УДК 004.415.25

Дьоміна А. О.

ОСОБЛИВОСТІ ПРОТОТИПУВАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ПРОГРАМНИХ СИСТЕМ

Предметом дослідження є прототипування програмних систем та їхніх компонентів як складової проектування програмних систем.

У статті розглянуто основи створення прототипу, що є частиною розподіленої програмної системи та використовується як уніфікований інструмент для тестування частин цієї системи. Досліджено базові вимоги до нього, сформульовано методичні рекомендації щодо розробки такого прототипу. Теоретична частина досліджень була застосована й перевірена на практиці під час розробки описаного прототипу.

Ключові слова: прототипування, проектування, розподілені програмні системи, повторне використання коду, вимоги до програмного забезпечення.

Вступ

Як зазначають Лефінгуел та Уідріг [3, с. 169], прототип програмних вимог є частковою реалізацією програмної системи й допомагає розробникам, користувачам і замовникам краще зрозуміти вимоги до системи. Таким чином, прототипування стає важливою складовою проектування програмної системи, але найчастіше прототип розглядають саме як інструмент для уточнення вимог. На думку Джона Крінніона: «Було визначено, що прототипування є дуже ефективним під час аналізу та розроблення он-лайн систем. Що більша взаємодія між користувачем і програмою, то більша вигода, отримана від швидкої розробки системи, яка може бути надана користувачеві для вивчення» [5, с. 18]. Тому логічно, що з розвитком веб-застосувань увага в розробці прототипів змістилася на візуальний компонент, тобто на оформлення веб-інтерфейсів. Натомість у цій статті розглянуто прототипування з іншої точки зору – прототип як уніфікований інструмент для тестування частин комплексної програмної системи. Такий прототип має свою специфіку, не відображену в сучасних дослідженнях.

Відповідно розглянуто такі питання:

- Основні підходи до прототипування, класифікація прототипів та їхнє використання залежно від особливостей завдання.
- Ключові вимоги до вертикального еволюційного прототипу порівняно з вимогами до програмного забезпечення взагалі, зокрема, щодо відповідності стандартам проектування та визначення значущих характеристик якості програмного забезпечення.
- Методичні рекомендації щодо побудови прототипу.

Визначення основних понять

У програмній інженерії прототипування означає створення часткових програмно-апаратних реалізацій моделей програмного забезпечення. Варто зауважити, що у програмну інженерію термін «прототип» потрапив з інших сфер інженерії (наприклад, машинобудування). Проте якщо в інших сферах прототипування – усталений процес, етап життєвого циклу виробництва, який працює за перевіреною десятиліттями схемою, то в програмній інженерії, що бурхливо розвивається, місце прототипування залишається неоднозначним.

Визначення прототипу, за працею Сміта [9, с. 43–44], дещо розпливчасте. Для уточнення нижче наведено перелік термінів, що асоціюються з предметом прототипування:

- *макет (breadboard)* є функціональним прототипом з обмеженим користувацьким інтерфейсом. Макети використовують для оцінювання продуктивності чи експериментальної розробки функцій і структур даних.
- *Дизайн (design)* є стадією розробки програмного забезпечення (ПЗ), коли планується форма системи. Прототипування – не техніка дизайну. Добрий дизайн важливий для ефективного прототипування, а прототипування може бути ефективним засобом оцінювання дизайну.
- *Імітація (mock-up)* – статичний, частково функціональний прототип, що демонструє, якого вигляду має набути система.
- *Модель (model)* – абстракція програмної системи в обмеженому масштабі. Моделі зазвичай не містяться у готовій системі, натомість їхні внутрішні механізми можуть бути використані. Моделі є корисними для тестування функціональності та визначення структур даних (рис. 1).

- *Симуляція (simulation)* – динамічна, математична чи алгоритмічна репрезентація активності. Симуляції так само, як і моделі, зазвичай не належать до готової системи. Використовуються в основному для оцінки ефективності.
- *Каркас (skeleton)* – широкомасштабний прототип, застосовується для оцінювання основних правил та обмежень, що накладаються на систему.

