РАДІОТЕХНІКА, ЕЛЕКТРОНІКА ТА ТЕЛЕКОМУНІКАЦІЇ

УДК 004.42

О.В. ПОМОРОВА, С.О. ПАРФЕНОВ
Хмельницький національний університет
О. О. ІЛЛЯШЕНКО
Національний аерокосмічний університет ім. М. Є. Жуковського«ХАІ», м. Харків

ВИКОРИСТАННЯ СЕРЕДОВИЩА IMB RATIONAL RHAPSODY DEVELOPER ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІДПОВІДНО ДО ВИМОГ

У статті описується процес розробки моделі програмного забезпечення у середовищі IBM Rational Rhapsody Developer відповідно до поставлених вимог, тестування моделі на відповідність вимогам та верифікація автоматично згенерованого коду засобами Understand Source Code Analysis and Metrics та PVS-Studio.

This article describes the process of software model development using IBM Rational Rhapsody Developer according to requirements, model testing for requirements compliance and automatically generated code verification using Understand Source Code Analysis and Metrics and PVS-Studio.

Ключові слова: моделювання, вимоги, верифікація, тестування на відповідність вимогам.

Вступ

Якість майбутнього програмного забезпечення можна прогнозувати ще на етапі розробки вимог. Неправильне формулювання або протиріччя у вимогах призводять до помилок у ПЗ та його невідповідності очікуванням замовника. Статистика показує, що витрати на усунення помилок у визначенні вимог до ПЗ у процесі розробки, в залежності від етапу життєвого циклу проекту, можуть досягати від 30 до 50% загального бюджету[1]. Для запобігання цьому створено правила розробки вимог та роботи із ними, що регламентуються міжнародними стандартами [2–5].

IEEE Standard Glossary of Software Engineering Terminology [12]визначає вимоги як:

1) умови чи можливості, необхідні користувачу для вирішення проблем або досягнення цілей;

2) умови чи можливості, якими повинна володіти система або системні компоненти щоб виконати контракт або задовольняти стандартам, специфікаціям чи іншим формальним документам;

3) документоване представлення умов чи можливостей для п. 1 та 2.

Виділяють три види вимог: функціональні, не функціональні та вимоги до предметної галузі [13, 14]. Функціональні вимоги визначають поведінку ПЗ та задачі, які воно вирішуватиме [14]. Не функціональні вимоги описують такі характеристики ПЗ, як надійність, ефективність, продуктивність та ін. До не функціональних вимог також відносять обмеження, які накладаються на ПЗ та окремі потреби користувачів [14]. Вимоги до предметної галузі характеризують галузь використання ПЗ. Такі вимоги можуть бути функціональними та не функціональними [14].

При розробці складних програмних систем проектувальник повинен мати можливість створити програмну модель для дослідження модульності, об’єктів системи та зв’язків між ними, процесів, які виконуються під час функціонування моделіта провести тестування на відповідність вимогам. Для даної мети широко використовуються засоби візуального моделювання, серед яких Enterprise Architect[6], IBM Rational Rhapsody [7], Eclipse UML [8], SpringRoo [9], Grails [10] та ін. Перелічені засоби вирішують задачі створення UML-моделей, автоматичної генерації коду, виконання тестування та створення документації. Для досліджень було обрано середовище IBM Rational Rhapsody Developer версії 7.6, оскільки, згідно з [11, 16], це середовище є лідером на ринку UML 2.1 Model Driven Development рішень та використовує ряд перспективних технологій, які забезпечують користувачам найбільш ефективні засоби розробки систем та програмного забезпечення.

Постановка задачі

Метою роботи є дослідження послідовності розробки моделі програмного забезпечення системи автоматичного захисту потяга у середовищі IBM Rational Rhapsody Developer згідно з вимогами, тестування моделі на відповідність вимогам та верифікація автоматично згенерованого коду.

