УДК 004.415.532

І. С. СКАРГА-БАНДУРОВА, Я. П. КОВАЛЕНКО

Східноукраїнський національний університет ім. В. Даля, Україна

ТЕХНОЛОГІЧНІ АСПЕКТИ СТАТИЧНОГО АНАЛІЗУ КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ ЗАЛІЗНИЧНОЇ АВТОМАТИЗАЦІЇ

Розглянуті особливості тестування програмного забезпечення систем залізничної автоматизації. Вперше запропоновано модель для тестування функціонального програмного забезпечення системи мікропроцесорної централізації МПЦ-У на основі автомату потоку управління. Математична модель, що складається з внутрішніх проміжних структур забезпечує адекватне урахування особливостей програмного забезпечення МПЦ-У і дозволяє проводити моделювання роботи програми в її природному оточенні. Викладено загальну методику пошуку помилок програмного забезпечення у вигляді комбінаційних схем. Одержані результати є основою для розроблення спеціалізованого програмного засобу, що дозволяє уникнути використання додаткових апаратних витрат і гарантує практично повне покриття всіх ділянок коду за рахунок аналізу потоку управління.

Ключові слова: *статичний аналіз, модель, потік даних, потік управління, тестування, код, програмне забезпечення, методика пошуку помилок.*

Вступ. Постановка проблеми

З ростом автоматизації на залізних дорогах України проблема тестування програмних компонент керуючих систем стає все більш важливою. Для забезпечення безпеки і надійності залізничного руху всі компоненти програмного забезпечення мікропроцесорних систем управління МПЦ-У повинні бути кваліфіковані на основі відповідних процесів верифікації та валідації.

Як зазначено в роботі [1], випробування програмного забезпечення МПЦ-У є відносно простими, але дорогими, стомлюючими і, більшою мірою, здійснюються людьми з залученням великого обсягу коду. Проблема тестування обумовлена також специфікою мов програмування МПЦ-У, які значно відрізняються від мов загального призначення С, С ++, С # або Java, і, як наслідок, відсутністю належних інструментів аналізу. І, нарешті, програмне забезпечення МПЦ-У складається з різних модулів та блоків (контроль рейкових кіл, управління стрілками, управління маршрутами і т.д.) і реалізується різними групами кодувальників, які беруть участь в розробці, що створює додаткові труднощі для контролю якості програмного забезпечення. Означені проблеми обумовлюють актуальність пошуку нових технічних рішень, які дозволяють підвищити якість програмного забезпечення, а також скоротити час і витрати на розробку програмного забезпечення.

Метою статті є представлення розроблених моделей і методів тестування функціонального програмного забезпечення (ФПЗ) системи мікропроцесорної централізації МПЦ-У, що впроваджується на Укрзалізниці на основі методу статичного тестування.

1. Огляд літературних джерел

Статичний аналіз є особливо потужним засобом, коли мова заходить про пошук помилок безпеки. Застосування статичного тестування для типових програм, дозволяє знаходити від 30% до 80% помилок логічного проектування і кодування [2, 3]. Даний метод сприяє суттєвому підвищенню продуктивності і надійності програм, дозволяє раніше виявити помилки, і, як наслідок, зменшити вартість виправлення.

Інструменти статичного аналізу використовуються в якості додаткового засобу до звичайних динамічних випробувань з метою отримання гарантій якості і безпеки розробки програмного забезпечення критичних систем, в тому числі, для пошуку коду, що потенційно містить уразливості [4]. В цьому контексті, статичне тестування повинно забезпечити перевірку внутрішньої несуперечності і повноти реалізації вимог і підтвердити відповідність програмного модуля, що випробовується вихідним функціональним вимогам, а також виявити всі невідповідності та відмінності між очікуваними і отриманими результатами для подальшого дослідження і усунення.

При статичному тестуванні програмний код не виконується, аналіз програм проводиться на основі вихідного коду, який вираховується вручну, або аналізується спеціальними інструментами. При цьому перевіряється правильність побудови елементів програми і їх взаємодія один з одним, виконується перевірка правил структурної побудови програми і обробки даних [5]. Тестуванню мають підлягати: внутрішні структури даних; розгалуження; обробки

помилкових ситуацій; інтерфейс модуля; граничні умови.

