

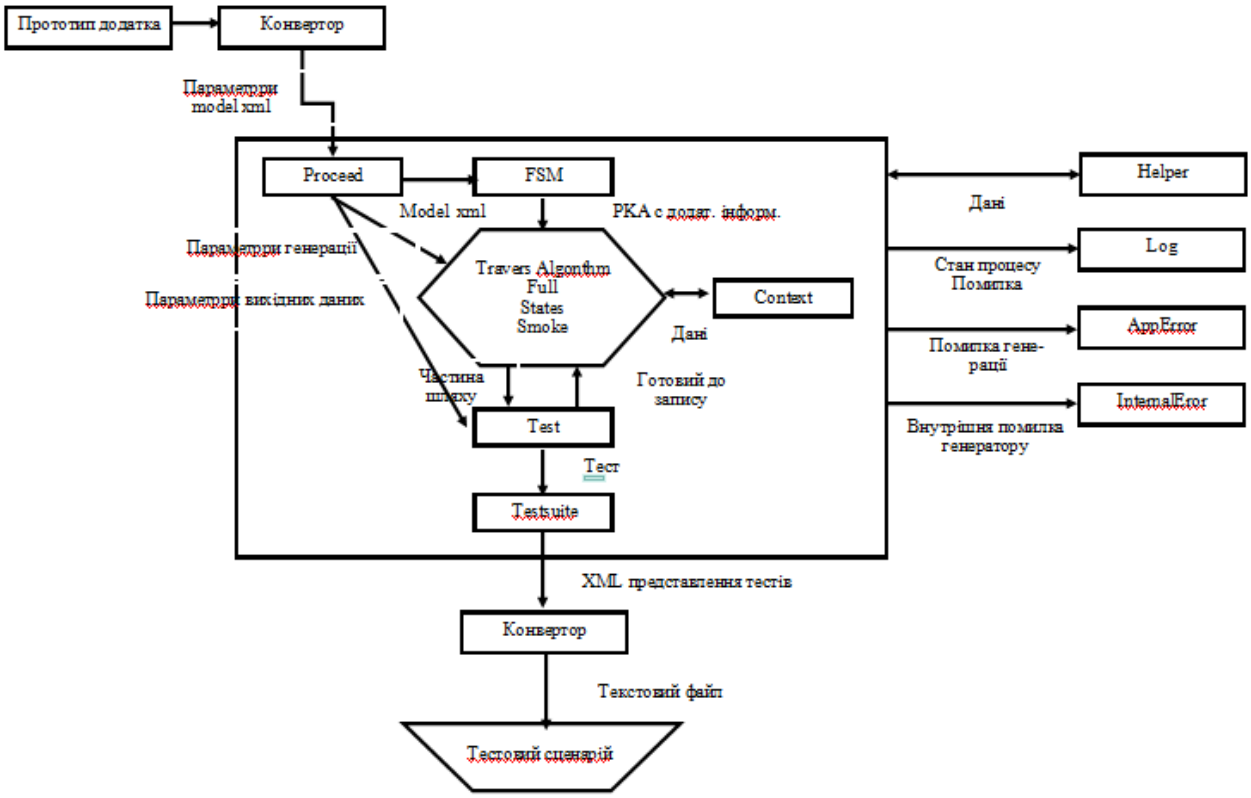

Рисунок 3.3 – Схема ПМАГТ

3.2.3 Вхідні параметри

На вхід ПМАГТ можливо подати параметри, контролюючі його роботу (рисунок 3.4).

3.2.4 Опис найбільш важливих вхідних параметрів

Параметр s – будь-який рядок, який відповідає назві початкового стану РКА.

Якщо наданого в секції стану не існує, ПМАГТ виведе відповідну помилку. Значення за замовчуванням «Start».

Параметр t – тип алгоритму генерації. Підтримуються наступні типи створення:

Full - генерація відбувається до тих пір, поки не будуть покриті всі

переходи між станами. Оскільки граф РКА для призначеного для користувача додатки є зв'язковим (щоб користувач потрапив в якийст стан застосунку, потрібно ввести певний набір команд), то за такої умови генерації покриваються не тільки всі переходи, а й їхні капітали РКА.

Параметр		Можливі значення [значення за замовчуванням]	Короткий опис
-h		Параметр присутній, [відсутній]	Показати коротку інструкцію по використанню ПМАГТ
-d		Параметр присутній, [відсутній]	Ввімкнути режим DEBUG
-f		/шлях/к/module.xml, [шлях/за/замовчуванням/module.xml]	Шлях до прототипу
-s		Назва початкового стану, [start]	Показати стан РКА, з якого потрібно починати генерацію
-e		Параметр присутній, [відсутній]	Генерація в розширеному режимі
-l		Назва лог файлу[/generator.log]	Повний шлях до файлу логування
-t		[Full], states, smoke	Тип генерації
-output		[txt], csv, robotium, selenium	Тип вихідного тестового набору
-delay		Час затримки, [0]	Час затримки при переході з одного стану РКА в інший
-filter		[dot], circo, twopi, neato	Тип візуального відображення РКА
-visualize		Параметр присутній, [відсутній]	Ввімкнути візуалізацію під час генерації
-servermode		Параметр присутній, [відсутній]	Ввімкнути роботу в серверному режимі

Рисунок 3.3 – Вхідні параметри

States - генерація відбувається до тих пір, поки не будуть покриті всі стани РКА. Цей спосіб генерації не забезпечує покриття всіх переходів між станами. Тому, в результаті роботи, будуть покриті всі стани програми, але не всі команди, що призводять додаток в даний стан.