Вимоги до кінцевого продукту

Система автоматичного захисту потяга (САЗП) – це складний комплекс програмних засобів для залізничної сигнальної системи [15]. Вона складається з двох частин: зовнішнього обладнання, розташованого вздовж залізниці, та бортового обладнання, розміщеного в кабіні машиніста (рис. 1). Зовнішнє обладнання – це ядро системи, що виконує усі логічні функції. Воно обчислює можливість подальшого руху потяга за маршрутом на основі отриманих даних про його місцезнаходження, маршрут, даних про залізничну лінію та стану обладнання. Бортове обладнання слугує для керування потягом та передачі даних про його місцезнаходження та стан.

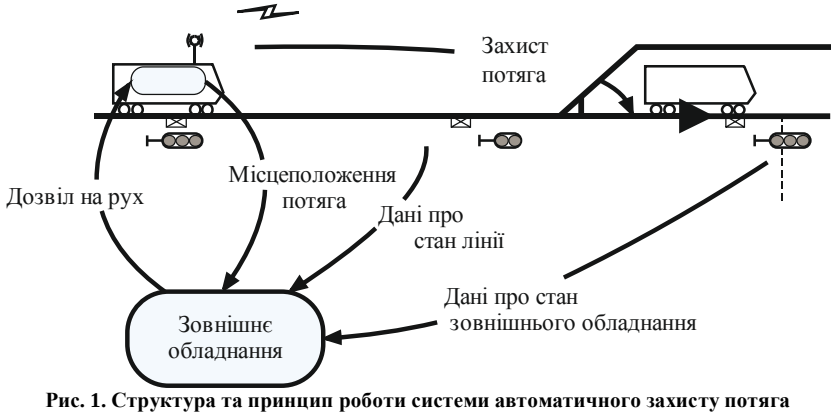

Рис. 1. Структура та принцип роботи системи автоматичного захисту потяга

Складовою бортового обладнання є кнопка підтвердження про отримання обмежувального сигналу семафора (далі «кнопка»). Призначення кнопки наступне: після того, як машиніст побачив сигнал певного роду (наприклад, жовтий сигнал залізничного семафора), він повинен натиснути кнопку для підтвердження. Кнопка може бути натиснута та відпущена лише двічі. У випадку, коли потяг минув семафор та кнопка не була натиснута і відпущена, система повинна активувати гальма до моменту повної зупинки потяга. Нижче наведено вимоги до САПЗ:

1. Кнопка повинна бути натиснута, а потім відпущена протягом 12 секунд до досягнення обмежувального сигналу.
2. Якщо кнопка була натиснута протягом часового інтервалу, більшого ніж 12 секунд, система повинна активувати гальма до моменту повної зупинки потяга.
3. Якщо швидкість потяга рівна нулю та гальма не активовані, система не повинна активувати гальма.
4. Якщо швидкість потяга рівна нулю та гальма активовані, система повинна деактивувати гальма.
5. Процедура натискання та відпускання кнопки повинна бути виконана в інтервалі між двохсот п’ятдесятьма та нулем метрів до обмежувального сигналу
6. Якщо обмежувальний сигнал пройдено і кнопка не була натиснута та відпущена, система повинна активувати гальма до моменту повної зупинки потяга.
7. Якщо обмежувальний сигнал не досягнуто протягом двоста п’ятдесяти метрів після останньої процедури натискання та відпускання кнопки, система повинна активувати гальма до моменту повної зупинки потяга.
8. Коли кнопка була натиснута, система повинна активувати сигналізатор натискання на одну секунду.
9. Кількість процедур натискання та відпускання кнопки обмежена двома. Коли процедура повторюється, система повинна обнулити лічильник дистанції до очікуваного обмежувального сигналу.
10. Якщо процедура натискання та відпускання кнопки повторюється третій раз, система повинна ігнорувати цю подію.
11. Якщо після процедури натискання та відпускання кнопки було пройдено обмежувальний сигнал, система не повинна активувати гальма.
12. У будь-якому разі, коли будь-який сигнал було пройдено, система повинна обнулити лічильник натискань кнопки та лічильник, що вимірює дистанцію до очікуваного сигналу.
13. Кожного разу, коли система активує гальма, вона повинна скидати усі лічильники.

