УДК 004.891.3: 004.3

Перевозніков С.І., Говорущенко Т.О.

Вінницький національний технічний університет, м.Вінниця;
Хмельницький національний університет, м.Хмельницький

СИСТЕМА ІДЕНТИФІКАЦІЇ ПРИХОВАНИХ ТИПОВИХ ПОМИЛОК ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В статті описано типові помилки проектування програмного забезпечення за Алленом [1], описано методи та операції виявлення типових помилок проектування; наведено поняття рівня категорійності прихованих помилок, які залишились в програмному продукті після основного тестування, а також поняття та призначення повторного тестування. Призначено рівні категорійності типовим помилкам проектування, які залишились в програмному продукті і стали прихованими. Описано систему ідентифікації прихованих типових помилок програмного забезпечення, яка вказує на наявність помилок та інформує про їх рівень категорійності на основі звіту про основне тестування програмного продукту.

In article typical software designing bugs (bug patterns) by Eric Allen [1], techniques and operations of typical software designing bugs finding are described; conception of category level of hidden bugs, which stayed in program after basic testing, and retesting conception and purpose are cited. Category level appointed to typical software designing bugs, which stayed in program and turned into hidden bugs. Hidden typical software designing bugs identification system is described. This system points to bugs presence and informs about their category level on basis of basic software testing reports.

Вступ

Надійність і достовірність роботи сучасних мікропроцесорних пристроїв, комп'ютерів та комп'ютерних систем є запорукою успішного вирішення задач, що розв'язуються ними, та довготривалої їх експлуатації. Одним з основних засобів, за допомогою котрого досягається висока надійність правильного функціювання обчислювальної техніки та вірність розв'язку прикладних задач, є діагностування програмного забезпечення (ПЗ) на різних етапах його життєвого циклу, зокрема, на етапах проектування, розробки та експлуатації.

Однією з основних складових діагностування ПЗ є тестування як процес виявлення помилок у програмах. Його роль тим більше зростає в зв'язку з тим, що ПЗ сучасних комп'ютерних систем є досить складним і не може бути бездефектним. Причиною невиявлення помилок у розроблюваному ПЗ швидше за все слід вважати недосконаліть тестів, а не бездефектність програми.

Виявлення недоліків методів та засобів тестування ПЗ, у тому числі ідентифікація прихованих помилок програмних продуктів, є актуальною задачею розвитку методів тестування ПЗ, що підвищує його достовірність.

Типові помилки проектування програмного забезпечення за Алленом

Ерік Аллен під програмним дефектом розуміє "поведінку програми, яка відхиляється від поведінки, визначеної специфікацією" [1]. Під це визначення не підпадають: низька продуктивність, нефективний або незручний користувацький інтерфейс; недостатня функційність, відсутність будь-якої функції, що не входить до специфікації програми.

Ерік Аллен виділяє наступні різновиди типових помилок проектування програмного забезпечення [1]:

–"Фальшива черепиця". програма поводить себе так, як якщо б виправлена раніше помилка залищилась на своєму місці. Причиною такої поведінки є той факт, що принаймні одна копія розтиражованого фрагменту програми містить дефект, виправлений в інших копіях.

–Нульові вказівники (Null Pointer) та нульове виключення (NullPointerException) – виникають внаслідок присвоєння змінним, на які вони вказують, нульового значення; такі вказівники неінформативні та невловимі, можуть призводити до дуже серйозних помилок та дефектів програми.

–"Незакріплений компонент" - виникає при

визначенні рекурсивного типу даних таким чином, що деяким базовим випадкам визначення не виділяються власні класи. Замість того в різні складені типи даних вставляються нульові вказівники, після чого екземпляри типів даних використовуються так, як якщо б ці нульові вказівники були заповнені відповідним чином.

–“Нульовий прапорець” - важко діагностоване виключення. Причиною є те, що викликаючі методи не перевіряють значення, що повертається, на рівність нулю. Симптомом є той факт, що блок програми, в якій використовується нульовий вказівник як ознака виключної ситуації, сигналізує виключенням NullPointerException.

–“Подвійний спуск” - програма рекурсивно спускається по складеній структурі даних, так що за один рекурсивний виклик виконується більше, ніж один крок донизу. Причиною є той факт, що фрагмент програми спускається на два рівні за одне звертання до методу баз відповідної обробки другого спуску.

