УДК 004.891.3: 004.3

Т.О. ГОВОРУЩЕНКО, Р.А. МАЛЯРЧУК
Хмельницький національний університет

АНАЛІЗ ПРОЦЕСУ ВИБОРУ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ, МЕТОДОЛОГІЇ ТА СЕРЕДОВИЩА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даній статті виконано аналіз сучасних технологій проектування, методологій та середовищ розроблення програмного забезпечення (ПЗ), в результаті якого було визначено основні переваги, недоліки та використовуваність технологій/методологій/середовищ. При наявній великій кількості технологій/методологій/середовищ частка проблемних проектів залишається сталою, а процес розроблення ПЗ залишається недетермінованим. Автори провели аналіз процесу оцінювання та вибору технології/методології/середовища і показали, що на даний момент в галузі панує повний суб'єктивізм вибору. У роботі доведено актуальність задачі підтримки процесу вибору технології/методології/середовища для ПЗ.

Ключові слова: програмне забезпечення (ПЗ), життєвий цикл програмного забезпечення (ЖЦ ПЗ), технологія проектування програмного забезпечення (ТППЗ), методологія розроблення програмного забезпечення (МРПЗ), середовище розроблення програмного забезпечення (СРПЗ).

T.O. HOVORUSHCHENKO, R.A. MALYARCHUK
Khmelnytsky National University

ANALYSIS OF PROCESS OF SELECTION OF DESIGN TECHNOLOGY, METHODOLOGY AND ENVIRONMENT OF SOFTWARE DEVELOPMENT

Abstract – The paper is dedicated to substantiation of actuality of task of support of the process of evaluation and selection of software design technology, software development methodology and environment.

In this article, the analysis of modern software design technologies, software development methodologies and environments was performed. As its results the main advantages, disadvantages and the possibilities of using of software technologies/methodologies/environments were determined. The large number of technologies/methodologies/environments are available, but share of problem software projects remains constant, and the process of software development is nondeterministic. The authors analyzed the process of evaluation and selection of technology/methodology/environment and showed, that currently the full subjectivism of choice prevails in the field.

The authors have formulated the tasks of further research. The solving of these tasks will make reasoned and informed decisions regarding the predominance and choice of software design technology, software development methodology and environment.

Keywords: software, software lifecycle, software design technology (SDT), software development methodology (SDM), software development environment (SDE).

Вступ

В реальних умовах проектування програмного забезпечення (ПЗ) – це пошук способу, який забезпечує необхідну функційність програмної системи засобами наявних технологій з врахуванням заданих обмежень.

Статистика успішності програмних проектів за даними The Standish Group International (Chaos reports) [1, 2] представлена на рисунку 1.

	2004	2006	2008	2010	2012
Успішні	29%	35%	32%	37%	39%
Неуспішні	18%	19%	24%	21%	18%
З недоліками	53%	46%	44%	42%	43%

Рис.1. Статистика щодо успішності впровадження програмних проектів проектів за даними The Standish Group International (Chaos reports) [1, 2] у 2004–2012 роках

Аналіз даних рисунку 1 дав можливість побачити приріст кількості успішних проектів та спад кількості неуспішних проектів, але в той же час частка проектів з недоліками є досить сталою величиною і складає мінімум 42%. При цьому лише невелика кількість (максимум 39% за 1994–2012 роки) проектів має необхідні функціональні можливості при існуючих обмеженнях за вартістю та терміном [2].

При розробленні програмних продуктів постає проблема формулювання бізнес-вимог, які висуваються до програмного продукту. Значною мірою помилки проектування, які призводять до неуспішності або недоліків програмних проектів, обумовлені неточним вираженням цих вимог. З аналізу сучасного стану галузі створення ПЗ випливає, що наразі при створенні ПЗ програмістам доводиться використовувати комплексні засоби розроблення, які складаються з великої кількості важливих та необхідних компонентів для створення повнофункціональних програмних додатків. Такими засобами і компонентами, які об'єднують, доповнюють та покращують один одного, є технологія проектування ПЗ, методологія розроблення ПЗ та середовище розроблення ПЗ. Сукупність таких програмних засобів забезпечує підтримку всіх етапів розроблення будь-якого ПЗ – написання сирцевого коду програми, його

редагування та виправлення, швидке під'єднання додаткових модулів, можливості зручної взаємодії з іншими засобами та інструментами, а також тестування та налагодження створюваного ПЗ.

Технологія проектування програмного забезпечення (ТППЗ) – це впорядкована сукупність взаємозв'язаних технологічних процесів в межах життєвого циклу програмного забезпечення [3]. ТППЗ – це сукупність трьох складових: 1) покрокової процедури, яка визначає послідовність технологічних операцій; 2) критеріїв та правил, використовуваних для оцінки результатів виконання технологічних операцій (відповідність стандартам); 3) нотацій (графічних та текстових засобів), використовуваних для опису проектованої системи. ТППЗ представляє собою інженерний підхід до розроблення програмних засобів, який охоплює методологію програмування, проблеми забезпечення надійності програм, оцінки робочих характеристик та якості проектів. ТППЗ визначає професійну культуру фахівців, яка забезпечує заданий рівень продуктивності праці і якості одержуваної в результаті програмної продукції [3,4]. ТППЗ об'єднує в собі всі специфічні факти, рівні, важливі знання, заходи, операції, прийоми щодо керування розробленням ПЗ.

Методологія розроблення ПЗ – це набір методів та критеріїв оцінки, які використовуються для постановки задачі, планування, контролю і для досягнення поставленої мети [5]. Процес розроблення описується моделлю, яка визначає послідовність найбільш загальних етапів та одержуваних результатів. Методологія розроблення ПЗ – це система принципів, а також сукупність ідей, понять, методів, способів та засобів, які визначають стиль розроблення ПЗ. Саме методологія визначає, як виконуватиметься розроблення ПЗ. Методології представляють собою ядро теорії керування розробленням ПЗ [6].

Середовище розроблення ПЗ – це система програмних засобів, використовувана програмістами для розроблення програмного забезпечення [7]. Середовище розроблення включає в себе текстовий редактор, компілятор та/або інтерпретатор, засоби автоматизації та відлагоджувач. Іноді містить також засоби для інтеграції з системами керування версіями та різноманітні інструменти для спрощення конструювання графічного інтерфейсу користувача. Сучасні середовища розроблення ПЗ можуть містити також браузер класів, інспектор об'єктів та діаграму ієрархії класів для використання при об'єктно-орієнтованому розробленні ПЗ.

Останнім часом питанням вибору технології проектування, методології та середовища розроблення ПЗ приділяється підвищена увага: як показує досвід, без вірної технології/методології/середовища навіть невеликі проекти навряд чи можуть бути успішними, і сьогодні все більше розробників, аналітиків та керівників проектів починають це усвідомлювати.

