РОЗДІЛ «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ»

УДК 519.218

ДРАНИШНИКОВ Л.В., д.т.н., професор

Дніпровський державний технічний університет, м. Кам'янське

ІНФОРМАЦІЙНИЙ АНАЛІЗ АВАРІЙНОГО РИЗИКУ ОБ'ЄКТІВ ПІДВИЩЕНОЇ НЕБЕЗПЕКИ

Вступ. Сучасний рівень організації та управління виробництвом висуває вимоги розробки нових підходів, що базуються на використанні нових інформаційних технологій та інтелектуальних засобів підтримки і прийняття рішень з оперативного управління аварійними ситуаціями, пов'язаними з функціонуванням об'єктів підвищеної небезпеки (ОПН), а також з прогнозування та оцінки тяжкості наслідків аварій. Створення фундаментальних наукових, правових та економічних основ забезпечення безпеки є однією з цілей державної науково-технічної політики і державної науково-технічної програми з безпеки природно-технічної сфери, для підвищення безпеки у промисловому, енергетичному, транспортному, будівельному, нафтогазовому, гірничодобувному й оборонному комплексах. Подальша розробка і реалізація програм науково-технічного розвитку сучасної цивілізації неможлива без системного наукового підходу до вирішення проблем забезпечення безпечного функціонування структурно-складних систем, якими є об'єкти підвищеної небезпеки – об'єкти, на яких використовують, виробляють, переробляють, зберігають або транспортують пожежо- та вибухонебезпечні та (або) небезпечні хімічні речовини, що створюють реальну загрозу виникнення аварії, та без розробки математичного апарату для кількісної оцінки ризику.

З 2001 року діє Закон України про об'єкти підвищеної небезпеки (Відомості Верховної Ради, 2001, №15, с.73), з 2002 року – постанова №956 Кабінету Міністрів України від 11.07.2002 р.

У кінці минулого століття було висунуто концепцію прийнятного техногенного ризику [1]. Це означає, що забезпечення абсолютної безпеки об'єктів техносфери неможливе і треба домагатися їхньої відносної безпеки, доводячи аварійний ризик, пов'язаний з ними, до прийнятного, допустимого рівня. Рівень ризику, прийнятний для тієї чи іншої діяльності, визначається, виходячи з економічних і соціальних аспектів відповідно до принципів управління ризиком, які мають бути сформульовані для цієї мети.

Постановка задачі. Система забезпечення безпеки ОПН повинна бути комплексною і містити у своєму складі підсистеми науково-технічного, інформаційного, матеріально-технічного, кадрового та організаційного забезпечення. Найважливішою частиною системи науково-технічного забезпечення безпеки ОПН є підсистеми аналізу аварійного ризику і реагування на надзвичайні ситуації. Підсистема аналізу аварійного ризику призначена концентрувати інформацію про об'єкт у цілому, про систему його безпеки, про навколишнє середовище, а також прогнозувати можливі аварії та їх наслідки. Головна її функція – розробка рекомендацій з коригувальних впливів на об'єкт у цілому для того, щоб забезпечити зниження величини ризику та підтримання його на прийнятному рівні. Аналіз аварійного ризику являє собою складну комплексну процедуру, що включає цілий ряд етапів.

Результати роботи. Блок-схему аналізу аварійного ризику, що включає всі основні процедури аналізу ризику, наведено на рис.1. Вся процедура аналізу аварійного ризику може бути розбита на ряд порівняно самостійних, але взаємопов'язаних етапів: перший етап призначений для виявлення основних небезпек, які властиві даному об'єкту; на другому етапі виконується аналіз і кількісна оцінка можливих наслідків від прогнозованих аварій; третій етап являє собою частотний аналіз аварійних подій; він

полягає у визначенні інтенсивностей (частот) та ймовірностей аварійних подій; на четвертому етапі дані про очікуваний збиток і втрати від окремих аварій комбінуються з даними щодо можливих інтенсивностей та ймовірностей аварійних подій і відшукується величина прогнозованого аварійного ризику.

Управління аварійним ризиком носить циклічний характер: після реалізації зазначених впливів знову реалізуються блоки – етапи 1-4 – і так до того часу, поки не буде досягнуто прийнятне значення прогнозованого ризику.

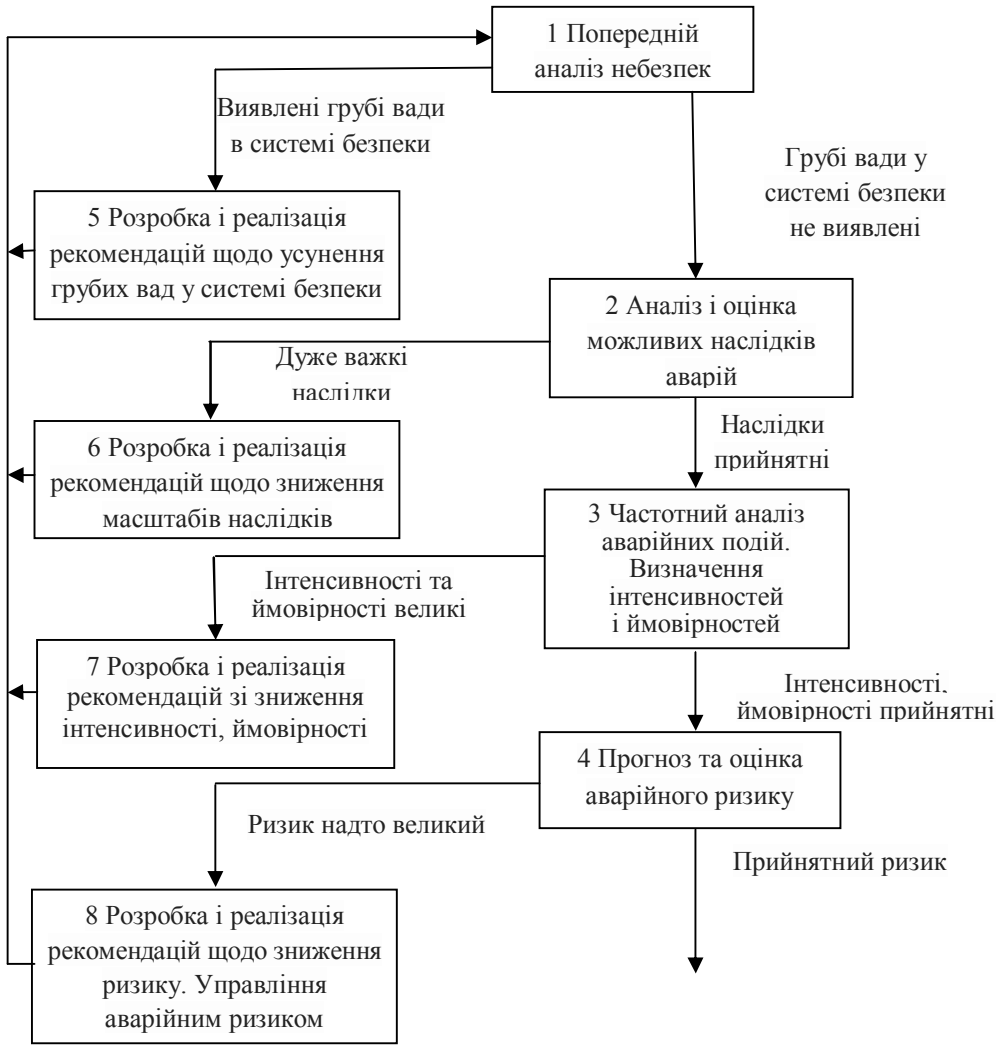


Рисунок1 – Блок-схема аналізу аварійного ризику

Попередній аналіз небезпек (ПАН). Його призначення: виявити, з яких причин можуть виникати аварії, ідентифікувати носії аварійної небезпеки; визначити, за якими сценаріями можуть розвиватися аварії; відібрати з них найбільш небезпечні. ПАН складається з низки етапів: 1) опис об'єкта; 2) опис носіїв небезпеки та їх класифікація; 3) виявлення можливих інцидентів (на цьому етапі найчастіше використовують метод дерев відмов (ДВ) у припущенні, що верхня небажана подія являє собою інцидент); 4) аналіз післяінцидентних поєднань аварійних подій, процесів і явищ, які можуть мати місце після інциденту; 5) відбір найбільш небезпечних інцидентів і формування остаточного підсумкового списку інцидентів; 6) складання сценаріїв аварій на основі підсумкового списку інцидентів; 7) розробка рекомендацій щодо зниження рівня небезпеки об'єкта.