–“Фальшиве представлення” - програма, яка використовує графічний користувацький інтерфейс, проходить набір тестів, але потім проявляє поведінку, яку ці тести повинні були заборонити. Причиною є те, що тести перевіряють лише аспекти моделі напряму. Представлення ж моделі відокремлено від самої моделі, тому зміна стану моделі не завжди відповідним чином відображається в її представленні.

–“Шкідливі дані” - дуже важко дістатись до істинної причини незапланованого функціювання програми, коли вона невірно функціює через помилки в даних. Симптоми такої помилки – програма, яка зберігає складні дані та керує ними, несподівано починає невірно функціювати, виконуючи завдання, схоже на інші задачі, виконання яких не викликає проблем. Причиною такої неполадки є пошкодження синтаксису чи семантики деяких внутрішніх даних.

–“Зламаний діспатч” - виникає при користуванні поліморфізмом наслідування в комбінації зі статичним перевантаженням. Симптоми – одразу після перевантаження методу, раптово не працює тест для програми, яку ви не змінювали. Причина - перевантаження методу призвело до того, що метод, який не змінювався, почав викликати не той метод, який було заплановано.

–“Тип-самозванець” - коли для того, щоб розрізняти типи об'єктів використовуються спеціальні тегові поля, можуть виникати помилки, при яких теги невірно помічають дані. Симптоми – програма однаковим чином обробляє принципово різні типи даних або не розпізнає певні типи даних. Слід підозрювати помилку “тип-самозванець”, якщо виникає невідповідність між абстрактним типом даних і тим способом, яким він обробляється в програмі.

–“Розщеплений чистильник” - симптоми - програма некоректно керує ресурсами: програма не звільняє всі ресурси (пам'ять, файли, з'єднання з базою даних) або звільняє їх надто рано. Причина – на деяких шляхах виконання програми ресурси не звільняються рівно один раз, як це повинно бути.

–“Фіктивна реалізація” - важко діагностований різновид помилок. Симптоми – клієнтський клас, який працює з вказаним інтерфейсом, не вірно працює, якщо використовується певна реалізація цього інтерфейсу. Причина – реалізація інтерфейсу не задовольняє деяким інваріантам, які містяться в ньому.

–“Осиротілий потік” - багатопоточна програма є недетермінованою, тому ймовірність появи помилок в ній досить висока, більш того, помилки, які виникають в такій програмі, набагато складніше відтворити, а, отже, й виправити. Симптоми – багатопоточна програма “зависає”, а вміст стеку виводиться в стандартний пристрій для виведення повідомлень про помилки. Причина – різні потоки програми зупинились в очікуванні вхідних даних від потоку, який припинив роботу, після того як було ініційовано виключення, що не перехоплюється.

–“Недостатня ініціалізація” - досить часто можна побачити визначення класу, в якому конструктор класу приймає недостатню кількість аргументів для коректної ініціалізації всіх полів класу. При використанні подібних конструкторів клієнтські класи повинні ініціалізувати об'єкти в декілька етапів. Така ініціалізація екземпляру являє собою процес, який може призводити до виникнення багатьох помилок. Симптоми – виключення NullPointerException, яке виникає при звертанні до одного з неініціалізованих полів. Причина – клас, конструктори якого не ініціалізують всі поля напряму: недостатня кількість аргументів

конструктора для ініціалізації полів класу; програмістом пропущено один з етапів ініціалізації; кроки ініціалізації можуть бути залежними один від одного, але виконані програмістом в невірному порядку; може бути змінено ініціалізований клас.

–Помилки, пов'язні з залежністю від платформи – такі помилки можуть бути залежними від виробника, від версії, від операційної системи. Виникають через різні правила поведінки апаратури різних виробників, різних версій програмного забезпечення та різних операційних систем. Найбільш поширені помилки, залежні від версій.

Методи та операції виявлення типових помилок проектування ПЗ

Для виявлення типових помилок проектування ПЗ вживаються наступні методи та операції:

- «Фальшива черепиця» - по можливості вичленення спільних фрагментів програми; перевірка всіх схожих фрагментів програми з метою виявлення розбіжностей між ними.

- Нульові вказівники та нульове виключення – перевірка, чи не дорівнюють 0 змінні, на які вказують будь-які вказівники.

- «Незакріплений компонент» - перевірка, чи представлені базові випадки; перевірка базових випадків на не протиріччя; перевірка на правильність використання нульових вказівників; перевірка на заповнення певних екземплярів типів даних.

- «Нульовий прапорець» - перевірка на виключні ситуації; перевірка на значень, що повертаються, на рівність нулю.