Аналіз технологій проектування програмного забезпечення

Технологія проектування ПЗ повинна забезпечувати: 1) підтримку повного життєвого циклу ПЗ; 2) гарантоване досягнення цілей розроблення ПЗ; 3) можливість виконання великих проектів у вигляді підсистем; 4) можливість ведення робіт з проектування окремих підсистем; 5) мінімальний час одержання роботоздатного ПЗ; 6) можливість керування конфігурацією проекту, ведення версій проекту та його складових, можливість автоматичного випуску проектної документації та синхронізацію її версій з версіями проекту; 7) незалежність виконуваних проектних рішень від засобів реалізації ПЗ. Крім цього, ТППЗ повинна підтримуватись комплексом узгоджених CASE-засобів [3].

Провідними CASE-технологіями проектування програмного забезпечення наразі є [3, 4]: Rational Unified Process (RUP); Oracle; Borland; Computer Associates.

Rational Unified Process (RUP) – одна з найбільш досконалих технологій, яка пропонує ітеративний підхід до проектування. RUP значною мірою відповідає стандартам, зв'язаним з процесами життєвого циклу (ЖЦ) ПЗ та оцінкою технологічної зрілості організацій. RUP спирається на інтегрований комплекс інструментальних засобів Rational Suite. Технологія RUP – це зв'язний цикл проектування: від постановки задачі до готового продукту, тому ця технологія надає ряд *переваг*: 1) зміни і уточнення вимог виявляються вже на ранніх етапах розроблення, коли трудомісткість внесення змін є досить низькою; 2) можливість залучення фахівців замовника для оцінювання проміжних версій (прототипів) вже на ранніх етапах розроблення, яка надає високу ймовірність того, що кінцевий продукт робитиме саме те, чого від нього очікує замовник; 3) зниження архітектурних та інтеграційних ризиків завдяки стійкій, перевіреній архітектурі; 4) більш повне повторне використання програмних елементів, що сполучається з ідеями об'єктно-орієнтованого програмування. Суттєвим *недоліком* технології RUP є слабкі інструментальні засоби для системного аналізу, який передує розробленню специфікації. Через слабкість інструментальних засобів технологія RUP не може повною мірою надати розробникам свої переваги. Технологія проектування RUP наразі *часто використовується* при створенні систем автоматизованого проектування, CASE-систем, систем з елементами штучного інтелекту, а також систем підтримки структурно-параметричного синтезу об'єктів [8].

ТППЗ Borland (Borland Together, Borland ECO) – це технологія проектування ПЗ компанії Borland, яка ґрунтується на інтегрованому комплексі інструментальних засобів ALM, що реалізують керування повним життєвим циклом додатків. *Переваги ТППЗ Borland Together* [9]: 1) створення та використання шаблонів, завдяки якому відбувається стимулювання повторних розроблень успішних програмних проектів, скорочення витрат на розроблення ПЗ; 2) багаторівневе тестування та аудит з використанням метрик; 3) підтримка стандартів програмування; 4) відповідність промисловим стандартам моделювання UML, XMI, OCL, QVT; 5) автоматична синхронізація моделей та коду. *Недоліками ТППЗ Borland Together* [10] є її

орієнтація, в першу чергу, на створення коду, а не на створення моделі, а також завищені вимоги до ресурсів робочої станції, на якій використовуватиметься розроблюваний програмний продукт. Технологія проектування Borland *може застосовуватись* як софтверними компаніями, які реалізують великі дорогі програмні проекти, так і невеликими софтверними компаніями завдяки демократичним цінам багатьох редакцій цієї технології.

Методологічну основу *ТППЗ корпорації Oracle* складають наступні методи, які охоплюють більшість процесів ЖЦ ПЗ: CDM – розроблення прикладного ПЗ; PJM – керування проектом; AIM – впровадження прикладного ПЗ; BPR – реінжиніринг бізнес-процесів; OCM – керування змінами. ТППЗ Oracle спирається на інтегрований комплекс інструментальних засобів Oracle Developer Suite. *Переваги технології Oracle*: 1) підтримка стандарту UML-моделювання об'єктних додатків, завдяки якій на основі змодельованих класів та робочих процесів відбувається генерація коду для середовища BC4J; 2) збереження моделей в загальному репозиторії Oracle, що забезпечує зручний контроль за версіями об'єктів; 3) підтримка XML для обміну даними з іншими UML-інструментами; 4) існування репозиторію для підтримки роботи великих колективів розробників, який дозволяє керувати процесом оновлення версіями об'єктів; 5) спрощення розроблення та підтримки Web-додатків завядки набору JavaBean-компонентів інтерфейсу користувача, які забезпечують узгодженість інтерфейсу та полегшення його налагодження та локалізації. *Недоліки технології Oracle*: 1) підвищена складність проектування та розроблення прикладних програмних систем через величезну кількість можливостей при їх реалізації; 2) зростання ризиків у проектах прикладних програмних систем через підвищену складність проектування та розроблення; 3) труднощі при організації розроблення через стандартизацію зовнішнього вигляду програм. Технологія Oracle містить функційно повний набір інтегрованих *засобів розроблення для* швидкого створення та розгортки Інтернет-додатків, динамічних Web-порталів та розгортання Web-сервісів [11].

Технологія Computer Associates (CA) ґрунтується на комплексах інструментальних засобів підтримки різних процесів ЖЦ ПЗ, зокрема на CASE-засобах AllFusion Modeling Suite, AllFusionComponentModeler. *Переваги технології CA*: 1) підтримка всіх етапів життєвого циклу ПЗ; 2) скорочення витрат та підвищення продуктивності процесу розроблення; 3) оптимізація діяльності організації та перевірка її на відповідність стандартам ISO 9000 [12], що полегшує сертифікацію; 4) підтримка трьох нотацій моделювання – IDEF0 (функційне моделювання), IDEF3 (моделювання потоків робіт), DFD (моделювання потоків даних); 5) можливість виявлення недоліки та помилки проектування; 6) сумісний доступ та редагування моделей (наприклад, функційної разом з об'єктною); 7) інтеграція з іншими технологіями та продуктами Computer Associates, Microsoft, Rational Software; 8) низька вартість та широке поширення. *Недоліками технології CA* є: 1) складне керування системами; 2) витрати більшого часу на виконання тих же задач. *CASE-засоби технології CA використовуються*, наприклад: AllFusion Modeling Suite – для задоволення всіх потреб компаній-розробників ПЗ, AllFusionComponentModeler – для моделювання компонентів ПЗ та генерації об'єктного коду додатків на основі створених моделей як при створенні нових додатків, так і при зміні або об'єднанні існуючих [13]. Технологію CA використовують такі потужні компанії, як NATO, StanCorp, Portman, QuantumResearch, НИИ «Восход» [13].

