УДК 681.3.06

В.А. Алексєєв, В.С. Терещенко

РОЗВИТОК СПІРАЛЬНОЇ МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНИХ СИСТЕМ

Пропонуються два нових види спіральної моделі життєвого циклу програмної системи для різних режимів розробки: доводки характеристик системи до характеристик, закладених у вимогах замовника, та створення нової версії системи після її дослідної експлуатації відповідно до нових вимог замовника. Надається узагальнена спіральна модель, в яку інтегровані характеристики перших двох моделей.

Вступ

Життєвий цикл програмного забезпечення (програмного продукту, програмної системи) визначається як термін часу, який починається від момента прийняття рішення про створення програмного забезпечення й закінчується у момент його повного вилучення з експлуатації, скоріше за все, з метою встановлення та інсталяції нового продукту. Саме — нового, а не нової версії того ж продукту, що вилучається. Життєвий цикл поділяється на етапи, упродовж кожного з яких створюється нова версія програмного продукту. Під моделлю життєвого циклу програмного забезпечення сприймається структура, що визначає послідовність виконання і взаємозв'язок процесів, дій і задач упродовж життєвого циклу. Головним нормативним документом, що регламентує склад процесів життєвого циклу, є міжнародний стандарт ISO/IEC 12207: 1995 "Information Technology — Software Life Cycle Processing" та відповідний йому Державний стандарт України ДСТУ 3918-99 "Інформаційні технології. Процеси життєвого циклу програмного забезпечення" [1]. В усіх роботах, в яких так чи інакше висвітлюється життєвий цикл програмних систем, наприклад [2, 3], є посилання на ці стандарти. Крім того, в них наводяться приклади каскадної моделі життєвого циклу та для порівняння на її тлі — спіральної моделі, що має більш придатні властивості для моделювання процесів життєвого циклу. Але, якщо для каскадної моделі життєвий цикл включає стадію експлуатації та супроводу, то для спіральної моделі (її графічного зображення) життєвий цикл закінчується стадією тестування. Інакше кажучи, така спіральна модель відображає не життєвий цикл програмної системи, а життєвий цикл її розробки. Цей погляд на спіральну модель дещо розбігається з міжнародним та державним стандартами. Саме цій розбіжності і присвячена дана стаття.

1. Спіральна модель життєвого циклу програмної системи

Спіральна модель життєвого циклу розробки (саме розробки [2, с. 20]) програмної системи є подальшим розвитком водоспадної або каскадної моделі, в якій кожний завій спіралі відповідає одній з версій розробленої системи [2]. Точніше було б сказати: кожний завій спіралі відповідає одному з етапів життєвого циклу, під час якого розробляється версія програмної системи. Ця модель позбавляє замовника і розробника від необхідності повного і точного формулювання вимог до системи на початковій стадії, оскільки вони уточнюються на кожному завії спіралі (ітерації або етапі розробки версії системи). Таким чином, постійно поглиблюються та конкретизуються деталі проекту і, в результаті, вибирається обґрунтований варіант, який доводиться до реалізації [3]. Спіральна модель (у всякому разі її графічне зображення), тобто кожний з її завіїв — етапів розробки, складається з чотирьох стадій: аналізу вимог, проектування, реалізації та тестування. Під стадією розробки розуміється частина етапу процесу розробки програмної системи, що

обмежена у часі і завершується створенням конкретного продукту, визначеного у вимогах. Проходження всіх стадій повторюється на кожному етапі розробки, кожному завії спіралі, окрім останнього, який завершується виводом системи з експлуатації. Відміною такої моделі від попередньої (каскадної) є можливість багаторазового повернення до стадії формулювання вимог до розробки з будь-якої стадії робіт, якщо виявиться необхідність внесення змін. Це позитивна властивість моделі, бо на кожній стадії життєвого циклу може виникнути потреба змін, а внесення змін на будь-якій стадії обов'язково починається з внесення змін до попередньо зафіксованих вимог. Це може бути пов'язане з виявленням у процесі розробки недостатньої обґрунтованості сформульованих вимог або неузгодженості окремих позицій такого формулювання, відсутності потрібних інформаційних або технічних ресурсів, зниження обсягів фінансового забезпечення тощо. Такі зміни носять локальний поточний характер і не приводять до принципових змін вимог, зафіксованих у документах. Але і в цьому випадку рішення про внесення змін повинно бути виваженим, ретельно обґрунтованим і не виходити за рамки розроблюваної версії, тому що велика кількість змін і, таким чином, часте повернення до попередніх стадій можуть внести додаткові труднощі в організацію робіт, пов'язаних з розробкою. Більш суттєві пропозиції щодо змін, які пов'язані з розширенням функціональних можливостей програмної системи, накопичуються у замовника, узгоджуються з розробником і вносяться у список для подальшого включення у нові вимоги до розробки системи з метою створення її нової, більш досконалої, версії.