Кількісний аналіз аварійних подій базується на використанні методів математичного моделювання [2, 4, 5]. На цьому етапі використовуються математичні моделі різних класів. На рис.2 зображено блок-схему математичного моделювання аварійних ситуацій.

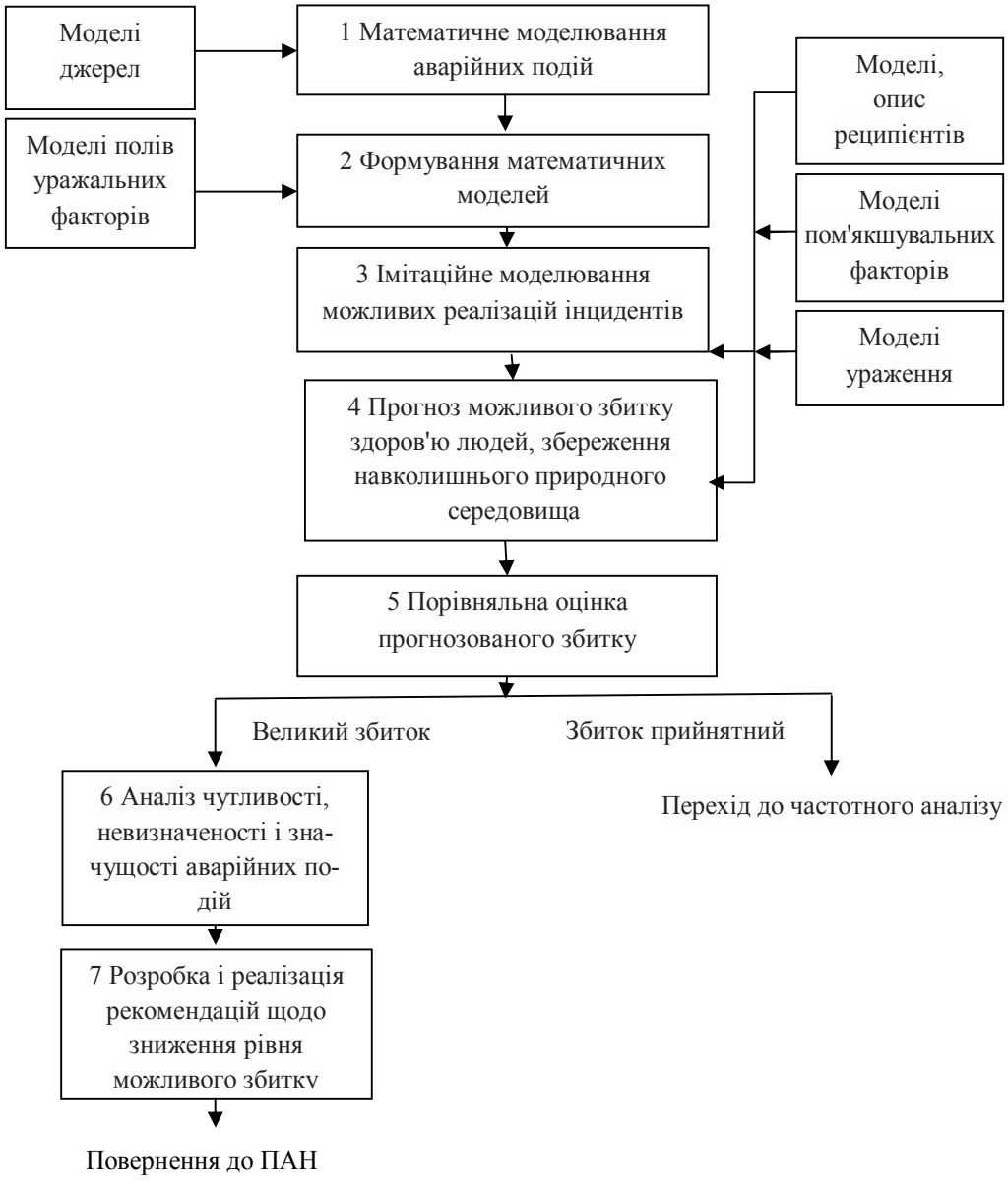


Рисунок 2 – Блок-схема математичного моделювання аварійних подій

Перший етап полягає в математичному моделюванні поєднань аварійних подій. На даному етапі на моделях досліджуються небезпечні ініціюючі події. За допомогою моделей імітуються різні комбінації аварійних подій. Тут необхідно описати множину пов’язаних одна з одною подій для кожного інциденту, прийнятого для розгляду, починаючи від подій, пов’язаних з вивільненням токсичного і/або енергетичного потенціалу, і закінчуючи ураженням людей.

При формуванні математичних моделей проявів інцидентів велике значення надається правильному вибору моделей джерел. До таких моделей відносяться, насамперед, моделі витоків речовин. Вихід з-під контролю процесу вивільнення енергії або неконтрольований викид отруйних речовин – все це може стати причиною ризику. Поширення викидів небезпечних речовин в атмосфері та їх вплив на людей і навколишнє се-

редовище є важливими факторами, що визначають тяжкість наслідків аварій на небезпечних виробничих об'єктах. До теперішнього часу склалися три основних підходи для кількісного опису процесу розсіювання викиду газоподібних речовин в атмосфері: 1) гаусові моделі розсіювання, іноді так звані дисперсійні моделі; 2) моделі розсіювання, що базуються на інтегральних законах збереження або у хмарі в цілому (залповий викид), або у поперечному перерізі хмари (тривалий викид); 3) моделі, побудовані на чисельному рішенні системи рівнянь збереження в їх оригінальному вигляді (методи прямого чисельного моделювання).

Для аналізу ризику використовують логічні і логіко-графічні моделі [3, 6], перші з яких є сукупністю логічних виразів і висловлювань, що характеризують послідовність розвитку аварійних подій, а другі дозволяють встановити причинно-наслідкові зв'язки між вихідними ініційованими подіями виникнення аварійних ситуацій та їх розвитком, що призводить до різних видів ризиків.

Імітаційне моделювання можливих реалізацій інцидентів спирається на використання моделей джерел, моделей полів уражальних факторів, моделей опису реципієнтів, моделей пом'якшувальних факторів та моделей ураження реципієнтів.

Моделі полів уражальних факторів включають моделі концентраційних полів токсичних речовин у різних середовищах, моделі температурних полів, що виникають у разі пожеж і вибухів, моделі розподілу тиску й осколків при вибухах. Для оцінки токсичних наслідків аварій будують моделі міграції токсикантів у повітряному середовищі.

Під моделями опису реципієнтів мають на увазі моделі їх розподілу за видами і факторами ураження. До них приєднуються моделі пом'якшувальних факторів, в яких відображається захищеність реципієнтів від впливу уражальних факторів. До моделей ураження відносять моделі токсичного ураження людей, моделі термічного ураження, а також моделі баричного й осколкового ураження.

