Шаров С.В.
кандидат педагогічних наук, доцент,
завідувач кафедри комп'ютерних наук
Трачов В.В.
здобувач вищої освіти
Таврійський державний агротехнологічний
університет імені Дмитра Моторного

ПРИЧИННО-ТРИГЕРНИЙ ПІДХІД ДО ВИЗНАЧЕННЯ ПРИЧИН ЗБОЇВ ІНФОРМАЦІЙНИХ СИСТЕМ ТА НАПРЯМКІВ ЇХ ВИРІШЕННЯ

Анотація. *У статті висвітлюються особливості причинно-тригерного підходу до визначення причин збоїв інформаційних систем, аналізуються підходи до пошуку та виявлення дефектів програмного забезпечення, висвітлюються фактори, що можуть бути віднесені до тригерів збоїв, пропонуються заходи, спрямовані на уникнення або вирішення проблем, пов'язаних з якістю програмного забезпечення.*

Ключові слова: *якість програмного забезпечення, тригер, виявлення дефектів, інформаційна система, причинно-тригерний підхід.*

Sharov S., Trachov V. Cause-Trigger Approach to Identifying the Causes of Information System Failures and Directions for Their Resolution. *The article highlights the features of the cause-trigger approach to identifying the causes of information system failures, analyzes approaches to detecting and identifying software defects, outlines factors that can be considered as triggers for failures, and proposes measures aimed at avoiding or resolving issues related to software quality.*

Key words: *software quality, trigger, defect detection, information system, cause-and-trigger approach.*

Актуальність. Чітке розуміння того, звідки беруться дефекти, як правильно аналізувати причини їх появи і як при цьому нічого не пропустити, може дати розробникам хорошу основу для того, щоб надалі зменшити їх вплив, правильно боротися з причинами збоїв і врешті-решт отримати більш надійне програмне забезпечення. Тому застосування

методів виявлення дефектів програмного забезпечення, зокрема інформаційних систем, є важливим етапом забезпечення якості програмного забезпечення. Одним з таких підходів є причинно-тригерний підхід, який дозволяє після попереднього виявлення потенційних помилок у програмному забезпечення знайти справжні причини дефекту.

Метою статті є висвітлення особливостей причинно-тригерного підходу до визначення причин збоїв інформаційних систем та формування переліку заходів, спрямованих на уникнення або зменшення кількості дефектів програмного забезпечення.

Виклад основного матеріалу. Будь-яке програмне забезпечення (ПЗ), у тому числі інформаційні системи [6, с. 17], повинні відповідати стандартам якості. В даному випадку якість ПЗ оцінюється через комплекс критеріїв, що характеризують функціональність, надійність, сумісність, зручність використання та супроводу, портативність [1, с. 85] тощо. Одним з підходів до оцінки якості ПЗ є тестування програмного забезпеченнях, що передбачає пошук дефектів функціональних та нефункціональних вимог [5, с. 93]. Тестування ПЗ надає можливість забезпечити високу якість ще на етапі розробки та передбачає дослідження функціональності програмного забезпечення за допомогою численних тестів [2, с. 249].

На жаль, на даний момент ні в англомовній, ні в україномовній літературі для багатьох понять у галузі програмного забезпечення немає єдиних загальноприйнятих визначень. Це стосується і понять, пов’язаних з помилками в інформаційних системах. Тому спочатку треба необхідно визначитися з основними поняттями. Сучасна термінологія в цій галузі в більшості спирається на термінологічну модель Авізенісу та його колег [8]. В її основі лежать три основні концепції: збій, помилка та дефект. З часом їхні формулювання були дороблені та адаптовані під певні напрямки діяльності з інформаційною системою. Далі наведені визначення цім концепціям, які ми будемо використовувати в роботі:

– збій (failure) – це подія, коли послуга, що надається, відхиляється від правильної з точки зору користувача;
– помилка (error) – це частина стану системи, яка може викликати наступний збій;

– дефект (fault, bug) – це мінімальний набір відхилень коду від правильного, при якому виконання коду, що відхиляється, може викликати помилку [9].

У цьому переліку можна побачити чітку ієрархічну структуру, на нижчому рівні якої знаходиться дефект і саме його необхідно буде знайти, виправити, а потім знайти причину, що викликала цей дефект та спробувати її усунути чи зменшити вплив.

Пошук причин виникнення дефекту в складних інформаційних системах є комплексним та доволі ресурсомістким процесом. Адже якщо існує ланцюжок подій, після яких з'явився дефект і між ними спостерігаються причинно-наслідкові зв'язки, то будь-яку з цих подій можна назвати причиною його появи, оскільки вона була в минулому і після неї з'явився цей дефект. Як наслідок, дефекти програмного забезпечення може призвести до зменшення функціональності або продуктивності, неправильної роботи системи [7, с. 104], втрати інформації, втрати конференційних даних [3, с. 235], збоїв на рівні програмного забезпечення тощо.