Найбільш суттєві зміни у вимогах (по суті — створення нових вимог) замовника можуть виникнути тільки після дослідної, а, можливо, й під час промислової експлуатації програмної системи. Наведена модель (графічне її зображення) не враховує цієї стадії етапу життєвого циклу, хоча в згаданих роботах перед тим на цьому наголошувалось. Тому пропонуємо для режиму розробки нової версії програмної системи з метою її вдосконалення та розвитку у зв'язку з появою нових вимог замовника для досягнення більш високих характеристик, аніж ті, що були закладені у початкових, використовувати не чотирьохстадійну, а п'ятистадійну спіральну модель життєвого циклу розробки, тепер вже і супроводу (дослідної або промислової експлуатації) програмної системи. Шосту стадію — вивід системи з експлуатації, про яку ведеться мова в роботах, застосовувати у моделі не будемо, тому що на ній закінчується спіраль, розвиток процесу (його динаміка) завершується і предмет дослідження (життєвий цикл) зникає. Це, на наш погляд, не стадія, а, скоріше, кінцева фаза одного (останнього) з етапів розвитку подій. І без уваги він не залишиться, але про нього трохи пізніше. Графічно п'ятистадійна модель зображується не у вигляді чотирьох квадрантів на тлі Декартових (прямокутних) координат, а у вигляді п'яти секторів полярної сітки координат (рис. 1). Весь простір полярної координатної сітки на площині п'ятьма променями (*OA*, *OB*, *OC*, *OD* та *OE*) розбитий на сектори, кожний з яких разом з променем відповідає певній стадії етапу життєвого циклу розробки. Промені проградуйовані відповідно до шкали одиниць вимірювання процесів, що відбуваються упродовж кожної з цих стадій. Чотири з них такі ж, як і у попередній моделі: аналіз вимог, проектування, реалізація та тестування, а п'ята — дослідна (а може, й промислова) експлуатація разом з авторським супроводженням розробником. Розглянемо більш детально призначення кожної стадії.

На першій стадії (стадія А) кожного етапу по результатах збирання вимог (вивчення директивних, договірних та нормативних галузевих документів замовника, опитування його уповноважених на це фахівців тощо), аналізу вимог замовника (виявлення функ-

Рис. 1. Спіральна модель життєвого циклу програмної системи (режим розробки)

ціональних і нефункціональних вимог, визначення їх пріоритетності тощо) будується модель проблеми для нової версії програмної системи. На першому етапі у функціональній точці старту S починається процес розробки першої версії програмної системи. Умовними одиницями вимірювання вимог (шкали цієї стадії) можуть бути інтегровані показники, що враховують кількісні і якісні показники вимог, їх складність, взаємопов'язаність, або, навпаки, їх локалізованість і незалежність, трудомісткість та ресурсоємність при реалізації тощо.

На другій стадії (стадія B) кожного етапу під час проектування версії програмної системи відбувається процес трансформації побудованої моделі проблеми у структуровану множину проектних рішень щодо побудови системи відповідно до вимог замовника — побудова архітектури самої системи, її бази даних, інших компонентів і модулів. Одиницями вимірювання шкали цього процесу також можуть бути інтегровані показники, що враховують кількість компонентів, їх архітектурний склад тощо.

