УДК. 004.054

Павло Юрійович Катін

МЕТОДИКА ПОБУДОВИ МАТЕМАТИЧНИХ МОДЕЛЕЙ ДЛЯ ДОКУМЕНТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІМПЕРАТИВНИХ МОВ ПРОГРАМУВАННЯ

Одним з актуальних питань, яке досі не вирішено, є забезпечення безпеки програмного продукту (ПП), при розробці програмного забезпечення (ПЗ), в тому числі військового призначення. Загальновідомий факт, що уразливі місця в ПП зустрічаються навіть у відомих розробників ПЗ. Сказане підтверджує процес постійного вдосконалювання ПЗ шляхом встановлення "латок" (patch) на готові ПП. Одним зі способів забезпечення безпеки ПП у ході його вдосконалення є його коректне документування. Воно дозволяє виправити виявлені в ході експлуатації ПП помилки, проводити тестування для поліпшення якості ПП [1—3]. Важливість документування ПЗ в військової сфері підтверджує наявність військових стандартів по документуванню ПЗ, прикладом якого є MIL-STD-498. Однак, незважаючи на практичну важливість стандартів, вони мають загальноорганізаційний і рекомендаційний характер.

Аналіз публікацій і літератури показує [4—5], що сьогодні найпоширенішим способом документування ПП для його тестування є фрагмент коду з коментаріями. В [6] доступно описаний процес блокового тестування ПП на імперативній мові програмування С++ з документуванням у вигляді коду з коментаріями.

Але такий спосіб документування має певні недоліки. До них відносяться: недостатня універсалізація документування, неможливість використання математичних методів аналізу, необхідність глибокого знання імперативної мови програмування (ІМП) експертами, які проводять тестування.

Метою цієї статті є розробка методики побудови математичних моделей (ММ) для документування ПЗ імперативних мов програмування з використанням математичного апарату мереж Петрі (МП). Для досягнення поставленої мети пропонується формалізація блоків ПП у вигляді МП. Практична побудова ММ здійснюється для мови С++, проте розроблена методика придатна для інших ІМП.

Аналіз останніх досліджень у даному напрямку показує, що є роботи, в яких реалізовані спроби використання МП на етапі проектування. Далі математична модель МП перекладається в програмний код [7—10]. Проте способи використання мереж Петрі для документування ПП є недостатньо вивченими.

Формальна постановка завдання. Необхідно розробити методику перетворення блоку програмного коду ІМП у формальні конструкції МП. Потрібно обумовити особливості роботи МП, що формалізує блок коду.

Введемо певні обмеження на ММ, оскільки МП є формальною математичною моделлю.

Розглянута методика призначена для опису роботи ІМП. Використовується одноріневна МП, всі її спрацьовування послідовні. На відміну від класичних МП в методиці передбачається наявність механізму упорядкування спрацьовування переходів. Цей механізм впроваджується у вигляді призначення для переходів певної ієрархії. Передбачається наявність строгої початкової розмітки перед виконанням МП і місткість для кожного з місць.

Введемо формальний опис МП із використанням відомої теорії [11—12]. Графічно МП представляється у вигляді двочасткового орієнтованого графа з двома типами вершин — місцями й переходами. Місце позначається у вигляді кола, перехід — рискою або прямокутником. У цілому, одноріневна МП повністю описується наступними елементами: множиною позицій, переходів, вхідних дуг, вихідних дуг, міток мережі.

У математично формалізованому виді [11—13] МП є впорядкованою множиною, що включає чотири елементи: $C=(P, T, I, O)$, де $P=\{p_1, p_2, p_3, p_4, ..., p_n\}$ є кінцевою множиною місць $n \geq 0$, а $T = \{t_1, t_2, t_3, t_4, ..., t_m\}$ являє собою множину переходів $m \geq 0$. При-

чому множини місць P і переходів T не перетинаються. Вхідна функція є відображенням переходів у комплекти місць виду $I: T \to P^{\infty}$. Вихідна функція є відображенням з переходів у комплекти місць виду $O: T \to P^{\infty}$ [14, 15]. Граф МП можна представити у вигляді множини $G=(V, A)$, де $V=\{u_1, u_2, u_3, ..., u_s\}$ — множина вершин, а множина $A=\{a_1, a_2, a_3, ..., a_t\}$ є комплектом спрямованих дуг, які визначають взаємозв'язок між вершинами [11—12]. Множина V розбита на дві підмножини, які не перетинаються: P і T.

