УДК 509.7

**О. О. ЛЕТИЧЕВСЬКИЙ**

***Інститут кібернетики ім. В. М. Глушкова НАН України, Україна***

# СИМВОЛЬНІ МЕТОДИ У ВЕРИФІКАЦІЇ ТА ТЕСТУВАННІ ВИСОКОНАДІЙНИХ СИСТЕМ

*В роботі вирішується задача розробки високонадійних систем, зокрема тих, що критичні до безпеки. Розглядаються проблеми тестування та верифікації, які актуальні на кожній стадії процесу розробки. Окрім того при розробці мають бути дотримані такі стандарти розробки систем як досяжність 100% тестового покриття та застосування технології модельного тестування для розподільних та недетермінованих систем. Пропонується вирішення проблеми з використання символьних методів на основі теорії інсерційного моделювання та предикатних перетворювачів. В якості формальних специфікацій, що визначають модель системи використовується мова Live UCM, як композиція стандартної мови UCM (Use Case Maps) та мова базових протоколів. Запропоновані методи реалізовані в низці програмних систем та апробовані на прикладах перевірки властивостей безпеки в моделях у проектах в різних галузях сучасної індустрії.*

***Ключові слова:*** *модельне тестування, метод «чорної скриньки», метод «білої скриньки», символьне моделювання, предикатний перетворювач, верифікація, інсерційне моделювання*

## Вступ

Створення систем з підвищеною надійністю є однією з актуальних задач сучасної програмної інженерії. Відповідність високим вимогам стандартів тестування та верифікації зумовило необхідність використання формальних методів у високоточних індустріях. До таких систем відносяться аерокосмічні, автомобільні системи, а також системи, що працюють з мережевими протоколами, системи управління залізницями. Створення формальних методів та теорій, що вирішують складні проблеми відповідності стандартам надійності, зокрема на етапах розробки систем є важливою науковою проблемою. Такими проблемами є тестування систем з великою або нескінченною кількості станів, систем реального часу та досяжність 100% покриття коду тестами.

Проблема відповідності до вимог міжнародних стандартів для тестування та верифікації є надзвичайно складною, зокрема при проектуванні розподілених або недетермінованих систем. Існуючі методи модельного тестування не завжди можуть задовольнити такі вимоги. Тому розробка відповідної технології є однією з найважливіших задач, як у теоретичному плані так і в промисловості.

Для систем з великою кількістю станів процес генерації тестів або верифікації може бути нездійсненним у зв'язку з явищем комбінаторного вибуху. Методи конкретного моделювання або використання методів теорії ймовірності не вирішують повністю дану проблему. Тому використання символьних (алгебро-логічних) методів є однією з важливих задач як у теорії так і в практиці.

В Інституті кібернетики ім. В. М. Глушкова розробляються методи верифікації та модельного тестування із застосуванням символьних методів.

## 1. Символьна технологія

*Символьна технологія* базується на використанні дедуктивних засобів та систем розв'язувачів задач у різних теоріях.

Така технологія активно розвивається в методах верифікації та тестування. У якості прикладів використання символьних методів у тестуванні можна розглянути інструмент RT-tester [1], що працює з UML/SysML моделями, трансформуючи їх у внутрішнє представлення, як структури Кріпке. Вимоги до моделі втілюються в термінах LTL-формул. Тести генеруються за допомогою SMT-розв'язувача для різних критеріїв покриття.

Для інших різновидів тестування використовують початковий код програми системи, написаний на деякій мові програмування, та здійснюють його символьне виконання. При цьому генеруються сценарії, що містять у якості вхідних та вихідних сигналів формули, розв'язуючи які можна отримати конкретні значення. У процесі символьного виконання програми перевіряються порушення властивостей безпеки, таких, як звернення до нульової пам'яті, ділення на нуль, вихід за межі структур даних. Крім того, перевіряються властивості або анотації, задані користувачем у різних точках програми. До таких інструментів належить розробка Microsoft

SAGA [2]. Дана система активно використовується в тестуванні властивостей безпеки й велику кількість серйозних помилок було виявлено завдяки їй. Інструмент PathFinder [3] виконує символьне моделювання Java коду, перевіряючи безпеку й задані користувачем властивості. Такими інструментами також є KLOVER [4] и Barad [5], які використовуються для тестування графічного інтерфейсу, використовуючи моделювання символьних констант.