Серед існуючих методів статичного аналізу програмного забезпечення виділяють аналіз тексту вручну; машинну обробку тексту без формування допоміжних даних; обробку з використанням проміжних структур. Істотним недоліком аналізу коду вручну є її вкрай висока вартість і низька швидкість. Машинна обробка тексту без формування допоміжних даних являє собою пошук за регулярними виразами і пошук шаблонів типових конструкцій. Такий аналіз не може дати повноцінного уявлення про поведінку програми при виконанні, тому має досить обмежене застосування. На даний момент найбільш ефективним методом статичного аналізу є обробка коду з використанням внутрішніх проміжних структур (проміжного подання) [5]. За своєю суттю проміжне подання - це набір даних, що створюється програмним аналізатором, на основі яких виконується аналіз. Проміжне подання еквівалентно вихідному коду за заданими критеріями і може задаватися в різних формах, що використовують різні моделі представлення коду: реляційній [5], за допомогою кінцевих автоматів [6], синтаксичне дерево розбору [7], дерево Канторовича або абстрактне синтаксичне дерево (abstract syntax tree, AST) [8], граф потоку керування (control flow graph, CFG) [9], граф залежності за даними (data flow graph, DFG) [5], модель одноразового статичного присвоювання (static single assignment, SSA) [5], абстрактний семантичний граф (abstract semantic graph, ASG) [7] тощо.

2. Завдання дослідження

– аналіз особливостей тестування ПЗ для систем залізничної автоматизації;

– вибір способу представлення вихідного коду функціональних програм для управління системами залізничної автоматизації;

– розробка моделей тестування ФПЗ МПЦ-У на основі методу статичного тестування;

– розробка методики пошуку помилок ПЗ систем залізничної автоматизації.

3. Особливості тестування ПЗ систем залізничної автоматизації

Для тестування функціонального програмного забезпечення комп'ютерних систем високої надійності, зокрема систем мікропроцесорної централізації МПЦ-У проміжне подання задається у вигляді типових внутрішніх наборів даних. Таке подання вважається оптимальним, тому що операторами використовуваної технологічної мови є логічні функціональні блоки.

Алгоритми, на підставі яких виконується кодування, реалізуються у вигляді комбінаційних схем. Операторами технологічної мови є певні макрокоманди, які мають наступний вигляд:

```
## <Ім'я макрокоманди> [<ім'я функції>] {<пере-
          лік вхідних параметрів>},
       {<перелік вихідних параметрів>},
     {<перелік настроювальних параметрів>}
```

Нижче наведено приклад опису комбінаційних схем (рис. 1, 2) та їх кодування на мові технологічного програмування для обробки сигналів стану контактів реле схеми сполучення з ключем-жезлом.

Обробка сигналів стану контактів реле схеми сполучення з ключем-жезлом з контролем на достовірність і допустимість комбінацій стану контактів реле виконується за наступними параметрами: достовірність сигналів (<ім'я перегону>_KG.STCP) і стан ключа-жезла (<ім'я перегону>_KG.SOST) визначаються станом нормально замкнутого контакту реле сигналу KG (SR_<ім'я перегону>_KG.NZKD), станом контактів реле сигналу KG (SR_<ім'я перегону>_KG.SD) і ознакою достовірності сигналу (SR_<ім'я перегону>_KG.STCP):

{сигнал стану ключа-жезла недостовірний (<ім'я перегону>_KG.STCP = 1)},
якщо
{контакти реле сигналу від ключа-жезла
непрацездатні (SR_<ім'я перегону>_KG.SD=1)
або сигнал від ключа-жезла недостовірний (SR_<ім'я перегону>_KG.STCP=1},
інакше
{сигнал стану ключа-жезла достовірний
(<ім'я перегону>_KG.STCP=0)}.