Маркування МП математично формалізується у вигляді функції, що відображає множину місць у множину натуральних чисел N $\mu: P \to N$.

Або у векторному вигляді $\mu=(\mu_1, \mu_2, \mu_3, ..., \mu_n)$. Уведемо функцію $\delta(\mu, t_i)$, що називається функцією наступного стану МП, і початкове маркування μ^0. Приклади побудови моделей і їхнього опису для ІМП приводяться далі й показані на рис. 1—3. Робота МП здійснюється у вигляді послідовності спрацьовування переходів і зміни маркування. Повний опис роботи і способи аналізу МП докладно описані в [11—12] і в статі не розглядаються.

Практика розробки і документування ПЗ дозволяє представити методику побудови ММ блоку програми у вигляді МП в наступній послідовності.

1. Здійснюється оцінка і виділення основних операторів ІМП в блоці ПП.
2. Будується типова модель у вигляді МП для кожного з операторів ІМП блоку програмного продукту. При цьому:
   - переходи МП ототожнюються з виконанням функцій або окремих дій операторів ІМП;
   - позиції ототожнюються з результатами виконання функцій, результатами дій операторів ІМП, зупинками при виконанні програми або готовністю до виконання функцій, операторів ІМП;
   - формалізуються логічні операції у вигляді МП.
3. Будується модель для блока з урахуванням зв'язків між операторами ІМП.
4. Для упорядкування виконання МП кожному переходу визначається ієрархія. Найвища ієрархія встановлюється для переходів, з яких починається виконання програми й далі зменшується по ходу програми.
5. Місткість кожного місця встановлюється шляхом оцінки коду ПП, залежно від умов виконання.

Реалізуючи п. 2 методики, побудуємо типову ММ для оператора ІМП *switch()* (мова С++).

Готовність оператора до виконання передбачає наявність змінних для обчислення виразу в дужках *switch(вираження)*. Цей процес в ММ формалізує місце *P1* (ємність місця дорівнює 1), при цьому в ньому розміщується мітка (рис. 1). Процес множинного вибору — *case:* формалізують місця *P2—P4* (ємність місць дорівнює 1) і переходи *T1—T3*. Наявність мітки в одному з місць *P2—P4* свідчить про виконання однієї з альтернативних умов. У цілому фрагмент мережі Петрі *P2, T1, P5* (рис. 1) формалізує одну альтернативу. При збільшенні альтернатив вибору в операторі *switch()*, кількість конструкцій *P2, T1, P5; P3, T2, P6; P4, T3, P7* (рис. 1) збільшується відповідно до кількості альтернатив.

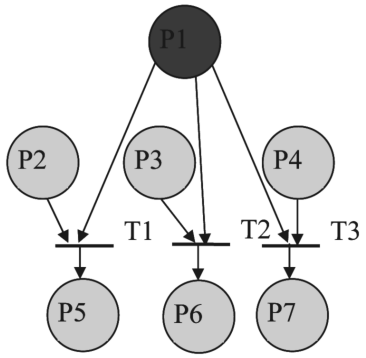

Рис. 1. Модель оператора switch

Виконання ММ у вигляді МП оператора мови С++ на рис. 1 здійснюється в наступному порядку. При переході програми до оператора *switch()* в моделі МП в місті *P1* розміщується мітка. За збігом результату обчислення виразу в *switch()* і константи в *case:* у відповідне місце *P2—P4* (рис. 1) розміщується мітка. Спрацьовує відповідний перехід (*T1—T3*) (рис. 1), і в наступне за переходом місце міститься мітка. Це формалізує готовність до виконання наступного оператору.

Ієрархія переходів у моделі однакова. Таким чином, оператор *switch()* можна представити у вигляді математичної моделі МП.