Моделювання кнопки підтвердження про отримання обмежувального сигналу семафора

Основними компонентами моделі є компоненти «кнопка» (button), «гальма» (brakes) та «обмежувальний сигнал» (Restrictive Signal). Тому до складу моделі увійшли три програмних модулі: модуль кнопки (button), модуль гальм (brakes) та модуль, який описує систему відслідковування обмежувального сигналу (RSsystem). Початкова модель зображена на рис. 2.

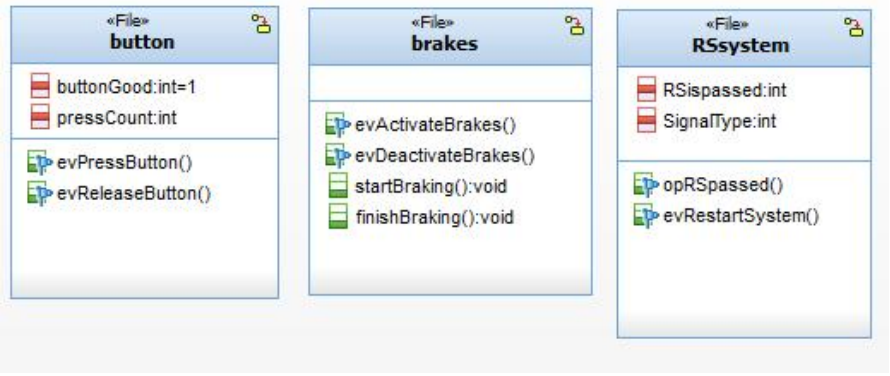

Рис. 2. Компоненти початкової моделі

Кожен програмний модуль містить процедури та змінні, за допомогою яких можна описати поведінку відповідних компонентів реальної системи. Модель, зображена на рис. 2 містить 1221 рядок коду.

На наступному етапі відбувалось доповнення моделі для задоволення усім поставленим вимогам. Аналіз показав, що виконання вимог №12 та №13 не потребує великих змін у моделі програмного продукту, тому їх реалізація виконувалась в першу чергу. Доповнення моделі для задоволення цих вимог обмежилось додаванням функцій ResetRSsystem(), та resetButtonPress() у програмні модулі «RSsystem»та «button».

Наступним етапом було розширення моделі для задоволення вимог №3, №4, №6 та №10. Для задоволення вимозі №6 додано функцію WhenPassed() та декілька додаткових умов. Для задоволення вимог №3 та №4 створено окремий програмний модуль OUTside, в якому визначені зовнішні змінні, видимі в межах усієї системи керування потягом. Для задоволення вимозі №10 додано одну умову в опис поведінки програмного модуля «кнопка». В загальному, реалізація цих умов не призвела до значного збільшення об'єму генерованого програмного коду.

Для задоволення вимог №1 та №2 створено окремий програмний модуль «timer», який описує поведінку компонента САПЗ «таймер натиснення кнопки». Модуль містить п'ять процедур та одну змінну. Також вносились певні зміни у вже існуючі файли для взаємодії інших програмних модулів з модулем «timer» Після усіх вищеописаних дій код моделі збільшився на 456 рядків. Таким чином, після внесених змін для задоволення вимог №1, №2, №3, №4, №6, №10, №12, №13 модель містила п'ять модулів (рис. 3.), та її код збільшився від 1221 рядка до 1972 рядків (разом із програмним модулем OUTside, який описує змінні та функції зовнішніх компонентів).