Потім передбачається оцінка отриманих значень прогнозованого збитку від різних можливих аварій і порівняння їх з допустимими критичними значеннями. При перевищенні останніх виявляються найбільш значущі аварійні події, які вносять найбільший вклад у значення збитку, визнаного недопустимим.

У результаті розробляються рекомендації, націлені на зниження рівня недопустимо великих значень збитків при тих чи інших аваріях, і забезпечується їх реалізація.

Частотний аналіз аварійних подій. Частотний аналіз є одним з основних етапів аналізу аварійного ризику і необхідною умовою для його прогнозування.

Частотний аналіз спирається на математичний апарат теорії ймовірності, математичної статистики, теорії надійності та алгебри логіки. Ймовірності несприятливих подій (аварій) можуть бути визначені трьома способами.

Перший пов'язаний зі статистичною обробкою емпіричних даних і з використанням експертних оцінок (придатні для визначення ініціюючих (базових) подій, а також для визначення ймовірностей самих аварій).

Другий спосіб пов'язаний з визначенням ймовірностей за допомогою дерев відмов (ДВ) і дерев подій (ДП) (включає якісний аналіз ДВ – побудова мінімальних аварійних поєднань подій і кількісний аналіз – метод мінімальних аварійних поєднань, метод функції алгебри логіки, а також метод статистичних випробувань Монте-Карло).

У дослідженнях ризику найбільш широко використовуються дерева подій. Таке дерево являє собою ієрархічну структуру, верхній рівень якої характеризує несприятлива подія, а нижній – набори факторів умов, при яких вона проявляється. При цьому умови більш високого рівня також представляються як дерева, утворені факторами нижніх рівнів, що формують їх. Побудова дерева подій одночасно дозволяє визначити значення ймовірності виникнення результуючої несприятливої події.

Логічна функція (Л-функція) ініціюючих подій записується як

$$Y = S_1 \vee S_2 \vee \ldots \vee S_n .$$

Завдання формулюється таким чином: неуспіх відбувається, якщо відбувається яка-небудь одна, два і т. і. або всі ініціюючі події.

Л-модель ризику в еквівалентній ортогональній формі записується як

$$Y = S_1 \vee S_2\overline{S}_1 \vee S_3\overline{S}_2\overline{S}_1 \vee ...$$

Після ортогоналізації Л-функції може бути записана наступна імовірнісна функція – функція (імовірнісний поліном):

$$P = P_1 + P_2 \cdot Q_1 + P_3 \cdot Q_1Q_2 + .. ,$$

де $P_1, P_2, ..., P_n$ – ймовірності подій-ознак $S_1, S_2, ..., S_n$; $Q_1 = 1 - P_1, Q_2 = 1 - P_2$.

У моделях аварій і катастроф, сформованих на базі дерев подій, можуть бути використані імітаційні методи для отримання можливих (допустимих) значень випадкових збоїв на елементах нижнього рівня, якщо ці значення визначені з великою помилкою внаслідок відсутності достатньої інформації. Якщо вдається приблизно оцінити значення цих ймовірностей, можна з використанням датчика випадкових чисел задавати можливі значення цих ймовірностей, а потім за моделлю дерева подій визначити відповідні їм значення ймовірностей аварій. Такий підхід використовують для оцінки того, як невизначеності вихідних значень ймовірностей збоїв на нижніх рівнях системи вплинуть на значення ймовірності аварії, можливих у процесі її функціонування і розвитку. Датчик випадкових чисел використовується в даному випадку для формування різних станів ймовірностей збоїв на елементах нижнього рівня.

Використання логіко-лінгвістичного моделювання для дослідження людино-машинних систем ґрунтується як на врахуванні впливу психофізіологічних властивостей персоналу на реалізацію ними алгоритмів діяльності, так і факторів, які визначаються ергономічністю і надійністю обладнання.

Третій спосіб пов’язаний з використання моделей стану досліджуваної системи, що виражаються диференціальними рівняннями Колмогорова (визначається ймовірність аварійного стану об’єкта), наприклад, для безперервного Марківського ланцюга:

$$\frac{dP_1(t)}{dt} = -\lambda_{12}P_1(t) + \lambda_{21}P_2(t);$$

……..……..……..……..……..……

$$\frac{dP_i(t)}{dt} = -\lambda_{i-1,i}P_{i-1}(t) + \lambda_{i+1,i}P_{i+1} -$$

$$-(\lambda_{i-1,i} + \lambda_{i,i+1})P_i(t), \qquad i = \overline{2, n-1};$$

....

$$\frac{dP_n(t)}{dt} = \lambda_{n-1,n}P_{n-1}(t) - \lambda_{n,n-1}P_n(t),$$

де $\lambda_{i,j}$ – густота ймовірності (інтенсивність) переходу.

Прогноз, порівняльна оцінка та управління аварійним ризиком. Визначення величини аварійного ризику, породжуваного ОПН, і розробка рекомендацій щодо його зниження грають виключно важливу роль у всій методології аналізу ризику, пов’язаного з аваріями. Ці процедури логічно завершують множину різних підходів, методів і прийомів, що входять до арсеналу методології аналізу аварійного ризику.

Цей етап аналізу ризику можна умовно розбити на дві частини: 1) прогноз і порівняльна оцінка ризику та 2) управління аварійним ризиком. У першому випадку необхідно зробити прогноз величини сукупного аварійного ризику з урахуванням можли-

вого збитку від кожної окремої аварії та її інтенсивності і порівняти її з допустимим критичним значенням. У другому випадку необхідно розробляти в ході проведення всіх попередніх етапів аналізу ризику рекомендації щодо зниження можливого збитку та інтенсивностей прогнозованих аварій, щоб досягти прийнятного критичного значення сукупного аварійного ризику при обмежених економічних витратах. Заключний етап аналізу аварійного ризику містить ряд послідовно виконуваних процедур.

Перш за все передбачається, що повинен бути вибраний тип або вид аварійного ризику і відповідна йому міра. Ризик – багатогранне поняття, і, навіть, якщо обмежитися ризиком, породжуваним аваріями на підприємстві, можна розрізняти ризики за видом небезпеки, за характером джерел ризику, за реципієнтами ризику, за масштабами зони ураження та одиниць вимірювання. Відповідно до видів ризику існують і міри ризику. Найбільше поширення отримав аварійний ризик для однієї людини – локальний та індивідуальний ризик, ризик для групи людей – груповий ризик, та індекси ризику.

Після того, як форма подання ризику обрана, складають модель прогнозу і виконують необхідні обчислення. Потім виконується процедура порівняльної оцінки рівня аварійного ризику, коли дослідник повинен прийняти рішення, прийнятний ризик чи ні. Це рішення приймається на основі зіставлення знайдених значень ризику з фоновими та критичними значеннями. Під фоновим ризиком для людини, наприклад, мається на увазі ризик, якому піддається людина за безаварійних умов від різних природних, побутових небезпечних подій у даній місцевості. Фоновий ризик служить відправною точкою для призначення критичного рівня ризику. Критичний рівень визначає межу, перевищення якої недопустиме. Величина критичного рівня базується на міжнародному досвіді і закладається в нормативні документи.