У зв'язку з цім з'явилось дуже багато методів та підходів до цього процесу, але найбільш популярними є три:

– 5 чому (5 whys);
– діаграма Ішикави (Ishikawa Diagram);
– аналіз кореневої причини (Root Cause Analysis, RCA).

Зупинимось на них більш детально. 5 чому – простий метод, що полягає у послідовному завданні питань «чому». Дозволяє виявити певні причини проблеми, але його результат дуже залежить від того хто саме задає питання, наприклад, менеджер, тестувальник чи програміст.

Діаграма Ішикави, відома також як діаграма «риб'ячої кістки» (Fishbone Diagram) – візуалізує потенційні причини проблеми у вигляді горизонтальної стрілки (дефекту), від якої під нахилом відходять стрілки причин, що досліджуються [4, с. 90]. Такий підхід дуже добре себе зарекомендував в автомобільної промисловості, але при розробці програмного забезпечення він зазвичай є слабо сфокусованим тому не дає можливості виділити основні причини.

Аналіз кореневої причини – це метод, основною метою якого є зрозуміти, чому сталася проблема, а не просто усунути її симптоми. У контексті пошуку причин дефектів у програмному забезпеченні цей

метод допомагає виявити фундаментальні причини помилок, не тільки тимчасові або поверхневі. Але ж практика показує, що нечасто проблему викликує тільки одна причина – як правило їх декілька та вони тісно пов'язані між собою, тому визначити одну кореневу буває дуже складно.

Крім перелічених вище існує ще декілька дієвих методологій, але на практиці у спеціаліста, що відповідає за аналіз збоїв не завжди є достатньо часу чи бажання досконально всі можливі варіанти причин. Саме тут на допомогу може прийти причинно-тригерний підхід до визначення причини збою. Такий підхід передбачає поділ всіх подій, що потенційно можуть бути потрібними причинами на власне причини та тригери. Такий підхід дозволить після попереднього аналізу відкинути тригерну складову та сфокусуватися на пошуку справжньої причини дефекту.

По-перше потрібно визначитись з тим, що причиною треба вважати тільки ту подію, на яку можна вплинути і змінити що-небудь у інформаційної системі для ліквідації дефекту або зменшення його впливу. По-друге – будь-який збій починається з якоїсь події, яка безпосередньо запускає її – це і є тригер. Так, наприклад, якщо збій стався в результаті виходу з ладу обладнання, то було б зручно сказати, що його причиною є поломка обладнання. Однак це не так, адже обладнання завжди ламалося і буде ламатися. Навіть якщо змінити його виробника це скоріш за все продовжуватиме відбуватися (в кращому випадку з меншою частотою). Тому, вихід із ладу устаткування у цьому прикладі – це тригер, а причиною є відсутність досить працездатних механізмів захисту від поломок устаткування.

За цім підходом, при пошуку причини виникнення збою всі підозрілі події необхідно зіставити з можливими тригерами і якщо вона є у цьому списку – переходити до іншої. В узагальненому вигляді до тригерів збоїв можна віднести такі фактори:

– стихія та зовнішній техногенний фактор – урагани, землетруси, а також пожежі, відключення електроенергії та інше;
– реліз – коли новий функціонал містить дефектний код;
– відмова обладнання;
– зміни в діях користувачів – зміни в тому який функціонал чи з якою інтенсивністю використовується;

– переповнення – переповнення диска, оперативної пам'яті, портів, лічильників бази даних та інше;
– випадковий збіг факторів;
– зміна роботи зовнішніх систем – збій чи зміна поведінки систем, з якими інтегрована наша;
– дія навчених фахівців – дії спеціалістів, що працюють зі системою але напряму не пов’язані з її програмуванням (системний адміністратор, інженер по автоматизації технологічних процесів та ін.);
– дія третіх осіб – дії зловмисників, хакерів та ін.

Далі необхідно знайти потрібну подію в переліку саме причини та визначитися с підходами до її вилучення чи зменшення її впливу. Далі представлені деякі основні причини с заходами по боротьбі з ними.

1. Помилка (недоробка) у програмному коді, в архітектурі застосунку (наприклад, синхронний чи асинхронний підхід) чи в архітектурі системи (однорівнева чи мікросервісна).

Вирішення такого роду проблем залежить від багатьох чинників, наприклад від того звідки цей код: написаний всередині компанії, чи він є опенсорсний або був куплений. Також багато залежить від того чи є ця проблема технічним боргом (тобто вона та її рішення відомі в компанії) або проблема нова чи для неї ще нема рішення. Якщо код написаний всередині компанії, то рекомендовано вжити перелічених далі заходів.

Встановити стандарти кодування у команді. Це включає угоди про форматування, іменування змінних і функцій, коментування коду, тощо. Для моніторингу дотримання цих стандартів можна використовувати так звані лінтери, які розроблені для більшості мов програмування.