На третій стадії (стадія C) кожного етапу відбувається реалізація проектних рішень, тобто створення нової версії програмної системи шляхом трансформації проекту в код реалізації відповідно до вибраної технології (структурного, об'єктно-орієнтованого, імперативного методу) за допомогою сучасних новітніх програмних інструментальних CASE-засобів проектування і генерації розподілених баз даних (як, наприклад, S-Designer) або інших програмних засобів такого класу, що підтримують візуальне об'єктно-орієнтоване моделювання (наприклад, UML), програмних засобів розробки функціональних модулів (наприклад, Java, PowerBuilder). І тут в шкалі цієї стадії одиницями вимірювання процесу можуть бути інтегровані показники, що враховують якісні показники застосованого програмного інструментарію, обсяг програмних компонент тощо.

На четвертій стадії (стадія D) кожного етапу під час тестування (семантичної перевірки програми) відбувається верифікація — перевірка правильності трансформації проекту в код, а також валідація — перевірка відповідності функцій працюючої версії програмної системи вимогам замовника до системи. Якщо будуть виявлені відмови, дефекти або помилки у системі, то виконуються цикли доведення, які здійснюються згідно другої, теж спі-

ральної, моделі життєвого циклу доведення (налагоджування) програмних засобів, зображеної на рис. 2: проектування, а може, й перепроектування програмного компонента, залежно від виявленого дефекту, реалізація і знову тестування. Після відпрацювання цієї моделі (моделі доведення) у повному обсязі (можливо, за кілька циклів доведення) розробка знову повертається до першої моделі — моделі розробки програмних засобів. Але опису моделі доведення та циклам доведення програмної системи буде приділена увага трохи пізніше. Одиницями вимірювання результатів тестування в шкалі цієї стадії можуть бути інтегровані показники, що враховують кількість виявлених дефектів, їх складність, час на їх виправлення, питому вагу серед інших дефектів, кількість компонент з дефектами тощо.

На п'ятій стадії (стадія Е) кожного етапу життєвого циклу (вже не розробки, а супроводження) програмної системи під час її дослідної (а може, й під час промислової) експлуатації відбувається перевірка адекватності і працеспроможності не самого продукту (це вже було виявлено під час тестування), а тих вимог, ідей, що були закладені у розробку при її проектуванні. Саме під час цієї стадії етапу життєвого циклу програмної системи і визріває потреба у замовника на вдосконалення самої системи, на її розвиток, розширення та поглиблення властивостей, характеристик у зв'язку з накопиченням досвіду її експлуатації, виникненням нових задач, нових технічних (а може, й фінансових) можливостей тощо. Інакше кажучи, настає новий етап ітерації розробки системи, розробки її нової версії, нового завію спіралі моделі життєвого циклу не тільки розробки, але й супроводу під час експлуатації — дослідної або промислової. Після реалізації такої версії, тестування та дослідної експлуатації знову може виникнути потреба в удосконаленні системи, її розвитку та створенні нової версії — нового завію спіралі на моделі життєвого циклу. І так весь час, тому що загальновизнаним законом програмної інженерії є закон еволюції, котрий формулюється так: кожна діюча програмна система з часом потребує змін або перестає використовуватись [2, с. 19].

Під час дослідної, а в окремих випадках і промислової експлуатації системи обов'язково відбувається супроводження її розробником. Метою такої дії є:

Рис. 2. Спіральна модель життєвого циклу програмної системи (режим доведення)

- усунення дефектів у програмних засобах, які не були виявлені під час тестування і можуть бути виявлені під час цієї стадії життєвого циклу (коригуюче супроводження);
- адаптація системи до умов експлуатації замовником, консультування, а можливо, і навчання фахівців замовника кваліфікованій експлуатації системи (адаптивне супроводження);
- збирання й накопичення саме розробником ідей з вдосконалення системи, підвищення її програмно-технічних характеристик та якості для включення їх у списки змін вимог до наступної версії системи (вдосконалювальне супроводження).