Проведений аналіз популярних CASE-технологій проектування ПЗ показав, що більшість існуючих CASE-засобів базуються на методологіях структурного або об'єктно-орієнтованого аналізу та проектування, який використовує специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів. Згідно огляду передових технологій (Survey of Advanced Technology) [14], складеному фірмою Systems Development Inc. на основі анкетування більше 1000 американських фірм, CASE-технології наразі потрапили до найбільш стабільних інформаційних технологій (їх використовували 50% всіх опитаних користувачів більш ніж у третині своїх проектів, 85% з яких завершились успішно). Однак, при всіх потенційних можливостях CASE-технологій, існує чимало прикладів їх невдалого впровадження, в результаті яких програмні проекти стають «полочним» ПЗ (shelfware). В зв'язку з цим слід відзначити наступне: 1) CASE-технології не обов'язково дають негайний ефект – він може бути одержаний лише через якийсь час; 2) реальні витрати на впровадження CASE-технології зазвичай набагато перевищують витрати на їх придбання; 3) CASE-засоби забезпечують можливості для одержання існуючої вигоди лише після успішного завершення процесу їх впровадження.

Під впровадженням ТППЗ розуміють всі дії – від оцінювання початкових потреб до повномасштабного використання ТППЗ у різних підрозділах організації [3]. Процес впровадження складається з наступних етапів [3]: 1) визначення потреб у ТППЗ, характеристик об'єкту впровадження та проектів створення ПЗ; 2) визначення вимог, які висуваються до ТППЗ (аналіз характеристик об'єкту впровадження та проектів, обґрунтування вимог до ТППЗ, визначення пріоритетів вимог); 3) *оцінювання варіантів ТППЗ – попереднє експертне оцінювання, яке полягає у аналізі доступних ТППЗ на предмет відповідності вимогам, та деталізоване оцінювання, яке полягає у формуванні детального опису кожної ТППЗ-претендента*; 4) визначення потреб у ТППЗ, характеристик об'єкту впровадження та проектів; 5) *вибір ТППЗ на основі порівняльного аналізу технологій та з врахуванням експертної оцінки*; 6) адаптація ТППЗ до умов застосування шляхом формування конкретної робочої конфігурації ТППЗ, адаптованої до умов об'єкту впровадження.

Сьогодні процеси оцінювання та вибору ТППЗ не забезпечені математичним підґрунтям або

рекомендаціями стандартів щодо їх проведення, базуються на експертних оцінках, які часто є суб'єктивними, або на оцінках-описах різних ТППЗ, тому актуальною задачею наразі є підтримка процесів оцінювання та вибору технології проектування програмного забезпечення.

Аналіз методологій розроблення програмного забезпечення

Процес розроблення ПЗ описується моделлю, яка визначає послідовність найбільш загальних етапів та одержуваних результатів. Наразі відомі та вживані декілька основних моделей ЖЦ ПЗ: каскадна модель ("waterfall", "водоспад"), інкрементна модель, спіральна модель ("spiral"), еволюційна модель (RAD). Крім цього, широко використовується парадигми гнучкого розроблення та екстремального програмування.

Каскадна модель – модель процесу розроблення програмного забезпечення, в якій процес розроблення виглядає як потік, що послідовно проходить етапи визначення та аналізу вимог, проектування, реалізації (конструювання, кодування), тестування та налагодження (верифікації), інтеграції та підтримки. Перехід від одного етапу до іншого відбувається тільки після повного і успішного завершення попереднього етапу. *Недоліки каскадної моделі* [15]: 1) потреба у точному наборі автономних етапів процесу розроблення ПЗ; 2) необхідність фіксації всіх вимог на початковому етапі проекту; 3) неврахування змінюваних потреб користувачів та змін у зовнішньому середовищі; 4) істотні труднощі та великі витрати часу та коштів на усунення помилки внаслідок великого розриву між часом внесення помилки та часом її виявлення; 5) можливість тестування, оцінювання якості та внесення зауважень лише після завершення роботи над ПЗ; 6) відсутність можливості трактування ПЗ як процесу розв'язування задачі. *Фактори ризику* каскадної моделі [15]: 1) недостатня чіткість формулювання вимог до ПЗ; 2) можливе погіршення процесу розроблення ПЗ через внесення швидких змін у технологію та вимоги; 3) звуження можливостей реалізації ПЗ через обмеження на ресурси; 4) непридатність отриманого продукту через недостатнє розуміння розробниками вимог або через недостатність тестування.

В основу *інкрементної моделі* ЖЦ ПЗ покладено наступну концепцію: перша створювана проміжна версія ПЗ (випуск 1) реалізує частину вимог, в наступну версію (випуск 2) додають додаткові вимоги і так доти, доки не будуть остаточно виконані всі вимоги та вирішені завдання розроблення ПЗ. Для кожної проміжної версії на етапах ЖЦ виконуються необхідні процеси, роботи та завдання, в тому числі аналіз вимог та створення нової архітектури, які можуть бути виконані одночасно. Наразі інкрементна модель частіше розглядається в контексті поступового нарощування функціональності створюваного продукту. Інкрементну модель ЖЦ *доцільно використовувати* у випадках, коли: бажано реалізувати деякі можливості ПЗ швидко за рахунок створення проміжної версії продукту; система розбивається на окремі складові частини, які можна реалізовувати як деякі самостійні проміжні або готові продукти; можливе збільшення фінансування на розроблення окремих частин ПЗ. *Фактори ризику*, які можуть мати місце при застосуванні інкрементної моделі ЖЦ ПЗ: 1) складання вимог з врахуванням можливості їх зміни при реалізації; 2) необхідність реалізації всіх можливостей ПЗ через високу швидкість зміни технологій та вимог до ПЗ; 3) затягування термінів здачі ПЗ в експлуатацію через обмеження в ресурсному забезпеченні.

Спіральна модель – це основа циклічного процесу розроблення ПЗ. Процес починається з початкового планування і закінчується впровадженням (готового ПЗ) з циклічними взаємодіями між цими етапами. Внесення змін у спіральній моделі орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації (опис вимог проекту, коментарі різного виду і т.і.) не відповідають дійсному стану розроблення. Відмінність спіральної моделі від каскадної полягає у можливості забезпечувати багаторазове повернення до процесу формулювання вимог і до повторного розроблення будь-якого процесу виконання робіт. Серед *переваг* спіральної моделі слід відмітити спрощення внесення змін в проект при зміні вимог замовника, оскільки елементи ПЗ інтегруються поступово. *Недоліком* спіральної моделі є важкість визначення моменту переходу розроблення на наступний етап.