Аналогічно побудуємо типову ММ, реалізуючи п. 2 методики, для оператора ІМП *for*, що має наступний формат *for (вираз 1; вираз 2; вираз 3)*. Місце *P1* (рис. 2) в ММ формалізує готовність оператора *for* до роботи, перехід *T1* (рис. 2) формалізує обчислення *виразу 1*. Місце *P2* (рис. 2) формалізує готовність до обчислення *виразу 2*, перехід *T2* (рис. 2) формалізує обчислення *виразу 2*. Робота моделі при виконанні оператора *for* визначається в наступної послідовності. Обчислюється *вираз 1*, обчислюється *вираз 2*. Якщо значення *виразу 2* відмінно від логічного нуля (*true*), мітка розміщується в місце P_{true} (рис. 2), виконується тіло циклу (перехід *T3*) і здійснюється перехід до обчислення *T2* (рис. 2), цикл повторюється. Цикл закінчується якщо вираз 2 дорівнює логічному нулю (*false*). У цьому випадку мітка

розміщається в P_{false} (рис. 2), це відповідає закінченню циклічних обчислень і виходу з циклу. Таким чином, оператор ІМП *for* можна представити у вигляді математичної моделі МП. Ієрархія переходів установлюється наступним чином в порядку зниження: *T1, T2,* (*T3, T4* однакова для двох).

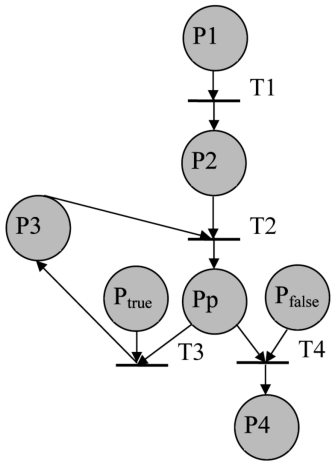

Рис. 2. Модель оператора for

Аналогічно можна побудувати модель для кожного з операторів будь якої ІМП. В процесі роботи були побудовані ММ у вигляді МП для всіх нетривіальних операторів ІМП С++.

Розглянемо формалізацію й опис практичного прикладу блоку програми з використанням моделі. В якості прикладу обраний блок ІМП С++. Частина його вихідного коду показана нижче. Блок програми виконаний з використанням стандартних операторів ІМП С++ и може бути скомпільований в більшості середовищ програмування.

Робота програми представляється в наступній послідовності. На початку програми відображається головне меню (“GENERAL MENOO”) і запит на введення знаку з клавіатури. У випадку набору “1” виконується список операторів і відображається запрошення на запит “CHAPTER1”, при наборі “2” виконується список операторів і запит на запрошення “CHAPTER2”. При наборі іншого знаку з клавіатури — здійснюється вихід із програми.

Якщо знаходимось в меню “CHAPTER1” і відповідно до запрошення набираємо “1” або “2”, то виконується послідовність функцій, відображається відповідна інформація і програма повертається у вихідне меню “CHAPTER1”. При наборі іншого знаку із клавіатури (який відрізняється від “1” або “2”) — здійснюється вихід в “GENERAL MENOO”.

Аналогічний порядок дій виконується при знаходженні в меню “CHAPTER2”.

Блоки коду програми призначені для використання в інших ПП. Програма складається з наступних файлів: файл *general content.h* містить оголошення класів для файлу *Main.cpp*; файл *Main.cpp* містить головну функцію програми; файл *section_1.h* містить оголошення класів для файлу *chapter_1.cpp;* файл *section_2.h* містить оголошення класів для файлу *chapter_2.cpp; General_content_CPP.cpp* містить опис функцій для головної програми; файл *chapter_1.cpp* містить опис функцій класів для відображення змісту розділу 1; файл *chapter_2.cpp* містить опис функцій класів для відображення змісту розділу 2.

Далі приводиться зміст файлів програми які застосовуються для розробки ММ у вигляді МП.

Зміст файлу general_content.h

```
# include <iostream>
# include “section_1.h”
# include “section_2.h”
namespace Inform_menoo{
        class General_menoo: public Class_CONTROL_content_section_2 {
                public:
                void Class_show_general_menoo_content();
                void Class_CONTROL_general_menoo_content(char Var_general_menoo_contntent);
                int foo1();//————————————————————1
                int foo2();//————————————————————2
        };
}
```