Рис. 1. Основні проблеми, які розв'язує прототипування

На рис. 1 наведено схему з [9, с. 43], де відображено основні проблеми, яких стосується прототипування. Це виявлення, демонстрація й оцінювання таких характеристик програмної системи:

- структура та вимоги до даних;
- функціональні вимоги;
- операції та ефективність;
- організаційні потреби і проблеми.

Класифікація прототипів

Існують численні способи розбиття прототипів на категорії. Наприклад, у праці Девіса [7, с. 45] виділено такі різновиди прототипів: одноразові, еволюційні та операційні; вертикальні й горизонтальні; користувацькі інтерфейси та алгоритмічні; тощо. Різновид прототипу залежить від специфіки задачі, яка має бути розв'язана за його допомогою.

Розглянемо різновиди прототипів детальніше за [3, с. 160]. Якщо ризик проекту пов'язаний із застосуванням певного технологічного підходу, тобто певна технологія не застосовувалася у такій задачі раніше і невідомо, чи забезпечить вона належні виробничі характеристики, то варто створити архітектурний прототип, який демонструє можливість використання цієї технології. Якщо ж основний ризик в проекті становить користувацький інтерфейс і основною проблемою є уточнення вимог користувача, потрібен прототип вимог. Для архітектурних прототипів і для прототипів вимог характерний поділ на одноразові й еволюційні. Одноразові прототипи дають відповідь на певне питання щодо технології чи специфікацій, після чого більше їх не використовують у розробленні системи. Єдиний результат такого прототипу – знання. На відміну від них, еволюційні прототипи використовують як фундамент і подальший розвиток системи відбувається на базі доробок, отриманих при втіленні прототипу. У [9, с. 62] запропоновано інший погляд на одноразовий прототип: автор стверджує, що такий протип не відкидається, а трансформується, і так само, як еволюційний, стає основою системи, але неявно.

Також варто розглянути виміри прототипування. З точки зору вимірів виділяють горизонтальні та вертикальні прототипи.

Співвідношення вертикального і горизонтального прототипування наочно представлене на рис. 2 з [2, с. 75]. Горизонтальний прототип ілюструє характеристики системи без реалізації функціональності, а вертикальний прототип реалізує частину функціональності без демонстрації нереалізованих характеристик.

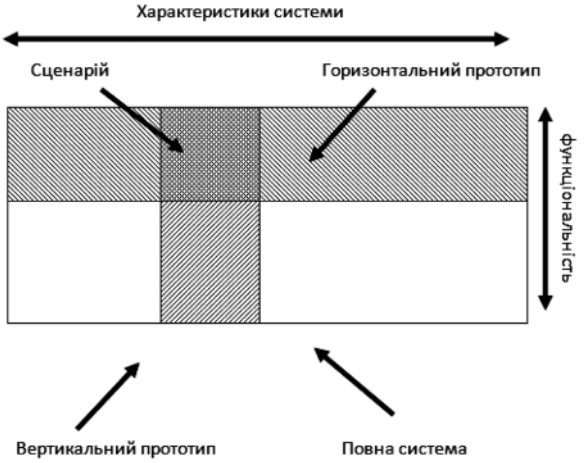

Рис. 2. Співвідношення горизонтального та вертикального прототипів

За [3, с. 159], горизонтальний прототип забезпечує широке представлення повної системи або підсистеми, зосереджуючись на користувацькому інтерфейсі більше, ніж на низькорівневій системній функціональності як, наприклад, доступ до бази даних. Горизонтальні прототипи використовують для підтвердження призначених для користувача інтерфейсних вимог і системного контексту, створення демонстраційних версій системи в маркетингових цілях, приблизної оцінки часових рамок і ресурсних затрат, яких потребуватиме проект.

Вертикальний прототип характерний повнішою розробкою деякої підсистеми або функції. Це корисно для отримання детальних вимог щодо певної функціональності. Вертикальне прототипування застосовують, наприклад, для вдосконаленого проектування бази даних, отримання інформації стосовно обсягів даних, системних інтерфейсних потреб, визначення характеристик

мережі та вимог щодо потужностей, прояснення складних вимог, оскільки воно дає змогу побудувати весь ланцюг взаємодій фактичної системної функціональності.