- «Подвійний спуск» - вичленення фрагменту програми, який відповідає за перетворення типу, в окремі методи кожного класу; перевірка інваріантів на успішність перетворення типів; «запакування» кожного оператора перетворення типів в перевірку instanceof; перевірка всіх рекурсивних викликів на факт спуску по складеній структурі даних не більше, ніж на один крок.

- «Фальшиве представлення» - незалежне тестування моделі і представлення, а також тестування їх комбінації; перевірка не лише аспектів моделі напряму, а і представлення моделі.

- «Шкідливі дані» - тестування на цілісність даних; перевірка цілісності тих даних, що залишились після видалення існуючих пошкоджених даних; граматичний розбір вхідних даних; контроль типів з метою перевірки цілісності програми на семантичному рівні; перевірка цілісності даних в режимі offline; перевірка синтаксису внутрішніх даних; перевірка семантики внутрішніх даних.

- «Зламаний діспатч» - перегляд набору методів, наданих різними класами; виявлення фрагменту програми, який не був змінений, але його тест сигналізує про неполадки; перевірка методу після його перевантаження.

- «Тип-самозванець» - розподіл принципово різних типів даних на окремі класи з метою виявлення помилки обробки різних типів даних однаковим невірним чином або не розпізнавання певних типів даних; перевірка абстрактного типу даних та способу його обробки в програмі на відповідність.

- «Розщеплений чистильник» - перевірка, чи відбувається звільнення кожного ресурсу і чи відбувається воно лише один раз.

- «Фіктивна реалізація» - перевірка початкової умови (чи виконана деяка умова до входження в програмний блок); перевірка вихідної умови (чи виконана деяка умова на виході з програмного блоку); перевірка інваріанту (чи виконана деяка умова під час виконання програмного блоку); блочне тестування з метою визначення, чи виконуються додаткові інваріанти інтерфейсу.

- «Осиротілий потік» - знаходження помилки в потоці, який некоректно працює або «завис»; перевірка на перехоплення виключень, ініційованих в різних потоках, до припинення роботи; блочне тестування з метою виявлення помилок в багато потокових додатках; перевірка, чи жоден потік не зупинився в очікуванні вхідних даних.

- «Недостатня ініціалізація» - перевірка, чи приймає конструктор достатню кількість аргументів для ініціалізації полів класу; перевірка, чи не пропущено жоден етап ініціалізації; перевірка, чи не виконано залежні кроки ініціалізації в невірному порядку; перевірка, чи не змінився ініційований клас.

- Помилки, пов’язані з платформою – перевірка, чи всі функції програмного продукту підтримуються тією платформою (виробник, версія, операційна система), на якій повинен працювати даний програмний продукт.

Прихованість помилок та рівні категорійності прихованих помилок

Прихованою помилкою назвемо будь-яку

помилку ПЗ, що залишилась у програмному продукті після його діагностування у процесі розроблення та налагодження. Приховані помилки відрізняються від виявлених тим, що вони на певний момент часу після діагностування у процесі розроблення та налагодження ПЗ існують і ще не виявлені. При цьому помилки, зумовлені дефектом, певним чином впливають на систему через програмне забезпечення

Виявлення прихованих помилок проводиться після розроблення і налагодження ПЗ, де тестування програм здійснювалось як часткова технологічна операція під час цих процесів (етапів). Вважатимемо його окремим технологічним процесом. У такому разі умовно розділимо тестування на два види: *основне* та *повторне*.

Основним тестуванням вважатимемо тестування дефектів (помилок), яке здійснюється на етапах розроблення та налагодження ПЗ і є частковою складовою цих процесів.

Повторним тестуванням вважатимемо тестування з метою виявлення прихованих помилок, яке здійснюється після розроблення та налагодження ПЗ і є окремим технологічним процесом.

Повторне тестування здійснюється на етапі вхідного контролю, який здійснює замовник, тобто допомагає замовнику оцінити якість тестування програмного забезечення, яке приймається, і вказує на наявність в ньому прихованих помилок.

Щодо розподілу помилок ПЗ взагалі за їх видами і впливом на роботу комп'ютерних систем, то в літературі відомий їх розподіл за пріоритетами і категоріями [2]. Розподіл помилок за пріоритетами здійснюється у відповідності до нормативів з визначення рівня серйозності помилок.

Проведемо уточнення цього підходу щодо опису прихованих помилок ПЗ [3, 4]. Всі приховані помилки розподілимо за видами на незначні (НПП), помірні (ППП), серйозні (СПП) та катастрофічні (КПП) приховані помилки.