Smoke - генерація відбувається до тих пір, поки не будуть покриті всі основні стани програми. У будь-якому ПП можна виділити основні і допоміжні стани. У даній роботі, РКА володіють додатковими властивостями, які зберігають інформацію, корисну для тестування. У тому

числі і інформацію про основні стани застосунку. Подібний спосіб генерації дозволяє згенерувати тестовий набір з інформацією про передбачуваний результатах виконання тесту.

Delay - час затримки між послідовними переходами РКА з одного стану в інший в процесі генерації тестових сценаріїв. Збільшення цього часу дозволяє візуалізувати процес генерації і поспостерігати за роботою ПМАГТ.

Filter - для візуалізації процесу генерації використовується утиліта graphviz. Цю утиліту потрібно передати на вхід набір станів РКА і переходів між ними, а на виході можна отримати його графічне представлення. Утиліта підтримує різні фільтри для графічного представлення - dot, circo, twopi, neato, fdp, sfdp. Даний ключ дозволяє задати певний фільтр для візуалізацій процесу генерації. Значення за замовчуванням «dot».

Visualize - ключ дозволяє включити візуалізацію процесу генерації

Servermode - ключ, що запускає модель алгоритму в режимі сервера. В цьому режимі модель алгоритму працює як сервер і «спілкується» з клієнтською частиною, яка реалізує візуалізацію процесу генерації.

3.2.5 Вибір мови програмування для реалізації ПМАГТ

Дотримуючись методології організації процесів тестування, викладеної в першому розділі, можна припустити, що процес генерації тестових сценаріїв не відбуватиметься часто в порівнянні з іншими процесами, описаними в методології. Процес генерації не є лімітуючою стадією. Таким чином, час генерації тестових сценаріїв не є критичним. Тому для реалізації ПМАГТ потрібно вибрати найбільш високорівневу мову програмування, яка володіє об'єктною орієнтацією та зручним інтерфейсом для роботи з різними XML структурами.

Також необхідно вибрати інтерпретовану мову, що підтримує метапрограмування, оскільки в процесі генерації виникає необхідність динамічно перевіряти умови переходів і програмно «запам'ятовувати»

поточний стан кінцевого автомата. Однією з таких мов є ruby.

3.2.6 Програмне уявлення розширеного кінцевого автомата прототипу

Прототип описує тестований додаток. Прототип можна представити у вигляді кінцевого автомата, використовуючи мову XML (35). Прототип складається з станів, які може приймати автомат і переходів між цими станами. Кожне стан і кожен перехід може мати набір додаткових властивостей, значення яких використовуються при генерації тестових сценаріїв.

Атрибут стану	Опис атрибуту	Обов'язковий атрибут?
Id	Унікальний ідентифікатор стану всередині даного прототипу	Так
Name	Назва стану, відображається в генеруючих тестових сценаріях	Так
Description	Опис стану, відображається в генеруючих тестових сценаріях	Ні
Main	Прапор, вказує - чи є стан головним. Цей прапор використовується при генерації тестових сценаріїв для закінчення даного тесту і початку нового тесту	Так
Inputs	Атрибут, що забезпечує можливість задати класи еквівалентності вхідних даних.	Ні
elements	Атрибут, що задає набір UI елементів даного стану. Кожен елемент має тип і зміст	Так

Рисунок 3.4 – Стану кінцевого автомата прототипу

Атрибут стану	Опис атрибуту	Обов'язковий атрибут?
Id	Унікальний ідентифікатор переходу всередині даного прототипу	Так
Name	Назва переходу, відображається в генеруючих тестових сценаріях	Так
Source	Стан-джерело для даного переходу	Так
Target	Стан-приймач для даного переходу	Так
Action	Дія переходу. Задає значення параметрів розширеного кінцевого автомата прототипу	Ні
Condition	Умова переходу. Логічна умова, яка визначає - чи можливий перехід при даних значеннях параметрів розширеного кінцевого автомата	Ні
Chance	Імовірність переходу. Використовується для виділення найбільш ймовірних переходів, тим самим спрямовуючи алгоритм генерації в ту чи іншу сторону	Ні
Weight	Вага переходу. використовується алгоритмом генерації для зважування всіх переходів динамічно, після кожного кроку, в залежності від поточного значення змінних і умови даного переходу	Ні
type	Тип переходу. Задається в залежності від елемента UI прототипу. Відповідає одному з детермінованих або варіаційних дій	Так

Рисунок 3.5 – Переходи між станами кінцевого автомату прототипу

3.2.7 Основні моделі ПМАГТ

Програмний код основних модулів представлений в застосунку «Основні модулі програмної моделі алгоритму генерації ». Модуль ініціалізації прототипу та вхідних параметрів (Proceed) Даний модуль відповідає за завантаження XML прототипу застосунку в оперативну пам'ять для подальшої обробки. Цей модуль є процедурним і алгоритм його роботи може бути представлений у вигляді блок-схеми.