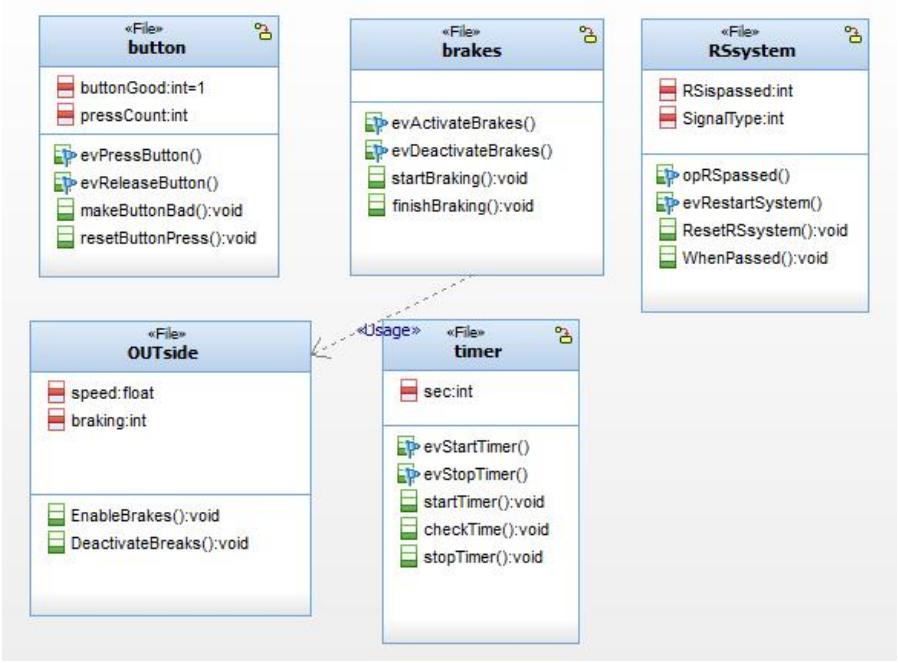

Рис. 3. Компоненти удосконаленої моделі

Для задоволення вимог №5 та №7 створено окремий модуль «Лічильник дистанції», який відповідає за вимірювання дистанції до очікуваного обмежувального сигналу. Для його опису створено файл Distance2RS, що містить дві змінні та шість функцій. Модуль «Лічильник дистанції» взаємодіє з модулем «кнопка» (після «відпускання» кнопки відбувається скидання та повторний запуск лічильника) та використовує змінну «speed», що відповідає за швидкість, та знаходиться у модулі OUTside. Генерований код для програмного модуля «Лічильник дистанції» складає 433 рядка.

Задоволення вимог №9 та №11 обмежилось додаванням декількох умов до вже існуючих функцій програмних модулів, що призвело до незначного збільшення генерованого програмного коду.

Для задоволення останній вимозі №8 створено окремий програмний модуль «Сигналізатор натиснення кнопки», що описаний у модулі ButtonAlarm. Хоча, для взаємодії модулів «кнопка» та «Сигналізатор натиснення кнопки» не вносились суттєві зміни в існуючу модель, опис цього модуля збільшив генерований програмний код на 407 рядків.

Для дослідження підходу до наповнення моделі функціональністю відповідно до поставлених вимог 1–13 оцінимо динаміку зростання об'єму програмного коду у програмних модулях, що описують компоненти моделі при доповненні її функціональності. На початковому етапі модель містила 1221 рядок програмного коду. Задоволення вимог №1, №2, №3, №4, №6, №10, №12, №13 призвело до збільшення об'єму програмного коду на 751 рядок. Задоволення вимог №5 та №7 призвело до збільшення об'єму програмного коду на 433 рядка. Задоволення вимоги №8 призвело до збільшення об'єму програмного коду моделі на 407 рядків. Таким чином, кінцева модель програмного продукту, що задовольняє усім вимогам (рис. 4) містить 2822 рядки генерованого програмного коду, що понад вдвічі більше від початкової моделі.