Незначними прихованими помилками (НПП) програмного забезпечення вважатимемо такі, що не впливають на дії користувача, програмний продукт з їх наявністю придатний для використання.

Помірними прихованими помилками (ППП) програмного забезпечення вважатимемо такі, що впливають на дії користувача. Програмний продукт з їх наявністю придатний для використання з частковою втратою функційності.

Серйозними прихованими помилками (СПП) програмного забезпечення вважатимемо такі, що призводять до помилкових результатів, внаслідок чого програмний продукт непридатний до використання.

Катастрофічними прихованими помилками (КПП) програмного забезпечення вважатимемо такі, що призводять до спотворення інформації (даних), внаслідок чого програмний продукт непридатний до використання і намагання його опрацювати призводить до відмови програмної системи.

Незначним прихованим помилкам присвоїмо найнижчий рівень категорійності (РК) – перший. Помірним прихованим помилкам присвоїмо, відповідно, рівень 2; серйозним – рівень 3. Найвищим рівнем вважатимемо катастрофічний – рівень 4. Таким чином, рівнів прихованих помилок буде чотири.

Рівні категорійності прихованих типових помилок проектування програмного забезпечення

Відповідно вищеописані типові помилки проектування, які залишаються в програмному продукті після його основного тестування і переходять в розряд прихованих, за допомогою групи експертів з 9 експертів провідної фірми-розробника програмного забезпечення Sitronics Telecom Solutions (Хмельницька філія), можна віднести до наступних рівнів категорійності:

–"Фальшива черепиця" - рідко переходить в розряд прихованих, помірна прихована помилка (2 РК), оскільки досить швидко проявляється в поведінці програми.

–Нульові вказівники та нульове виключення – більшість крахів програм пов'язані якраз з такими різновидом помилок, тому таку помилку можна віднести до катастрофічних прихованих помилок (4 РК).

–"Незакріплений компонент" - помірна прихована помилка (2 РК).

–"Нульовий прапорець" - серйозна прихована помилка (3 РК).

–"Подвійний спуск" - помірна прихована помилка (2 РК).

–"Фальшиве представлення" - дуже часто переходить в розряд прихованих помилок, серйозна прихована помилка (3 РК).

–“Шкідливі дані” - дуже часто переходять в розряд прихованих помилок, оскільки не всі функції звертаються до пошкоджених даних, які знаходяться в системі невизначено довго, не завдаючи шкоди, доки не відбудеться звертання до небезпечного біту даних; катастрофічна прихована помилка (4 РК).

–“Зламаний діспатч” - складно діагностована помилка, оскільки такі помилки можуть з'являтись при додаванні нових методів, причому програма може продовжувати роботу протягом деякого часу, доки не виявиться дана проблема, тому такі помилки часто переходять в розряд прихованих помилок; помірна прихована помилка (2 РК).

–“Тип-самозванець” - рідко переходить в розряд прихованих; незначна прихована помилка (1 РК).

–“Розщеплений чистильник” - помірна прихована помилка (2 РК).

–“Фіктивна реалізація” - дуже часто переходить в розряд прихованих помилок, оскільки може залишатися непоміченою, доки рідко виконуваний шлях програми не виявить її; катастрофічна прихована помилка (4 РК).

–“Осиротілий потік” - серйозна прихована помилка (3 РК).

–“Недостатня ініціалізація” - серйозна прихована помилка (3 РК).

–Помилки, по'язані з залежністю від платформи – рідко стають прихованими помилками; катастрофічні приховані помилки (4 РК).

Система ідентифікації прихованих типових помилок програмного забезпечення

Було розроблено структурну схему системи [5, 6] ідентифікації прихованих помилок програмного забезпечення (рис.1).

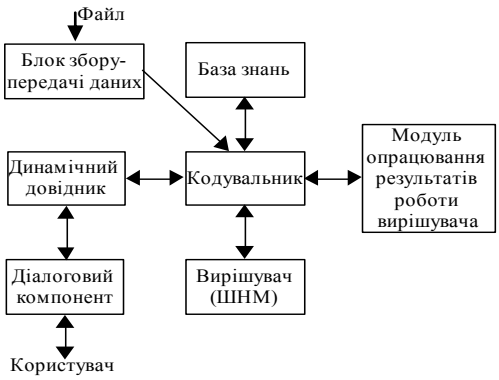

Рис.1. Структурна схема системи ідентифікації прихованих типових помилок проектування програмного забезпечення