Якщо рівень аварійного ризику прийнятний, аналіз аварійного ризику закінчується. В іншому випадку, коли ризик або можливі втрати визнаються недопустимо високими, виконується дослідження чутливості, ступеня невизначеності і значущості складових аварійного ризику. Визначається «найбільш вузька ланка» у системі забезпечення безпеки об’єкта. І відповідно до цього, а також з урахуванням економічних аспектів, розробляються рекомендації щодо зниження рівня ризику. Реалізація подібних рекомендацій дозволить знизити рівень небезпеки об’єкта. Проте, до її фактичного здійснення повинна бути знову проведена процедура аналізу аварійного ризику, включаючи попередній аналіз небезпек, математичне моделювання наслідків, частотний аналіз, і, нарешті, прогноз аварійного ризику і його порівняльну оцінку. Передбачається, що процедура аналізу ризику в загальному випадку повинна мати ітераційний характер.

Для виконання описаних вище завдань необхідно: розробити банк верифікованих математичних моделей, що використовуються при аналізі ризику; розвинути методологію математичного моделювання аварійних подій в умовах дефіциту вихідної інформації; створити інформаційне забезпечення, що містить, зокрема, відомості про параметри математичних моделей прогнозу аварійних ситуацій і умов їх застосування; розробити сучасне програмне забезпечення для вирішення завдань прогнозу техногенних аварій.

Висновки. Результати аналізу ризику використовуються при декларуванні об’єктів промислової безпеки небезпечних виробничих об’єктів, експертизі промислової безпеки. Необхідно відзначити, що загальноприйнятих значень рівня ризику в світі поки немає. Розкид порогових значень ступеня ризику (ізолініями на карті виділяють зони з різним ступенем небезпеки і вимірюють її в заданій точці ймовірністю ураження) пояснюється різним ставленням до ризику (добровільний або примусовий), рівнем розвитку промислової безпеки в країні, а також відмінностями в методології аналізу ризику. Одним з найважливіших підходів при визначенні прийнятного ризику є підхід, що полягає у визначенні ризику загинути протягом року людині будь-якого віку (як від

різних окремих причин, так і їх сукупності). Зокрема вважається, що максимально допустимою величиною ризику (критерієм небезпеки за рівнем ризику) є очікувана частота загибелі людини $5 \cdot 10^{-5}$ на рік. Залежно від очікуваних вигод може обговорюватися рівень ризику в діапазоні $10^{-3} \div 10^{-6}$.

Стратегія управління ризиком не є однозначною і багато в чому залежить від загального стану, пріоритетів і тенденцій розвитку економіки країни, від існуючої законодавчої та нормативної бази, налагодженості механізмів економічного і правового управління безпеки й охорони навколишнього середовища в промисловості та від ряду інших факторів.

ЛІТЕРАТУРА

1. Качинський А.Б. Безпека, загрози і ризик: наукові концепції та математичні методи / А.Б.Качинський. – Київ, 2004. – 472с.
2. Управління техногенною безпекою об'єктів підвищеної небезпеки / [В.Ф.Стоєцький, Л.В.Дранишников, А.Д.Есипенко та ін.]. – Тернопіль: Видавництво Астон, 2006. – 408с.
3. Дранишников Л.В. Системний ризик-аналіз техногенних аварій / Л.В.Дранишников // Математичне моделювання. – Дніпродзержинськ: ДДТУ. – 2015. – №1(32). – С.22-28.
4. Теоретические основы техногенной и экологической безопасности. Часть 2. Методы анализа и оценки риска аварий / Л.В.Дранишников, Ю.Н.Матвеев, Б.В.Палюх, В.Н.Богатиков. – Тверь: ТвГТУ, 2013. – 160с.
5. Стоецкий В.Ф. Оценка риска при авариях техногенного характера / В.Ф.Стоецкий, Л.В.Дранишников, В.И.Голинько // Науковий вісник НГУ. – 2014. – №3 – С.117-124.
6. Егоров А.Ф. Анализ риска, оценка последствий аварий и управление безопасностью химических, нефтеперерабатывающих и нефтехимических производств / А.Ф.Егоров, Т.В.Савицкая. – Москва: КолосС, 2010. – 626с.

Надійшла до редколегії 27.02.2017.

УДК 004.432

ЖУЛЬКОВСКАЯ И.И., к.т.н., доцент
ЖУЛЬКОВСКИЙ О.А., к.т.н., доцент
БИЛЬО В.В., студент

Днепровский государственный технический университет, г. Камянское

ТИПИЗАЦИЯ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Введение. Эффективность машинных алгоритмов во многом зависит от используемых языков и систем программирования. При этом повышение эффективности языков и систем программирования тесно связано с совершенствованием следующих средств: анализа и обработки информации; информационного обеспечения, организации обработки и управления данными, процедурами, моделями, декомпозиции больших программ, семантических, синтаксических и морфологических возможностей языка; адаптации к внутренним и внешним условиям использования языка; технологии программирования и т.п.

Прогресс компьютерных технологий сопровождается созданием новых и совершенствованием существующих средств общения программистов с ЭВМ – языков программирования (ЯП). Со времени создания первых компьютеров человечество придумало уже более восьми с половиной тысяч ЯП.

Чтобы разобраться во всем многообразии ЯП, нужно знать их классификацию, историю создания, эволюцию и тенденции развития.

Постановка задачи. В настоящее время существует большое количество работ, посвященных общему анализу ЯП. В то же время актуальной остается проблема систематизации ЯП в соответствии с методологией хранения и обработки данных.

Целью настоящей работы является исследование особенностей типизации наиболее популярных (рейтинговых) на текущий момент ЯП для правильного выбора необходимого программного обеспечения при решении широкого класса задач.

Результаты роботы. Для определения рейтинга популярности ЯП может служить индекс TIOBE (TIOBE *programming community index*) [1].

На индекс TIOBE ориентируется большинство авторов в научных публикациях при сравнении популярности ЯП, несмотря на его косвенные, проприетарные методики и платность набора исходных данных [2].

Для формирования индекса используется поиск на нескольких, наиболее посещаемых (по данным Alexa.com), порталах: Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon. Полученные результаты нормализуются по некоторой формуле, которая и определяет место языка в рейтинге. Если считать первые 50 ЯП за 100%, то рейтинг TIOBE в числовом выражении показывает долю, занимаемую каждым языком. В табл.1 представлено десять первых позиций современного рейтинга TIOBE.

Таблица 1 – Индекс TIOBE за февраль 2017 года

Февраль 2017	Февраль 2016	Язык программирования	Рейтинг	Изменение
1	1	Java	16.676%	-4.47%
2	2	C	8.445%	-7.15%
3	3	C++	5.429%	-1.48%
4	4	C#	4.902%	+0.50%
5	5	Python	4.043%	-0.14%
6	6	PHP	3.072%	+0.30%
7	9	JavaScript	2.872%	+0.67%
8	7	Visual Basic .NET	2.824%	+0.37%
9	10	Delphi/Object Pascal	2.479%	+0.32%
10	8	Perl	2.171%	-0.08%

Как известно, все ЯП можно разделить на типизированные и нетипизированные (бестиповые). К бестиповым обычно относятся низкоуровневые (ассемблер) или эзотерические языки. В этих языках все хранимые данные – это просто последовательность бит различной длины.

Операция назначения типа, называемая типизацией, придаёт смысл цепочкам бит, таким как значение в памяти компьютера, или объектам, таким как переменная.

Система типов – это гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений [3].

В состав ЯП включается система типов для осуществления проверки типов во время компиляции или во время исполнения, требующая явного провозглашения типов или выводящая их самостоятельно. Именно поэтому типизированные языки разделяются на категории [4].

Статическая типизация – приём, широко используемый в ЯП, при котором переменная, параметр подпрограммы, возвращаемое значение функции связываются с типом в момент объявления, и тип не может быть изменён позже (переменная или параметр будут принимать, а функция – возвращать значения только этого типа). Стати-

ческая типизация означает, что все проверки типов данных выполняются на этапе компиляции, а не на этапе выполнения программы. Проверки типов данных, выполняемые на этапе компиляции, используют только сам код программы. Преимуществом статической типизации является то, что такие проверки достаточно выполнить только один раз. Кроме того, отсутствие проверок, выполняемых на этапе выполнения программы, и знание всех типов данных на этапе компиляции позволяет сделать скомпилированную программу более эффективной. Статическая типизация используется в большинстве компилируемых ЯП.