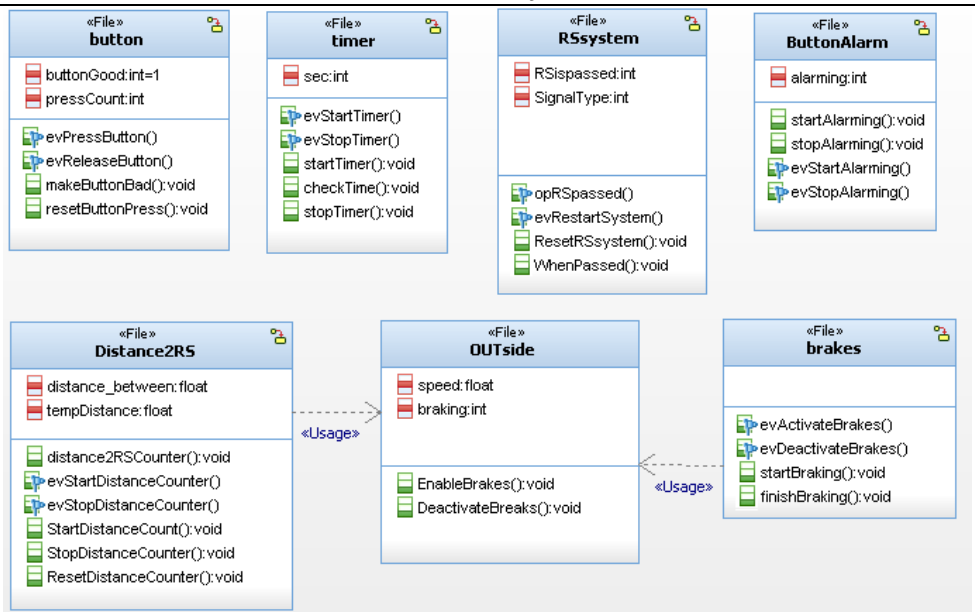

Рис. 4. Компоненти кінцевої моделі ПЗ

- Проведемо аналіз відповідності моделі, представленої на рис. 4 переліку вимог до САПЗ. Для цього розроблено прототип панелі керування потяга (Рис. 5.), який містить наступні елементи:
- *кнопка натиснута (сек.) (Buttonispressed (sec))* – рахує час (в секундах) протягом якого натиснута кнопка;
- *лічильник натискань кнопки (Buttonpressingcounter)* – лічильник, що рахує кількість натиснень кнопки;
- *сигнал пройдено (Signalpassed)* – сигналізатор проходження сигналу;
- *тип сигналу (Signaltype)* – вказівник (присутній лише під час моделювання) на тип пройденого сигналу;
- *КНОПКА (натиснути, відпустити) (BUTTON (Press, Release))* – комплекс для моделювання натиснення та відпущення кнопки (присутній лише під час моделювання);
- *кнопка справна (Buttonisgood)* – індикатор справності кнопки;
- *сигналізатор натиснення кнопки (Buttonalarm)* – сигналізатор натиснення кнопки;
- *швидкість потяга (Trainspeed)* – аналоговий тахометр швидкості потяга;
- *швидкість (Speed)* – регулятор швидкості (присутній лише під час моделювання);
- *гальма (Braking)* – сигналізатор процесу гальмування потяга;
- *швидкість (км/год) (Speed (km/h))* – цифровий тахометр швидкості потяга;
- *пройдений шлях (м) (Passeddistance (m))* – лічильник дистанції до очікуваного обмежувального сигналу;