Модуль програмного уявлення кінцевого автомата (FSM)

Даний модуль є реалізацією КА (FSM) за допомогою рубі класу. Даний клас реалізує модель розширеного КА і містить наступні основні складові:

Елементи класу	Тип класу	Короткий опис
Transitions	Масив	Набір переходу між станами
States	Масив	Набір станів
fsmContext	Об'єкт	Контекст поточного стану кінцевого автомату
Current_state	Об'єкт	Поточний стан кінцевого автомату
Start_state	Рядок	Початковий стан кінцевого автомату
Set_state	Метод	Установити початковий стан КА
Find_state_by_id	Метод	Знайти стан за допомогою ідентифікатору
Find_state_by_name	Метод	Знайти стан за допомогою імені
Find_transition_by_id	Метод	Знайти перехід за допомогою ідентифікатору
Find_transition_by_stateNames	Метод	Знайти перехід за допомогою імені стану
New_state	Метод	Створити новий стан
New_transition	Метод	Створити новий перехід
Make_transition	Метод	Зробити перехід(змінити поточний стан КА)
Transitions_from	Метод	Масив переходів з поточного стану
Transitions_to	Метод	Масив переходів в поточному стані
Next_for	Метод	Масив послідовників поточного стану або переходу
clon	Метод	Створити клон д=поточного КА

Рисунок 3.6 – Елементи класу FSM

Всередині класу FSM стани і переходи представлені у вгляді окремих класів - State, Transition.

Елементи класу	Тип елементу	Короткий опис
Id	Число	Унікальний ідентифікатор стану всередині поточного КА
Name	Рядок	Ім'я стану, відображається в кінцевому результаті
Description	Рядок	Опис стану
Main	Прапор	Покажчик - чи є стан «Головним»
In	Масив	Масив вхідних переходів
Out	Масив	Масив вихідних переходів
Inputs	Масив	Масив класів еквівалентності вхідних даних
elements	Масив	Масив об'єктів, відповідних елементів UI

Рисунок 3.7 – Елементи класу State

Елементи класу	Тип елементу	Короткий опис
Id	Число	Унікальний ідентифікатор переходу всередині поточного КА
Name	Рядок	Ім’я стану, відображається в кінцевому результаті
Source	Об’єкт	Стан-джерело
Target	Об’єкт	Стан-приймач
Condition	Рядок	Умова переходу
Action	Рядок	Дія під час переходу
Chance	Рядок	Вірогідність переходу
Wight	Число	Вага переходу
type	Рядок	Тип переходу

Рисунок 3.8 – Елементи класу

У таблицях наведені основні елементи програмного уявлення КА.

Всі ці елементи використовуються алгоритмом генерації тестових сценаріїв при роботі ПМАГТ.

Модуль контекстного зберігання стану кінцевого автомата (Context)

Цей модуль дозволяє зберегти контекст КА, тобто значення змінних для даного стану КА. Це дозволяє враховувати умови переходів, в залежності від поточних значень змінних, які були отримані змінними в попередніх переходах. Розглянемо РКА, представлений у вигляді графа, зображеного на рисунку 3.9.

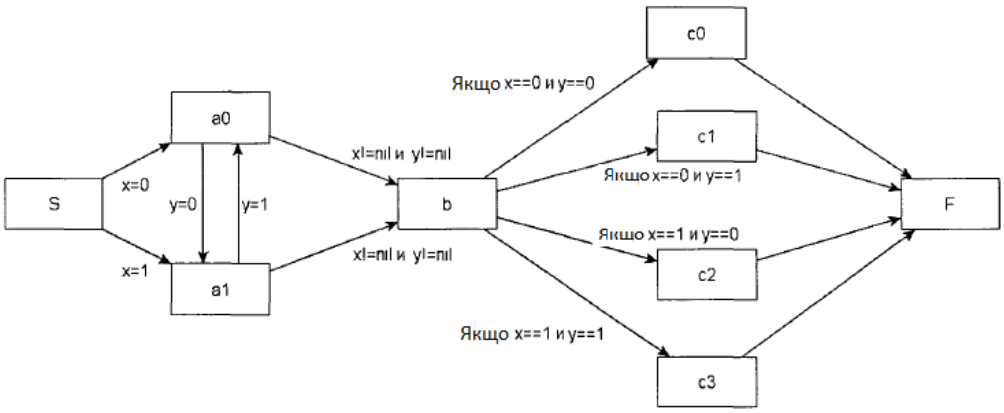

Рисунок 3.9 – Граф деякого розширеного кінцевого автомата

В процесі генерації тестових сценаріїв нам потрібно буде зробити обхід графа, з огляду на значення змінних x і y. У кожному стані значення змінних залежать від попередніх переходів. Тобто в кожному стані значення змінних залежать від контексту стану КА. Уявімо, що один чоловік починає обхід даного графа, тоді для опису всіх можливих контекстів КА нам буде потрібно чотири клони даної людини (рисунок 3.10).

