УДК 004.41.42

О.С. Волковський, А.О. Дмитренко

МЕТОДИКА РОЗРОБКИ СКЛАДНИХ ПРОГРАМНИХ СИСТЕМ

Анотація. Дана оцінка існуючим методам розробки складних програмних систем. Враховуючи виявлені проблеми розробки складного програмного забезпечення, запропонована методика, використання якої дозволить контролювати та продовжувати життєвий цикл системи, зменшити витрати на її модифікацію та супровождення, а також гнучко адаптувати її до нових потреб користувача, наглядно відображаючии структуру системи.

Вступ. Розробка складного програмного забезпечення (ПЗ) є молодою галуззю, яка швидко розвивається в умовах сучасної комп'ютерної науки, котра схильна до постійних і швидких змін. У зв'язку з ускладненням ПЗ стало очевидно, що його важко модифікувати. З'явилася потреба в створенні технології розробки складних програмних засобів і алгоритмічних методів їх проектування для значного і суттєвого поліпшення продуктивності праці розробників. Глобальні зміни в галузі створення ПЗ були обумовлені швидким зростанням попиту на ринку програмного продукту – особливо тієї частини розроблених програм, яка отримується користувачем у вигляді готових до експлуатації пакетів програм різного призначення.

Світовий досвід розробки ПЗ дозволив виділити декілька загальноприйнятих моделей створення складних програмних систем (СПС). Ці моделі призначені для встановлення чіткої регламентації етапів і змісту робіт на кожному кроці, методів і процедур виконання самих робіт, складу і змісту розроблюваної документації. Чіткі моделі дозволяють істотно підвищити ефективність процесу розробки складних програмних комплексів, оптимально організувати управління розробкою, планувати і контролювати терміни виконання окремих етапів, правильно розподілити роботу в колективі розробників. В результаті вдається помітно знизити витрати на розробку програмного продукту і підвищити його якість.

Одним з базових понять методології проектування СПС є поняття життєвого циклу її ПЗ. Життєвий цикл ПЗ – це безперервний процес,

© Волковський О.С., Дмитренко А.О., 2017

який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

Для успішної реалізації проекту об'єкт проектування інформаційної системи (ІС) повинен бути перш за все адекватно описаним, повинні бути побудовані повні й несуперечливі функціональні та інформаційні моделі ІС [1].

Накопичений до теперішнього часу досвід проектування ІС показав, що це логічно складна, трудомістка і тривала за часом робота, котра вимагає високої кваліфікації фахівців, які беруть участь у ній. Однак до недавнього часу проектування ІС виконувалося в основному на інтуїтивному рівні із застосуванням неформалізованих методів, заснованих на практичному досвіді, експертних оцінках і дорогих експериментальних перевірках якості функціонування ІС. Окрім того, в процесі створення і функціонування ІС інформаційні потреби користувачів можуть змінюватися чи уточнюватися, що ще більше ускладнює розробку і супровід таких систем.

Управління конфігурацією є одним з допоміжних процесів, які підтримують основні процеси життєвого циклу ПЗ, перш за все процеси розробки і супроводу ПЗ. При створенні проектів складних ІС, що складаються з багатьох компонентів, кожен з яких може мати різновиди чи версії, виникає проблема обліку їх зв'язків і функцій, створення уніфікованої структури і забезпечення розвитку всієї системи. Управління конфігурацією дозволить організувати, систематично враховувати і контролювати внесення змін в ПЗ на всіх стадіях ЖЦ.

Постановка проблеми. Користувачі, котрі працюють з СПС, мають у процесі своєї роботи не помічати внесення в неї нових функцій або усування помилок. Необхідна також наявність проектної документації, яка дозволяла б розвивати програму, навіть не тим розробникам, які її створювали без великих витрат на зворотну розробку. Окрім того, додатковими факторами, що збільшують складність розробки програмних систем, є:

- складність формального визначення вимог до програмних систем;
- відсутність задовільних засобів опису поведінки дискретних систем з великим числом станів при недетермінованій послідовності вхідних впливів;
- колективна розробка;
- необхідність вирішення проблеми дублювання коду.