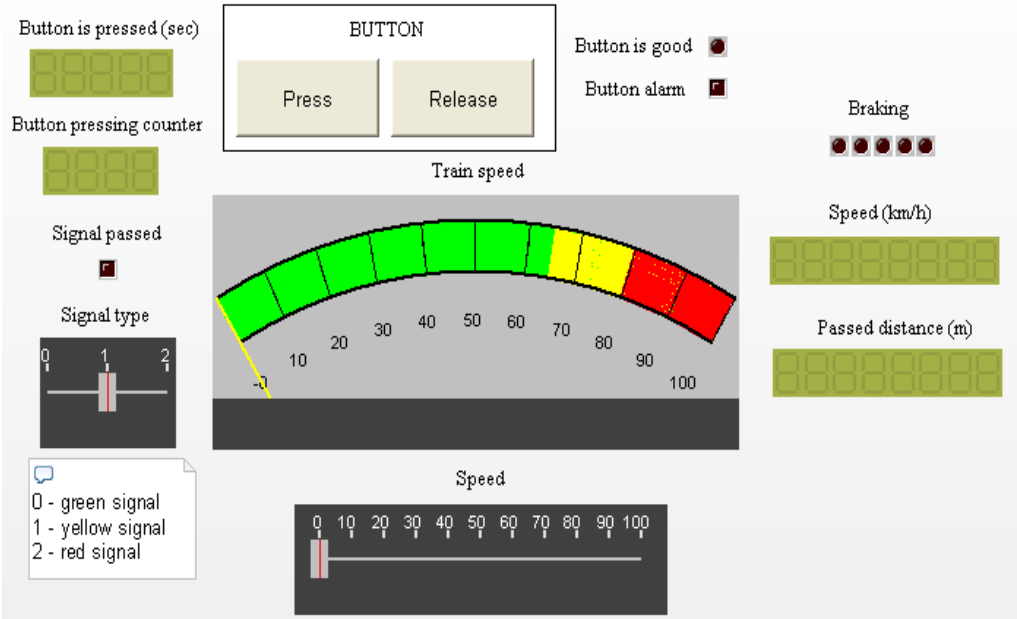

Рис. 5. Прототип панелі керування потягом

Для проведення тестування моделі розроблено ряд тестів, які наведено у таблиці 1.

Таблиця 1

Тести для визначення покриття вимог

Номер вимоги	Опис вимоги	Тест	Покриття
1	2	3	4
1	Кнопка повинна бути натиснута, а потім відпущена протягом 12 секунд до досягнення обмежувального сигналу	- Початкова швидкість потяга більше 0 - Кнопка нажата та відпущена за час, менший ніж 12 секунд - Лічильник дистанції активований, лічильник натиснень кнопки збільшився на 1	+
2	Якщо кнопка була натиснута протягом часового інтервалу, більшого ніж 12 секунд, система повинна активувати гальма до моменту повної зупинки потяга	- Початкова швидкість потяга більше 0 - Кнопка «зажата» на час, більший ніж 12 секунд - Стан кнопки змінено на «зламана» - Система активувала гальма до моменту повної зупинки потяга (швидкість == 0)	+
3	Якщо швидкість потяга рівна нулю та гальма не активовані, система не повинна активувати гальма	- Швидкість потяга рівна нулю - Гальма не активовані - Система не активувала гальма при надходженні відповідного сигналу	+
4	Якщо швидкість потяга рівна нулю та гальма активовані, система повинна деактивувати гальма	- Початкова швидкість потяга більше нуля - Гальма активовані (надійшов сигнал активації гальм) - Швидкість потяга зменшилась до нуля (зупинка потяга) - Гальма деактивовані (надійшов сигнал деактивації гальм)	+
5	Процедура натиснення та відпускання кнопки повинна бути виконана в інтервалі між двохсот п’ятдесятьма та нулем метрів до обмежувального сигналу	- Вимога уточнює вимоги№6 та №7 і не може бути протестована як окрема вимога. Вона виконується у випадку виконання вимог №6 та №7	+
6	Якщо обмежувальний сигнал пройдено і кнопка не була натиснута та відпущена, система повинна активувати гальмо до моменту повної зупинки потяга	- Початкова швидкість потяга більше нуля - Процедура натиснення та відпускання кнопки не відбувалась - Сигнал пройдено (відповідний тригер змінив значення) - Гальма активовано (отримано сигнал активації гальм) - Швидкість знизилась до нуля - Гальма деактивовано (отримано сигнал деактивації гальм)	+
7	Якщо обмежувальний сигнал не досягнуто протягом двоста п’ятдесяти метрів після останньої процедури натиснення та відпускання кнопки, система повинна активувати гальма до моменту повної зупинки потяга.	- Початкова швидкість потяга більше нуля - Кнопка була натиснута та відпущена - Лічильник дистанції до очікуваного сигналу налічив 250 метрів від моменту останньої процедури натиснення та відпускання кнопки - Активовано гальма потяга	+
		- Початкова швидкість потяга більше нуля - Кнопка була натиснута та відпущена - Сигнал пройдено (відбулась зміна значення відповідного тригера) до моменту, коли лічильник дистанції до очікуваного сигналу налічив 250 метрів - Гальма не активовано, лічильник обнулено	+
8	Коли кнопка була натиснута, система повинна активувати сигнал натиснення на одну секунду	- Кожного разу, коли кнопка натиснута, система активує сигналізацію - Після секундної сигналізації, система деактивує останню	+