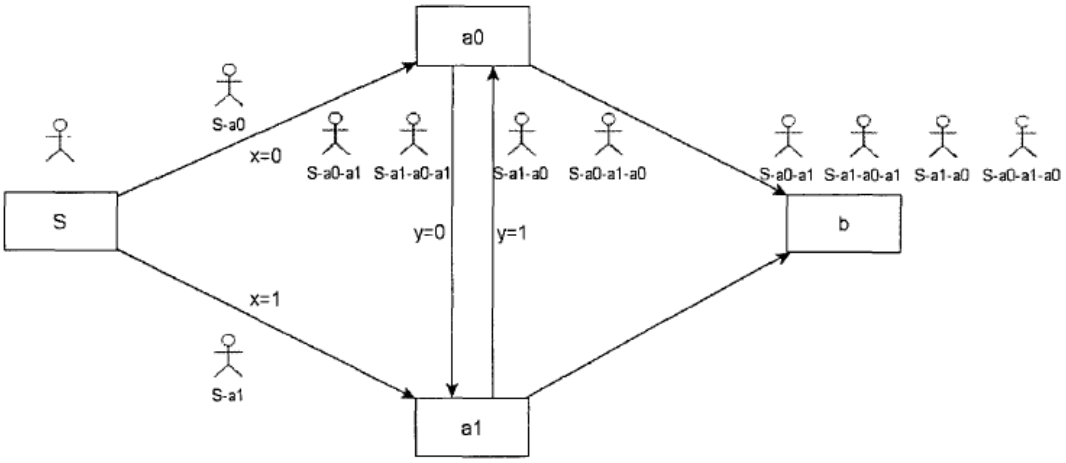

Рисунок 3.10 – Контекст кінцевого автомату в залежності від стану

Даний модуль реалізований у вигляді окремого рубі класу Context, який має всього один метод, представлений в таблиці 10.

Таблиця 3.1 – Елементи класу Context

Елементи класу	Тип елементу	Короткий опис
getBinding	Метод	Метод, повертає контекст змінних класу в поточному стані

Цей клас використовується у вигляді атрибуту fsmContext класу FSM. У свою чергу, при здійсненні переходу КА, перевіряється умова переходу в поточному контексті КА. При клонуванні КА, клонується контекст КА (тобто

значення всіх його змінних в даний момент часу). Більш повний опис використання контексту КА можна знайти в застосунку «Основні модулі програмної моделі алгоритму генерації».

Модулі генерації вихідних даних (Test, testsuite)

Обидва модулі реалізовані у вигляді окремих рубі класів, які використовуються алгоритмом генерації для запису результатів обходу графа КА. Опис класів представлено в таблицях 3.2 та 3.3.

Таблиця 3.2 – Елементи класу Test

Елементи класу	Тип елементу	Короткий опис
Name	Рядок	Ім’я тестового сценарію
Steps	Масив	Кроки тестового сценарію
Precondition	Рядок	Передумова тестового сценарію
Check	Рядок	Очікуваний результат виконаного тестового сценарію
New_step	Метод	Створити новий крок тестового сценарію

3.2.8 Допоміжні модулі ПМАГТ

Модуль допоміжних методів (Helper)

Даний модуль містить допоміжні методи, необхідні для роботи ПМАГТ. Опис модуля представлено в таблиці 3.4.

Таблиця 3.3 – Елементи модуля Helper

Елементи модулю	Короткий опис
setGlobalParams	Установити глобальний параметр генерації
setTestsuite	Ініціалізувати набір тестових сценаріїв
Write_tests	Запис набору тестових сценаріїв у файл
GetXmlFile	Повернути зміст файлу XML прототипа

Модуль логування роботи ПМАГТ (Log)

Даний метод встановлює глобальні параметри логування. Опис модуля представлено в таблиці 3.4.

Модулі обробки помилок (AppError, InternalError)

Дані модулі реалізовані у вигляді рубі класів. вони дозволяють ініціювати запуск винятків певних типів:

AppErrог - виняток, відповідний помилці в ході генерації.

3.2.9 Візуалізація роботи процесу генерації

Вибір технології для візуалізації.