Теоретичною основою запропонованої технології є по-перше *теорія інсерційного моделювання* [6], що визначає методи взаємодії агентів та середовищ в складних розподілених мультиагентних системах.

Агент розглядається як розмічена транзиційна система з деякою множиною дій. Стани агента розглядаються як формули у деякій теорії. Поведінка агента визначається функцією занурення у середовище, яка представляє зміну символьних станів агенту.

Друга складова символьної технології – це *теорія предикатних перетворювачів* [7], які є основою символьного моделювання систем. Предикатний перетворювач є функцією, аргументами і значенням якої є формули в деякій теорії. Визначено теорію предикатних перетворювачів для цілочисельних, перелічувальних типів даних та логіки першого порядку з універсальним квантором. Предикатний перетворювач є функцією, що визначає новий символьний стан агента.

На відміну від конкретних, символьні методи оперують з множинами значень атрибутів системи, що дає змогу здійснити вичерпну верифікацію або модельне тестування для досяжності максимального тестового покриття.

## 2. Процес розробки систем, що критичні до безпеки

У розробці високонадійних систем, зокрема систем, що критичні до безпеки, процес перевірки безпеки інтегрується у стандартний процес у вигляді верифікації та тестування на кожній фазі (Рис. 1). У процесі розробки програм при користуванні модельним способом у кожній фазі розробки, як кінцевий результат, створюються артефакти — моделі, що є об'єктом верифікації. Фаза усталення вимог до системи, що представлені у відповідних формальних специфікаціях та фаза дизайну проекту визначаються формальними моделями. Етап кодування завершується створенням початкового коду, що є входом для фази тестування.

Верифікація формальних вимог та моделі дизайну є невід'ємною частиною інтегрованого процесу розробки систем, що критичні до безпеки. На цих етапах ми розглядаємо два різновиди верифікації: з одного боку – це досяжність властивостей, а з другого – тестування розробником формальних специфікацій відповідними тестами. Останнє розглядається як спеціальний етап після кодування та верифікації початкового коду.

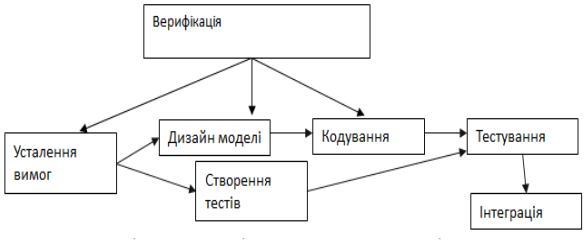


Рис. 1. Процес розробки програмного забезпечення систем, що критичні до безпеки, інтегрований з процесом верифікації

Проведенню процедури верифікації передує стадія формалізації – найбільш трудомісткий процес. Формалізація представляє собою подання вимог у деякій формальній мові, що може бути сприйнятою на вхід верифікаційної програми.

Формальні мови, за допомогою яких представляються вимоги, мають широке висвітлення в літературі. Оскільки верифікаційні алгоритми базуються на алгебраїчних перетвореннях, то початково вхідною мовою слугували традиційні математичні теорії, такі, як теорія множин, логіка першого або більш високих порядків, темпоральна логіка, теорія автоматів, мережі Петрі. Пізніше, у зв'язку з активним використанням верифікації у процесі розробки, ці мови стали основою інженерних мов, таких, як UML (Unified Modeling Language, SDL (Specification Description Language), MSC(Message Seuence Charts).