Система ідентифікації прихованих помилок програмного забезпечення складається з наступних компонентів:

- блок збору – передачі даних – підключає наданий користувачем файл з результатами основного тестування, представленими у вигляді журналу “Метод тестування – Операція тестування – Різновид виявленої типової помилки проектування”;

- кодувальник – виконує перетворення вхідних даних з лінгвістичної форми представлення в кількісну форму за допомогою таблиць присвоєння номерів методам, операціям основного тестування та різновидам типових помилок проектування бази знань, заповнення таблиці кількісного представлення вхідних даних бази знань вхідними даними та формування вхідних векторів для модуля вирішувача. Кодувальник також перевіряє вхідні дані на правильність та достатність при формуванні вхідних векторів; якщо дані недостатні або вони невірні, то кодувальник передає динамічному довідникові своє повідомлення з пропозицією сформувати ще один файл такої ж форми, як і попередній, з додатковими результатами, що перетворюються в кількісну форму аналогічно даним основного файлу, після чого заносяться в базу знань. Здійснюється заповнення бази знань вихідними даними, перетворення результуючих векторів вирішувача з кількісної в лінгвістичну форму за допомогою таблиці присвоєння номерів рівням категорійності прихованих помилок бази знань та передачу їх модулю опрацювання результатів роботи вирішувача;

- база знань – містить таблиці присвоєння номерів методам і операціям основного тестування, різновидам типовиих виявлених помилок проектування та присвоєння номерів рівням категорійності прихованих помилок; таблицю кількісного представлення вхідних даних, в якій містяться вхідні дані, перетворені кодувальником в кількісну форму; таблицю текстового представлення результуючих векторів вирішувача (ШНМ), в якій представлені результуючі вектори, перетворені кодувальником в лінгвістичну форму; таблиці відповідності методу основного тестування, операцій основного тестування, типів виявлених під час основного тестування помилок, відповідності між номером методів тестування ПЗ та рівнем категорійності прихованих помилок ПЗ, відповідності між

операціями тестування ПЗ та рівнем категорійності прихованих помилок, на основі яких система формує висновок про метод, яким рекомендується проводити повторне тестування прикладного ПЗ, а також правила для формування висновку про необхідність та метод повторного тестування;

- вирішувач – штучна нейронна мережа [3, 4, 5], на входи якої подається інформація про методи і операції основного тестування та різновиди виявлених під час основного тестування типових помилок проектування, а на виході одержується рівень категорійності прихованих помилок;

- модуль опрацювання результатів роботи вирішувача – на основі правил та таблиці результатів роботи вирішувача, взятих з бази знань, генерує висновок про необхідність та метод повторного тестування, який передається користувачу через кодувальник, динамічний довідник та діалоговий компонент;

- динамічний довідник – надає користувачу довідку про формат вхідного файлу, про відомі системі методи і операції основного тестування ПЗ, різновиди типових виявлених під час основного тестування помилок проектування ПЗ, а також представляє в наглядній формі всі повідомлення будь-якого з компонентів системи;

- діалоговий компонент –візуалізує повідомлення динамічного довідника та видає їх користувачу в зрозумілій для сприйняття формі.

Запропонована система ідентифікації прихованих помилок програмного забезпечення дозволяє користувачу, на основі звіту про результати основного тестування, одержати висновок про необхідність повторного тестування, а саме: про наявність у програмному забезпеченні прихованих типових помилок проектування та про метод, яким рекомендується здійснювати повторне тестування.

Систему ідентифікації прихованих помилок програмного забезпечення було реалізовано в Borland C++ Builder 6.0 із застосуванням системи управління базами даних Paradox7.

Після активізації системи користувачем на екрані з’являється наступне вікно (рис.2).

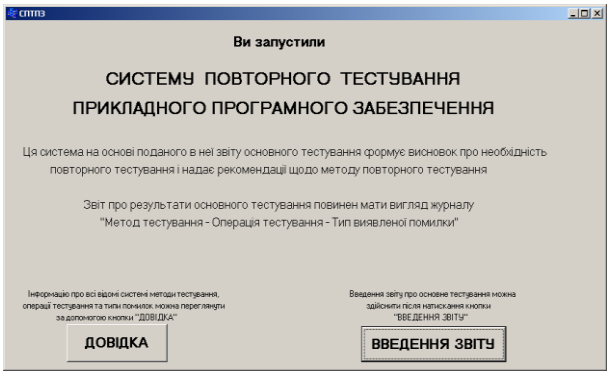

Рис.2. Перше діалогове вікно системи ідентифікації прихованих типових помилок проектування програмного забезпечення

При натисканні в першому діалоговому вікні кнопки “ВВЕДЕННЯ ЗВІТУ” користувач має змогу подати по рядку звіт в систему повторного тестування прикладного програмного забезпечення.