Необхідно задовольнити наступні умови при виборі технології візуалізації:

Кросплатформеність – користувачі будь-яких операційних систем повинні мати доступ до використання ПМАГТ. Простота інтерфейсу - оскільки модель алгоритму виробляє проміжний продукт, необхідно максимально зменшити час, необхідний для вивчення його інтерфейсу. Простота підтримки - необхідно максимально спростити спосіб доставки

користувачам нових версій ПМАГТ. Всім цим умовам задовольняє клієнт-серверна технологія. Модель алгоритму запускається у вигляді ВЕБ сервісу і надає ВЕБ інтерфейс для кінцевих користувачів. При цьому: Підтримується кроссплатформеність, оскільки в будь-який операційній системі існує додаток для роботи з ВЕБ сервісами (браузер).

ПМАГТ має ВЕБ інтерфейс, а значить, кінцевий користувач спочатку представляє функціонал елементів інтерфейсу ПМАГТ і має деякий досвід користувачів для роботи з ним.

Для випуску нових версій досить оновити запущений ВЕБ сервіс. Ніяких додаткових дій від користувача не потрібно.

Вибір конкретного рішення для візуалізації

Через ряд причин, обговорених вище, для реалізації ПМАГТ було обрано мову програмування «Рубі». Тому для кращої інтеграції ПМАГТ з сервером візуалізації, доцільно вибрати клієнт-серверну технологію, підтримувану цією мовою. Існує два найбільш відповідного рішення:

- Ruby on Rails;
- Sinatra.

Оскільки інтерфейс ПМАГТ досить простий, слід вибрати найбільш найпростіше рішення – Sinatra.

Сервер візуалізації.

Сервер візуалізації ПМАГТ реалізований як клієнт-серверна технологія.

Модель алгоритму запускається у вигляді окремого процесу на сервері.

Спілкування ПМАГТ і сервера відбувається за допомогою Tcp-сокетів. Схема взаємодії представлена на рисунку 3.11. Клієнтом в цій архітектурі є будь-який браузер. Спілкування між браузером користувача та сервером під час генерації відбувається за допомогою Ajax технології.

Передача даних між сервером візуалізації та ПМАГТ.

Сервер візуалізації дозволяє стежити за роботою ПМАГТ в «онлайн» режимі. Для цього в моделі алгоритму реалізований функціонал конвертації

поточного стану КА в graphviz формат. На кожному кроці генерації (При включених параметрах серверного режиму і візуалізації), модель алгоритму зберігає і передає на сервер graphviz модель КА за допомогою TCP сокетів. На стороні сервера використовується утиліта graphviz, яка дозволяє побудувати граф переданої graphviz моделі КА. Така поведінка описується схемою, зображеної на рисунку 3.11.

Рисунок 3.11 – Схема серверу візуалізації

Передача даних між сервером та браузером

Передача даних відбувається за допомогою асинхронних HTTP запитів (Ajax) (41). Таким чином, кінцевий користувач стежить за роботою ПМАГТ за допомогою свого браузера, без перезавантаження сторінки. В процесі роботи ПМАГТ змінюється кількість пройдений гілок, що візуалізується за допомогою поновлення SVG зображення (42) моделі КА відбувається з регульованою частотою.

3.3 Висновки третього розділу

У цьому розділі представлені аналітична і програмна моделі алгоритму генерації тестових сценаріїв.

Аналітична модель являє собою модифікований алгоритм A*.