Формалізація присутня і на етапі створення тестів для тестування модельним методом *(Model based testing)* який полягає в автоматизації створення тестових наборів відповідно до деяких критеріїв, визначених користувачем, такими як покриття, метод тестування, набір властивостей, що тестуються, включно з властивостями безпеки. В MBT-методі тестові набори створюються автоматично з використанням моделей, які описують поводження системи, що тестується, на відміну від традиційного способу, де тест-інженер повинен створювати кожен тест вручну.

У низці стандартів DO-178C (аерокосмічна індустрія), IEC 62304 (медичне приладобудування), CENELEC EN 50128 (системи управління залізницями), ISO 26262 (автомобільна індустрія) визначено необхідність використання формальних методів у кожній фазі процесу розробки системи. Процесу верифікації підлягає артефакт кожної стадії процесу розробки, починаючи з усталення вимог та закінчу-

ючи тестуванням. Тестування модельним методом також є необхідною умовою відповідності стандартам. До інших вимог, на відміну від стандартного процесу розробки систем, додається етап інтеграції, на якому також визначаються процедури верифікації.

## 3. Методи верифікації та тестування

Основні методи верифікації моделей [8] визначаються властивостями, що перевіряються, до яких відносяться:

– класичні властивості несуперечливості переходів (відсутність недетермінізму), повноти моделей (відсутність тупикових станів та взаємного блокування);

– властивості життєздатності;

– локальні властивості (інваріанти, контракти) та властивості, що визначають відповідність моделям;

– властивості, що специфічні до певних класів систем, що проектуються – конкуруючі процеси, взаємне виключення, когерентність станів, динамічне взаємне блокування.

Кожна властивість визначається формулою у певній теорії та процес верифікації полягає у досяжності істинності або порушення цієї формули. В загальному вигляді проблема досяжності є нерозв'язною, тому для подолання комбінаторного вибуху використовується символьне моделювання.

Основні методи модельного тестування [9,10], що використовують символьний підхід:

*Метод «чорної скриньки»* із застосуванням символьної генерації тестів із моделі. В цьому методі із моделі дизайну або формальних вимог генеруються тести, що містять замість конкретних величин формули, які покривають деяку множину значень атрибутів системи.

*Методи символьного виконання тестів* використовують символьне моделювання для виконання програми, що тестується. При цьому кожен стан програми представляє собою формулу над її змінними. Досліджуючи формули стану виконуваної програми та очікуваної формули в тесті можна зробити висновок про наявність дефекту у випадку коли формули не є еквівалентними.

*Метод «білої скриньки»* використовується, коли відомий початковий код програми, що тестується. В цьому випадку моделюється програма, що тестується та генеруються можливі сценарії її поведінки, які далі порівнюються на відповідність із моделлю вимог або дизайну.

*Інтеграційне символьне тестування* використовується, коли модулі системи тестуються та верифікуються окремо. За допомогою символьних методів визначається перетин властивостей, які реалізують модулі та проводиться тестування лише тієї частини, що відноситься до цього перетину.

*Регресивне символьне тестування* використовується при повторних циклах після корекції коду. За допомогою символьних методів визначається та частина коду, що має бути протестована згідно виправленому коду.

## 4. Мова Live UCM

Для побудови моделей вимог та дизайну було створено мову Live UCM [11].

В основі семантики Live UCM лежать основні семантичні конструкції мови UCM (Use Case Maps), викладені в термінах інсерційного моделювання. Мова UCM є однією із стандартних мов для формалізації вимог до програмних систем. UCM, складається з множини вкладених карт. Кожна з карт містить мережу спрямованих шляхів. Шляхи можуть розгалужуватися і зливатися. Є два види розгалуження і злиття: паралельний і альтернативний. Крім розгалужень і злиттів, на шляхах можуть бути ще два види символів. Перший – це символ зобов'язання, що позначається хрестиком, і другий символ – це стаб, що позначається ромбом. З кожним стабом асоційована деяка карта, на яку посилається цей стаб.