Зміст файлу Main.cpp.

```
#include <iostream>
#include “general_content.h”
using namespace Inform_menoo;
using namespace std;

int main (int argc, char *argv[]){
char _Var_main; //————————————————————————————————————————————————————1
General_menoo * _Main= new General_menoo;//————————————————————————————2
do {
 _Main->Class_show_general_menoo_content();//————————————————————————————3
 cin>_Var_main; //
          _Main->Class_CONTROL_general_menoo_content(_Var_main);//———4
         }
```

```
        while (_Var_main=='1'||_Var_main=='2');//-------------------------------------------5
delete _Main;//-------------------------------------------------------------------------------------6
        return 0;//-----------------------------------------------------------------------------------7
        }
```

Зміст файлуGeneral_content_CPP.cpp.

```
# include <iostream>
# include <string>
# include "general_content.h"
using namespace std;
namespace Inform_menoo {
int General_menoo :: foo1(){///-----------------------------------------------------------------------2
char Var_in_use;
Class_CONTROL_content_section_1 * Exemp1_section_1 = new ;
 do {
                Exemp1_section_1->Show_menoo_content_section_1();
 cin>Var_in_use;
        Exemp1_section_1->Choose_content_section_1(Var_in_use);
        }
        while (Var_in_use=='1'||Var_in_use=='2'||Var_in_use=='3'||Var_in_use=='4'||Var_in_use=='5');
        delete Exemp1_section_1;
        return 0;
}
int General_menoo :: foo2(){///-----------------------------------------------------------------------3
char Var_in_use;
Class_CONTROL_content_section_2 * Exemp1_section_2= new ;
do {
                Exemp1_section_2->Show_menoo_content_section_2();
 cin>Var_in_use;
        Exemp1_section_2->Choose_content_section_2(Var_in_use);
        }
        while (Var_in_use=='1'||Var_in_use=='2'||Var_in_use=='3'||Var_in_use=='4'||Var_in_use=='5');
        delete Exemp1_section_2;
        return 0;
}
void    General_menoo    ::Class_CONTROL_general_menoo_content(char    Var_general_menoo_contntent){
//-------------------------------------------------------------------------------1
                switch (Var_general_menoo_contntent){
        case '1':
                foo1();
                break;
        case'2':
                foo2();

                break;
        }
cout<" \n";
 }
void General_menoo :: Class_show_general_menoo_content(){
 cout<" \n\n";
        cout<"\n";-------------------------------------------------------------------------------|\n";
 cout<" |GENERAL MENOO: |\n";
 cout<" |Tape number Section, any key for EXID |\n";
        cout<"\n";-------------------------------------------------------------------------------|\n";
        cout<" |1. CHAPTER 1 |\n";
        cout<" |2. CHAPTER 2 |\n";
        cout<" |3. Eny tipe for exid |\n";
        cout<"\n\n\n";-----------------------------------------------------------------------|\n\n\n";

}
}
```

Для реалізації методики використаємо готові ММ у вигляді МП основних операторів ІМП С++. Будувати будемо ММ фрагменту блоку програми. Обраний фрагмент програми, для якої будується ММ, охоплює файл *Main.cpp*. і частину файлів в котрих описані функції *Main.cpp*. Для покращення наочності пояснень оператор do *while()* в *Main.cpp*. (позначений у коментарях //5) і частина внутрішніх функцій (файл *chapter_1.cpp*) у моделі не формалізуватимуться.

Будова ММ починається у файлі *Main.cpp* з формалізації операторів позначених в коментарях файлу *Main.cpp* *//1, //2*. Формалізація операторів здійснюється у вигляді *P1, T1* (рис. 3).

Місце *P2* (рис. 3) і перехід *T2* — формалізують готовність до виконання і виконання функції, позначеної в коментарях //3 (файл *Main.cpp*).

Місце *P3* (рис. 3) — результат виконання функції //3 (файл Main.cpp), факт відображення меню "GENERAL MENOO" і запит на введення знаку із клавіатури.