Замовник, як вже було сказано, теж збирає ідеї з функціонального вдосконалення системи, розширення її функціональних можливостей, розв'язання нею нових задач тощо з тією ж метою — для включення їх у списки змін вимог до наступної версії системи. Таких версій на рис. 1 зображено три, але, зрозуміло, що їх кількість може обмежуватись, скоріше за все фінансовими можливостями замовника, а визначатися буде залежністю від кількості, масштабності, а головне — змінюваності (у часі) функціональних задач, що стоять перед замовником.

2. Спіральна модель життєвого циклу для налагоджувального режиму розробки

На четвертій стадії (стадія D) кожного етапу неодмінною складовою процесу розробки є режим доведення (налагоджування) програмної системи. Потреба в ньому виникає у тому разі, коли під час тестування системи виявляється, що її характеристики не відповідають замовленим. Є розбіжності між характеристиками, зафіксованими у вимогах на створення системи, і характеристиками, отриманими під час тестування реалізованої програмної системи. Це може статися за різних обставин, але ясно одне — систему (програмний продукт) треба доводити, тобто вилучати помилки, усувати дефекти, вносити зміни в окремі програми, а може, й перепроектувати якісь фрагменти системи. Після цього знову необхідно реалізувати систему і ще раз тестувати. Можливо, і на цей раз будуть виявлені розбіжності в характеристиках, але значно менші. Знов повторення вказаного циклу доведення. Вочевидь — новий завій спіралі, але не тої спіралі, що розходиться, як це здійснюється у моделі розробки (рис. 1), а тої, що зходиться, як це здійснюється у моделі доведення (наладки) версії програмного продукту (дещо нагадує спіраль Архімеда з однією гілкою) і показано на рис. 2.

Ця спіральна модель зображена на тлі прямокутних (Декартових) координат, координатні вісі яких поділяють простір площини на чотири квадранти, кожний з яких разом зі своєю напіввіссю (*oa*, *ob*, *oc*, *od*) відповідає певній стадії режиму доведення програмної системи. Суттєвою відміною цієї моделі є те, що вона притаманна іншому режиму розробки програм, навіть не розробки, а доведення їх до кондиційних умов, досягнення рівня адекватності програм висунутим до них вимогам, тобто отримання необхідної якості програмного продукту. Ця модель має малі цикли (на відміну від великого життєвого циклу) доводки програмних засобів. Про це вже згадувалось вище. Причиною проведення цих циклів доводки є не вимоги замовника, закладені у технічному завданні, а дефекти програмних засобів, що виникли під час розробки за вини самого розробника чи то через неадекватно сприйняті розробником вимоги замовника, чи то через невдало вибрану технологію проектування, кодування й реалізації. Саме це й впливає на характеристики системи, вносить розбіжності між ними і характеристиками, зазначеними у вимогах. Ці розбіжності відкладаються на ординаті "Аналіз вимог і розбіжності характеристик". На першому циклі — аналіз вимог, а на інших — розбіжності характеристик. Треба зауважити, що за своєю функціональною навантаженістю напіввісь *oa* (стадія a) моделі доведення співпадає з функціональною

навантаженістю променя *OA* (стадія A) моделі розробки. Те ж саме можна сказати про відповідності інших стадій: *ob* ↔ *OB* (стадія B), *oc* ↔ *OC* (стадія C), *od* ↔ *OD* (стадія *D*). Ця схожість не випадкова. Вона говорить, що процеси, які відбуваються під час функціонування цих моделей, за природою схожі. Вони тільки мають різну направленість. Процес у першій моделі (моделі розробки) направлений на збільшення можливостей системи завдяки розробці нових версій, а процес у другій моделі направлений на зменшення дефектів в системі внаслідок запровадження циклів доведення програмного забезпечення. На рис. 2 таких циклів зображено три, але в житті, залежно від ваги, кількості і складнощів дефектів, а також від акуратності і досвідченості аналітиків-проектувальників та програмістів, таких циклів може бути від одного до кількох. Роль ординати (напіввісь *oa*) цієї стадії нагадує роль елемента порівняння в моделях систем автоматичного регулювання, який генерує керуючий сигнал, ґрунтуючись на відхиленнях порівнюваних параметрів. Продовжуючи порівняння моделі доводки з моделями систем автоматичного регулювання, можна сказати, що роль зворотного зв'язку тут виконує стадія тестування. Проте з кожним таким циклом доводки на стадії тестування дефектів виявляється все менше і менше і, зрозуміло, й величини розбіжностей весь час зменшуються, тому на одному з чергових тестувань їх може не виявитись зовсім, що й видно на моделі (рис. 2) — розбіжність рівняється нулю і спіраль зайшла у центр координат (в усякому разі так повинно бути в ідеалізованому уявленні).