Підсумуємо розглянуті класифікації. На рис. 3 схема, яка базується на запропонованому Лефінгуелом та Уідрігом [3, с. 160] дереві рішень, яке робить можливим коректно обрати спосіб прототипування для проекту.

Рис. 3. Схема вибору способу прототипування

Переваги та недоліки прототипування

Використання прототипів під час моделювання та розроблення програмних систем базується на низці переваг, які надає прототипування. Водночас прототипування як модель життєвого циклу ПЗ нерідко зазнає критики, обгрунтованої певними недоліками. Переваги й недоліки прототипування наведено нижче.

Переваги прототипування

- *Мінімізація часових і фінансових витрат*: прототипування може підвищити якість вимог і специфікацій, що надаються розробникам. Раннє визначення дійсних вимог і потреб користувача спонукає до швидшої та дешевшої розробки програмного забезпечення.
- *Вдосконалене залучення кінцевого користувача до процесу створення*: наявність прототипу, досліджуваного користувачем, запобігає багатьом непорозумінням, які зазвичай виникають між програмістами та замовниками.
- *Навчання користувача:* прототип може слугувати для навчання користувачів роботі зі складною системою, доки сама система ще перебуває у розробленні [10].

Недоліки прототипування

- *Недостатній аналіз:* концентрація уваги на обмеженому прототипі може завадити коректному аналізу повного проекту. Це може призвести до вибору не найліпших рішень, підготовки неповних специфікацій або перетворення обмежених прототипів у неправильно спроектовані системи, які важко підтримувати.
- *Непорозуміння з користувачем, який не розуміє різниці між прототипом та завершеною системою*: якщо прототип містить тільки візуальну оболонку, замовникові може бути важко пояснити, що створення візуальної оболонки становить значно менше часу, ніж втілення всієї внутрішньої логіки, передбаченої специфікаціями.
- *Неправильне використання часу розробником:* ключова особливість прототипу – швидкість його створення. Якщо побудова прототипу та поліпшення його функціональності тривалі, відбувається спотворення основної парадигми прототипування, результатом чого є проблеми з часовими рамками проекту.
- *Зацикленість на прототипі:* після виготовлення прототипу в програмістів виникає спокуса подальшого його використання як бази проекту і розробка зводиться до надбудови модулів над моделлю. Такий підхід стає причиною некоректних архітектурних рішень.

Зі сказаного можна підсумувати, що прототипування має низку беззаперечних переваг, а недоліки прототипування не є вадами цього підходу, а тільки результатом його невдалого застосування.

Модель програмної системи, для якої будується прототип

Нехай маємо розподілену програмну систему, різні компоненти якої незалежні (і можуть виготовлятися різними організаціями). Взаємодія компонентів стандартизована і відповідає певним нормативним документам, де описано інтерфейси взаємодії. До того ж увесь комплекс має проходити випробування і сертифікуватися для застосування надалі. На етапі сертифікації та попереднього тестування доцільно деякі частини комплексу замінити прототипами. Особливості побудови таких прототипів, що слугують уніфікованими інструментами для тестування, – предмет дослідження тут.

Основне завдання прототипу в такий системі – перевірка можливості взаємодії модулів, виготовлених різними незалежними колективами розробників. Проте традиційне завдання прототипування також береться до уваги: прототип має бути побудований з максимізацією відсотку коду, який надалі використовуватимуть як у модулі, що в результуючій системі замінить прототип, так і в інших модулях розподіленої системи.

Прототип, що досліджується, можна класифікувати як вертикальний та еволюційний.