У випадку *еволюційної моделі* ЖЦ програмне забезпечення розробляється у вигляді послідовності блоків структур (конструкцій). На відміну від інкрементної моделі ЖЦ, вимоги встановлюються частково і уточнюються в кожному наступному проміжному блоці структури ПЗ. Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб замовника проекту та аналізу можливості застосування цієї моделі для реалізації. Модель *застосовується* для розроблення нескладних і некритичних систем, для яких головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені одразу і повністю. *Переваги* застосування даної моделі ЖЦ наступні: швидка реалізація деяких функціональних можливостей ПЗ та перевірка їх роботоздатності; використання проміжного продукту в наступному прототипі; виділення окремих функціональних частин для реалізації їх у вигляді прототипу; зворотній зв'язок із замовником для уточнення функціональних вимог; спрощення внесення змін в зв'язку з заміною окремої функції; дозволяє знизити ступінь невизначеності із завершенням кожної ітерації циклу; тестування продукту можна починати вже на ранніх стадіях життєвого циклу. При використанні еволюційної моделі слід враховувати *фактори ризику*: 1) громіздкість ПЗ через одночасну реалізацію всіх його функцій; 2) зайнятість і так обмежених людських ресурсів протягом тривалого часу; 3) можливість збільшення фінансування ПЗ.

Отже, проведений аналіз моделей життєвого циклу ПЗ дав можливість зробити висновок, що глобально всі моделі ЖЦ ПЗ можна розділити на 2 великі групи – каскадне розроблення та ітеративне розроблення. Під час розроблення завжди виявляються нові вимоги або змінюються вже виявлені,

з'являються нові обмеження, врахувати їх повною мірою вдається саме при ітеративному розробленні, тому саме ітеративні методології є найбільш поширеними через свої очевидні переваги.

Сьогодні найбільш відомими та використовуваними є наступні методології: AGILE (SCRUM, XP, DYNAMIC SYSTEM DEVELOPMENT METHOD (DSDM), KANBAN), MICROSOFT SOLUTIONS FRAMEWORK (MSF), RATIONAL UNIFIED PROCESS (RUP).

Основні засади методології розроблення *RUP* збігаються з основними засадами технології RUP, в яку вона входить: 1) рання ідентифікація та неперервне усунення основних ризиків; 2) концентрація на виконанні вимог замовника; 3) очікування змін у вимогах, проектних рішеннях та реалізації в процесі розроблення; 4) компонентна архітектура, реалізована та тестована на ранніх етапах проекту; 5) постійне забезпечення якості на всіх етапах розроблення проекту; 6) робота над проектом в команді, ключова роль в якій належить програмним архітекторам [16]. *Перевагами* RUP є: 1) максимум уваги усуненню ризиків; 2) забезпечення надійного контролю процесу розроблення; 3) орієнтація на максимально точне виконання вимог замовника; 4) зміна ступеня формалізації в залежності від потреб проекту. *Використовується* для масштабних та тривалих проектів.

Методологія *MSF* розрахована на створення готового продукту, який відповідає бізнес-інтересам замовника, чіткий розподіл відповідальності між учасниками проекту, керування ризиками та точну розстановку пріоритетів. Базові концепції та принципи: 1) єдине бачення проекту; 2) керування компромісами; 3) гнучкість; 4) концентрація на бізнес-пріоритетах; 5) заохочення вільного спілкування всередині проекту; 6) створення базової версії. *Переваги* MSF: гнучкість, масштабованість та відсутність жорстких інструкцій, завдяки чому методологія здатна задовольнити потреби організації будь-якого розміру. Тому методологія MSF є *оптимальною для* великих проектів, які вимагають дотримання балансу між ресурсами, часом розроблення та можливостями.

AGILE – це об'єднана назва гнучких методологій розроблення ПЗ. Гнучке розроблення – найкращий засіб для підвищення продуктивності розробників програмного забезпечення [16]. Мета всіх AGILE-методологій полягає у мінімізації ризиків шляхом розроблення серії коротких циклів, названих ітераціями. Кожна ітерація виглядає як програмний проект в мініатюрі, включає всі задачі, необхідні для видачі міні-приросту по функційності: планування, аналіз вимог, проектування, кодування, тестування та документування. Хоча окрема ітерація недостатня для випуску нової версії продукту, передбачається, що гнучкий програмний проект готовий до випуску в кінці кожної ітерації. По завершенню кожної ітерації команда виконує переоцінку пріоритетів розроблення. AGILE-методології орієнтовані на безпосереднє спілкування проектної команди та замовників [17].

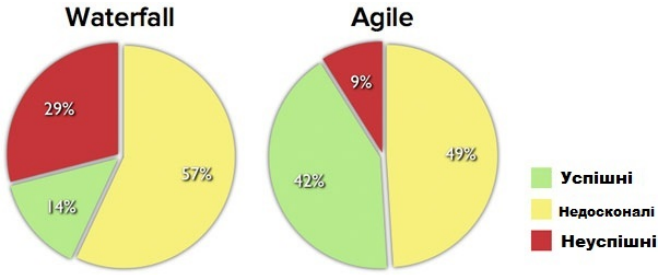
Відомою та використовуваною AGILE-методологією є методологія *SCRUM* [18], основною *перевагою* якої є її простота. SCRUM робить акцент на командній роботі, гнучкості, оперативності та контрольованості процесу розроблення. Призначена для невеликих команд (до 10 чоловік). Методологія *ідеальна для* невеликих та середніх проектів, особливо, якщо під час процесу розроблення очікується внесення численних змін.

Методологія *XP* (екстремальне програмування) – це також гнучка AGILE-методологія, розрахована на взаємодію з представниками замовника та швидке реагування на зміни у вимогах до продукту. Вона *розрахована*, в основному, *на* невеликі проекти, в яких не виникає необхідності у створенні детальної документації та регламентації всіх кроків розроблення [19].

AGILE-методологія DSDM [6] з'явилась в результаті консорціуму 17 англійських компаній. Згідно цієї методології, робота над програмним проектом розпочинається з вивчення здійсненності програми (а також питання, чи підходить DSDM для цієї програми) та галузі її застосування. Далі процес поділяється на три взаємозв'язаних цикли: цикл функціональної моделі (створення аналітичної документації та прототипів), цикл проектування та конструювання (приведення системи в робочий стан), цикл реалізації (забезпечення розгортання програмної системи). Базові принципи: активна взаємодія з користувачами, часті випуски версій, самостійність розробників у прийнятті рішень, тестування протягом всього циклу робіт. *Особливий акцент* робиться на високій якості роботи та адаптованості до змін у вимогах.

Наступною AGILE-методологією є методологія *KANBAN* [6], яка орієнтована на задачі. Основними правилами цієї методології є: 1) візуалізація розроблення; 2) обмеження робіт, які виконуються одночасно, на кожному етапі розроблення; 3) вимірювання часу циклу та оптимізація процесу. *Переваги* KANBAN: 1) зменшення кількості паралельно виконуваних задач, що значно зменшує час виконання кожної окремої задачі; 2) швидке виявлення проблемних задач; 3) обчислення часу на виконання усередненої задачі.