Динамическая типизация – приём, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Динамическая типизация означает, что большая часть проверок типов данных выполняется на этапе выполнения программы, хотя некоторые проверки (например, проверка синтаксической корректности кода) выполняются на этапе компиляции. Динамическая типизация позволяет создавать более гибкое программное обеспечение, хотя и ценой большей вероятности ошибок типизации. Модульное тестирование приобретает особое значение при разработке программного обеспечения на ЯП с динамической типизацией, т.к. оно является единственным способом нахождения ошибок типизации, допущенных в редко используемых ветвях логики программы.

Язык с явной типизацией предполагает, что программист должен указывать типы всех переменных и функций, которые объявляет.

Неявная типизация не требует явного объявления типа для используемых переменных. Этот вид типизации обычно связывают с динамической типизацией, для которой типы данных ассоциируются с конкретными значениями, а переменной можно присвоить значение любого типа. Неявная типизация также может быть реализована средствами статической типизации, если язык позволяет вывод типов, т.е. автоматическое определение типа переменной путем вычисления соответствующего выражения.

Используя вышеописанную классификацию, далее представлен анализ современных ЯП в хронологическом порядке.

ЯП *C* разработан в 1972 г., автор – Деннис Ричи (*Bell Labs*). *C* – поддерживает процедурную парадигму программирования и статическую явную типизацию, являясь компилируемым языком общего назначения.

ЯП *C++* появился в 1983 г., автор – Бьёрн Страуструп (*Bell Labs*). *C++* – компилируемый ЯП общего назначения, поддерживает объектно-ориентированную и процедурную парадигмы программирования, а также статическую явную типизацию. После стандарта *C++11* получил поддержку неявной типизации с помощью ключевых слов *auto* и *decltype*. Поддерживает динамическую типизацию при использовании библиотеки *Boost*.

ЯП *Perl* разработан Ларри Уоллом (*Unisys*) в 1987 г. *Perl* – высокоуровневый интерпретируемый ЯП общего назначения. Одним из главных достоинств языка является поддержка различных парадигм (процедурный, объектно-ориентированный и функциональный стили программирования). Типизация – динамическая неявная, с версии *Perl5.6* довольно ограниченно можно воспользоваться преимуществами статической типизации.

ЯП *Python* опубликован сотрудником голландского института *CWI* Гвидо ван Россумом в 1991 г. *Python* поддерживает несколько парадигм программирования, в том числе объектно-ориентированное, функциональное, процедурное. Эталонной реализацией *Python* является интерпретатор *CPython*, поддерживающий большинство активно используемых платформ. Есть реализации интерпретаторов для *JVM* (с возможностью

компиляции), *MSIL* (с возможностью компиляции), *LLVM* и других. Типизация – динамическая неявная.

В 1995 г. вышел первый публичный релиз ЯП *PHP*, автор – датский программист Расмус Лердорф. *PHP* – интерпретируемый (интерпретатор компилирующего типа), сценарный язык общего назначения, интенсивно применяемый для разработки вебприложений. *PHP* поддерживает объектно-ориентированную, функциональную, процедурную парадигмы программирования. Типизация – динамическая неявная.

Java – объектно-ориентированный ЯП, разработанный компанией *Sun Microsystems* (в последующем приобретённой компанией *Oracle*) в 1995 г. Приложения *Java* обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной *Java*-машины. Типизация – статическая явная.

JavaScript – прототипно-ориентированный сценарный ЯП (компания *Netscape*, 1995 г.). *JavaScript* поддерживает несколько парадигм программирования, в т.ч. объектно-ориентированное, функциональное, процедурное. Типизация – динамическая неявная.

ЯП *C#* разработан в 1998-2001 г.г. группой инженеров под руководством Андерса Хейлсберга (компания *Microsoft*) как язык разработки приложений для платформы *Microsoft .NET Framework*. Класс языка – мультипарадигмальный (процедурный, объектно-ориентированный и функциональный стили программирования). Типизация – статическая явная. Также *C#* поддерживает динамическую типизацию посредством специального псевдо-типа *dynamic* с версии 4.0. Поддерживает неявную типизацию с помощью *dynamic* и *var*.

С ЯП *Delphi* обычно ассоциируется среда разработки приложений на основе языка *Object Pascal*, разработанного фирмой *Borland*. Этот язык является наследником *Turbo Pascal* с объектно-ориентированными расширениями, который, в свою очередь, ведет свою историю от классического *Pascal*, созданного Никлаусом Виртом в 1970 г. Впоследствии, в 2002 г. разработчики из компании *Borland* официально поставили знак равенства между языками *Delphi* и *Object Pascal*. *Delphi* – императивный, структурированный, объектно-ориентированный ЯП со статической явной типизацией переменных. *Delphi* поддерживает динамическую типизацию посредством специального типа *Variant*.

ЯП *Visual Basic .NET* – объектно-ориентированный язык, который можно рассматривать как очередной виток эволюции *Visual Basic*, реализованный на платформе *Microsoft .NET*. В 2002 г. появился первый выпуск *Visual Basic .NET*. С этого момента обратная совместимость с классической версией *Visual Basic* оказалась нарушена. Тип исполнения – компилируемый, интерпретируемый ЯП. Типизация – динамическая явная.

В табл.2 отмечено наличие/отсутствие возможностей типизации в популярных по рейтингу TIOBE ЯП.

Таблица 2 – Типизация языков программирования

Язык программирования	Статическая	Динамическая	Явная	Неявная
1	2	3	4	5
Java	+	-	+	-
C	+	-	+	-
C++	+	-	+	-/+
C#	+	-/+	+	-/+
Python	-	+	-	+
PHP	-	+	-	+
JavaScript	-	+	-	+

Продолжение таблицы 2

1	2	3	4	5
Visual Basic.NET	-	+	+	+
Delphi/Object Pascal	+	-/+	+	-
Perl	+/-	+	-/+	+

Примечание:
+ – указанная возможность присутствует;
- – указанная возможность отсутствует;
-/+ – возможность поддерживается очень ограниченно;
+/- – возможность поддерживается не полностью.

Выводы. Проведен общий анализ современных ЯП. Выполнено исследование особенностей типизации наиболее популярных (рейтинговых) ЯП для правильного выбора необходимого программного обеспечения при решении широкого класса задач. Результаты исследований показали низкую заинтересованность разработчиков программного обеспечения в нетипизированных языках.

ЛИТЕРАТУРА

1. TIOBE Index for February 2017. [Электронный ресурс] – Режим доступа: https://www.tiobe.com/tiobe-index/.
2. Жульковская И.И. К выбору стратегии преподавания информатики для инженерных специальностей университетов / Жульковская И.И., Жульковский О.А. // Зб. наукових праць ДДТУ (технічні науки). – Дніпродзержинськ: ДДТУ. – 2013. – №.1 (21). – С.171-176.
3. Пирс Б. Типы в языках программирования / Пирс Б.; перевод с англ. – М.: Издательство «Лямбда пресс»-«Добросвет», 2011. – 680с.
4. Груздев Д. Ликбез по типизации в языках программирования [Электронный ресурс] / Груздев Д. – Режим доступа: https://habrahabr.ru/post/161205/.

Поступила в редколлегию 29.03.2017.