Аналіз вимог до прототипу об'єктно-орієнтованої програмної системи

Відмінність вимог до прототипу від вимог до готової програмної системи визначається харак-

теристиками прототипу. Горизонтальні одноразові прототипи, зокрема прототипи інтерфейсу користувача, можуть відповідати мінімальному набору вимог, серед яких провідне місце посідає зручність. Окрім того, за Е. Йорданом [2, с. 77], горизонтальний прототип, що ілюструє повний набір функцій майбутньої системи, не повинен декларувати додаткову функціональність, утілення якої неможливе (наприклад, через часові та фінансові обмеження). Натомість різновид прототипу, розглянутого тут, – еволюційний вертикальний прототип, – наближений до готової програмної системи, отже, має задовольняти ряд вимог до програмних систем. Вимоги до прототипу визначеного метою його створення, яка виражена у двох підзавданнях:

- прототип має реалізовувати не тільки візуальні, а й специфічні функціональні вимоги в обсязі, достатньому для проведення повного і всеохопливого тестування інших частин програмної системи, з якими взаємодіє прототип;
- базова функціональність, притаманна не лише прототипу, а й модулю, що його замінить, має бути реалізована за допомогою компонентів, найадаптованіших для повторного використання.

Відповідність стандартам

Як зазначає М. Сміт, «внутрішня технічна якість прототипу є серйозною проблемою. Погіршення внутрішньої якості та відповідності стандартам може несприятливо вплинути на кошторис, довгострокову підтримку й ефективність» [9, с. 24].

Як і в будь-якій галузі промисловості, в ІТ-сфері стандарти є ключовими документами, що регулюють розробку програмних систем, маючи на меті працездатність, масштабованість та інтероперабельність даних систем. Аналогічно до повноцінної програмної системи, еволюційні прототипи мають відповідати стандартам, за якими розроблятиметься програмний продукт. Прототипи можуть не виконувати вимог стандартів у повному обсязі, проте вони не мають містити логіки, що суперечить положенням стандартів.

Розробка прототипів вимог, горизонтальних та одноразових прототипів за допомогою стандартизованих засобів мінімізує необхідність роботи зі стандартами. По суті, єдиним стандартом є специфікації замовника. Проте у випадку прототипування внутрішньої логіки системи (вертикальний прототип) дослідження стандартів, зокрема, специфічних для даної предметної області, є першочерговим завданням, яке має передувати прототипуванню.

Модульність та зв'язки між модулями. Підходи до проектування

У дослідженні Дж. Саметінгера зазначено: «Повторне використання коду та модульність програмних систем ідеально доповнюють одне одного. Використання компонентів для побудови програмних систем майже автоматично призводить до повторного використання коду. І намагання створити код, придатний для повторного використання, майже автоматично призводить до компонування програмного продукту з компонентів» [8, с. 12]. Таким чином, розглядаючи модульну структуру прототипу та підходи до її проектування, постійно матимемо мету – створення модулів, придатних для повторного використання.

Вертикальний еволюційний прототип закладає основу всієї системи, отже, правильне його проектування вкрай важливе. Водночас прототип складається як із фрагментів, що увійдуть до наступних версій системи, тому мають бути спроектовані гнучко з можливістю зручного розширення, так і з частин, що слугують заглушками.

За [8, с. 12], є такі техніки повторного використання коду: композиційне повторне використання та генеративне повторне використання. Композиційне повторне використання дає змогу збирати системи з готових низькорівневих компонентів. Генеративне повторне використання адаптує спеціальні системні структури, архітектури, інтерфейси до конкретних компонентів. Доробки, отримані внаслідок прототипування, найкраще використовувати такими шляхами:

- *композиція компонентів*. Компоненти, що розробляються, потрапляють до готової системи;
- *копіювання дизайну та коду.* Для копіювання коду з прототипу братимуть фрагменти коду, для копіюванні дизайну – великі фрагменти коду, в яких змінюватимуть деталі, а каркас залишатимуть.

Враховуючи специфіку задачі, повторно використовуватимуть класи, що відображають базові сутності предметної області. Класи, що втілюють бізнес-логіку, у прототипі можуть представляти ланки між робочими частинами та заглушками, тобто слугуватимуть для копіювання дизайну та коду.

Програмування на основі взірців

Деякі автори, зокрема С. Беркун [1, с. 139], пропонують погляд на прототип як на сукупність ідей: може існувати набір альтернатив, кожна з яких прототипується в загальних рисах, після чого обираються найвдаліші ідеї. Кожна окрема

вдала ідея може бути прикладом вдалого застосування готових взірців проектування або може стати новим взірцем.