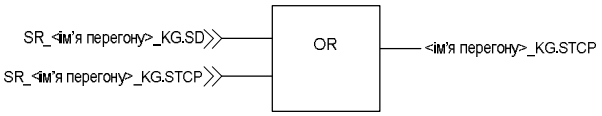


Рис. 1. Логічна схема формування параметру <ім'я перегону>_KG.STCP

{стан ключа-жезла - "відсутній" (<ім'я перегону>_KG.SOST = 1)},
якщо
{встановлений сигнал КЖ (НЗ контакт реле КЖ замкнутий
(SR_<ім'я перегону>_KG.NZKD =1) і сигнал достовірний(<ім'я перегону>_KG.STCP=0)},
інакше
{стан ключа-жезла - "на місці" (<ім'я перегону>_KG.SOST = 0)}.

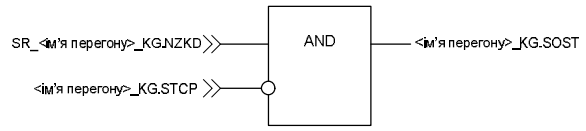


Рис. 2. Логічна схема формування параметру <ім'я перегону>_KG.SOST

Лістинг файлу прикладної програми PG_KG.dfb для обробки сигналів стану контактів реле схеми сполучення з ключем-жезлом з контролем на достовірність і допустимість комбінацій ста-

ну контактів реле має наступний вигляд:

```
#include "pg.inc";
#DECLARE_FB "Взаємодія з ключом-жезлом"
/*Оголошення формату оператора виклику
#DFB_CALL PG_KG {
/*вхідні сигнали від схеми сполучення з
перегоном
SR_KG=A1  /* Сигнали від ключа-жезла
},{
/*Вихідні параметри, сформовані алгоритмом
KG=B1, /* Стан ключа-жезла
}
/* Оголошення кількості параметрів
#DFB_ALL PG_KG {
/* вхідні сигнали від схеми сполучення з
ключем-жезлом
X1,X2,X3 /* A1 Сигнали від ключа-жезла
(KG) .NZKD, .SD, .STCP
},{
/*Вихідні параметри, сформовані алгоритмом
Y1,Y2 /* B1 Стан ключа-жезла .SOST, .STCP
},{},{}
/* Оголошення характеристик вхідних пара-
метрів
#TYPEX X1={BOOL,A1.NZKD}, /* Значення сиг-
налу КЖ .NZKD
X2={BOOL,A1.SD}, /* Стан контактів реле
сигналу КЖ .SD
X3={BOOL,A1.STCP} /* Ознака достовірності
сигналу
КЖ .STCP
/* Оголошення характеристик вихідних пара-
метрів
#TYPEY
Y1={GROUP="YP_PG_AB",B1.SOST,DECL},/* Стан клю-
ча-жезла
Y2={GROUP="YP_PG_AB",B1.STCP,DECL} /* Дос-
товірність стану ключа-жезла
/* ОПИС АЛГОРИТМУ СЕКЦІЇ ВИКОНАННЯ
#BEGIN
   ##OR {X2,X3},{Y2},{}
   ##AND_NOT1 {Y2,X1},{Y1},{}
#END
#END_DECLARE_FB
```

Лістинг файлу PG.inc, що містить оголошення групи атрибутів параметрів, необхідних для прикладної програми:

```
/* атрибути, породжувані PG_KG.dfb
#TYPEDEF_GROUP
P_PG_AB{SOST:BOOL,STCP:BOOL}
```

Лістинг файлу виклику прикладної програми PG_KG.itx:

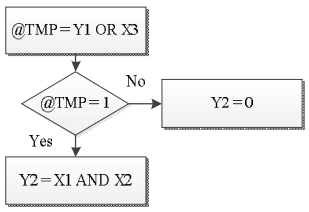
```
#SECTION_1 HIDDEN
#PART1: "Частина 1"
 #CALL PG_KG {
/* вхідні сигнали від схеми сполучення з
перегоном
SR_KG=SR_PGCB_KG,  /* Сигнал від ключа-
жезла
},{
/* Вихідні параметри, сформовані алгорит-
мом
KG=PGCB_KG,       /* Стан ключа-жезла
}
#SECTION_END
```

В процесі тестування проводиться виділення і розпізнавання обов'язкових базових лексем (#DECLARE_FB, #DFB_CALL, #DFB_ALL, #TYPEX, #TYPEY, #BEGIN, #END, #END_DECLARE_FB), що знаходиться у вхідній послідовності символів. У разі коректного опису виконується подальший аналіз макрокоманд секції виконання. У разі неуспішного розпізнавання лексеми користувачеві видається повідомлення про помилку і подальше розпізнавання припиняється.