Продовження табл. 1

1	2	3	4
9	Кількість процедур натиснення та відпускання кнопки обмежена двома. Коли процедура повторюється, система повинна обнулити лічильник дистанції до очікуваного обмежувального сигналу.	- Початкова швидкість потяга більше нуля - Кнопка натиснута та відпущена вперше - Активовано лічильник дистанції до очікуваного сигналу - Кнопка натиснута та відпущена вдруге (показники лічильника дистанції менші за 250 метрів) - Лічильник дистанції обнулено та запущено повторно	+
10	Якщо процедура натиснення та відпускання кнопки повторюється третій раз, система повинна ігнорувати цю подію	- Початкова швидкість потяга більше нуля - Кнопка натиснута та відпущена вперше - Лічильник дистанції активовано - Кнопка натиснута та відпущена вдруге (пройдена дистанція менше 250 метрів) - Лічильник дистанції обнулено та запущено повторно - Кнопка натиснута і відпущена втретє - Лічильник дистанції продовжує підрахунок дистанції без скидання.	+
11	Якщо після процедури натиснення та відпускання кнопки було пройдено обмежувальний сигнал, система не повинна активувати гальма	- Початкова швидкість потяга більше нуля - Кнопка була натиснута і відпущена - Протягом 250 метрів отримано сигнал, відмінний від обмежувального - Сигнал активації гальм не отримано, гальма не активовані	+
12	У будь-якому разі, коли будь-який сигнал було пройдено, система повинна обнулити лічильник натискань кнопки та лічильник, що вимірює дистанцію до сигналу	- Початкова швидкість потяга більше нуля - Кнопка була натиснута і відпущена двічі - Сигнал пройдено (відбулась зміна значення відповідного тригера) - Усі лічильники скинуто	+
13	Кожного разу, коли система активізує гальма, вона повинна скидати усі лічильники	- Початкова швидкість потяга більше нуля - Отримано сигнал активації гальм - Усі лічильники скинуто	+

З результатів наведених тестів слідує, що розроблена модель в повній мірі задовольняє усім висунутим вимогам. На цьому етап проектування моделі завершено.

Верифікація моделі

Верифікація програмного забезпечення виконувалась засобами Understand Source Code Analysis and Metrics [17, 18] та PVS-Studio[17,19]. Understand Source Code Analysis and Metrics виконує перевірку на відповідність стандартам EffectiveC++ (3rdEdition) Scott Meyers, MISRA-C 2004 та MISRA-C++ 2008 [18].

PVS-Studio – статичний аналізатор, який виявляє помилки у вихідному коді додатків на мовах C/C++/C++11[19]. Виділяють 3 набори правил, що входять до складу PVS-Studio:

1) діагностика 64-бітних помилок (Viva64);

2) діагностика паралельних помилок (VivaMP);

3) діагностика загального призначення;

Інструмент PVS-Studio інтегрується в середовище Visual Studio 2005/2008/2010. Повний перелік помилок, що відслідковуються цим аналізатором наведено в [19].