На рис. 2 зображено діаграму порівняння проектів, які були виконані з використанням AGILE-методології та каскадної методології [20]. Очевидно, що AGILE-проекти втричі успішніші за каскадні проекти. Аналіз відомих методологій розроблення ПЗ показав, що не існує якоїсь єдино вірної методології, оптимальної для будь-якого проекту. Отже, аналітичні процеси повинні адаптуватись під кожен проект. У рідкісних випадках можна виробити єдиний підхід до всіх проектів компанії. Саме з вибору методології та побудови аналітичних процесів необхідно починати планування, причому не тільки планування вимог, але й планування всього програмного проекту.



Джерело: The CHAOS Manifesto, The Standish Group, 2012.

Рис.2. Порівняння успішності програмних проектів, реалізованих за каскадними та AGILE- методологіями

Аналіз середовищ розроблення програмного забезпечення

Після етапу проектування перед будь-якою компанією, що займається розробленням програмного забезпечення, постає задача вибору середовища розроблення. Проаналізуємо найпопулярніші інтегрованих середовища розроблення програмного забезпечення:

1) *C++ Builder* – не є надто потужним середовищем. Середовище C++ Builder написане мовою Delphi. Використовує концепцію швидкого розроблення додатків (RAD), яка дозволяє суттєво зменшити обсяг "ручного" кодування, створювати інтерфейс користувача, використовуючи техніку drag-and-drop, підвищує продуктивність праці програмістів, скорочує тривалість циклу розроблення [21]. Має доволі зручний та зрозумілий інтерфейс з невеликою кількістю компонентів. У своєму арсеналі має лише одну мову програмування – Visual C. Дозволяє створювати стабільні функціональні програми як консольні додатки, так і додатки з графічним інтерфейсом користувача. Приховує деякі низькорівневі деталі програмування. *Недолік* полягає у тому, що прикладні програми працюватимуть тільки під ОС Windows. Має великий набір компонентів. Ціна середня.

2) *Code::Blocks* – вільно поширюване, кросплатформенне середовище розроблення. Code::Blocks написане мовою C++ і використовує бібліотеку wxWidgets [22]. Маючи відкриту архітектуру, може масштабуватися за рахунок додаткових модулів. Підтримує мови програмування C, C+ +, D (з обмеженнями). Code::Blocks розробляється для Windows, Linux і Mac OS X. Підтримує декілька компіляторів: GCC (MingW/GNU GCC), MSVC++, clang, Digital Mars, Borland C++ 5.5, Open Watcom. *Використовується* для багатоцільових програмних проектів.

3) *Dev-C++* – повнофункціональне інтегроване середовище розроблення мовами програмування C/C++ [23]. Використовує компілятор на основі GCC. Забезпечує комплексне відлагодження, підтримує профілювання, дозволяє швидке створення статичних та динамічних бібліотек, підтримує шаблони для створення власних типів проектів. Саме середовище Dev-C++ написане на Delphi. Розповсюджується згідно з ліцензією GPL. Проект підтримується SourceForge.

4) *Eclipse CDT* – інтегроване, повнофункціональне середовище розроблення для C/C++ [24]. Eclipse в цілому складається з базового робочого місця і розширюваної плагін-системи для налагодження середовища. Містить інструменти: браузер, браузер макровизначення, редактор коду з підсвіткою синтаксису, інструмент навігації за допомогою гіперпосилань, інструмент для рефакторингу та генерації сирцевого коду, візуальні засоби відлагодження. Написане в основному мовою Java, Eclipse *може використовуватися* для розроблення Java-додатків, а також за допомогою різних плагінів – для розроблення додатків іншими мовами програмування: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Perl, PHP, Prolog, Python, R, Ruby, Clojure, Groovy і Erlang. Середовище розроблення включає інструменти розроблення Eclipse Java (JDT) для Java і Scala, Eclipse CDT для C/C++ і Eclipse PDT для PHP.

5) *MonoDevelop* – вільне мультиплатформенне середовище розроблення, призначене для створення додатків мовамиC #, C, C++, Java, Visual Basic.NET, Vala, CIL, Nemerle, Boo [25]. На даний момент середовище є частиною проекту Mono. Середовище *дає можливість* розробникам швидко створювати додатки для кишенькових пристроїв та web-додатки ASP.NET для Linux, Windows, MacOSX. Полегшує розробникам перенесення .NET-додатків, створених у Visual Studio, на Linux та MacOSX. Дозволяє створення web-проектів з повною підтримкою автозавершення коду та тестуванням на XSP.

6) *Qt Creator* – кросплатформенне вільне середовище розроблення мовами C, C++ і QML [26]. Розроблене компанією Trolltech (Digia) для роботи з фреймворком Qt. Включає в себе налагоджувач з графічним інтерфейсом та візуальні засоби розроблення інтерфейсу з використанням як QtWidgets, так і QML. Підтримувані компілятори: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

7) *Microsoft Visual Studio* – дуже потужне середовище програмування від Microsoft. Дає змогу програмувати мовами: Visual C#, Visual Basic, Visual C++, Visual F#, JavaScript [27]. Програми, створені в цьому середовищі, орієнтовані не лише на операційні системи Windiws, але й на UNIX-системи. Має потужний інструментарій для створення програмного забезпечення. Існують декілька версій середовища, що призначені для задоволення певних потреб розробників і мають різний набір компонентів. Дане інтегроване середовище *дозволяє* розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі за підтримки технології Windows Forms, а також web-сторінки, web-додатки та web-служби. Є дуже зручні компоненти для проведення тестування. Ціна достатньо висока.

8) *NetBeans* – вільно поширюване середовище розроблення для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy [28]. Середовище може бути встановлене і з підтримкою окремих мов, і у повній конфігурації. Середовище розроблення NetBeans *підтримує розроблення* для платформ J2SE і J2EE.

Отже, очевидно, що існує значна кількість середовищ розроблення ПЗ. Безліч інформації про середовища розроблення програміст може знайти в мережі Інтернет, але ця інформація не є структурованою. Надзвичайно багато часу програміст витрачає на пошук оптимального рішення щодо вибору середовища розроблення, читаючи форуми та шукаючи порад більш досвідчених користувачів, але такі поради не можуть бути достатньо об'єктивними, адже, як відомо, вибір середовища програмування здебільшого є справою звички. Програмісти зазвичай використовують для розроблення ПЗ ті середовища, з якими вони вже давно працюють та до особливостей (переваг та недоліків) яких звикли. Освоєння нового середовища програмування вимагає значних витрат часу та коштів, причому ціна на сучасні засоби розроблення програмного забезпечення (ПЗ) може сягати 1000$ за одну ліцензію. Зрозуміло, що, якщо необхідно забезпечити середовищем програмування штат програмістів, припустимо, невеликої компанії з 10-30 працівників, то витрати стають досить значними. Важливим є питання, чи є реальною така висока ціна для існуючих середовищ програмування. Але значно важливішим є питання, чи є реальною висока ціна для середовища програмування в контексті поставленого завдання. Можливо, для виконання поставлених перед програмістом задач буде достатньо більш дешевого або взагалі безкоштовного (вільно поширюваного) середовища.