3. Узагальнена спіральна модель життєвого циклу програмної системи

Після усунення дефектів, що були виявлені на стадії тестування програмної системи, тобто після відпрацювання моделі доводки, можна повертатись до попередньої моделі, до моделі життєвого циклу розробки і продовжувати по всіх стадіях цей етап розробки нової версії програмної системи відповідно до наведеної моделі аж до стадії переходу до дослідної експлуатації та супроводження системи. І так до тих пір, поки розроблена програмна система не задовольнить замовника по всіх параметрах та характеристиках, коли він (замовник) прийме систему в промислову експлуатацію, як це показано на рис. 3.

Рис. 3. Узагальнена модель життєвого циклу розробки і доведення програмної системи

На цьому рисунку зображена узагальнена (об'єднуюча всі процеси) модель життєвого циклу розробки і доведення програмної системи. Процес режиму розробки системи (модель розробки) включає три версії та по два цикли режиму доведення системи (модель доведення) кожної версії. Зрозуміло, що це зображення умовне, тому що може буде достатньо і однієї версії, а можливо, й трьох буде замало. Це залежить, як вже згадувалось, від досвідченості фахівців замовника при формулюванні та висуванні вимог до програмної системи, від досвідченості та кваліфікації фахівців розробника при проектуванні та реалізації системи та багатьох інших об'єктивних і суб'єктивних причин. І не в останню чергу залежить і часто визначається типом створюваної програмної системи. Так, наприклад, автоматизовані системи управління організацією (підприємством) весь час потребують змін через появу нових або необхідності вдосконалення вже існуючих функціональних задач, через розширення виробництва, зміну партнерів, зміни в потребах ринку, котирування продукції, зміни нормативних актів тощо. Тому для такого типу систем потреба у нових версіях буде завжди, доки існуватиме організація (рис. 4, а). Процес розробки автоматизованих інформаційних систем у меншій мірі, але теж прихильний до багатоітераційності внаслідок включення до системи нових задач, розширення її меж (масштабів). Особливо це стосується територіально розподілених інформаційних систем, що охоплюють великі галузі народного господарства, органи державного управління, міністерства, відомства (рис. 4, б). Для процесу розробки програмних систем, призначених для наукових, інженерно-технічних розрахунків або навіть моделюючих програмних систем, така багатоітераційність менш характерна. Це пояснюється тим, що сама задача та її межі ясні замовнику з самого початку і вимоги свої він формулює чітко і конкретно. Якщо ж і з'явиться необхідність у доробці системи, то це стосуватиметься скоріш за все невеликих змін у алгоритмі розрахункової частини програмного забезпечення, що не вплине ні на архітектуру системи, ні на інформаційні потоки, ні на технічне та організаційне забезпечення. Але тип системи вплине на зовнішній вигляд самої моделі. Так, для моделюючих систем, ймовірно, спіральна модель матиме незначну кількість завіїв, до того ж вони більш щільно розміститься один до одного, якщо враховувати кількість та складність вимог (у вигляді інтегрованого показника) на стадії А та відкладати його на шкалі променя *OA* (рис. 4, в).

Кількість завіїв на спіралях моделей свідчить лише про тенденцію, що характерна для моделі того чи іншого типу автоматизованих програмних систем, а не про дійсну їх кількість. Це пояснюється: по-перше, браком місця для графічного зображення великої кількості завіїв спіральної моделі на рис.4, а по-друге, дійсна кількість завіїв визначається для кожної системи (навіть не типу системи, а самої системи)

Рис. 4. Спіральні моделі життєвого циклу різних типів автоматизованих систем:
а) автоматизована система управління; б) автоматизована інформаційна система;
в) моделююча автоматизована система та система для наукових розрахунків

залежно від того, як про це було згадано вище, від чіткості визначення та формулювання вимог замовником, складності системи, кваліфікації і досвіду фахівців розробника та ще багатьох інших чинників.