4. Математичні об'єкти для тестування програмного забезпечення

Для цілей роботи вихідний код програми був представлений у вигляді автомата потоку управління [10], що описує набір станів операторів і переходи між ними. В автоматі потоку управління кожен вузол автомата відповідає базовому блоку – прямолінійній ділянці коду. Приклад автомата потоку управління відповідного фрагменту вихідного коду наведено на рис. 3.

```
.....
##OR{Y1,X3},{@TMP},{}
#IF {@TMP,""}
     ##AND
{X1,X2},{Y2},{}
#ELSE
   ##ASSIGN
{0},{Y2},{}
#END_IF
.....
```



а б

Рис. 3. Приклади: а – фрагмент вихідного коду, б – автомат потоку управління

Для реалізації аналізу вихідного коду на підставі синтаксису мови складено потік управління і потік даних, що представляють собою модель виконання програми і модель залежностей змінних, що містить в собі перевірку коректності імен, а також відповідності типів і значень для заданих параметрів.

Модель потоку управління складається з внутрішньої проміжної структури (ВПС) і містить множину операторів секції виконання O_s і множину значень позицій для операторів розгалуження S_p.

$$M_{c.f.} = <O_s, S_p>. \quad (1)$$

Значення позицій реалізовують перехід для непрямолінійних ділянок коду в залежності від стану змінної визначає умови розгалуження.

Модель потоку даних являє собою ВПС опису формальних змінних вихідної програми. Дана модель містить множину формальних імен параметрів

D, типів параметрів T, формальних імен операндів N, імен атрибутів A, і множину значень змінних V:

$$M_{d.f.} = <D, T, N, A, V>. \quad (2)$$

При запуску моделювання роботи програми, модуль потоку управління виконує інтерпретацію і-го заданого логічного оператора секції виконання O_{si}, грунтуючись на описі роботи всіх операторів мови, і перевіряючи коректність змінних і їх значень, що описуються моделлю (2). Сформовані ВПС зберігаються в динамічній пам'яті програми і необхідні для виконання подальшого аналізу і подальшої обробки. Взаємодія ВПС в моделі потоку даних показана на рис. 4.

Маючи описані дані можна проводити моделювання роботи програми в її природному оточенні. Інтерпретація проводиться згідно логіки роботи поточного оператора <ім'я макрокоманди> та на підставі лексем <ім'я функції>, <перелік вхідних параметрів>, <перелік вихідних параметрів>, <перелік настроювальних параметрів>, виконується аналіз переліку вхідних, вихідних і настроювальних параметрів на наявність коректних імен, а також аналізуються типи та їх відповідності для заданих параметрів. Далі аналізується встановлений режим роботи інтерпретатора (покроково, зупинка по контрольним точкам, або за кількістю тактів). Кожен оператор описаний у вигляді окремої процедури, яка виконує його аналіз згідно з правилами синтаксису технологічної мови. Якщо поточний оператор відповідає коректному оператору мови, то виконується запуск відповідної процедури, яка виконує всі необхідні перевірки правильності опису оператора і, при відсутності помилок, логічну інтерпретацію оператора.

5. Загальна методика пошуку помилок ПЗ систем залізничної автоматизації

Для вирішення завдання добування інформації з вихідного тексту був складений список необхідних ключових дій, а саме:

– визначити порядок і правильність опису елементів програми;

– визначити всі змінні програми;

– визначити логічні оператори;

– зіставити опис програми та її виклик.

Витяг і подальша обробка інформації з вихідних даних виконується в наступній послідовності:

1. Аналіз параметрів запуску програми:

– видалення коментаря з вихідного тексту програмного модуля і запис у ВПС;

– видалення коментаря з виклику програмного модуля і запис в ВПС.

