H_M = (M + R1 * M (M1) +... + Rn * M (Mn) / (1 + R1 +... + Rn)

Де M – базова метрика, Mi – інші цікаві заходи, Ri – коректно підібрані коефіцієнти, M (Mi) – функції.

Функції M (Mi) і коефіцієнти Ri обчислюються за допомогою регресійного аналізу або аналізу задачі для конкретної програми.

В результаті досліджень, автор метрики виділив три моделі для заходів: Маккейба, Холстеда і SLOC, де в якості базової використовується міра Холстеда. Ці моделі отримали назву «найкраща», «випадкова» і «лінійна».

Підводячи підсумок, хотілося б відзначити, що жодної універсальної метрики не існує. Будь-які контрольовані метричні характеристики програми повинні контролюватися або в залежності один від одного, або в залежності від конкретного завдання, крім того, можна застосовувати гібридні заходи, проте вони так само залежать від більш простих метрик і також не можуть бути універсальними. Строго кажучи, будь-яка метрика – це лише показник, який сильно залежить від мови і стилю програмування, тому ні одну міру не можна зводити в абсолют і приймати будь-які рішення, ґрунтуючись тільки на ній.

ЛІТЕРАТУРА

1. http://www.ibm.com/developerworks/ru/library/r-rational_clear_case/
2. Липаев В.В. «Качество программных средств», М. – 2002
3. T.J. McCabe, «A complexity measure,» IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, December, 1976
4. Богданов Д.В., «Стандартизация жизненного цикла программных средств», СПб – 2000, 210 с.

Г. С. Павловська,
студентка VI курсу,
Національний технічний університет України
«Київський політехнічний інститут»

Ю. О. Олійник,
асистент кафедри автоматизованих систем
обробки інформації та управління,
Національний технічний університет України
«Київський політехнічний інститут»

ФОРМАЛЬНИЙ АПАРАТ ОПИСУ СПЕЦИФІКАЦІЇ ВИМОГ НА ОСНОВІ МОВИ UML

Процес опису специфікації вимог є дуже важливим етапом життєвого циклу програмного забезпечення (ISO/IEC 12207) та є обов’язковим елементом всіх відомих моделей життєвого циклу програмного забезпечення. Специфікація вимог є важливим елементом для розробки та тестування програмного забезпечення. Найбільш популярною мовою моделювання в області розробки програмного забезпечення на даний момент є мова UML, призначена для візуалізації, специфікування, проектування та документування програмних систем.

В той же час UML не може бути повноцінно використана для підготовки автоматизованого тестування, оскільки в ній відсутня чітка семантична основа. Тому необхідно дослідити можливості UML для специфікації вимог та вибрати формальний апарат такого опису специфікації використання у процесах розробки та тестування.

У даному дослідженні розглядаються лише вимоги до програмного забезпечення. Такі вимоги встановлюють основні угоди між користувачами (замовниками) та розробниками (виконавцями) відносно того, що робитиме система і чого від неї варто очікувати. Документ може включати процедури перевірки одержуваного програмного забезпечення на відповідність пропонованим йому вимогам, характеристики, що визначають якість і методи його оцінки, питання безпеки та багато іншого. Часто програмні вимоги описуються звичайною мовою. Водночас, існують напівформальні і формальні методи й підходи, що використовуються для специфікації програмних вимог. У кожному разі, завдання полягає в тому, щоб ці вимоги були ясні, зв'язки між ними прозорі, а зміст даної специфікації не допускав різночитань та інтерпретацій, здатних привести до створення програмного продукту, що не відповідає потребам зацікавлених осіб.

Одним із варіантів функціональних вимог є моделі вимог у вигляді діаграм, елементами яких є вимоги та відношення між ними. Вимоги до користувацького інтерфейсу можуть бути представлені у вигляді прототипу цього ж інтерфейсу. Модель функціональних вимог надалі може бути представлена у вигляді діаграм прецедентів.

Засобів мови UML в сукупності з потужними механізмами розширення достатньо для того, щоб описати будь-яку програмну систему, з усіх точок зору, актуальних на різних етапах життєвого циклу. Більше того, все ширше стає область застосування концепцій модельно-орієнтованої розробки (Model Drive Development), що відводять моделям головну роль у процесі створення та підтримки системи.

Процес об'єктно-орієнтованого аналізу і проектування тісно пов'язаний з підготовчим етапом або аналізом вимог, що включає в себе опис прецедентів. Найчастіше прецеденти потрібно продумувати досить детально. Але основна ідея полягає в дослідженні та формулюванні функціональних вимог шляхом написання історії «з життя системи». Ці історії допомагають сформулювати різні завдання і являють собою сценарії використання системи.