Live UCM (Рис.2) специфікаціями є композиція UCM-специфікацій та множини базових протоколів, де з кожним оператором типу зобов'язання зв'язаний базовий протокол, що визначає перехід системи за допомогою перед- та післяумови.

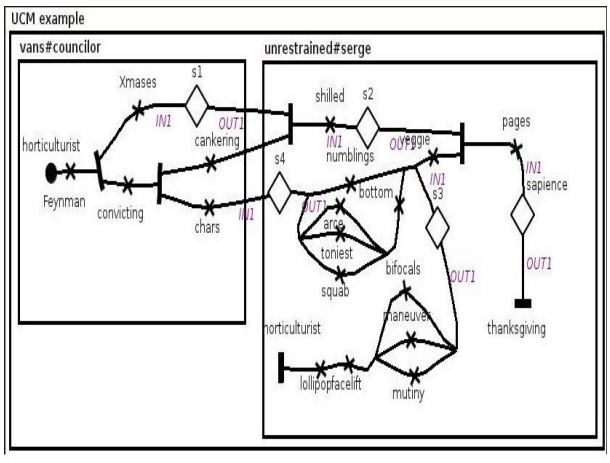


Рис. 2. Приклад моделі представленої за допомогою Live UCM специфікацій

*Базовий протокол* представляє собою формулу виду:

$$\forall x(\alpha(x) \rightarrow < P(x) > \beta(x)),$$

яку можна розглядати як формулу темпоральної або динамічної логіки, яка пов'язує поточний стан сис-

теми з майбутнім станом, до якого система перейде після закінчення процесу P(x).

Кожна карта розглядається як атрибутне середовище для занурених у неї агентів.

*Атрибутні середовища* базуються на деякій логічній базі. Ця структура включає множину типів (цілочисельні, раціональні, перелічувані, символьні, поведінкові і т. і.), інтерпретованих в деяких областях даних, символи для позначення констант з цих областей та множини типізованих функцій та предикатних символів. Деякі з цих символів є інтерпретованими (наприклад арифметичні операції та нерівності, рівності для всіх типів). Неінтерпретовані функціональні та предикатні символи називаються *атрибутами*. Неінтерпретовані функціональні символи арності 0 називаються простими атрибутами, а інші – функціональними атрибутами (неінтерпретований предикатний символ розглядається як функціональний з областю значень 0,1). Функціональні символи використовуються для визначення структур даних таких як масиви, списки, дерева тощо.

*Базова логічна мова* будується над логічною структурою атрибутного середовища. Зазвичай це є мова першого порядку. Якщо необхідно, вона може включати деякі модальності або нечітку логіку, якщо вони спрощують виведення у специфічній предметній області. Формули базових протоколів визначаються в термінах базової мови.

## 5. Системи верифікації і тестування

Перелічені символьні методи реалізовані в наступних програмних системах:

*Система VRS* (Verification of Requirement Specifications) призначена для верифікації вимог до програмних систем та створена за замовленням компанії Motorola.

*Система GTG* (Generic Trace Generator) призначена для генерації трас, які використовуються для створення тестових наборів.

*Система «Помічник програміста»*, що призначена для аналізу програм та специфікацій у формальних мовах згідно запитам користувача з метою верифікації та тестування.

Розглянуті системи використовують в повній мірі символьні методи, що реалізовані на моделях представлених за допомогою Live UCM специфікацій. Їх впроваджено в стандартний процес розробки програмного забезпечення та дозволяють вирішити основні проблеми якості для програмних систем, що критичних до безпеки. В сукупності розглянуті системи є розвинутим інструментарієм, що має більш ніж 10 річну історію розвитку як алгоритмів і теорії, так і програм. Їх використання ефективно підвищує надійність та якість продукту.

## 6. Приклади використання технології на системах, що критичні до безпеки

Одним з прикладів використання символьних методів тестування та верифікації є перевірка властивостей безпеки в задачі «інтерлокінга» залізничних колій.