Перехід *T3* (рис. 3) — виконання функції *cin>* і перехід до функції, позначеної в коментарях //4 (файл *Main.cpp*). Оголошення класу й опис даної функції надані у файлах

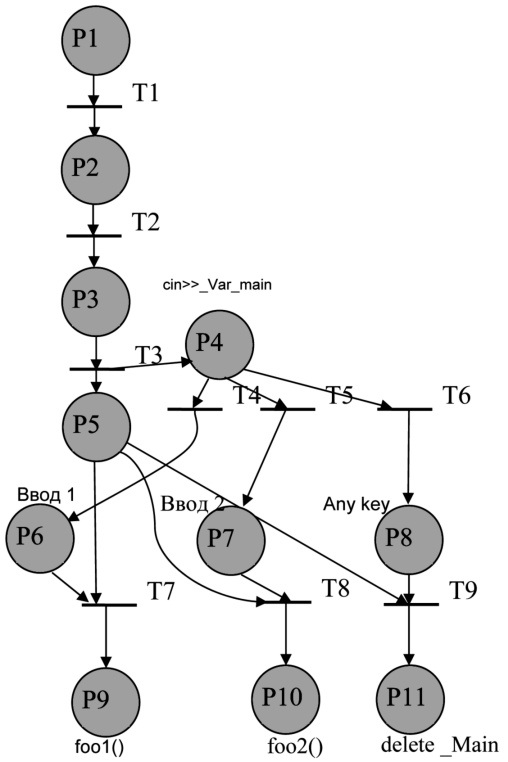

Рис. 3. Модель элемента програми.

General_content.h і *General_content CPP.cpp*. Як видно з файлу *General_content CPP.cpp,* ця функція позначена в коментаріях //1 (файл *General_content CPP.cpp*)) і містить один оператор *switch()*.

Використовуючи розроблену ММ оператора *switch()* (рис. 2) і зважаючи на наявність трьох альтернатив, функція що позначена в коментаріях //1 (файл *General_content_CPP.cpp*) буде формалізована у вигляді місць *P5—P8* (рис. 3).

Перехід *T7* (рис. 3) і місце *P9* (рис. 3) формалізує набір оператором "1" і перехід до виконання функції *foo1(),* оголошену в *General_content.h* (позначена в коментарях //1) і задана в *General_content_CPP.cpp* (коментар //2).

Перехід *T8* (рис. 3) і місце *P10* (рис. 3) формалізує набір оператором "2" і перехід до виконання функції *foo2(),* оголошену в *General_content.h* (позначену в коментарях //2) і заданну в *General_content_CPP.cpp* (коментар //3).

Перехід *T9* (рис. 3) і місце *P11* (рис. 3) формалізує перехід до виконання крайніх операторів головної функції програми (коментарі //6, 7 файлу *Main.cpp*) і вихід з програми.

Порядок спрацьовування переходів встановлюється в наступній послідовності у порядку зменшення ієрархії: *T1, T2, T3,* (*T4, T5, T6* ієрархія однакова для трьох), (*T7, T8, T9* ієрархія однакова для трьох).

Таким чином з використанням розробленої методики побудована ММ для ІМП у вигляді МП. Вона дозволяє зробити математичний аналіз у ході тестування ПП.

Для перевірки адекватності моделі здійснювалась компіляція і запуск програми в операційній системі Windows. При цьому здійснювалось порівняння ММ і ходу виконання ПП. Результати показали, що ММ у вигляді МП є адекватній коду вихідної програми.

Висновки за результатами статті. Розроблена модель дозволяє спростити процес тестування блоку ПП, зробити його наочним і застосувати формальні математичні методи аналізу. Оцінюючи відомі методики тестування [14—16] можна зробити висновок, що з використанням розробленої ММ можна здійснити блокове тестування (Unit Testing) ПП по методології "білого ящика".

Запропонована методика побудови ММ не може бути використана для інтеграційного тестування (Integration Testing) або для системного тестування (System Testing), оскільки однорівнева МП передбачає велику складність ММ і зменшує загальну наочність.

Перевагою даної методики є:

- можливість використання методики побудови ММ у вигляді МП для більшості ІМП;
- можливість використання методів математичного аналізу МП при блоковому тестуванні ПП;
- можливість перевірки адекватності моделі МП і блоку ПП;
- можливість побудови формалізованої системи при створенні ПП для документування й блокового тестування програмного забезпечення.

До недоліків необхідно віднести:

- збільшення складності і зменшення наочності ММ при формалізації складних ПП;
- неможливість здійснити інтеграційне (Integration Testing) і системне (System Testing) тестування;
- неможливість використання методики для функціональних мов програмування;
- не повною мірою відображена формалізація особливостей об'єктно-орієнтовного програмування.

Таким чином, в якості *напрямку для подальших досліджень* необхідна доробка методики побудови ММ у вигляді МП для документування складних ПП з урахуванням особливостей об'єктно-орієнтовного програмування ІМП.