На першому етапі проектування створюється модель, елементами якої є функціональні вимоги до програмного забезпечення, що розробляється. Визначаються всі необхідні дані щодо кожної вимоги, а також існуючі зв'язки між ними.

До кожної визначеної вимоги створюються об'єкти взаємодії, що реалізують вимоги. Цими об'єктами можуть бути актори, прецеденти та елементи графічного інтерфейсу. В результаті отримується діаграма прецедентів.

Діаграми прецедентів – це відмінне зображення системного контексту, оскільки вони відображають межі системи, зовнішні для системи поняття і способи використання системи. А також полегшують розуміння систем, підсистем або класів, представляючи погляд ззовні на те, як дані елементи можуть бути використані у відповідному контексті. Діаграма підсумовує поведінку системи та її виконавців.

У UML концептуальні відношення між елементами, що існують в різних моделях, можна моделювати за допомогою відношення трасування. Трасування неможна застосовувати до елементів в рамках однієї моделі. Вимоги в діаграмах прецедентів можна зв'язати з іншими вимогами, до яких вони можуть мати відношення. Трасування прецедентів до бізнес-вимог, вимог можливостей, тестам або іншим прецедентам допомагає оцінити вплив змін на зв'язані з ним вимоги та перевірити їх виконання.

З конкретного зв'язку між прецедентом (з діаграми прецедентів) та вимогою (модель вимог) формується сценарій певних дій визначених актором над програмним забезпеченням, що розробляється. Але також було б непогано відокремити і проілюструвати операції системи, виконання яких запрошує зовнішній виконавець, оскільки вони важливі для розуміння поведінки системи. В якості системи позначень до складу мови UML входять діаграми послідовностей. З їх допомогою можна проілюструвати взаємодію виконавця з системою і операції, виконання яких при цьому ініціюються.

Виходить, що при описі програмного забезпечення за допомогою мови UML всі діаграми пов'язані та створюються одні на основі інших. Тому пройшовши шлях від моделювання вимог до створення діаграм діяльності можна створити інші діаграми, що є необхідними для переходу до формальних апаратів, а саме діаграми послідовності та діаграми станів.

Мови специфікацій, що використовуються для формального опису властивостей програм, знаходяться на більш високому рівні, ніж мови програмування. Їх можна класифікувати за такими категоріями:

– універсальні мови з загальноматематичною основою (наприклад, RAISE, Z, API, VDM та інші);

– мови специфікації проблемних областей (наприклад, мови програмування, мови специфікацій ПЗ або доменів – DSL та інші);

– спеціалізовані мови специфікації (наприклад, мови таблиць, логіки, рівностей і підстановок та ін);

– мови, орієнтовані на специфікацію паралельних процесів (наприклад, CIP-L, Ada-68 Concurrent Pascal та ін).

Опис завдання на мові специфікації включає в себе опис загального контексту всіх понять, через які визначаються поняття, які беруть участь у формулюванні завдання або в описі моделі ПЗ.

Опис завдання дається у вигляді аксіом, тверджень, перед і постумов, що вимагають для їх реалізації не систем програмування, а спеціального апарату для доведення або верифікації опису завдань, зокрема інтерпретаторів або метасистем.

Графічна мова описів і специфікацій SDL є одним з найбільш відомих методів формального опису поведінки реактивних і розподілених систем. Він розроблений Міжнародним союзом електрозв'язку (ITU-T) і входить до Рекомендації ITU-T серії Z.100. Від початку націлена на телекомунікаційні системи, в сьогоденні мова широко застосовується для опису систем управління процесами і систем реального часу.

Перевага мови SDL – у зручності створення великих моделей. Він володіє такими корисними для розробки властивостями, як наочність і модульність, оснащений розширеним набором, що спрощують опис допоміжних конструкцій.

Мова SDL видається більш узгодженою, ніж UML і має більш чітку семантичну основу. З іншого боку, UML більш виразний і широко використовуваний. Мета об'єднання двох мов полягає в тому, щоб дозволити використовувати виражені можливості UML в поєднанні з перевагами SDL, до яких в першу чергу відносяться узгодженість і семантика.

З UML отримується діаграма кінцевих автоматів та застосовується метод трансформації виділення в метод частини кінцевого автомата, який має на увазі перенесення в виділений метод не тільки дій, пов'язаних з переходом, а й самих переходів і станів.