Обидва статичні аналізатори не виявили помилок в програмному коді моделі. Це вказує на відповідність коду стандартам EffectiveC++ (3rdEdition) Scott Meyers, MISRA-C 2004 та MISRA-C++ 2008 та відсутність помилок, описаних в [19].

Висновки

1. Дослідження послідовності розробки моделі програмного забезпечення згідно з вимогами показало, що при розподілі проектування на етапи, на кожному з яких виконується доповнення моделі для задоволення однієї або декількох вимог, не є оптимальним рішенням. Найчастіше це призводить до необхідності внесення змін у вже існуючі компоненти моделі або їх властивості, що, в свою чергу, може бути причиною втрати відповідності вимогам, що були задоволені на попередніх етапах. Для запобігання цьому слід ретельно планувати проектування ПЗ та групувати вимоги за їх впливом на окремі компоненти моделі.

2. Тестування моделі показало, що існують вимоги, для яких розробити набір тестів неможливо або

досить складно. В загальному, ці вимоги наводяться для уточнення інших вимог. Без таких вимог інші вимоги стають незрозумілими або невичерпними. При тестуванні такі вимоги слід об'єднувати з тими вимогами, які вони уточнюють.

3. Засобами Understand Source Code Analysis and Metrics та PVS-Studio проведено верифікацію автоматично згенерованого коду. Результати верифікації утилітою Understand Source Code Analysis and Metrics вказали на відповідність програмного забезпечення стандартам EffectiveC++ (3rdEdition) Scott Meyers, MISRA-C 2004 та MISRA-C++ 2008. Результати верифікації утилітою PVS-Studio вказали на відсутність помилок, описаних в [19].

Література

1. Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный поход / Д. Леффингуэлл, Д. Уидриг ; пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 448 с.

2. ISO/IEC TR 90005:2008 Software engineering – Guidelines for the application of ISO 9001:2000 to system life cycle processes

3. ISO/IEC 25051:2006 Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing.

4. ISO/IEC 90003:2004 Software engineering – Guidelines for the application of ISO 9001:2000 to computer software.

5. ISO/IEC 12207:2008 System and software engineering – Software life cycle processes.

6. http://www.sparxsystems.com/products/ea/index.html

7. http://www-01.ibm.com/software/rational/products/rhapsody/developer/features/?S_CMP=rnav

8. http://www.forum-omondo.com/documentation_eclipseuml_2008/eUML_EclipseUML_features_chart.html

9. http://www.springsource.org/spring-roo

10. http://grails.org/doc/latest/guide/single.pdf

11. Губский А.Е. Использование Rational Rhapsody для эффективной разработки и тестирования моделирующих систем на базе UML. Информатика и компьютерные технологии / А.Е. Губский, А.И. Андрюхин. – 2011. – С. 214–217.

12. IEEEStd 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, ISBN 1-55937-067-X.

13. FP6-027685 MESH D6.3, User and System Requirements, 2006, WP6 System Architecture and Integration.

14. Соммервилл И. Инженерия программного обеспечения / Соммервилл И. ; [пер. с англ.]. – М. : Вильямс. 2002.

15. Haifeng Wang, Chunhai GAO, Shuo LIU, Model-based Software Development for Automatic Train Protection, 2009 Second Asia-Pacific Conference on Computational Intelligence and Industrial Applications.

16. Tom Rittenbach, Hiroshi Satake, Eric Redding, Karen Perry, Mahendra Thawani, Dr. Carl Dietrich, Rithrong Thandee, GRA Model Driven Design Process, The 2010 Military Communications Conference – Unclassified Program – Systems Perspective Track.

17. Ivo Gomez, Pedro Morgado, Tiago Gomes, Rodrigo Moreira, An overview on the Static Code Analysis approach in Software Development, Faculdade de Engenharia da Universidade do Porto, Portugal, 2009.

18. http://www.scitools.com/features/codeCheck.php

19. http://www.viva64.com/ru/d/full/

Статтю представляє: д.т.н. Поморова О.В.
Надійшла 12.2.2012 р.