Беркун зазначає, що хоча досвідчені проектувальники та розробники з більш розвиненими творчими навиками можуть одразу виділити альтернативні напрямки ідей, менш досвідченій команді групування ідей допоможе визначити кількість комбінацій, не беручи до уваги інтерпретації кожної окремої ідеї.

Під час декомпозиції прототипу та проектування його частин на основі взірців варто дотримуватися, на думку С. Беркуна, таких принципів:

- «з кожної групи ідей варто обрати єдину та скомбінувати дані ідеї в єдиний задум.
- Варто створити невеликі прототипи для кожної з груп ідей і проаналізувати результат. Необхідно визначити, наскільки кожен з напрямів відповідає загальному задуму.
- Варто створити перелік найскладніших чи найважливіших питань проектування та прототипи, які б допомогли на них відповісти. Але слід брати до уваги, що складніший прототип дає змогу отримати відповідь на більш складніші питання. Поверхневі ескізи, зроблені нашвидкуруч, дадуть відповідь лише на початкові та дуже приблизні питання, а для отримання специфічної інформації необхідно розробити більш суттєву модель.

Під час створення прототипів стає зрозуміло, які ідеї можуть бути додатково використані, а які більше не вписуються в загальний контекст. Як у мозаїці, деякі елементи підходять за змістом більше, ніж інші, проте для пошуку відповідностей необхідно діяти методом спроб і помилок. Коли наявні численні точки зору, передбачити, які напрями будуть працездатними, дуже важко. Для цього й існують прототипи» [1, с. 140].

Аналіз залежностей між модулями

Ділячи компоненти прототипу на призначені для подальшого використання та заглушки, необхідно дослідити залежності між цими компонентами. Модулі, потрібні для подальшого використання, в жодному разі не можуть залежати від модулів-заглушок. Для забезпечення цього принципу необхідно створювати ієрархії, які в прототипі здаються зайвими, проте слугують для подальшої можливості гнучкого використання коду. Для дослідження залежностей між компонентами прототипу зручно побудувати граф посилань.

Врахування характеристик якості у прототипах програмних систем

Розглянемо характеристики якості програмного забезпечення стосовно прототипів, перелік яких взято із «Посібника Microsoft з проектування архітектури застосувань» [4, с. 169]. Визначимо, які характеристики є важливими для прототипу як уніфікованого інструменту для тестування модулів розподіленої програмної системи, а які стосуються тільки готових програмних систем. Результати проведеного аналізу відображено у табл. 1 (перші 3 колонки базуються на [4, с. 169–172], а в останній наведено значущість критеріїв для аналізованого прототипу з точки зору автора).

Таким чином, є низка характеристик якості програмного забезпечення, які важливі для прототипування. Для їхнього забезпечення слід звернутися до джерел, що описують забезпечення даних характеристик для програмних систем загалом [4; 11].

Зокрема дослідження [11] показує, що неправильна обробка некоректних вхідних параметрів часто стає причиною помилок у роботі програмних систем, а також дає зловмисникам можливість зламу системи. Отже, всі набори вхідних значень системи та реакція на них мають бути проаналізовані на етапі прототипування.

Розробка методичних рекомендацій до побудови прототипу

Автор цієї роботи на основі зазначених принципів побудови прототипів та обмежень, що на них накладаються, розробив методичні рекомендації для побудови прототипу компоненту, що використовуються для тестування пов'язаних із ним компонентів розподіленої системи.

Е. Сміт [9, с. 50] описує процес прототипування за допомогою діаграми, наведеної на рис. 4. Нижче вказано рекомендації, які уточнюють схему Сміта, враховуючи особливості прототипу та сучасні підходи до проектування програмних систем.

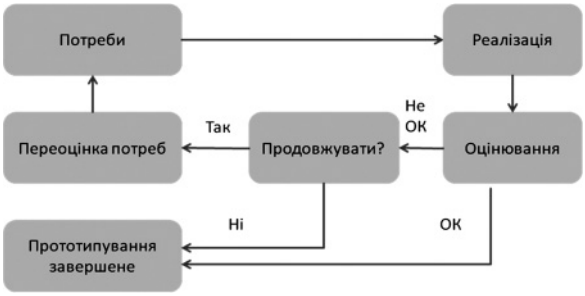

Рис. 4. Процес прототипування

Прототипування варто здійснювати поетапно.