 ISSN 1562-9945

На складність розроблюваного програмного продукту також впливає і прагнення до створення бібліотек компонентів, які можна було б використовувати в подальших розробках. Однак в цьому випадку компоненти доводиться робити більш універсальними, що в кінцевому підсумку збільшує складність розробки.

Разом узяті ці всі фактори суттєво збільшують складність процесу розробки. Однак очевидно, що всі вони безпосередньо пов'язані зі складністю об'єкта самої розробки - програмної системи.

На сьогоднішній день, виходячи з вищезазначеного, актуальним є створення єдиної методології складних інформаційних систем, яка вирішує вищеперераховані проблеми ПЗ.

Аналіз останніх розробок й публікацій. На сьогоднішній день не існує програмних (алгоритмічних) рішень в створенні технологій розробки складних комп’ютерних систем. Зазвичай, цю проблему намагаються вирішити за допомогою використання інструментів планування виконання завдань. Прикладом такого інструментарію може служити методологія Scrum, найважливіші умови якої це незмінність певних функцій під час виконання однієї ітерації і суворе дотримання термінів випуску чергового релізу, навіть якщо до його випуску не вдасться реалізувати весь запланований функціонал. Керівник розробки проводить щоденні 20 хвилинні наради, які так і називають - scrum, результатом яких є визначення функцій системи, реалізованих за попередній день, виниклі складності і план на наступний день. Такі наради дозволяють постійно відстежувати хід проекту, швидко виявляти виниклі проблеми і оперативно на них реагувати.

Основна частина. В даній роботі представлений підхід до вирішення описаних вище проблем розробки СПС на основі CASE-фреймворку Activiti та фреймворку роботи з БД Liquibase.

Activiti - це програмний фреймворк, спрямований на спрощення процесу розробки та уніфікацію програмного коду за допомогою зв'язку опису алгоритму на мові UML та виконуваного програмного коду на мові Java. Activiti розуміє опис бізнес-процесу в форматі файлів BPMN (Business Process Model and Notation), який є одночасно і описовим і виконуваним, тобто в одному файлі зберігається інформація не тільки про виконувану роботу та зв'язки, але також є інформація про становище цих елементів в просторі відносно один до одного, якщо потрібно буде побачити діаграму процесу. Також BPMN підтримує можливість редагування на мові XML, що дозволяє виключити помилки, що виникають через людсь-

кий фактор. В цьому стандарті є події, логічні оператори; потік управління може розгалужуватися за умовами, може розпаралелюватися і потім сходитися в одну точку [2]. Окрім процесів, є можливість виклику підпроцесу. У доповнення до Activiti йде веб-додаток, що надає інтерфейс для редагування та перегляду бізнес-процесів та плагін до інтегрованої системи розробки Eclipse. З точки зору розробника, Activiti – це набір jar-бібліотек, які можна підключати безпосередньо до веб-додатку та виносити на рівень моделі або на окремий сервер завдяки REST-інтерфейсу. Це означає, що можна відводити навантаження, пов'язане з роботою бізнес-процесів на інший фізичний сервер. Activiti добре підтримує кластеризацію і працює за власною схемою бази даних - можна додати скрипти створення таблиць прямо в свою робочу базу або винести базу на окремий фізичний сервер і отримати незалежність від навантажень власне Activiti і самого додатка. Це може бути дуже корисно, якщо потрібен моніторинг роботи процесів при серйозному навантаженні на основну базу додатка [3,4].