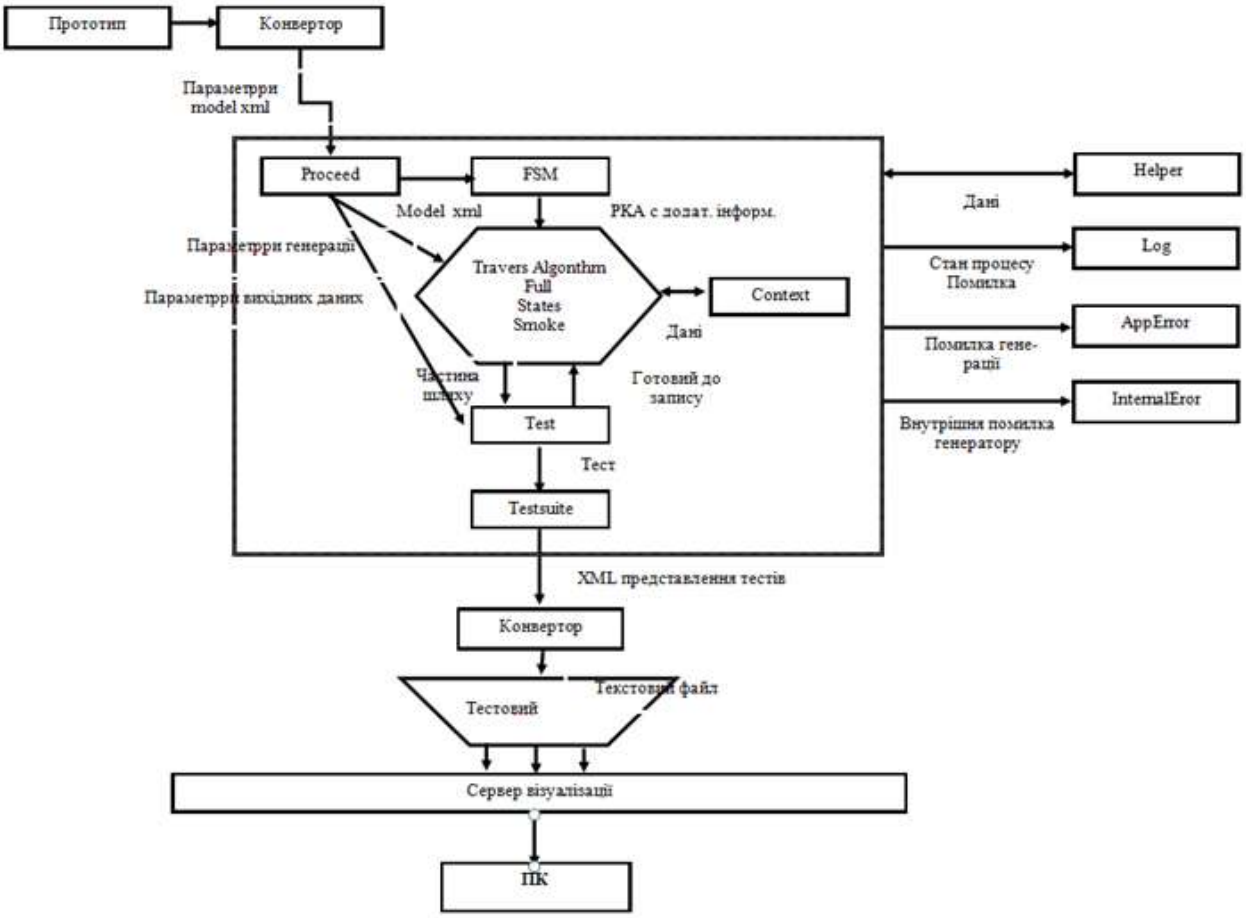

Рисунок 3.12 – Загальна схема роботи ПМАГТ

Програмна модель являє собою програму, реалізовану на мові програмування рубі, з використанням фреймворку Sinatra як компоненти візуалізації роботи. Загальна схема програмної моделі представлена на рисунку 3.12.

4 РОЗРОБКА МОДЕЛІ ТЕСТУВАННЯ ЗАСТОСУНКІВ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

4.1 Мобільні операційні системи

Існує широкий клас операційних систем, які підтримуються смартфонами і комунікаторами. Прикладами таких систем можуть бути:

- Symbian OS (виробники: Nokia, Samsung, Sony Ericson і ін.);
- Windows Mobile (виробники: HTC, T-mobile, Samsung і ін.);
- Palm OS (виробники: Palm і ін.). Android (виробники: Samsung, LG, Palm і ін.);
- iOS (виробники: Apple);
- RIM (виробники: Blackberry);
- Bada (виробники: Samsung);
- Windows Phone (виробники: HTC, Nokia і ін.);
- інші.

Відповідно до статистики на даний момент Android ОС є найперспективнішою і поширеною мобільної ОС у світі. Таким чином, для побудови прототипів ЗМП з урахуванням наступної генерації тестових сценаріїв доцільно вибрати інструмент Axure.

4.2 Модуль конвертації прототипу в формат вхідних даних алгоритму генерації

Цей модуль являє собою спеціальний скрипт, який бере на вхід прототип користувальницького застосунку і формує XML прототип застосунку, «зрозумілий» ПМАГТ. гр - це формат прототипу застосунку, отриманого в результаті прототипирования за допомогою інструменту AxureRP. AxureRP надає API для роботи з прототипом. Цей програмний

інтерфейс дозволяє конвертувати прототип застосунку в необхідний XML прототип застосунку. Вихідний код модуля представлений в застосунку «Модулі конвертації». Схема конвертування представлена на рисунку 4.1

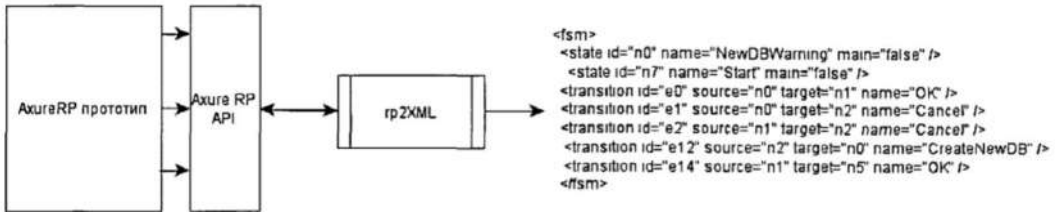

Рисунок 4.1 – Схема конвертування

4.3 Інструмент автоматизації тестування застосунків для мобільних пристроїв

З урахуванням проведеного дослідження, необхідно шукати інструмент для автоматизації тестування для ОС Android. Оскільки тестові сценарії будуть згенеровані, необхідний програмний інструмент автоматизації. На момент написання роботи існує тільки один інструмент, що підходить за параметрами - Robotium. Для використання даного інструменту в рамках запропонованого методу тестування, необхідно створити програмний інтерфейс букв алфавіту тестування, використовуючи Robotium framework.