1. Аналіз стандартів і задокументованих вимог, що описують функціональність програмної системи і, відповідно, прототипу. Перелік аналізованих стандартів залежить від рівня абстракції та об'ємів повторно використовуваного коду.

Таблиця 1. **Критерії якості програмного забезпечення**

Кат.	Критерій якості	Опис	Значущість для прототипу
Дизайн	Концептуальна цілісність	Визначає узгодженість і зв'язність дизайну в цілому. У рамках критерію розглядають те, як спроектовані компоненти та модулі, а також стиль написання коду	Висока значущість. Оскільки прототип є еволюційним, код, що використовуватиметься надалі, має повністю задовольняти цей критерій
	Зручність і простота обслуговування	Визначає здатність системи змінюватися. Це стосується зміни компонентів, сервісів, функцій та інтерфейсів при додаванні чи зміні функціональності, виправленні помилок і реалізації нових бізнес-вимог	Висока значущість. Вертикальний еволюційний прототип однозначно буде змінюватися, коли на його основі створюватиметься повноцінна система. Окрім того, компоненти прототипу планується використовувати в інших модулях, які розроблятимуть інші програмісти
	Можливість повторного використання	Визначає пристосованість компонентів та підсистем до використання в інших застосуваннях і сценаріях. Можливість повторного використання забезпечує зниження дублювання компонентів, а також скорочення часу на реалізацію	Висока значущість для окремо визначених компонентів
Час виконання	Доступність	Визначає, яку частину часу система функціональна і працює. Доступність може бути виміряна як процентне відношення часу простою системи і заданого часу. На доступність впливають помилки системи, проблеми інфраструктури, атаки зловмисників і навантаження системи	Значущість залежить від контексту. Найчастіше цей критерій доцільно перевіряти на готовій системі, а не на прототипі з обмеженою функціональністю. Проте у випадку тестування доступності всіх модулів розподіленої системи доступність прототипу не повинна спотворювати загальної картини
	Можливість взаємодії	Здатність системи успішно працювати через взаємодію та обмін даними з іншими зовнішніми системами, створеними і виконуваними зовнішніми сторонами. Можливість взаємодії спрощує обмін і повторне використання даних як ззовні, так і всередині	Висока значущість через особливості прототипу (як уніфікованого інструменту для тестування)
	Керованість	Визначає, наскільки легко системним адміністраторам керувати застосуванням. Зазвичай визначена засобами достатнього і корисного інструментарію, що надається для використання в системах моніторингу, а також для налагодження та налаштування продуктивності	Невисока значущість, яка залежить від контексту використання прототипу. Критерій важливий лише в ситуації, коли прототип буде надано для багаторазового використання стороннім особам
	Продуктивність	Це показник, що характеризує швидкість, з якою система виконує будь-яку дію в заданий проміжок часу. Продуктивність вимірюється в показниках затримки чи пропускної спроможності. Затримка – це час, необхідний для відповіді на будь-яку подію. Пропускна спроможність – це число подій, що відбуваються за певний проміжок часу	Найчастіше низька значущість. Загалом значущість залежить від контексту, від того, наскільки продуктивність компонентів прототипу, що використовуватимуться надалі, впливає на продуктивність системи в цілому. Зокрема, коли допустимий час виконання операції кілька секунд і кількість одночасних операцій невелика, окремі тести на продуктивність є недоцільними, і значущість цього критерію для прототипу і майбутньої програмної системи низька. Якщо ж майбутня система має ефективно використовувати ресурси, то для забезпечення вимог продуктивності оптимізація має стосуватися всіх рівнів. Прототип не відображатиме реальної картини щодо цього параметру
	Надійність	Здатність системи зберігати працездатність протягом певного часу. Надійність визначається як ймовірність того, що система зможе виконувати передбачені функції протягом заданого проміжку часу	Значущість (бажано висока) залежить від контексту. У випадку тестування надійності всіх модулів розподіленої системи надійність прототипу не повинна спотворювати загальної картини. Окрім того, надійними мають бути компоненти, призначені для подальшого використання
	Масштабованість	Здатність системи справлятися зі збільшенням навантаження без впливу на її продуктивність або здатність легко розширюватися	Невисока значущість, якщо тільки прототип не призначений для тестування розподілу навантаження