Системи управління залізницями відносяться до систем, які екстремально критичні до безпеки. Це означає, що порушення умов безпеки може привести до серйозних збитків та пораненню і загибелі людей. Рух потягів на станції керований системою взаємоблокуючих реле, якими керує автоматизована система.

Ми розглядаємо дві головні умови безпеки на станції при русі потягу:

– потяг не може прибувати на колію, де вже знаходиться інший потяг, що може призвести до зіткнення. Відповідні переведення стрілок мають вести на вільну колію;

– потяг не може використовувати стрілкові переведення, якщо воно знаходиться в процесі переводу руху з однієї колії на іншу, що може призвести до сходу потяга з колій.

В системі GTG ця властивість безпеки перевіряється оберненим символьним моделюванням від символьного стану, що визначає аварійну ситуацію. Таке моделювання приводить до тупикового стану, тобто початковий стан системи керування коліями недосяжний, а відповідно недосяжна і аварійна властивість.

Інший приклад – контролер світлофора зв’язаний із системою запалювання світла та датчиками, що дають інформацію про прибуваючий транспорт. Система недетермінована і вичерпна верифікація та тестування ускладняється комбінаторним вибухом. Для вирішення задачі розглядаємо тестування контролера з використанням методу «білої скриньки» тобто з початку генеруються сценарії поведінки контролера із початкового коду таким чином, щоб покрити всі рядки коду. Далі ми використовуємо ці сценарії на сумісність із моделлю формальних вимог шляхом їх символьного моделювання.

Наступний приклад демонструє використання технології для систем із великою або нескінченою кількістю агентів.

Ми розглядаємо систему телефонної мережі з необмеженою або великою кількістю підключених телефонів в якій фігурують різні властивості, такі як базовий виклик абонента, переадресація зайнятої лінії, трьохсторонній виклик, затриманий виклик та інші. Взаємодія різних властивостей та велика кількість агентів унеможливлюють вичерпну верифікацію конкретними методами. Використання інкреме-

нтальної верифікації та символьного інтеграційного тестування дає значне підвищення якості системи за рахунок більшої кількості проаналізованих станів системи, які подано у символьному вигляді замість конкретного.

Розглянуті проекти є прикладами типового використання символьних методів у модельному тестуванні та верифікації систем, що критичні до безпеки.

Хоча розглянутими техніками використання символьних методів не обмежується, представлені приклади дають повну уяву як в розробці систем застосовується символьна верифікація та модельне тестування. Дану технологію застосовано у верифікації та тестуванні більш ніж 150 проектів, що дало значний досвід у використанні символьних технологій.

## Висновки

Основним результатом роботи є створення технології верифікації та тестування інтегрованої в процес розробки програмного забезпечення на основі символьних методів. Така технологія, на відміну від існуючої, базується на використанні дедуктивних засобів та систем розв'язувачів задач у різних теоріях. Розроблено технологію тестування методами «чорної» та «білої скриньки», що дає значну ефективність при досяжності найбільшого покриття програмного коду системи, що тестується.

Створені методи модельного тестування застосовуються для недетермінованих та розподілених систем. Існуючі методи «чорної скриньки» не можуть використовуватись для систем з великою степеню паралелізму, а тільки для простих паралельних систем де можна передбачити всі можливі поведінки системи

В основі цих методів лежить мова Live UCM, яка інтегрує стандартну мову UCM з мовою базових протоколів, основною мовою специфікації взаємодії агентів та середовищ. Використання цієї мови для побудови формальних моделей складних розподілених систем, значно підвищує продуктивність праці розробників, та надійність створюваних моделей.

Використання символьної технології дає змогу виконати вимоги стандартів для систем, що критичні до безпеки, що важливо в аерокосмічній та автомобільній промисловості, виробництві медичних приладів, системах управління залізницею, виробництві зброї та інших галузях.