2. Пошук позицій ключових макрокоманд і аналіз порядку їх слідування.

Оператори конструктора прикладної програми (ДФБ) призначені для опису алгоритмів і правил виклику, визначених користувачем. Оператори оголошення ДФБ можуть розділятися рядками коментаря. Структуру оголошення ДФБ надано на рис. 5. Секції в структурі ДФБ, які відзначені пунктирними лініями можуть бути відсутніми.

На рис. 6 вказані можливі структури програмного модуля в залежності від внутрішньої організації програми:

–наявність секції ініціалізації (#BEGIN_INIZ, #END_INIZ);

–змінних для відновлення параметрів з історією (#RABP);

– настроювальних змінних (#TYPEP).

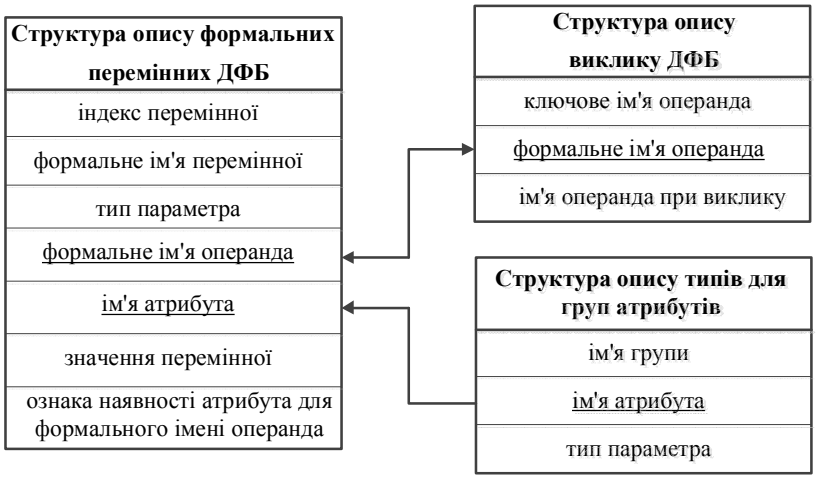


Рис. 4. Взаємодія ВПС в моделі потоку даних

Рис. 5. Структура програмного модуля ДФБ

3. Пошук допоміжних макрокоманд призначених для опису груп атрибутів і перевірка їх розташування, а також запис у ВПС.

4. Запис операторів секції виконання в ВПС, шляхом виділення команд із загальної структури.

5. Відображення операторів секції виконання з ВПС, нумерація рядків. У результаті, з ВПС видаляються всі оператори, починаючи з позиції макрокоманди опису секції виконання.

6. Розбір макрокоманди, яка оголошує форму виклику програмного модуля. Отримання переліків формальних операторів виклику. Виконання підрахунку кількості операндів, запис в масиви ключових і формальних імен.

7. Розбір макрокоманди оголошення кількості параметрів. Отримання списку і підрахунок кількості змінних. Заповнення відповідних масивів. Розбір макрокоманд оголошення характеристик усіх параметрів. Дана процедура здійснює запис для відповідних формальних імен - типу, атрибута, та установку початкового значення змінних у масивах.

Фрагмент алгоритму розбору формального імені змінної - <ФІП> наведено на рис. 7, де <ПІБ> - формальне ім'я операнда, <ІА> - ім'я атрибута.

8. Пошук робочих змінних поточного такту роботи програмного модуля:

– пошук в ВПС вихідного коду програми опису типу робочих змінних;

– аналіз порядку проходження опису робочих змінних щодо макрокоманд;

– запис у масив опису робочих змінних ідентифікатора робочої змінної, типу робочої змінної, установка початкового нульового значення.