Аналіз процесу вибору технології проектування, методології та середовища розроблення ПЗ

З аналізу відомих технологій проектування, методологій та середовищ розроблення очевидним є існування значної кількості технологій/методологій/середовищ за відсутності універсальної технології/методології/ середовища, яка підходила б для будь-якого програмного проекту, а також за відсутності чітких стандартизованих критеріїв вибору технології/методології/середовища.

Наразі *оцінювання і вибір технології проектування* базується на експертних оцінках, які часто є суб'єктивними. Задача комплексного оцінювання, обґрунтованого вибору та ефективного застосування технології проектування програмного забезпечення у великомасштабних проектах залишається *актуальною*. Неможливо досягти задовільних результатів від застосування навіть найдосконаліших технологій, якщо вони застосовуються безсистемно, розробники не мають необхідної кваліфікації для роботи із ними, і сам проект виконується і керується хаотично. Систематичний, обґрунтований підхід до вибору та застосуванню ТППЗ може скоротити час і підвищити якість розроблення ПЗ, забезпечити високий ступінь його незалежності від конкретних розробників, а також знизити витрати на розроблення та супровід ПЗ.

Вибір методології розроблення взагалі вимагає врахування багатьох різноманітних умов та факторів, які стосуються не лише процесів розроблення ПЗ, але й багатьох інших сфер. В кожному конкретному випадку правильний вибір методології розроблення залежить від таких факторів [19]: 1) масштабу проекту; 2) критичності проекту; 3) кількості та розподілу повноважень учасників проекту; 4) ступеня новизни проекту; 5) очікуваної тривалості проекту; 6) вимог замовника. При виборі методології розроблення необхідно враховувати велику кількість різних умов, таких як [9]: 1) стандарти і політики, прийняті в компанії-виконавця; 2) знання і досвід проектної команди; 3) уподобання або вимоги замовника; 4) розмір проектної команди; 5) специфіка та складність проекту; 6) стабільність та зрілість процесів у компанії; 7) особисті якості співробітників; 8) вимоги до проекту.

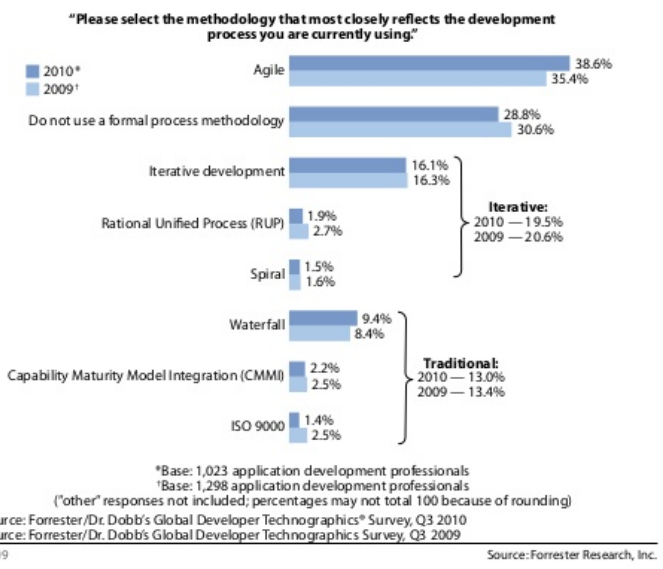


Рис.3. Статистика використання методологій розроблення ПЗ

Як правило, вибір між традиційними та гнучкими (AGILE-) методологіями значною мірою визначається бажаним ступенем формалізації процесу розроблення. Для великих, розрахованих на тривалий термін, проектів, як правило, використовуються методології з більшим ступенем формалізації – традиційні та ітеративні методології, в той час як для менш масштабних проектів, розрахованих на оперативну реалізацію, підходять менш формалізовані гнучкі методології розроблення [19].

Статистика використання методологій розроблення ПЗ наведена на рис.3 [29].

З рис.3 очевидно, що у 2010 році 30,6% софтверних компаній взагалі не використовують методологій розроблення ПЗ; 38,6% софтверних компаній використовують AGILE-методології; 19,5% компаній використовують ітеративні методології розроблення; 13% софтверних компаній досі

використовують традиційні методології розроблення (в тому числі 9,4% – каскадну методологію, яка є втричі менш успішною, ніж гнучкі методології [20]). Крім цього, аналіз рис.3 дає можливість зробити наступні висновки: 1) зростає кількість компаній, які використовують методологію розроблення ПЗ; 2) зростає кількість компаній, які використовують AGILE-методології; 3) зменшується кількість компаній, які використовують ітеративні та традиційні методології, але в той же час серед традиційних методологій зростає використовуваність саме каскадної методології.

Наразі відомі наступні використання існуючих методологій: 1) AGILE-методології використовується компанією Google; 2) MSF-методологія – компанією Microsoft; 3) традиційні методології (зокрема, CMM) – компаніями Boeing, Northrop-Grumman, Lockheed-Martin. Отже, компанії абсолютно суб'єктивно обирають методологію розроблення – немає ані критеріїв вибору методологій, ані систем підтримки прийняття рішень щодо вибору методологій, ані рекомендацій у стандартах. Вибір методології розроблення, як і вибір технології проектування відбувається на розсуд розробників, хоча задача оцінювання та вибору методології розроблення є дуже *актуальною*, оскільки саме вірно обрана методологія може підвищити успішність програмного проекту.

Вибір *середовища розроблення ПЗ* зводиться до багатокритеріальної задачі і далеко не очевидний. Багатокритеріальність вибору середовища програмування полягає у тому, що кожне існуюче середовище програмування слід оцінювати не за одним критерієм, а за сукупністю багатьох показників (критеріїв), що розглядаються одночасно. Наразі вибір середовища розроблення теж відбувається суб'єктивно – на розсуд програміста. При виборі середовища програмування розробники зазвичай виділяють наступні проблеми [30]: 1) відсутність єдиної бази знань про середовища розроблення ПЗ; джерела знань про середовища є розосередженими; основна інформація здебільшого знаходиться на сайтах розробників, проте користувач часто змушений порівнювати декілька середовищ і змушений користуватися великою кількістю джерел (сайти розробників, форуми, статті); наявність лише одного джерела значно спростила б пошук; 2) відсутність єдиного стандарту оцінки середовищ розроблення ПЗ; 3) відсутність єдиних критеріїв, за якими оцінюються середовища: кожен тестувальник чи розробник висвітлює різні характеристики, що ускладнює порівняння декількох середовищ одночасно. Отже, наразі *актуальною* задачею є оцінювання та підтримка вибору середовища розроблення ПЗ, яка полягає у виборі найкращого рішення з безлічі можливих та упорядкуванні можливих рішень за переважністю.