УДК 519.688

ПИШНИЙ М.А., аспірант
МАРЧЕНКО О.О., магістр
КОСУХІНА О.С., к.ф.-м.н., доцент
ГУЛЄША О.М., к.пед.н., доцент

Дніпровський державний технічний університет, м. Кам'янське

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ ОБРОБКИ ДАНИХ: МУРАШИНІ АЛГОРИТМИ

Вступ. В останні два десятиліття при вирішенні комбінаторно-логічних задач проектування, конструювання та штучного інтелекту, а також при оптимізації складних систем дослідники все частіше застосовують природні механізми пошуку оптимальних рішень. Багато методик, які стосуються м'яких обчислень, експлуатують ідею того, що множина відносно простих об'єктів, працюючих за цілком зрозумілими правилами, об'єднуючись, демонструють поведінку, що набагато перевищує за інтелектуальністю поведінку окремого індивідуума. Мурашині алгоритми – це перспективний метод оптимізації, що базується на моделюванні поведінки колонії мурах. Оригінальний алгоритм (Ant Colony Optimization) виник в процесі спостереження за тим, як живі мурахи виду Argentine Ant обстежують територію навколо мурашника, знаходять їжу і несуть її в мурашник, постійно оптимізуючи (скорочуючи) шлях, який проходить кожен з мурах. Ці дослідження проводилися в 1989-му році Госсом і в 1990-му – Денеборгом [1]. Перша

математична формалізація алгоритму запропонована в 1992-му році Марко Доріго [2]. Застосування мурашиних алгоритмів для різного роду задач, таких як задача комівояжера, задача календарного планування, транспортна задача, квадратична задача про призначення, задача оптимізації мережевих трафіків, задача розмальовки графа, розробка оптимальної структури мереж GPS в рамках створення високоточних геодезичних та знімальних технологій тощо, розглядаються і в роботах вітчизняних вчених, наприклад, Штовби С.Д., Рудого О.М. [3-4], Колеснікова К.В, Карапетяна А.Р., Кравченка О.В. та ін.

Колонія мурах може розглядатися як багатоагентна система, в якій кожний агент (мураха) функціонує автономно за дуже простими правилами. На противагу майже примітивній поведінці агентів поведінка всієї системи виходить напрочуд розумною.

Живі мурахи під час пошуків їжі ходять навколо мурашника випадковим чином по стежках, які не є фізичними доріжками, «протоптаними» поколіннями комах. Основна орієнтація у мурах відбувається за рахунок феромонів, до яких вони дуже чутливі і якими вони позначають все навколо. Стежки стають все більш помітними. Але іноді мурашки збиваються з шляху і тоді випадковим чином шукають місце, позначене феромонами. Це може призводити до знаходження коротшого шляху. У цей час відбувається процес випаровування феромону. Якби його не було, то початковий шлях завжди мав би найсильніший запах, і процес пошуку коротшого шляху не відбувався.

До найбільш вдалих вдосконалених алгоритмів відносять:

- Elitist Ant System. Вдосконалення полягає у введенні в алгоритм так званих «елітних мурах». Досвід показує, що проходячи ребра, які входять в короткі шляхи, мурахи з більшою ймовірністю будуть знаходити коротші шляхи. Ефективною стратегією є штучне збільшення рівня феромонів на найвдаліших маршрутах. Для цього на кожній ітерації алгоритму кожна з елітних мурах проходить шлях, який є найкоротшим із знайдених на даний момент. Експерименти показують, що, до певного рівня, збільшення числа елітних мурах є досить ефективним, дозволяючи значно скоротити число ітерацій алгоритму. Однак, якщо число елітних мурах занадто велике, то алгоритм досить швидко знаходить субоптимальне рішення і застряє в ньому. Як і інші змінні параметри, оптимальне число елітних мурах слід визначати дослідним шляхом;
- Ant-Q. Цей мурашиний алгоритм отримав свою назву за аналогією з методом машинного навчання Q-learning. В основі алгоритму лежить ідея про те, що мурашину систему можна інтерпретувати як систему навчання з підкріпленням. Ant-Q підсилює цю аналогію, запозичуючи багато ідей з Q-навчання. Алгоритм зберігає Q-таблицю, що зіставляє кожному з ребер величину, яка визначає «корисність» переходу по цьому ребру. Q-таблиця змінюється в процесі роботи алгоритму – відбувається навчання системи. Значення корисності переходу по ребру обчислюється, виходячи із значень корисності переходу по наступних ребрах в результаті попереднього визначення можливих наступних станів. Після кожної ітерації корисності оновлюються, виходячи з довжин шляхів, до складу яких було включено відповідні ребра;
- Ant Colony System. Для підвищення ефективності в порівнянні з класичним алгоритмом введено три основних зміни: по-перше, рівень феромонів на ребрах оновлюється не лише в кінці чергової ітерації, але і при кожному переході мурах з вузла у вузол; по-друге, наприкінці ітерації рівень феромонів підвищується тільки на найкоротшому із знайдених шляхів; по-третє, алгоритм використовує змінене правило переходу: або, з певною часткою ймовірності, мураха безумовно вибирає краще ребро у відповідності до довжини і рівня феромонів, або робить вибір так само, як і в класичному алгоритмі;
- Max-min Ant System. У цьому мурашиному алгоритмі підвищення концентрації феромонів відбувається тільки на кращих шляхах з тих, що пройшли мурахи. Така велика увага до локальних оптимумів компенсується введенням обмежень на максимальну і мінімальну концентрації феромонів на ребрах, які вкрай ефективно захищають алгоритм від передчасної збіжності до субоптимальних рішень. На етапі ініціалізації концентрація феромонів на всіх ребрах встановлюється рівною максимальній. Після кож-

ної ітерації алгоритму тільки одна мураха залишає за собою слід: або найбільш успішна на даній ітерації, або, аналогічно алгоритму Elitist Ant System, елітна. Цим досягається, з одного боку, більш ретельне дослідження області пошуку, з іншого – його прискорення.

Постановка задачі. У роботі представлена алгоритмічна реалізація мурашиного алгоритму для задачі комівояжера, та її програмне втілення за допомогою мови програмування C#.

Задача комівояжера формулюється як задача пошуку мінімального за вартістю замкнутого маршруту за всіма вершинами без повторень на повному зваженому графі з n вершинами. Змістовно вершини графа є місцями, які повинен відвідати комівояжер, а вага ребер відображає відстань (довжину) або вартість проїзду.

Розглянемо більш формально один з найпростіших мурашиних алгоритмів. Лабіринт задається графом (вершини і ребра). Вважаємо, що мурахи шукають оптимальний шлях (найкоротший) між двома вершинами (мурашник і їжа), поки (не виконані умови виходу):

1 – створюємо мурах. Початкові точки, куди поміщаються мурахи, залежать від обмежень задачі. У найпростіших випадках ми можемо їх всіх помістити в одну точку або випадково розподілити по площині графа. На цьому ж етапі кожне ребро графа позначається невеликим позитивним числом, що характеризує запах феромону. Це потрібно для того, щоб на наступному кроці у нас не було нульових ймовірностей;

2 – мурахи шукають рішення. Визначаємо ймовірність переходу з вершини i у вершину j за формулою

$$P_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[d_{ij}\right]^{-\beta}}{\Sigma_{l \in connected\_nodes}\left[\tau_{ij}(t)\right]^{\alpha}\left[d_{ij}\right]^{-\beta}}, \quad (1)$$

де $\tau_{ij}(t)$ – рівень феромону; d_{ij} – евристична відстань; α, β – константи.