Проводити регулярні код-рев’ю – це дозволяє членам команди виявляти та виправляти помилки до того, як код потрапить в основну гілку проекту. Цей процес можуть сильно спростити системи контролю версій, наприклад Git та створені на його основі сервіси (GitHub, BitBucket та ін.).

Проводити юніт-тестування та відстеження покриття тестами всього коду, який можна тестувати.

Використовувати логування (запис інформації про події в системі в файли чи базу даних) для відстеження та аналізу помилок у реальному часі.

Підтримувати код у хорошому стані через регулярні рефакторинги. Це може допомогти усунути технічний борг, що накопичився, і зменшити ймовірність помилок.

2. Невдало спроектовані процеси – процес автоматичної інтеграції та розгортання проекту (CI/CD), процес узгодження робіт. Таке зустрічається коли процес є хибним або підштовхує співробітника до помилок. Для вирішення цієї проблеми необхідно спростити процес і зробити його зручнішим, щоб не хотілося його обходити. Або ж підняти його важливість, проводити роз'яснювальні роботи та обмежити доступ до засобів для здійснення цього процесу без погодження керівника.

3. Помилка працівника при проведенні робіт. В цьому випадку дії можуть залежати від того чи це просто помилка співробітника чи адміністративний інтерфейс провокує на цю помилку. Тому треба відпрацьовувати дії спеціалістів в стресових ситуаціях або удосконалити адміністративний інтерфейс.

4. Проблеми з в комунікацією та документацією. Для вирішення подібних проблем необхідне проведення мітапів та впровадження механізмів доведення інформації до співробітника (у разі його відсутності). При цьому для боротьби з перенасиченням повідомленнями можна використовувати різні рівні їх доставки, наприклад, повідомлення, записи обговорень і короткі курси (з іспитами та підтвердженнями).

Також необхідно підтримувати документацію в актуальному стані, що особливо важливо для нових членів команди.

5. Пряма людська проблема. Сюди крім вигоряння, лінощів і втоми слід також віднести і ігнорування ризиків за бажання досягти мети будь-якою ціною. У такій ситуації можна переглянути графіки завдань проекту, а також різними мотиваційними та спрямованими на психологічну підтримку заходами. Крім того, не можна нехтувати відкритим обговоренням з демонстрацією прикладів проектів, де ігнорування можливих ризиків призвело до серйозних наслідків.

6. Навмисне заподіяння шкоди. Цю проблему можна розділити на дві категорії: заподіяння шкоди співробітниками та третіми особами. Але в обох випадках необхідно вдосконалити алгоритми, які забезпечують безпеку, а також проводити регулярні аудити безпеки, щоб оцінити рівень захисту в офісі.

Висновки. Отже, дефекти та збої програмного забезпечення можуть призвести до втрати інформації, зменшення функціональності тощо. Серед відомих підходів до виявлення дефектів програмного забезпечення слід виокремити діаграму Ішикави, аналіз кореневої причини, підхід «5 чому». Водночас, застосування причинно-тригерного підходу до визначення причин збоїв програмного забезпечення, у тому числі інформаційних систем, надає можливість відкинути несуттєві причини та сфокусуватися на пошуку справжньої причини дефекту.

Література

1. Грицюк Ю. І. Система комплексного оцінювання якості програмного забезпечення. *Науковий вісник НЛТУ України*. 2022. Т. 32, № 2. С. 81–95.
2. Коломоєць Д. А. Особливості автоматизованого тестування якості програмного забезпечення. *Українські студії в європейському контексті*. 2023. № 7. С. 247–252.
3. Лубко Д. В., Мірошниченко М. Ю. Аналіз сучасних підходів та методик в області захисту інформації та даних. Вісник ХНТУ. Серія: Інформаційні технології. 2024. № 1. С. 231–236.
4. Огірко О. І., Пілат О. Ю., Романюк О. П. Моделювання інформаційних технологій діаграмами Ісікави. *Кваліlогія книги*. 2016. № 1. С. 90–99.
5. Руда О. А., Моденов Ю. Б. Методи та моделі тестування програмного забезпечення. Problems of Informatization and Control. 2013. Т. 2, № 42. С. 93–98.
6. Шаров С. В., Нікітенко Д. С. Визначення та завдання довідково-інформаційних систем. *Зб. тез доповідей Всеукраїнської наукової Internet-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення»*. 2014. № 1. С. 15–19.
7. Яковина В. С., Симець І. І. Прогнозування дефектів програмного забезпечення ансамблем нейронних мереж. *Науковий вісник НЛТУ України*. 2021. Т. 31. № 6. С. 104–111.
8. Avizienis A., Laprie J., Randell B., Landwehr C. E. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*. 2004. Vol. 1. Pp. 11–33.
9. Pretschner A., Holling D., Eschbach R., Gemmar M. A generic fault model for quality assurance. Model-Driven Engineering Languages and Systems. Springer, 2013. P. 87–103.