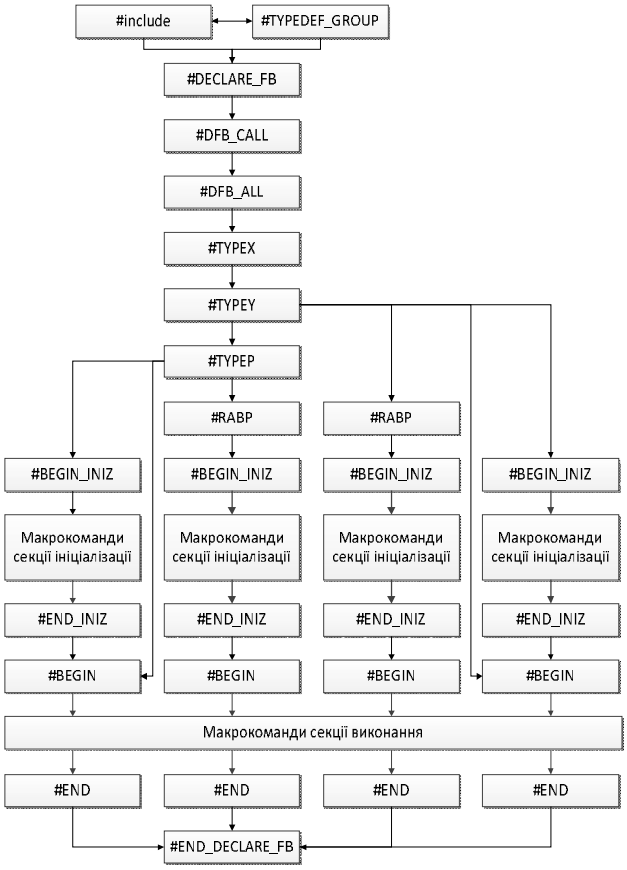


Рис. 6. Схема аналізу структури програмного модуля ДФБ

9. Пошук позицій ключових макрокоманд і аналіз порядку їх слідування в описі виклику прикладної програми.

10. Розбір макрокоманди виклику програмного модуля, для отримання формату виклику операндів. Запис в масив виклику операндів для відповідних списків. При наявності настроювальних змінних, виконана установка початкових значень для них.

11. Пошук позицій операції розгалуження в секції виконання.

12. Видалення попереднього лог-файлу, що містить вихідний код програмного модуля.

13. Послідовна перевірка всіх макрокоманд секції виконання.

14. Обнуління значень змінних після початкової перевірки.

15. При наявності секції початкової ініціалізації змінних, запис її операторів у відповідну ВПС.

16. Пошук позицій операції розгалуження в секції ініціалізації.

17. Інтерпретація макрокоманд секції ініціалізації.

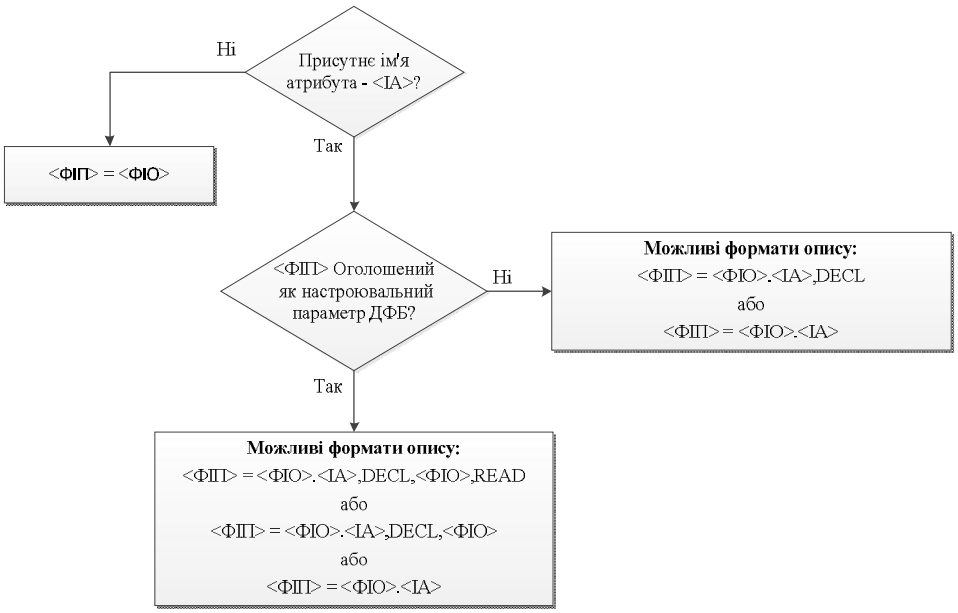


Рис. 7. Алгоритм розбору формального імені змінної

18. Створення потоку для контролю змін значень змінних.