## Література

*1. Peleska, J. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges [Text] / J. Peleska // Proceedings Eighth Workshop on Model-Based Testing. – 2013. – P. 3-28. doi: 10.4204/EPTCS.111.1.*

*2. Godefroid, P. SAGE: Whitebox Fuzzing for Security Testing [Text] / P. Godefroid, M. Y. Levin, D. Molnar // Communication of the ACM. – 2012.– Vol. 10, Issue 1. – P. 40-44. doi: 10.1145/2093548.2093564.*

*3. Visser, W. Model Checking Programs with Java PathFinder [Text] / W. Visser, P. Mehlitz // Lecture Notes in Computer Science. – 2005. – Vol. 3639. – 27 p.*

*4. Li, G. KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs [Text] / G. Li, I. Ghosh, S. P. Rajan // Lecture Notes in Computer Science. – 2011. – Vol. 6806. – P. 609-615.*

*5. Test Generation for Graphical User Interfaces Based on Symbolic Execution [Text] / S. Ganov, C. Killmar, S. Khurzid, D. E. Perry // Proceedings ISCE Workshop. – 2008. – P. 33-40.*

*6. Insertion modeling and symbolic verification of large systems [Text] / O. Letychevskyi, A. Letichevsky, V Peschanenko, T. Weigert // LNCS 9369 SDL 2015: Model-Driven Engineering for Smart Cities. – Springer, 2015. – P. 3-18.*

*7. Свойства предикатного трансформера системы VRS [Текст] / А. А. Летичевский, А. Б. Годлевский, В. С Песчаненко, С. В. Потиенко // Кибернетика и системный анализ – 2010. – № 4. – С. 3–16.*

*8. Метод статической проверки полноты и непротиворечивости в формальных моделях распределенных программных систем [Текст] / А. А. Летичевский, А. А. Летичевский, А. В. Колчин, С. В. Потиенко // Проблеми програмування. – Київ, 2014. – № 2-3. – С. 146–150.*

*9. Летичевский, А. А. Парадигми модельного и символьного тестирования программных систем [Текст] / А. А. Летичевский // Кибернетика и системный анализ. – 2015. – № 5. – С. 31–44.*

*10. Летичевский А. А. / Эксперименты с дедуктивным тестированием реактивных систем [Текст] / А. А. Летичевский // Математичні машини і системи. – 2013. - №4. – С.20-28.*

*11. Летичевский, А. А. Генерация символьных трасс в системе инсерционного моделирования [Текст] / А. А. Летичевский, В. С. Песчаненко, А. С. Губа // Кибернетика и системный анализ. – 2015. – № 1. – С. 3–17.*

## References

1. Peleska, J. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. *Proceedings Eighth Workshop on Model-Based Testing*. 2013, pp. 3-28. doi: 10.4204/EPTCS.111.1.

2. Godefroid, P., Levin, M. Y., Molhar, D. SAGE: Whitebox Fuzzing for Security Testing. *Communication of the ACM*, 2012, vol. 10, issue 1, pp. 40-44. doi: 10.1145/2093548.2093564.

3. Visser, W., Mehlitz, P. Model Checking Programs with Java PathFinder. *Lecture Notes in Computer Science*, 2005, vol. 3639, p. 27.

4. Li, G., Ghosh, I., Rajan, S. P. KLOVER: A Symbolic Execution and Automatic Test Generation

Tool for C++ Programs. *Lecture Notes in Computer Science*, 2011, vol. 6806, pp. 609-615.

5. Ganov, S., Killmar, C., Khurzid, S., Perry, D. E. Test Generation for Graphical User Interfaces Based on Symbolic Execution. *Proceedings ISCE Workshop*, 2008, pp. 33-40.

6. Letychevskyi, O., Letichevsky A., Peschanenko V., Weigert T. Insertion modeling and symbolic verification of large systems. *LNCS 9369 SDL 2015: Model-Driven Engineering for Smart Cities*, Springer, 2015, pp. 3-18.