Якщо $\alpha = 0$, то найбільш вірогідним є вибір найближчого сусіда, і алгоритм стає «жадібним». У разі, коли $\beta = 0$, найбільш імовірний є вибір тільки на основі рівня феромону, що призводить до того, що мурахи залишаються на вже «протоптаних» шляхах. Як правило, використовується деяке компромісне значення цих величин, яке підбирається експериментально для кожної задачі. Застосування такого ймовірнісного правила забезпечує реалізацію однієї з основних складових самоорганізації – випадковості;

3 – оновлення рівня феромону відбувається за формулою

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Sigma_{k \in used\_edge(i,j)}\frac{Q}{L_k(t)}, \quad (2)$$

де p – параметр, що задає інтенсивність випаровування; L_k – ціна поточного розв'язання для *k*-го мурахи; Q – характеризує порядок ціна оптимального рішення. Таким чином вираз $\frac{Q}{L_k(t)}$ визначає кількість феромону, яким мураха *k* відмітив ребро (i,j) [5].

Результати роботи. Для розв'язання задачі комівояжера розроблено програму в C# з використанням Win Forms, оскільки задачі, пов'язані з графами, незручно розглядати в консолі. Створена програма складається із декількох основних модулів.

Вихідні дані. Маємо граф з сімнадцятьма вершинами (випадкове значення), кожна з вершин повинна бути включеною в маршрут (відповідно до умов задачі комівояжера). Вершини розміщуються випадковим чином, так само як обираються початкові параметри системи.

За допомогою мурашиного алгоритму та блок-схеми, зображеної на рис.1, знаходимо оптимальний маршрут.

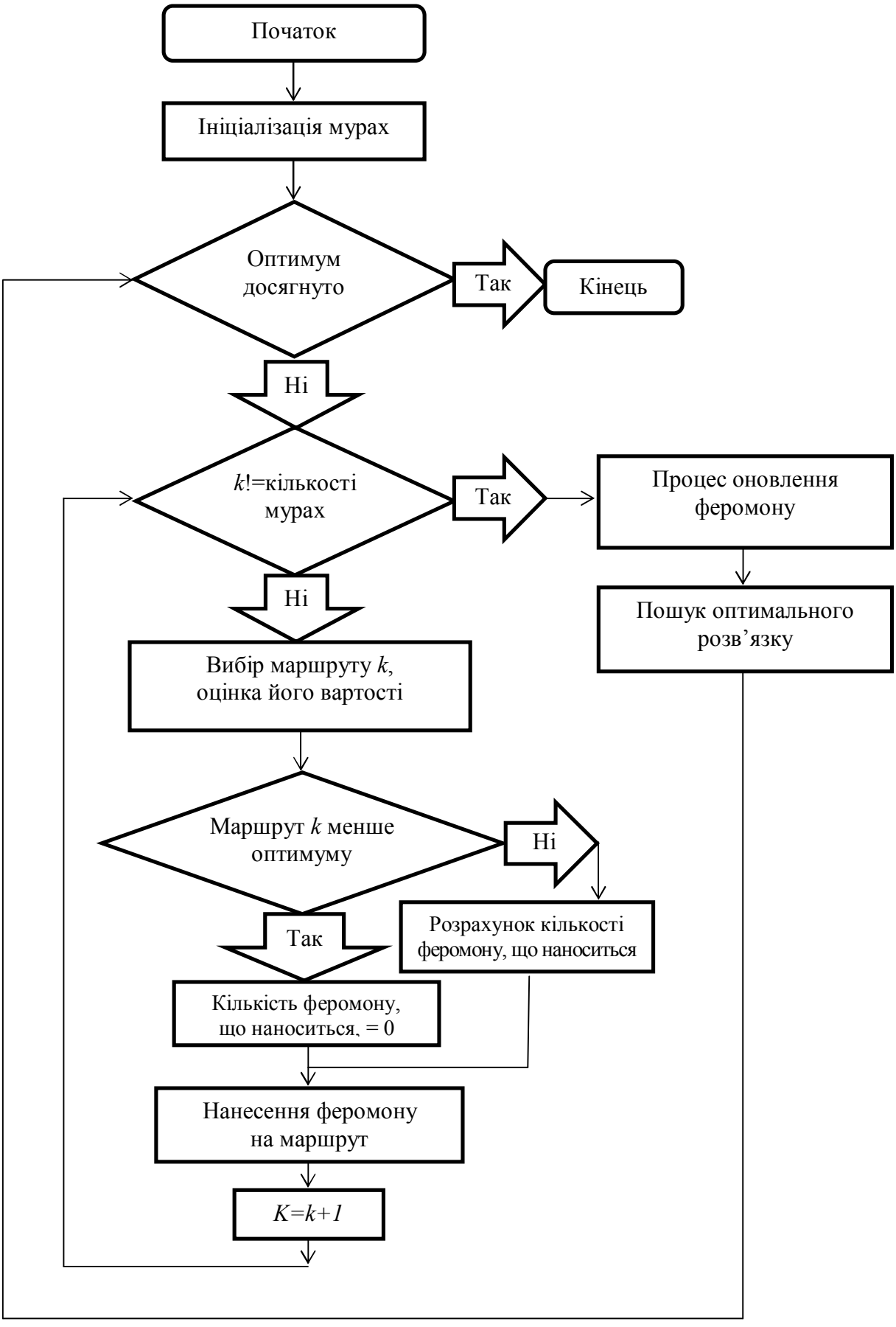


Рисунок 1 – Детальна блок-схема мурашиного алгоритму

На рис.2 зображено кращий, як для третього кроку алгоритму, маршрут та відображено довжину кращого шляху, яка складає приблизно 4.9 одиниць.

Рис.3 дає змогу спостерігати чотирьохсотий крок алгоритму, який відображає фінальну стадію задачі. З рис.3 видно, що довжина знайденого оптимального шляху зменшилася приблизно до 3.89 одиниць, що є на 8% (вісім відсотків) краще, ніж демонстрував рис.2. Проте, зважаючи на те, що різниця між рис.2 і 3 полягає у 311 кроках алгоритму, можна стверджувати, що для досягнення фінальної стадії оптимізації мурашиному алгоритму знадобився, можливо, надмірно великий час з огляду на порівняно невеликий відсоток оптимізації.

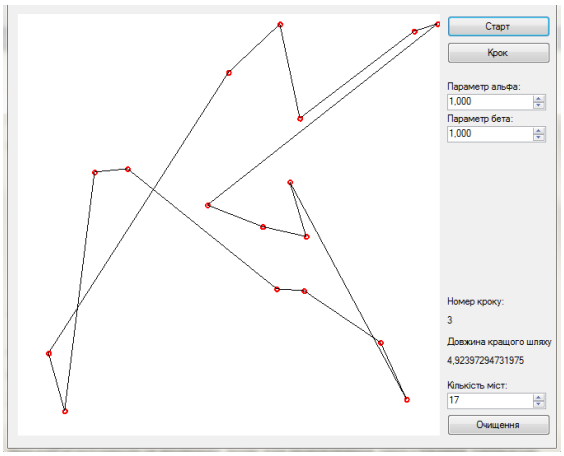


Рисунок 2 – Початок роботи програми

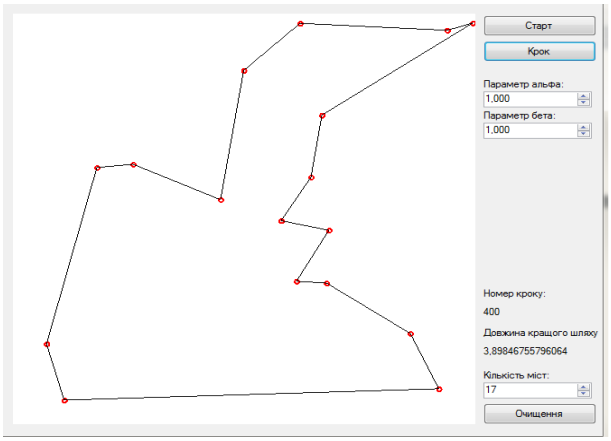


Рисунок 3 – Оптимальний маршрут знайдено

Звісно, кожний випадок буде мати свою швидкість збіжності, і для деяких з них оптимізація буде швидшою або повільнішою. Також не можна відкидати вірогідність того, що навіть після 12 тисяч ітерацій можливо отримати більш оптимальний маршрут. Проте, якщо казати про більш тривіальні випадки, наприклад, наведений вище, то можна стверджувати, що рис.2, 3 показують усереднену динаміку рішення задачі, а також, що основна оптимізація у класичному мурашиному алгоритмі, застосованому до графа з невеликою кількістю (приблизно до 50) вершин, відбувається протягом перших 100 ітерацій.