4.4 Модуль конвертації тестових сценаріїв в автоматичні тести

Даний модуль реалізований у вигляді окремого рубай скрипта, який приймає на вхід згенеровані тестові сценарії в уніфікованому XML поданні і видає на виході тестові сценарії в двох варіантах:

– тестові сценарії для автоматизованого тестування за допомогою програмного інтерфейсу алфавіту тестування, розробленого з використанням Robotium framework.

– тестові сценарії для ручного тестування функціоналу.

Схема модуля представлена на рисунку 4.2

Рисунок 4.2 – Модуль конвертації тестів

4.5 Середовища тестування застосунків для мобільних пристроїв

В якості тестової середовища необхідно вибрати IDE, що дозволяє виконувати автоматичне тестування і збирати результати тестування. Таким IDE є Eclipse (51) з відповідними застосункуми (плагінами).

4.6 Схема імітаційно-статистичної моделі

В результаті вибору всіх інструментів, отримуємо наступну схему імітаційно-статистичної моделі тестування ЗМП, зображену на рисунку 4.3

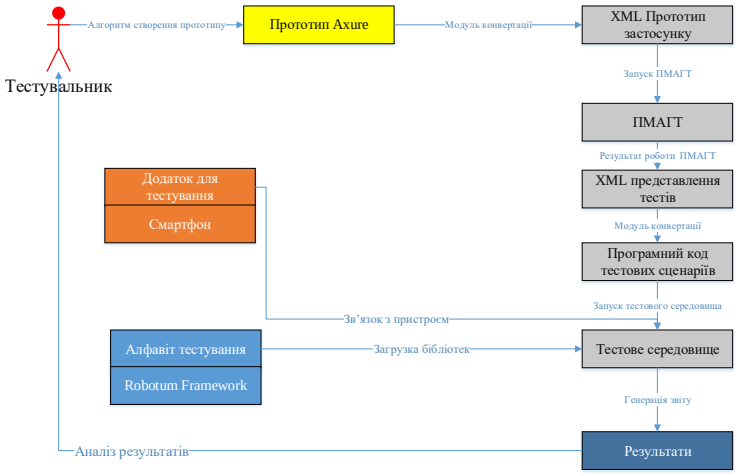

Рисунок 4.3 – Схема імітаційно-статистичної моделі тестування ЗМП

4.7 Ефективність запропонованого алгоритму генерації тестових сценаріїв

Цільова функція ефективності генерації тестових сценаріїв описується формулою (2.3):

$$Eff(G) = \frac{N_e}{N * L} 100\%, \qquad \text{де}$$

G – граф прототипу ЗМП

N*L – загальна кількість переходів, здійснених алгоритмом в внаслідок обходу.

$N_e = \sum_{s_i \epsilon s} \max(s_i^{in}, s_i^{out})$ – мінімальне і достатня кількість переходів, які потрібно здійснити для покриття всіх переходів РКА (відповідно до теореми Ейлера).

В середньому кількість станів ПМУ лежить в наступних межах: К ∈ [30, ..., 70]. Відповідний граф розширеного кінцевого автомата повинен бути зв'язковим, отже, найменшу кількість переходів одно: MMIN= К-1. Оцінка максимальної кількості переходів:

$M_{MAX} = \frac{K*Max}{2}$, де $Max = \max_{i \in Nodes} (in_i + out_i)$ – максимальна сума вихідних і вхідних гілок в вузол. Таким чином, середня кількість гілок графа:

$$M_{MAX} = \frac{K * Max}{2} + K - 1 \approx \frac{Max + 2}{4} K.$$

Для оцінки ефективності роботи моделі разробленого алгоритму був розглянутий набір РКА однакової структури, але різною кількістю параметрів (станів, переходів, умов, змінних). Структура прототипів підбиралася так, щоб задовольнялися 2 умови:

- РКА повинен емулювати реальний додаток;
- РКА повинен мати гарну масштабованість. У мобільних застосункух взаємодії користувача і додатки в загальному випадку відбувається за

наступним сценарієм:

- користувач знаходиться у вигляді, де передбачено введення даних;
- користувач вводить дані;
- додаток перевіряє валідність введених даних.;
- якщо дані невалідні, додаток виводить на екран наступний вигляд з повідомленням про помилку. Користувач приймає повідомлення і повертається на екран введення даних;
- якщо дані валідність, користувач потрапляє в новий вид, де можуть бути відображені результати його дій;
- користувач переходить в новий вид, де передбачено введення даних або повертається до попереднього вигляду. Назвемо РКА, відповідний описаним сценарієм «ядром тестових РКА».