Після введення одного рядка звіту потрібно натиснути кнопку “Запам’ятати обрані дані”, яка дозволяє перетворити обрані текстові дані в кількісну форму. Після цього натискаємо кнопку “Наступний рядок” для введення наступного рядка звіту. Якщо ж введення звіту закінчено, потрібно натиснути кнопку “Закінчити введення звіту і передати введений звіт на вирішувач”.

Для формування висновку вводяться величини порогових значень кількостей помилок кожного рівня категорійності.

Порогові значення вводяться на основі евристичних оцінок, зроблених на основі опрацювання експертної інформації, наступним чином:

1.якщо кількість помилок 4-го рівня категорійності (катастрофічних) перевищує 1, то повторне тестування необхідне за причини можливості відмови програмної системи;

2.якщо кількість помилок 3-го рівня категорійності (серйозних) перевищує 2, то повторне тестування необхідне за причини виникнення помилок вищого рівня категорійності;

3.якщо кількість помилок 2-го рівня категорійності (помірні) дорівнює або перевищує 50% від загальної кількості виявлених під час основного тестування помилок, то повторне тестування необхідне за причини виникнення помилок вищих рівнів категорійності;

4.якщо кількість помилок 1-го рівня категорійності (незначні) дорівнює або перевищує 75% від загальної кількості виявлених під час основного тестування помилок, то повторне тестування необхідне за причини виникнення помилок вищих рівнів категорійності.

Після аналізу поданого в систему звіту основного тестування із застосуванням введених порогових значень система видає висновок, потрібне чи не потрібне повторне тестування і рекомендує метод проведення повторного тестування, якщо воно потрібне.

Висновки

В статті розглядались типові помилки проектування програмного забезпечення за Алленом, їх симптоми прояву та причини появи. На основі цих причин появи сформовано методи та операції виявлення типових помилок проектування програмних продуктів, потрібні для створення таблиць бази знань системи ідентифікації прихованих типових помилок проектування програмного забезпечення. Введено поняття прихованої помилки, рівнів категорійності прихованих типових помилок проектування ПЗ, а також поняття та призначення повторного тестування. На основі експертних даних прихованим типовим помилкам проектування ПЗ призначено рівні категорійності, які необхідні для навчання ШНМ, що входить до складу системи ідентифікації прихованих типових помилок проектування програмного забезпечення, та для визначення системою, чи потрібне повторне тестування даного програмного продукту. Створено структуру системи ідентифікації прихованих типових помилок проектування програмного забезпечення, яка на основі опрацювання звіту основного тестування програмного продукту дає висновок про необхідність повторного тестування та його методи.

Література

1. *Э.Аллен*. Типичные ошибки проектирования. Библиотека программиста. - СПб.: Питер, 2003. - 224 с.
2. *Калбертсон Р., Браун К., Кобб Г.* Быстрое тестирование: Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 384с.
3. *Локазюк В.М., Пантелєєва (Говорущенко) Т.О.* Категорійна модель процесу повторного тестування дефектів програмного забезпечення // Вісник Технологічного університету Поділля – Хмельницький: ТУП, 2004. – ч.1, т.1, с. 53 – 58
4. *V.Lokasyuk, O.Pomorova, T.Govorushchenko.* Neural Nets Method for Estimation of the Software Retesting Necessity // Proceedings of the 2008 international workshop on Software Engineering in east and south Europe – Germany, Leipzig, 2008. – pp. 9-14. ISBN 978-1-60558-076-0. (*http://doi.acm.org/10/1145/1370868.1370871*)
5. *Говорущенко Т.О.* Підвищення достовірності тестування програмного забезпечення // Вісник Національного університету “Львівська політехніка” “Комп’ютерні науки та інформаційні технології” – Львів: Видавництво Національного університету “Львівська політехніка”, 2007 – с.186-196
6. *Говорущенко Т.О.* Реалізація та функціювання системи повторного тестування прикладного програмного забезпечення // Вісник Хмельницького національного університету – Хмельницький: ХНУ, 2007 - №2, т.2, с. 113-120

Perevoznikov S.I., Govorushchenko T.O. Hidden Typical Software Designing Bugs Identification System.