Продовження табл. 1

Кат.	Критерій якості	Опис	Значущість для прототипу
	Безпека	Здатність системи запобігати зловмисним чи випадковим діям, не передбаченим при проектуванні, або не допускати розголошення чи втрати даних	Висока значущість. По-перше, значно легше закласти підвалини реакції на некоректні значення та виняткові ситуації на етапі розробки системи ніж додавати їх пізніше. По-друге, оскільки з прототипом зазвичай взаємодіє користувач, ця частина системи найчутливіша до некоректно введених даних. Водночас за відсутності належної обробки у користувача може виникнути враження, що система загалом працює нестабільно. Більш того, некоректна реакція прототипу на певний набір вхідних даних спотворює результати тестів
Якості системи	Забезпечення технічною підтримкою	Здатність системи надавати інформацію, необхідну для виявлення та діагностики проблем при некоректній роботі	Висока значущість. Прототип має надавати точну інформацію щодо локалізації помилки (чи вона спричинена некоректністю прототипу як інструменту для тестування, чи некоректністю модуля, що тестується)
	Тестованість	Міра того, наскільки легко створити критерій перевірки для системи та її компонентів, виконати ці тести. Гарна тестованість означає високу ймовірність того, що збої в системі можуть бути своєчасно та ефективно ізольовані	Висока значущість, оскільки досліджуваний прототип є інструментом для тестування
	Легкість і зручність використання	Визначає, наскільки застосування відповідає вимогам користувача з точки зору зрозумілості, легкості локалізації й глобалізації, забезпечення гарної взаємодії з користувачем загалом	Часткова значущість. Прототип має бути зручним у використанні, проте, оскільки точно відомо, що він використовуватиметься обмеженим колом користувачів, немає необхідності адаптувати його інтерфейс для різних вимог (локалізувати, адаптувати для людей з особливими потребами тощо)

Під час використання готових низькорівневих бібліотек вважають, що дані бібліотеки вже забезпечують відповідність низькорівневим стандартам.

Найбільш наочним є приклад для мережної частини ПЗ. При написанні програми, що працює з протоколами прикладного рівня, вважають, що протоколи нижчих рівнів моделі OSI реалізуються операційною системою та готові сокети, що використовують, наприклад, Java, .NET бібліотеки. Водночас при написанні драйвера мережної карти аналізу підлягатимуть протоколи канального рівня моделі OSI.

Після аналізу стандартів необхідно виділити ті вимоги, які має реалізовувати прототип. Вимоги можуть поділятися на такі, що мають бути реалізовані в повному обсязі, частково або не реалізовані взагалі. Результатом аналізу стандартів має бути перелік функціональних вимог до прототипу програмної системи, сформульованих у прийнятому в колективі розробників вигляді (наприклад, UML-діаграми, таблиці тестових сценаріїв тощо).

2. Детальний аналіз предметної області, побудова онтології або відповідних діаграм предметної області (за наявності попередніх діаграм здійснюється їхній перегляд). Цей етап дає змогу виокремити сутності, які є спільними для різних частин розподіленої програмної системи, визначити, яким чином реалізувати зв'язки між сутностями, зокрема побудувати ієрархію класів.

3. Вибір фреймворку і технологій для втілення прототипу, складання загальної схеми компонентів. На цьому етапі важливим є виділення частин, що використовуватимуть готовий код частин, для яких розроблятиметься код, що ввійде в готову систему, і частин, які будуть створені тільки для цього прототипу. Зазначимо, що етапи 2 та 3 є пов'язаними, уточнення онтології можуть призводити до змін у діаграмі компонентів, а діаграма компонентів може спонукати до деталізації онтології.