7. Letichevskij, A. A., Letichevskij, A. Ad., Godlevskij, A. B., Peschanenko, V. S., Potienko S. V. Svojstva predikatnogo transformera sistemy VRS [Properties of predicate transformer HRV system]. *Kibernetika i sistemnyj analiz*, 2010, no. 4, pp. 3–16.

8. Letichevskij, A. A., Kolchin, A. V, Potienko, S. V. Metod staticheskoj proverki polnoty i neprotivorechivosti v formal'nyh modeljah raspredelennyh programmnyh sistem [The method of static verification of the completeness and consistency of formal models of distributed software systems]. *Problemi programuvannja*, 2014, no. 2-3, pp. 146–150.

9. Letichevskij, A. A. Paradigmi model'nogo i simvol'nogo testirovanija programmnyh system [Paradigms modeling and character testing of software systems]. *Kibernetika i sistemnyj analiz,* 2015, no. 5, pp. 31–44.

10. Letichevskij, A. A. Jeksperimenty s deduktivnym testirovaniem reaktivnyh system [Experiments with deductive testing rocket systems]. *Matematichni mashini i sistemi*, 2013, no. 4, pp. 20-28.

11. Letichevskij, A. A., Letichevskij, A. Ad., Peschanenko, V. S., Guba, A. S. Generacija simvol'nyh trass v sisteme insercionnogo modelirovanija [Generation of character insertion simulation runs in the system]. *Kibernetika i sistemnyj analiz*, 2015, no. 1, pp. 3–17.

*Поступила в редакцию 5.03.2016, рассмотрена на редколлегии 14.04.2016*

## СИМВОЛЬНЫЕ МЕТОДЫ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ ВЫСОКОНАДЕЖНЫХ ПРОГРАММНЫХ СИСТЕМ

***А. А. Летичевский***

В работе рассматривается задача разработки высоконадежных систем, в том числе тех, которые критичны к безопасности. Рассматриваются проблемы тестирования и верификации, которые актуальны на каждой стадии процесса разработки программ. Кроме того, при разработке должны быть соблюдены стандарты разработки систем такие как 100% достижимость тестового покрытия кода и применения технологии модельного тестирования для распределенных и недетерминированных систем. Предлагается решение с использованием символьных методов на основе теории инсерционного моделирования и предикатных преобразователей. В качестве формальных спецификаций, которые определяют модель системы используется язык Live UCM, как композиция языков UCM (Use Case Maps) и языка базовых протоколов. Предложенные методы реализованы в ряде программных систем и апробированы на примерах проверки свойств безопасности в моделях из проектов в различных современных индустриальных отраслях.

**Ключевые слова:** модельное тестирование, метод «черного ящика», метод «белого ящика», символьное моделирование, предикатный преобразователь, верификация, инсерционное моделирование.

## SYMBOLIC METHODS OF VERIFICATION AND TESTING OF HIGH RELIABILTY SOFTWARE

***O. O. Letychevskyi***

The problem of high reliability software development especially safety critical systems is considered in the paper. The problems of testing and verification arised on every stage of software development process are presented. Moreover the standards of system development such as 100% coverage of code by tests and utilization of model-based testing technology for non-deterministic and distributed systems shall be met. The symbolic approach is proposed for resolving of problem based on the insertion modeling theory and theory of predicate transformers. Live UCM language is proposed as formal specifications for model definition. It is defined as composition of standard UCM (Use Case Maps) and basic protocols language. The proposed methods are realized in a number of program systems and deployed on the examples of safety property checking in the projects from different modern industries.

**Key words:** model-based testing, “black box” method, “white box” method, symbolic modelling, predicate transformer, verification, insertion modelling.

**Летичевський Олександр Олександрович** – д-р фіз.-мат. наук, старший науковий співробітник Інституту кібернетики ім. В.М.Глушкова НАН України, e-mail: lit@iss.org.ua.

**Letychevskyi Oleksandr Oleksandrovich** – doctor of physical and mathematical sciences, senior researcher of Glushkov Institute of cybernetic NAS of Ukraine, e-mail: lit@iss.org.ua.