Крім того перспективним напрямком є розробка ММ у вигляді МП для практичного функціонального програмування.

Література

1. Синицын С.В. Верификация программного обеспечения / С. В. Синицын, Н. Ю. Налютин — М. : БИНОМ, 2008. — 368 с. **2. Бейзер Б.** Тестирование чёрного ящика. Технологии функционального тестирования

программного обеспечения и систем / Б. Бейзер. — СПб. : Питер, 2004. — 320 с. **3. Структуры** и алгоритмы обработки данных / [Матьяш В.А, Путилов В.А., Фильчаков В.В., Щёкин С.В.]. — СПб. : Аппатиты, КФ ПетрГУ, 2000. — 80 с. **4. Дехтярь М.И.** Введение в схемы, автоматы и алгоритмы / М.И. Дехтярь. — режим доступу: http://www.intuit.ru/department/ds/introsaa/ — Назва с екрану. **5. Богданов Д.В.** Стандартизация жизненного цикла и качества программных средств / Д.В. Богданов, В.В. Фильчаков. — С-Пб. : СПГУАП, 2000. — 210 с. **6. Солтер Николас.** C++ для профессионалов; [пер. с англ.] / Солтер Николас, Клепер Скотт Дж. ; пер. с англ. — М. : ООО "И.Д. Вильямс", 2006. — 912 с. **7. Применение** сетей Петри при разработке графического языка программирования : материалы Дальневосточной математической школы-семинара имени академика Е.В.Золотова / В.Е. Мельников, Д.И. Харитонов. — Владивосток : ИПМ ДВО РАН, 1998. — С. 59. **8. Анисимов Н.А.** Композиционные методы разработки протоколов на основе сетей Петри : дис. ... д-ра техн. наук : 05.13.11 / Анисимов Николай Александрович. — Владивосток, 1994. — 337 с. **9. Дубинин В.Н.** Языки логического программирования в проектировании вычислительных систем и сетей / В.Н. Дубинин, С.А. Зинкин. — Пенза : Изд-во Пенз. гос. техн. ун-та, 1997. — 100 с. **10. Федотов И.Е.** Некоторые приемы параллельного программирования / И.Е. Федотов. — М. : Моск. госуд. ин-т радиотехники, электроники и автоматики (технический университет), 2008. — 188 с. 11. Murata T. (1989) "Petri Nets: Properties, Analysis, and Applications" / T. Murata // Proceedings of the IEEE, Vol. 77. — № 4. P.p. 541—580. **12. Дж. Питерсон.** Теория сетей Петри и моделирование систем / Дж. Питерсон. — М. : Мир, 1984. — 264 с. **13. Котов В.Е.** Сети Петри / В.Е. Котов. — М. : Наука, 1984. — 160 с. **14. Криспин Л.** Гибкое тестирование: практическое руководство для тестировщиков ПО / Л. Криспин, Д. Грегори ; пер. с англ. — М. : Изд-во "Вильямс", 2010. — 464 с. **15. Канер К.** Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / Канер К., Фолк Дж., Нгуен Енг Кек ; пер. с англ. — К. : ДиаСофт, 2001. — 544 с. **16. Калбертсон Р.** Быстрое тестирование / Калбертсон Р., Браун К., Кобб Г. ; пер. с англ. — М.: Изд-во "Вильямс", 2002. — 374 с.

Статья раскрывает методику построения математических моделей в виде сетей Петри для документирования программного обеспечения. Методика и модели предназначены для императивных языков программирования, примеры представлены на языке C++. Предложенная методика позволяет исправлять ошибки, обнаруженные в процессе использования программного обеспечения, проводить поблочное тестирование для улучшения качества программного продукта, использовать математические методы анализа при поблочном тестировании.

Ключевые слова: сети Петри, документирование программного продукта, блочное тестирование, императивные языки программирования, математические модели.

Article opens a technique of construction of mathematical models in the form of Petri net for software documenting. A technique and models are intended for imperative programming languages, examples are presented in C++. The offered technique allows to correct the errors which have been found out in the course of maintenance software, to hold unit testing of software and for improvement of quality software, to use mathematical methods for the unit testing.

Key words: Peti net, software documenting, unit testing, imperative programming languages, mathematical models.