Застосування цієї трансформації пов'язана з наступною властивістю. Стани, що переносяться в метод, що виділяється, перестають належати вихідному автомату і, отже, команди переходу, що призводять з станів вихідного автомата в стан, перенесені у виділений автомат, некоректні. Такі команди переходу (зміни стану) повинні бути замінені командами виклику методу, що виділяється. Однак у автомата, що реалізує метод, може бути тільки одна вхідна точка, тому або всі такі команди повинні здійснювати перехід в один і той же стан, або можна використовувати цілочисельний параметр для передачі номера того стану, з якого має розпочатися виконання методу.

Позначимо через *RS(x, y)* множину, що містить всі стани автомата, в які можна потрапити зі стану *y*, не проходячи при цьому через стан *x*, включаючи *y* і виключаючи *x*. Спеціальний символ *stop* додається в множину *RS(x, y)*, якщо зі стану *y* за деяку кількість переходів можна дійти до дії, завершального роботу автомата *(stop)*. Позначимо множину всіх переходів деякого кінцевого автомата *A* через *All_T(A)*, а перехід зі стану *a* в стан *b* по сигналу *z* – через *t(a, z→b)*.

Множина станів *S* замкнута на множині переходів T, якщо не існує переходу:

$$t(x', e \to S) \in T : s \in S, x' \notin S$$

Трансформація виділення методу для переходу *t(x, e → y)* кінцевого автомата *A*, може бути застосована при виконанні наступних трьох умов:

x! = y, інакше *RS* порожньо і це буде випадок виділення автомата без станів;

множина *RS(x, y)* замкнута на множині переходів *All_T (A)\t(x, e→ y)*;

stop ∉ RS(x, y).

Трансформація виділення методу для переходу *t(x, e → y)* полягає в наступному:

- створюється і додається до активного клас метод *P* з реалізацією у вигляді кінцевого автомата;
- в цей метод переміщаються всі стани з множини *RS(x, y)*;
- дії, приписані переходу *t(x, e → y)*, стають діями, приписаними початковому переходу кінцевого автомата методу *P()*. Замість них у вихідний кінцевий автомат вставляється виклик методу *P()* і команда переходу в початковий стан *x*;
- всі команди переходу в стан *x* у створеному кінцевому автоматі замінюються на команди повернення з методу *(return)*.

Частина автомата, виділена в метод, володіє наступною семантикою: отримавши вхідний сигнал, автомат виконує деякі дії, починаючи з стану *y*, по завершенні яких повертається у стан *x*. Виконується логіка виконання завдання з наступним поверненням в початковий стан. Саме це і служить підставою для виділення методу.

У результаті перетворення виділяється структурна одиниця автомата – метод, а діаграма, що описує кінцевий автомат, зменшується, що спрощує його розуміння. А виділений метод можна використовувати повторно для зменшення дублювання коду.

Використовуючи описаний метод перетворення виходить схема (послідовність) методів і команд переходів між ними.

Отриману схему в подальшому буде легко виконати перетворення в сценарій тестування – набір тест-кейсів, зібраних в послідовність для досягнення певної мети. Він включає в себе вихідні дані, умови і послідовності виконання дій та очікувані результати.

В. В. Семерков,
студент IV курса,
Харьковский национальный университет радиоэлектроники

ПРИМЕНЕНИЕ АЛГОРИТМОВ ПРИМЕРНОГО СОПОСТАВЛЕНИЯ СЛОВ

Информация представляется в виде набора данных. Современные информационные системы основаны на концепции информации, представленной в виде данных и на концепции алгоритмов, реализованной в виде программного обеспечения. Алгоритмы имеют вспомогательный характер и необходимы для получения, сбора, обработки и преобразования данных. Следовательно, основой информационных систем являются данные и процедуры, организованные в базы данных, адекватно отражающие реалии действительности в той или иной предметной области.

Данный доклад посвящён исследованию методов решения проблемы поиска в базах данных, когда известно произношение текста, но неизвестно как он пишется. Рассматриваются некоторые фонетические алгоритмы. Также описываются метрики похожести текста, используемые в поиске, позволяющем ошибки. Информационная система, использующая фонетические алгоритмы, сможет предложить пользователю правильную выборку данных, даже если он ввёл не совсем корректный запрос для поиска. Данные алгоритмы сопоставляют словам со схожим произношением одинаковые коды, что позволяет осуществлять сравнение множества таких слов на основе их фонетического сходства. Поиск дублирующихся учетных записей, а, следовательно, нечёткое сопоставление записей, является одной из ключевых операций при работе с базами данных персонала крупных организаций [2; 3].