Висновки

Треба зазначити, що зображена на рис.3 узагальнена модель, як і попередні моделі (рис. 1 та рис. 2), наведені у цій статті, представлені в ідеалізованому вигляді. В них кутова відстань (розміри кута) між променями або напівосями не відповідає когнітивній відстані між стадіями (кількості інтелектуальних зусиль, які необхідно затратити розробникові програмного забезпечення для того, щоб перевести систему, що розробляється, з однієї стадії життєвого циклу в іншу) [2, с. 264]. Точки перерізу гілки спіралі з променями та напівосями не враховують величини інтегрованих показників, що повинні бути адекватними кількісним (розміри, об'єм, обсяг) чи якісним характеристикам тих процесів, що моделюються на цих стадіях життєвого циклу. Якби це зробити, то наведені моделі графічно виглядали б дещо інакше, що скоріше за все утруднювало б зорове сприйняття і розуміння складнощів всієї множини процесів, що протікають під час розробки програмної системи. Тим не менш зовнішній вигляд моделей спонукає до таких висновків:

- Нові вимоги до програмної системи виникають на стадії дослідної (промислової) експлуатації і, відповідно, супроводження програмної системи, причому замовник під час експлуатації напрацьовує вимоги до розширення можливостей (підвищення функціональних показників) системи, а розробник під час супроводження напрацьовує пропозиції на вдосконалення (покращення програмно-технічних характеристик) системи.
- На стадії тестування розробник визначає відхилення результуючих характеристик створеної системи (версії системи) від заданих у вимогах технічного завдання і переходить у режим налагоджування системи — доведення її характеристик до потрібних.
- Виведення системи з експлуатації не є стадією її життєвого циклу, як це стверджується в [3, с. 35], а є лише заключною фазою останнього етапу життєвого циклу, інакше на кожному етапі (завії спіралі) необхідно було б вилучати систему з експлуатації, а не замінювати її новою версією.

Хоча по цих моделях, вірніше, по їх графічному зображенню, наведеному у статті, не можна судити про когнітивну відстань між стадіями або про інженерію якості (процес надання продуктам програмного забезпечення властивостей якості, оцінки показників та кількісного вимірювання їх із зіставленням з зазначеними вимогами) створюваної системи, проте вона дає можливість показати читачеві, що таке (ідеалізоване) графічне зображення досліджуваних спіральних моделей життєвого циклу розробки й доведення програмного продукту дозволяє точніше сприймати багатоаспектні процеси побудови програмних систем. Спіральні моделі у такому вигляді є своєрідними атрибутами зручності, що надають можливість фахівцям з програмної інженерії, зокрема розробникам саме територіально розподілених інформаційних корпоративних інтегрованих систем, значно скоротити час на проектування, реалізацію, тестування, доведення та здачу в дослідну експлуатацію створеного програмного продукту завдяки більш ефективній організації процесів на всіх стадіях життєвого циклу цих систем.

1. *ДСТУ 3918-99* (ISO/IEC 12207: 1995) Інформаційні технології. Процеси життєвого циклу програмного забезпечення. — К.: Держстандарт України, 2002. — 49 с.
2. *Бабенко Л.П., Лавріщева К.М.* Основи програмної інженерії: Навч. посіб. — К.: Знання, КОО, 2001. — 209 с. — (Вища освіта XXI століття).
3. *Вендров А.М.* Проектирование программного обеспечения экономических информационных систем: Учебник. — М.: Финансы и статистика, 2000. — 352 с.

Отримано 31.07.03

Про авторів

Алексєєв Віктор Анатолійович,
кандидат технічних наук, завідувач відділом

Терещенко Валерій Савелійович,
кандидат технічних наук, старший науковий співробітник

Місце роботи авторів
Інститут програмних систем НАН України,
просп. Академіка Глушкова, 40,
Київ-187, 03680, Україна
Тел. (044) 266 4228, 266 6321
E-mail: terek@isofts.kiev.ua