Задачі в BPMN можна розділити на дві групи: ServiceTask (задачі, які виконуються без участі користувача) та UserTask (задачі, яким потрібна участь користувача). Для ServiceTask це означає реалізацію інтерфейсу, що містить метод, в який буде додаватися об'єкт для отримання доступа до поточного екземпляра процесу. Щодо компоненту UserTask, то процес при його обробці зупиняється і UserTask стає доступним певному раніше встановленому користувачу. Завданням користувача стає прийняття рішення про подальший хід бізнес-процесу, що має на увазі заповнення конкретними значеннями заздалегідь встановлених у UserTask змінних та передачу їх на обробку фреймворку Activiti за допомогою спеціальної команди (Complete).

Окрім підтримки та систематизації алгоритмічних компонентів складної програмної системи перед розробниками стає питання цілостності та збереження даних, з яким працює складна програмна система. Для цього було запропоновано використовувати фреймворк автоматичної міграції схем БД Liquibase, який дозволяє мігрувати дані додатку з підтримкою накату змін даних і їх відкату.

Для того, щоб внести зміни в БД, нам необхідно створити файл міграції (changeset), посилання на який потрібно буде вказати у файлі журналу змін (changelog), після чого міграція може бути успішно застосована до цільової БД. Беззаперечною перевагою такого підходу є можливість

 ISSN 1562-9945

виконання відкату створених змін. Liquibase має вбудовану підтримку відкату деяких типів чейнджсетів, наприклад «createTable». Якщо ми викликаємо Liquibase через командний рядок з аргументом «rollbackCount 1» замість «update», відбудеться відкат останнього чейнджсету, однак інші типи чейнджсетів не можуть бути видаленими автоматично.

При виконанні команди «update» для кожного з чейнджсетів відбувається перевірка, чи був він застосований до схеми. Якщо чейнджсет ще не був застосований, відбувається його виконання. Для цього Liquibase намагається зберегти дані в допоміжній таблиці DATABASECHANGELOGS, що містить вже застосовані чейнджсети, а також їх хеш-значення. Хеши використовуються для того, щоб ніхто не зміг змінити вже виконані чейнджсети [5,6].

Для демонстрації гнучкості розробленого інструментарію створення методології розробки СПС був розроблений гіпотетичний бізнес-процес призначення соціальної допомоги дітям (Рис. 1):

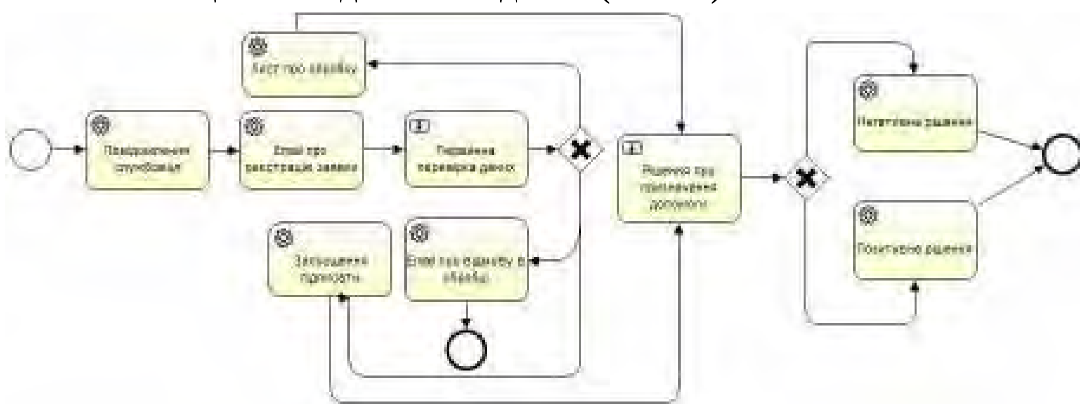

Рисунок 1 - Бізнес-процес соціальної допомоги дітям

Розглянемо реалізований алгоритм:

1. Відбувається старт процесу Activiti за допомогою компоненту StartEvent. StartEvent є базовим компонентом для всіх бізнес-процесів, що створюються за допомогою Activiti та служить як для позначення початку процесу, так і для опису необхідних змінних, які є вхідними параметрами системи. В нашому випадку - це необхідні дані громадянина для надання дитині соціальної допомоги згідно з чинним законодавством (ПІБ дитини, дата народження, свідоцтво про народження, e-mail громадянина тощо).