19. Формування списку всіх змінних для візуального відображення.

20. Пошук позицій операції розгалуження в секції виконання.

Запропонована методика реалізована у вигляді програмного засобу статичного тестування і на даний момент проходить впровадження на ПрАТ "СНВО "Імпульс" (м. Сєвєродонецьк).

Висновки

Розглянуті технологічні аспекти статичного аналізу коду програмного забезпечення систем залізничної автоматизації дозволили розробити математичну модель для тестування ФПЗ МПЦ-У і створити методику пошуку помилок у вихідному коді, алгоритми яких реалізуються у вигляді комбінаційних схем. Запропонований підхід дозволяє уникнути використання додаткових апаратних витрат і, за рахунок аналізу ВПС та використання моделей потоку управління та потоку даних гарантує майже повне покриття всіх ділянок коду. Подальші дослідження спрямовані на створення математичних моделей для автоматичного аналізу додаткових умов в програмах, що виникають як при істинних, так і при помилкових значеннях.

Література

1. Microprocessor Based Protection Systems [Text] / ed. A. R. Churchley. – Springer, 1992. – 290 p.

2. Dusting, E. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality [Text] / E. Dustin, T. Garrett, B. Gauf . – Addison-Wesley Professional, 2009. – 368 p.

3. Котляров, В. П. Основы тестирования программного обеспечения : учеб. пособие [Текст] / В. П. Котляров, Т. В. Коликова – М. : Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2006. – 285 с.

4. Livshits, B. Improving Software Security with Precise Static and Runtime Analysis [Text] : Ph.D. dissertation / B. Livshits. – Stanford Univ. – USA, 2006. – 229 p.

5. Глухих, М. И. Программная инженерия. Обеспечение качества программных средств методами статического анализа. : учеб. Пособие [Текст] / М. И. Глухих, В. М. Ицыксон – СПб: Изд-во Политехн. ун-та, 2011. – 150 с.

6. Опалева, Э. А. Языки программирования и методы трансляции [Текст] / Э. А. Опалева, В. П. Самойленко; СПб. : БХВ-Петербург, 2005. – 480 с.

7. Aho, A. V. Compilers: Principles, Techniques, and Tools [Text] / A. V. Aho , M. S. Lam, R. Sethi, J. D. Ullman ; second. ed. – Addison Wesley, 2006. – 1000 p.

8. Липаев, В. В. Тестирование компонентов и комплексов программ. [Текст] / В. В. Липаев. – М. : СИНТЕГ, 2010. – 400 с.

9. Schwartzbach, M. I. Lecture Notes on Static Analysis [Електронний ресурс] / M. I. Schwartzbach, University of Aarhus, Denmark - Режим доступу: http://www.itu.dk/people/brabrand/UFPE/Data-Flow-Analysis/static.pdf – 20.03.2016 р.

10. Kovalenko, Y. Software and algorithmic solutions for static testing software components of the safety-critical systems [Text] / Y. Kovalenko, I. Skarga-Bandurova // Theoretical and Applied Computer Science and Information Technology: Proc. of the 1st Int. Conf. TACSIT-2015. – Severodonetsk: EUNU, 2015. – pp. 54-57.

References

1. Churchley A. R. (Ed.) *Microprocessor Based Protection Systems*. Springer, 1992. 290 p.

2. Dusting, E, Garrett, T., Gauf, B. *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison-Wesley Professional, 2009. 368 p.

3. Kotlyarov, V. P., Kolikova, T. V. *Osnovy testirovaniya programmnogo obespecheniya* [Fundamental software testing] Internet-universitet informatsionnyh tekhnologii; BINOM. Laboratoriya znanii Publ., 2006. 285 p.

4. Livshits, B. *Improving Software Security with Precise Static and Runtime Analysis*. Ph.D. dissertation. Stanford Univ., USA, 2006. 229 p.