4. Аналіз послідовностей подій, виокремлення найскладніших ситуацій і дослідження альтернативних рішень. Цей етап прототипування варто зробити циклічним, тобто до вибору ідеї, якщо вона виявиться невдалою, слід повернутися до проектування відповідної частини знову. Таким чином, до прототипу також можливо застосувати спіральну модель життєвого циклу програмних систем, враховуючи, що кожен виток спіралі має бути якомога коротшим. Окрім того, варто прийняти рішення, чи є помилки проектування прототипу критичними.

Зазначимо кілька причин для застосування спіральної моделі до окремих частин прототипу (відповідно до циклу на рис. 3):

- ця частина прототипу має повторно використовуватися в готовій системі;
- прототип розробляється для тестування частин розподіленої системи. Такий прототип може використовуватися неодноразово і потребувати супроводу в разі виявлення невідповідностей у стикуванні модулів розподіленої системи. Отже, код має бути якомога читабельнішим і зручнішим у супроводі, тому заплутані ділянки коду мають бути замінені з використанням певних прийомів проектування, а також взірців. Варто зазначити, що у модулях прототипу, що не будуть повторно використовуватися, слід уникати зайвих ієрархій та узагальнених класів;
- обране рішення не забезпечує повного покриття функціональності прототипу.

5. Програмування та поєднання складових прототипу на основі попередньо проведеного проектування.

6. Тестування прототипу та внесення змін до нього.

Висновки

Основна мета прототипування – раннє втілення та перевірка функціональності проекту. На відміну від горизонтальних протоипів, які слугують для ілюстрації, вертикальні еволюційні прототипи є каркасом майбутньої системи, а тому їхнє створення потребує детального вивчення вимог до прототипу та застосування прийомів проектування, які зазвичай використовують в розробці повноцінних програмних систем.

Література

1. Беркун С. Искусство управления IT-проектами / Скотт Беркун. – М. : «Питер». – 2011. – 725 с.
2. Йордан Э. «Смертельный марш». Полное руководство для разработчика программного обеспечения по выживанию в безнадежных проектах / Эдвард Йордан ; пер. с англ. А. М. Вендрова. – Prentice Hall. – 1997.
3. Леффингуэлл Д. Принципы работы с требованиями к программному обеспечению. Унифицированный поход / Д. Леффингуэлл, Д. Уидриг; пер. с англ. Н. Ореховой. – М. : Издательский дом «Вильямс», 2002. – 448 с.
4. Руководство Microsoft по проектированию приложений. – 2-е издание. – Корпорация «Microsoft», 2009.
5. Crinnion J. Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology / John Crinnion. – New York : Plenum Press. – 1991. – P. 18.
6. Davis A. Software Prototyping / A. Davis // In Advances in Computers. Chestnut Hill, MA: Academic Press. – 1995. – Vol 40. – P. 39–62.
7. Nielsen J. Usability Engineering / Jacob Nielsen. – Boston : Academic Press. – 1993. – 343 p.
8. Sametinger J. Software Engineering with Reusable Components / Johannes Sametinger. – Springer-Verlag, 1997. – 511 p.
9. Smith M. F. Software prototyping: adoption, practice and management / M. F. Smith. – London : McGraw-Hill, 1991. – 368 p.
10. Teach ICT. – Режим доступу: http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/331_systems_cycle/prototyping_RAD/miniweb/pg5.htm. – Назва з екрану.
11. The 2009 CWE/SANS Top 25 Most Dangerous Programming Errors. – Режим доступу: http://cwe.mitre.org/top25/. – Назва з екрану.

A. Domina

FEATURES OF OBJECT-ORIENTED SYSTEMS PROTOTYPING

The subject of this article is a Software systems and their components prototyping as a part of software systems design.

Creation of the prototype which is a part of a distributed system and is used for unified testing instrument for the parts of this system is covered in the article. Basic requirements and development recommendations for such kind of prototype are investigated and formulated.

Theoretical part of the research was practically applied and verified while development of the described prototype, which was included into a business solution.

Keywords: prototyping, software architecture, distributed systems, reusable code, software requirements.

Матеріал надійшов 30.03.2012