Таблиця 4.1 – Результати роботи алгоритму генерації в розширеному режимі

Критерій оцінки	Результати оцінки	
	Генератор-опонент	Запропонована модель генерації
Середня ефективність алгоритму генерації	79%	77%
Середня швидкість роботи алгоритму	294 сек/додаток	0.18 сек/додаток
Середнє забезпечується тестове покриття	95%	100%
Можливість автоматичної генерації кроків на перевірку результатів тестування	ні	Так
Розширюваний формат вихідних даних	Ні	так

Таблиця 4.2 – Результати роботи алгоритму генерації в розширеному режимі

число ядер (И)	число станів	число переходів	число параметрів	число умов	Eff(A*)	Eff(опонент)
2	9	15	1	2	88	94
3	13	23	1	3	82	92
4	17	31	1	4	79	79
5	21	39	1	5	78	89
6	25	47	1	6	77	86
7	29	55	1	7	77	77
8	33	66	1	8	76	82
9	37	71	1	9	76	78
10	41	79	1	10	75	72
11	45	87	1	11	75	73
12	49	95	1	12	75	73
13	53	103	1	13	75	75
14	57	111	1	14	75	64
15	61	119	1	15	75	78
16	65	127	1	16	75	77
17	69	135	1	17	75	75

З наведеної таблиці випливає, що запропонована модель генерації перевершує генератор опонент по сукупності обраних критеріїв оцінки.

4.4 Ефективність запропонованого методу тестування

Відповідно до представлених критеріїв оцінки ефективності методів тестування, було проведено аналіз трудовитрат при використанні запропонованого прототипного підходу, на основі формули (2.2):

$$T = t\left[\begin{array}{c}\left(C_{\text{прог}} + \text{C}_{\text{оцін}}\right)(\text{К}+1) + \left(\text{C}_{\text{р мод}} + \text{C}_{n\ \text{спец}} + \text{C}_{n\ \text{тест}}\right)\left(1 + \frac{\text{К}\overline{\delta N}}{N}\right) + \\ + C_{\text{імп}} * N + \frac{\text{К}\overline{\delta N}}{N^2}\end{array}\right] \rightarrow \rightarrow min$$

За основу було взято час необхідний для текстового опису одного елемента користувальницького інтерфейсу застосунку (коефіцієнт $\text{C}_{n\ \text{спец}}$). Решта коефіцієнти у формулі розраховувалися по відношенню до даного коефіцієнту. Отримані наступні результати: $\text{C}_{n\ \text{спец}} =$ 2 - це значення відповідає сценарієм, в якому для складання специфікації по одному елементу інтерфейсу ПМУ, у аналітика йде 2 години. Збільшення ефективності щодо методу РР становить в середньому = 100% в залежності від кількості ітерацій повного циклу розробки. Збільшення ефективності щодо методу РА (програмний= підхід) складає в середньому 30%.

ВИСНОВКИ

Досліджені моделі та алгоритми тестування застосунків для мобільних пристроїв.

Було використано прототип застосунків для мобільних пристроїв; проведено аналіз метрик тестування застосунків для мобільних пристроїв, що враховують їх особливості та відповідні критерії оцінки ефективності методів тестування;

Проведено аналіз аналітичних та програмних моделей генерації автоматизованих тестів з прототипів застосунків для мобільних пристроїв.

Представлені результати роботи алгоритму генерації в розширеному режимі.

Відповідно до представлених критеріїв оцінки ефективності методів тестування, було проведено аналіз трудовитрат при використанні запропонованого прототипного підходу

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Браїла І.В., Дяченко В.О., Міхаль О.П. Модель тестування застосунків для мобільних пристроїв // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей дванадцятої міжнародної науково-технічної конференції 27-28 квітня 2022 р., т.1. Баку-Харків-Жиліна. 2022 р. С.71.

2. Хэнссон Д. Х., Томас, Д. Гибкая разработка ВЕБ-приложений в среде Rails. Санкт-Петербур : Питер, 2008.

3. MSDN Magazine, microsoft.com. [Інтернет] http://msdn.microsoft.com/ra- ru/magazine/dd419663 .aspx.

4. Andrews A., Offut J., Alexander R. Testing Web Applications by Modeling with FSMs. б.м. : National Science Foundation, 2005.

5. Makinen M. Model Based Approach to Software. Helsinki : Helsinki University of Technology, 2007.

6. Кулямин В. Компонентная архитектура среды для тестирования на основе моделей. Программирование. 2010 г., Т. 5.

7. Robinson H. 1999. Graph Theory Techniques in Model-Based Testing.

8. Heiskanen H., Maunumaa M., Katara, M. Test Process Improvement for Automated Test Generation. Tampere: Tampere University of Technology, Department of Software Systems, 2010.