5. Glukhikh, M. I., Itsykson, V. M. *Programmnaya inzheneriya. Obespechenie kachestva programmnykh sredstv metodami staticheskogo analiza*. [Software Engineering. Quality assurance by methods of static analysis]. SPb, Politehn. Univ., 2011. 150 p.

6. Opaleva, E. A., Samoilenko, V. P. *Yazyki programmirovaniya i metody translyatsii* [Programmimg languages and address mapping]. SPb. BHV-Peterburg Publ., 2005. 480 p.

7. Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D. *Compilers: Principles, Techniques, and Tools* second. ed. Addison Wesley, 2006. 1000 p.

8. Lipaev, V. V. *Testirovanie komponentov i kompleksov programm*. M. : SINTEG Publ., 2010. – 400 p.

9. Schwartzbach, M. I. *Lecture Notes on Static Analysis*. University of Aarhus, Denmark - Available at: http://www.itu.dk/people/brabrand/UFPE/Data-Flow-Analysis/static.pdf (accessed 20.03.2016).

10. Kovalenko, Y., Skarga-Bandurova, I. Software and algorithmic solutions for static testing software components of the safety-critical systems *Proc. of the 1st Int. Conf. "Theoretical and Applied Computer Science and Information Technology" TACSIT-2015.* – Severodonetsk, 2015. pp. 54-57.

Надійшла до редакції 21.03.2016, розглянута на редколегії 14.04.2016

ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ СТАТИЧЕСКОГО АНАЛИЗА КОДА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ ЖЕЛЕЗНОДОРОЖНОЙ АВТОМАТИЗАЦИИ

И. С. Скарга-Бандурова, Я. П. Коваленко

Рассмотрены особенности тестирования программного обеспечения железнодорожной автоматизации. Дан краткий обзор методов статического анализа. Впервые предложена модель тестирования функционального программного обеспечения для системы микропроцессорной централизации МПЦ-У на основе автомата потока управления. Математическая модель, состоящая из внутренних промежуточных структур, позволяет проводить моделирование работы программы в ее естественном окружении. Изложена общая методика поиска ошибок программного обеспечения в виде комбинационных схем. Полученные результаты являются основой для разработки специализированного программного средства, позволяющего избежать использования дополнительных аппаратных затрат и гарантирует практически полное покрытие всех участков кода за счет анализа потока управления.

Ключевые слова: статический анализ, модель, поток данных, поток управления, тестирование, код, программное обеспечение, методика поиска ошибок.

TECHNOLOGICAL ASPECTS OF STATIC CODE ANALYSIS FOR SOFTWARE COMPONENTS OF MICROPROCESSOR INTERLOCKING SYSTEM

I. S. Skarga-Bandurova, Y. P. Kovalenko

The paper discusses the features of railway automation software testing. A brief review of methods for static code analysis is discussed. The model based on the control flow graph is proposed for testing the software components of microprocessor interlocking system. The proposed mathematical model consists of the inner intermediate structures and allows simulate the work of program in its natural environment. The general research methodology of finding software errors is represented in the form of combinational circuits. The results allow avoiding additional hardware costs, and ensure complete coverage of all code areas through control flow analysis.

Key words: static analysis, model, data flow, control flow, test, code, software, error finding techniques.

Скарга-Бандурова Інна Сергіївна – д-р техн. наук, проф. кафедри комп'ютерної інженерії (КІ) Східноукраїнського національного університету ім. В. Даля, e-mail: skarga_bandurova@ukr.net.

Коваленко Ян Павлович – аспірант, співр. науково-дослідної групи при кафедрі КІ Східноукраїнського національного університету ім. В. Даля, e-mail: kovalenko16yan@gmail.com.

Inna S. Skarga-Bandurova – Doctor of Engineering Science, Professor at the Computer Engineering (CE) Department of V. Dahl East Ukrainian National University, Severodonetsk, Ukraine, e-mail: skarga_bandurova@ukr.net.

Yan P. Kovalenko – PhD student, researcher of the R&D group at the CE Department of V. Dahl East Ukrainian National University, Severodonetsk, Ukraine, e-mail: kovalenko16yan@gmail.com.