Висновки

У статті виконано аналіз сучасних технологій проектування, методологій та середовищ розроблення програмного забезпечення, в результаті якого було визначено основні переваги та недоліки технологій/методологій/середовищ, а також для яких програмних проектів та якими софтверними компаніями вони використовуються. Аналіз показав, що наразі розроблено чимало технологій/методологій/середовищ, які враховують життєвий цикл ПЗ, але при цьому частка проблемних проектів залишається сталою, а частка неуспішних програмних проектів зменшується незначно. При всій множині існуючих технологій/методологій/середовищ процес розроблення ПЗ залишається недетермінованим, і результат його завжди невідомий. Такий стан справ галузі програмної інженерії можна пояснити двома причинами, пов'язаними із технологіями/методологіями/середовищами: або їх недосконалістю, або невірністю їх вибору.

Автори провели аналіз процесу вибору технології проектування, методології та середовища розроблення ПЗ, показали, що програмний проект володіє рядом характеристик, які різною мірою впливають на вибір технології/методології/середовища. Виконаний аналіз довів, що на даний момент в галузі панує повний суб'єктивізм вибору, адже немає жодних чітко визначених критеріїв вибору, жодних рекомендацій у стандартах, жодних систем підтримки прийняття рішень щодо вибору технології/методології/середовища. Весь процес оцінювання та вибору технологій/методологій/середовищ покладено на розробників, які не завжди можуть прийняти оптимальне рішення, оскільки не завжди розуміють складність процесу розроблення ПЗ та ступінь впливу зміни вимог до ПЗ. Оскільки програмна інженерія, як і будь-яка інженерна дисципліна, вимагає чітких математичних моделей, критеріїв та методів, підтримка процесу оцінювання та вибору технології проектування, методології та середовища розроблення є *актуальною* науково-практичною задачею і потребує розв'язання.

Тоді *перспективними задачами наступних досліджень* в цьому напрямку є: 1) побудова математичних та інтелектуальних моделей технології проектування, методології та середовища розроблення ПЗ, які враховують важливі складові технології/методології/середовища, а також математичних моделей оцінювання та вибору технологій/методологій/середовищ з врахуванням життєвого циклу ПЗ, типів програмних проектів та всіх характеристик програмних проектів, які впливають на вибір; 2) розроблення математичних критеріїв та правил для оцінювання та вибору технології проектування, методології та середовища розроблення ПЗ; 3) розроблення математичних та інтелектуальних методів оцінювання та вибору технології проектування, методології та середовища розроблення ПЗ на основі розроблених математичних моделей; 4) побудова комплексних коефіцієнтів (оцінок) для вибору технології проектування, методології та середовища розроблення ПЗ, які враховуватимуть як важливі складові технології/методології/середовища, так і тип програмного проекту, життєвий цикл програмного проекту, характеристики програмного проекту, критичність проекту, кількість учасників проекту, ступінь новизни проекту, необхідність формалізації даних проекту; 5) проектування та реалізація інтелектуальної системи

підтримки прийняття рішень щодо вибору технології проектування, методології та середовища розроблення ПЗ.

Запропонований підхід дасть змогу приймати мотивовані та обґрунтовані рішення щодо переважності та вибору технології проектування, методології та середовища розроблення програмного забезпечення. На вирішення поставлених задач будуть спрямовані подальші зусилля авторів.

Література

1. Latest study shows rise in project failures [Електронний ресурс]. – Режим доступу : http://kinzz.com/resources/articles/91-project-failures-rise-study-shows
2. CHAOS Manifesto: Think Big, Act Small, 2013 [Електронний ресурс]. – Режим доступу : http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf
3. Чернев Д.А. Технология разработки программного обеспечения / Чернев Д.А. – Ташкент, 2004 – 224 с.
4. Greenfield J., Short K. et al. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools – John Wiley & Sons, 2004 – 666 p. [Електронний ресурс]. – Режим доступу : http://iclass.iuea.ac.ug/intranet/Ebooks/INFORMATION%20TECHNOLOGY/SOFTWARE%20ENGINEERING/%5BWiley%5D%20Software%20Factories%20-%20Assembling%20Applications%20with%20Patterns,%20Models,.pdf
5. Хаф Л. Методологии разработки програмного обеспечения [Електронний ресурс]. – Режим доступу : http://compress.ru/article.aspx?id=11321
6. Методологии разработки программного обеспечения [Електронний ресурс]. – Режим доступу : http://habrahabr.ru/sandbox/43802/
7. Среда разработки программного обеспечения [Електронний ресурс]. – Режим доступу : http://dic.academic.ru/dic.nsf/ruwiki/6703
8. RUP – рациональный унифицированный процесс разработки программного обеспечения [Електронний ресурс]. – Режим доступу : http://www.structuralist.narod.ru/dictionary/rup.htm
9. Средства разработки [Електронний ресурс]. – Режим доступу : http://www.avtorsoft.com/catalog/sredstva-razrabotki
10. Фаулер М. Рефакторинг: улучшение существующего кода / Фаулер М. – СПб : Символ-Плюс, 2012 – 489 с.
11. Oracle Developer Suite [Електронний ресурс]. – Режим доступу : http://www.interface.ru/oracle/OracleDS10g.htm
12. ISO 9000:2005 Quality management systems – Fundamentals and vocabulary [Електронний ресурс]. – Режим доступу : https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-3:v1:en
13. Маклаков С.В. Создание информационных систем с AllFusion Modelling Suite / Маклаков С.В. – М. : Изд-во Диалог-МИФИ, 2007 – 400 с.
14. Survey of Advanced Technology (SAT) [Електронний ресурс]. – Режим доступу : http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&SDDS=4223&InstaId=29938&SurvId=44344
15. Why the Waterfall Model Doesn't Work [Електронний ресурс]. – Режим доступу : http://www.infoq.com/resource/articles/scaling-software-agility/en/resources/ch02.pdf
16. Мартин Р. К. Быстрая разработка программ: принципы, примеры, практика / Р. К. Мартин, Д. В. Ньюкирк, Р. С. Косс. – М. : Вильямс, 2004. – 752 с.
17. Методологии разработки программного обеспечения (модели процесса) [Електронний ресурс]. – Режим доступу : http://www.dpgrup.ru/methodologies.htm
18. Обзор методологии разработки программного обеспечения SCRUM [Електронний ресурс]. – Режим доступу : http://www.dpgrup.ru/methodology-scrum.htm
19. Разработка программного обеспечения: Методологии разработки программного обеспечения [Електронний ресурс]. – Режим доступу : http://bms-soft.com.ua/ru/services/razrabotka-programmnogo-obespecheniya/metodologii-razrabotki-po
20. Agile Succeeds Three Times More Often Than Waterfall [Електронний ресурс]. – Режим доступу : http://www.mountaingoatsoftware.com/blog/agile-succeeds-three-times-more-often-than-waterfall
21. Borland C++ Builder 6. Руководство разработчика / [Д. Холингворт, Б. Сворт, М. Кэшмэн, П. Густавсон]. – М. : «Вильямс», 2004. – 976 с.
22. Code::Blocks features [Електронний ресурс]. – Режим доступу : http://www.codeblocks.org/features
23. Bloodshed Software - Dev-C++ features [Електронний ресурс]. – Режим доступу : http://www.bloodshed.net/devcpp.html
24. Eclipse CDT (C/C++ Development Tooling) [Електронний ресурс]. – Режим доступу : http://www.eclipse.org/cdt/
25. MonoDevelop [Електронний ресурс]. – Режим доступу : http://monodevelop.com/
26. Шлее М. Qt 4.5. Профессиональное программирование на C++ / Шлее М. – СПб : БХВ-Петербург, 2010. – 256 с.
27. Visual Studio 2010 для профессионалов / [Н. Рендольф, Д. Гарднер, М. Минутилло, К.