2. Після старту процесу виникає необхідність повідомити держслужбовця про надходження нової заявки. Це пропонується зробити за допомогою електронного листа, який відправляється на електронну скриньку за допомогою компоненту Servicetask. Компонент Servicetask дозволяє прив’язати клас із програми написаної на мові java та вказати, які змінні будуть передаватися у цю програму. В нашому випадку - це

програма відправки електронного листа, яка приймає на вхід такі значення як заголовок листа, тіло листа, цільовий e-mail. Використовуючи цю властивість Servicetask, відбувається відправка листа на заздалегідь заданий e-mail чиновника.

3. Окрім вищесказаного, є необхідність сповістити громадянина про успішне прийняття його заявки в обробку держслужбовцем. Задля цього у системі використовується відправка відповідного листа на e-mail через Servicetask. Як було описано раніше, завдяки властивостям Servicetask зв'язувати клас з бізнес-процесом ми можемо використати той самий код по відправці електронного листа, що й для пункту 2, замінив лише тіло листа та e-mail, значення якого ми візьмемо зі змінної процесу e-mail, яка була заповнена громадянином на StartEvent. Така ситуація прекрасно ілюструє, як фреймворк Activiti запобігає дублюванню коду та спрощує його підтримку: з одного боку, складна програмна система компонується із окремих програмних частин зв'язаних між собою лише засобами фреймворку та логікою бізнес-процесу, а з іншого - при необхідності замінити або модифікувати програмний код (у нашому випадку відправка листа) розробник буде працювати з одним класом, зміни якого автоматично підтягнуться в усі бізнес-процеси, які використовують цей клас.

4. На відміну від ServiceTask – компоненту, який використовується в операціях де безпосередня участь користувача непотрібна, фреймворк Activiti дозволяє впровадити прямий інтерфейс взаємодії користувача та бізнес-процесу за допомогою компоненту UserTask. У нашому випадку це необхідно задля прийняття рішення про коректність та повноту заповнених даних, коли бізнес процес доходить до цього етапу, його виконання припиняється, держслужбовець може оцінити вхідні дані, внесені громадянином на StartEvent, винести відповідне рішення та оповістити систему про нього, вибравши відповідне значення змінної (decision), після чого продовжити виконання процесу командою complete.

5. В залежності від значення decision, заповненої на попередньому кроці за допомогою компоненту Gateway, виконується умовний перехід або на наступний етап (значення змінної accept) або до завершення процесу (значення змінної reject). В залежності від результату роботи компоненту Gateway, користувачу відправляється лист з відмовою або з інформацію про переведення заявки на наступний етап.

 ISSN 1562-9945

6. За допомогою описаних раніше компонентів UserTask, ServiceTask та Gateway та у разі коректності заповнюваних даних соціальна допомога громадянину або призначається, або відмовляється у призначенні, про що громадянин оповіщається електронним листом. Процес завершується, коли досягає компоненту EndEvent.

Як можна було побачити з вищеописаного алгоритму, ми не маємо необхідності на постійне зберігання даних системи: усі дані, що використовуються в роботі процесу, існують лише від початку (StartEvent) до його завершення (EndEvent). Проте, у реальних системах така схема є незастосовною. Наприклад, кожне місто має у відповідності свій департамент державної установи, яка має призначати соціальну допомогу, інформація про що є стійкою та перебуває у вигляді довідника, з якого громадянин може вибрати, у якому місті та у якій установі він хоче замовити соціальну допомогу для дитини. У разі зберігання таких даних у БД, виникає необхідність редагування та повторного розгортання цих данних при змінах структури установ, що має на увазі залучення спеціалістів відповідної галузі, і, як наслідок – ускладнення роботи системи. Тому, для компонування збереження та автоматичного розгортання у БД такої інформації пропонується використовувати фреймворк Liquibase. Структура розробленої БД зображена на Рис. 2.:

1. Користувач вибирає місто (nIDCity) та регіон (nIDRegion).