Андерсон]. – М. : «Диалектика», 2011. —1184 с.

28. Монахов В.В. Язык программирования Java и среда NetBeans / Монахов В.В. – СПб : БХВ-Петербург, 2011. – 704 с.

29. West D. Water-Scrum-Fall is the reality of AGILE for most organizations today [Електронний ресурс]. – Режим доступу : http://www.slideshare.net/harsoft/water-scrumfall-isrealityofagileformost

30. Говорущенко Т.О. Підтримка вибору середовища програмування для системного програмного забезпечення / Т.О. Говорущенко, Р.А. Малярчук, В.В. Лаврінчук. // Збірник наукових праць III Всеукр. НПК молодих учених та студ. «Інтелектуальні технології в системному програмуванні». – Хмельницький : Гонта А.С., 2014. – С. 372–381.

References

1. Latest study shows rise in project failures // [Electronic resource] - Access mode: http://kinzz.com/resources/articles/91-project-failures-rise-study-shows

2. CHAOS Manifesto: Think Big, Act Small, 2013 // [Electronic resource] - Access mode: http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf

3. Chernev D.A. Technologiya razrabotki programmnogo obespecheniya – Tashkent, 2004 – 224 s.

4. J.Greenfield, K.Short et al. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools – John Wiley & Sons, 2004 – 666 p. // [Electronic resource] – Access mode: http://iclass.iuea.ac.ug/intranet/E-books/INFORMATION%20TECHNOLOGY/SOFTWARE%20ENGINEERING/%5BWiley%5D%20Software%20Factories%20-%20Assembling%20Applications%20with%20Patterns,%20Models,.pdf

5. L.Khaf. Metodologii razrabotki programmnogo obespecheniya // [Electronic resource] – Access mode: http://compress.ru/article.aspx?id=11321

6. Metodologii razrabotki programmnogo obespecheniya // [Electronic resource] – Access mode: http://habrahabr.ru/sandbox/43802/

7. Sreda razrabotki programmnogo obespecheniya // [Electronic resource] – Access mode: http://dic.academic.ru/dic.nsf/ruwiki/6703

8. RUP – ratsionalniy unificirovanniy process razrabotki programmnogo obespecheniya // [Electronic resource] – Access mode: http://www.structuralist.narod.ru/dictionary/rup.htm

9. Sredstva razrabotki // [Electronic resource] – Access mode: http://www.avtorsoft.com/catalog/sredstva-razrabotki

10. M.Fauler. Refaktoring: uluchshenie sushchestvuyushchego koda – SPb: Simvol-Plus, 2012 – 489 p.

11. Oracle Developer Suite // [Electronic resource] – Access mode: http://www.interface.ru/oracle/OracleDS10g.htm

12. ISO 9000:2005 Quality management systems – Fundamentals and vocabulary // [Electronic resource] – Access mode: https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-3:v1:en

13. Maklakov S.V. Sozdanie informacionnih sistem s AllFusion Modelling Suite – M.: Izdatelstvo Dialog-MIFI, 2007 – 400 s.

14. Survey of Advanced Technology (SAT) // [Electronic resource] – Access mode: http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&SDDS=4223&InstaId=29938&SurvId=44344

15. Why the Waterfall Model Doesn't Work // [Electronic resource] – Access mode: http://www.infoq.com/resource/articles/scaling-software-agility/en/resources/ch02.pdf

16. R.K.Martin, J.V.Newkirk, R.S.Coss. Bistraya razrabotka program: principi, primeri, praktika – M.:Williams, 2004. – 752 s.

17. Metodologii razrabotki programmnogo obespecheniya (modeli processa) // [Electronic resource] – Access mode: http://www.dpgrup.ru/methodologies.htm

18. Obzor metodologii razrabotki programmnogo obespecheniya SCRUM // [Electronic resource] – Access mode: http://www.dpgrup.ru/methodology-scrum.htm

19. Razrabotka programmnogo obespecheniya: Metodologii razrabotki programmnogo obespecheniya // [Electronic resource] – Access mode: http://bms-soft.com.ua/ru/services/razrabotka-programmnogo-obespecheniya/metodologii-razrabotki-po

20. Agile Succeeds Three Times More Often Than Waterfall // [Electronic resource] – Access mode: http://www.mountaingoatsoftware.com/blog/agile-succeeds-three-times-more-often-than-waterfall

21. D.Khallingvort, B.Swort, M.Kashman, P.Gustavson. Borland C++ Builder 6. Rukovodstvo razrabotchika – M.: Williams, 2004. – 976 s.

22. Code::Blocks features // [Electronic resource] – Access mode: http://www.codeblocks.org/features

23. Bloodshed Software - Dev-C++ features // [Electronic resource] – Access mode: http://www.bloodshed.net/devcpp.html

24. Eclipse CDT (C/C++ Development Tooling) // [Electronic resource] – Access mode: http://www.eclipse.org/cdt/

25. MonoDevelop // [Electronic resource] – Access mode: http://monodevelop.com/

26. Shlee M. Qt 4.5. Professionalnoe programmirovanie na C++ - SPb.: BHV-Peterburg, 2010 – 256 s.

27. N.Rendolf, D.Gardner, M.Minutillo, K.Anderson. Visual Studio 2010 dlya professionalov – M.: Dialektika, 2011 – 1184 s.

28. Monahov V.V. Yazik programmirovaniya Java I sreda NetBeans – SPb.: BHV-Peterburg, 2011 – 704 s.

29. D.West. Water-Scrum-Fall is the reality of AGILE for most organizations today // [Electronic resource] – Access mode: http://www.slideshare.net/harsoft/water-scrumfall-isrealityofagileformost

30. T.O.Hovorushchenko, R.A.Malyarchuk, V.V.Lavrinchuk. Pidtrymka viboru seredovishcha programuvannya dlya sistemnogo programnogo zabezpechennya // ITSP-2014. III Vseukrainska NPK molodih uchenih ta studentiv. Zbirnik naukovih prac – Khmelnitskiy: Gonta A.S., 2014 – s.372-381

Рецензія/Peer review : 18.10.2014 р. Надрукована/Printed :29.11.2014 р.
Рецензент: д.т.н., професор Ю. І. Шалапко