2. За допомогою запиту з використанням цих ідентифікаторів у проміжну таблицю SubjectOrganJoin ми зможемо знайти перелік органів (SubjectOrgan), які можуть надати цю послугу.

Фреймворк Liquibase дозволяє зберегти цю структуру та пов'язану з нею інформацію у CSV-файлах, після чого завантажувати та актуалізувати інформацію напряму в БД. Це дозволяє змінювати дані системи не залучаючи при цьому додаткові знання про БД та автоматично розгортати і гнучко змінювати структуру БД на основі чансетів та csv-файлів. Такий підхід повністю відв'язує дані системи від БД, створюючи проміжний контрольний рівень данних - liquibase, який не тільки самостійно контролює цілісність даних у бд а ще й уніфікує канал взаємодії системи з БД.

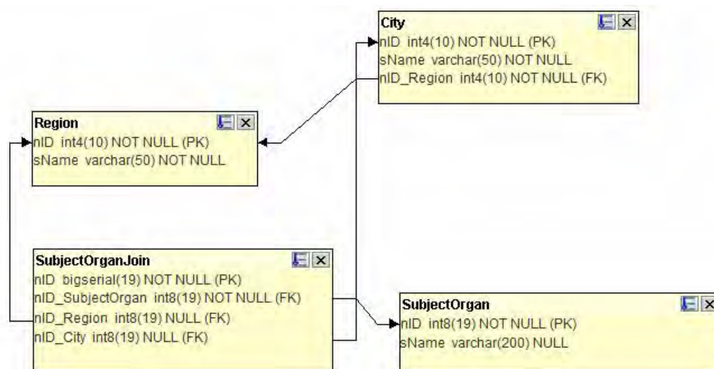

Рисунок 2 – Структура БД

Висновки. Під час дослідження були розглянуті проблеми розробки СПС та розроблена методика їх вирішення за допомогою програмних засобів Activiti та Liquibase. Сумісне використання цих фреймворків у складній програмній системі дозволяє винести архітектуру складної програмної системи над її програмною реалізацією, зробити її дискретною та стійкою до змін. Можливості, описані на прикладі конкретного бізнес-процесу, такі як зв'язок програмного коду та елементу бізнес-процесу, а також можливість впливу користувача на безпосередній хід виконання процесу, дозволяють усунути дублювання програмного коду та створити алгоритмічний рівень взаємодії користувача, з одного боку, та програмного коду, з іншого. Фреймворк Liquibase дозоляє зберігати та оброблювати дані поза межами БД, що вирішує проблему необхідності залучення спеціальних технічних знань щодо роботи з БД (таких як мова SQL) та контролбвати цілісність даних у системі. Така методологія дозволяє гнучко змінювати життєвий цикл системи, зменшити витрати на її модифікацію та супроводження, адаптувати її до нових потреб користувача та наглядно відобразити її структуру задля аналіза та оцінок.

ЛІТЕРАТУРА

1. Жалдак М.І., Хомік О.А., Володько І.В., Снігур О.М. Інформаційні технології. Навчально-методичний посібник. К.: 2003. – 194 с.

2. Вендров А.М. Case-технологии. Современные средства и средства проектирования информационных систем. – М.: ФиС, 1998. – 216 с.

3. Documentation Activiti [Електронний ресурс]. Режим доступу: http://alfresco.com/

4. Activiti 5.x business process management beginner's guide. Dr. Zakir Laliwala, Irshad Mansuri. Packt publishing – ebooks Account (March 2014). 276 p.

5. Documentation Liquibase: changes, refactorings [Електронний ресурс]. Режим доступу: http://www.liquibase.org/documentation/

6. https://htmlhook.ru/liquibase.html [Електронний ресурс].

 ISSN 1562-9945