У роботі проаналізовано також вплив деяких параметрів на ефективність отриманих рішень. Підібрати оптимальні значення параметрів α та β можна експериментальним шляхом. Для довільно згенерованих 60 вершин графа в табл.1 відображено отримані значення.

Таблиця 1 – Залежність ефективності рішень від вибраних параметрів на прикладі задачі з 60 вершинами графа

Параметри	Найкраще рішення	Найгірше рішення	Середній результат
1	2	3	4
$\alpha = 1$ та $\beta = 1$			
100 ітерацій	7.04	16.83	11.94
1000 ітерацій	6.52	6.82	6.67
$\alpha = 2$ та $\beta = 1$			
100 ітерацій	15.68	6.68	11.18
1000 ітерацій	6.68	6.68	6.68

Продовження таблиці 1

1	2	3	4
α = 5 та β = 1			
100 ітерацій	8.04	7.04	7.54
1000 ітерацій	6.99	6.89	6.94
α = 0 та β = 1			
100 ітерацій	7.57	8.39	7.99
1000 ітерацій	7.39	7.86	7.63
α =1 та β = 2			
100 ітерацій	3.01	3.34	3.16
1000 ітерацій	2.67	2.82	2.74
α = 1 та β = 5			
100 ітерацій	2.63	3.49	2.66
1000 ітерацій	2.60	2.61	2.61
α = 1 та β = 0			
100 ітерацій	11.66	11.68	11.67
1000 ітерацій	9.56	10.05	9.86

Вихідні дані. Маємо граф з 60 вершинами (випадкове значення), кожна з вершин повинна бути включеною в маршрут (відповідно до умов задачі комівояжера). Вершини розміщуються випадковим чином, так само як обираються початкові параметри системи. На рис.4, 5 наведемо отримані результати для α=1, β=1.

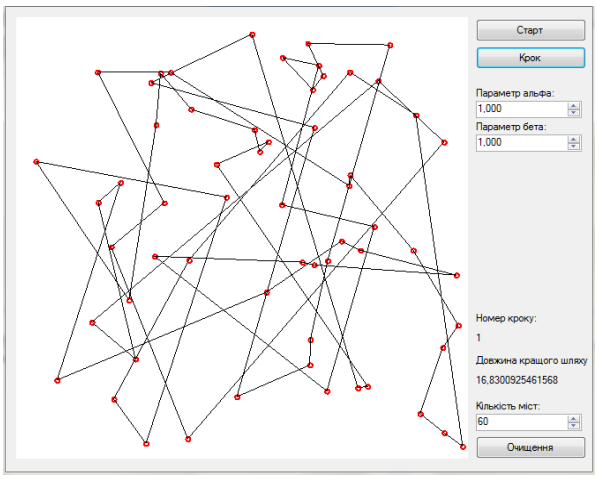


Рисунок 4 – Найгірше значення для 100 ітерацій при α=1, β=1

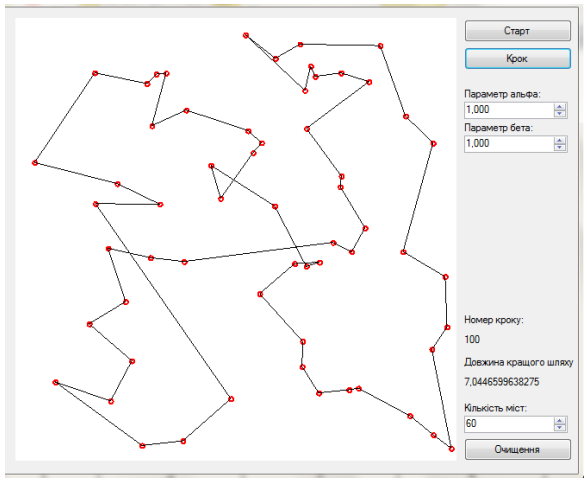


Рисунок 5 – Найкраще значення для 100 ітерацій при α=1, β=1

З наведених в табл.1 значень робимо висновок, що зміна значень параметрів α та β призводить до суттєвих змін в отриманих рішеннях. Найкращі результати (для 60 вершин графа) були отримані за наступних значень параметрів: (α=2, β=1) та (α=1, β=5). Збільшення значень параметра α (збільшується залежність отриманого рішення від концентрації феромону) призводить до збільшення діапазону рішень. Збільшення значень параметра β (збільшується залежність отриманого рішення від маршруту) призводить до зменшення діапазону рішень. Робота мурашиного алгоритму носить еврис-

тичний характер при більшому діапазоні рішень та „точний” характер – при невеликому діапазоні рішень.

Висновки. В роботі на прикладі задачі комівояжера розглянуто алгоритмічну реалізацію мурашиного алгоритму з впровадженням складових самоорганізації мурах, таких як випадковість, багатократність взаємодії, негативна і позитивна складові зв’язку. Програмна реалізація даного алгоритму здійснена за допомогою мови програмування C#.

ЛІТЕРАТУРА

1. Self-Organized Shortcuts in the Argentine Ant / Goss S., Aron S., Deneubourg J.L., Pasteels J.M. // Naturwissenshaften. – 1989. – № 76. – P.579-581.
2. Dorigo M. Swarm Intelligence, Ant Algorithms and Ant Colony Optimization / Dorigo M. // Reader for CEU Summer University Course «Complex System». – Budapest: Central European University. – 2001. – P.1-38.
3. Штовба С.Д. Муравьиные алгоритмы / С.Д.Штовба // Математика в приложениях. Exponenta Pro. – 2003. – №4. – С.70-75.
4. Штовба С.Д. Муравьиные алгоритмы оптимизации (на укр. языке) / С.Д.Штовба, О.М.Рудый // Вестник ВПИ. – 2004. – № 4. – С.62-69.
5. Шумейко А.А. Интеллектуальный анализ данных (Введение в Data Mining): учеб. пособ. / А.А.Шумейко, С.Л.Сотник. – Днепропетровск: издатель Белая Е.А. – 2012. – 210с.

Надійшла до редколегії 25.04.2017.

УДК 004.031.43

ЯШИНА К.В., к.т.н., доцент
ЯЛОВА К.М., к.т.н., доцент
СУГАЛЬ Є.О., студент

Дніпровський державний технічний університет, м. Кам’янське

ОГЛЯД ГНУЧКИХ МЕТОДОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Agile – це сімейство методологій розробки програмних продуктів, що орієнтовані на використання ітеративної технології з динамічним формуванням та уточненням системи вимог на кожному ітераційному кроці. Дискретизація локальних процесів передбачає організацію їх взаємодії в середині робочих груп, створених з кваліфікованих фахівців різного профілю. У світі розробки програмного забезпечення (ПЗ) Agile методології прийнято називати «гнучкими» або «легкими».

Гнучкі методології розробки ПЗ є досить молодими. Маніфест Agile прийнято у лютому 2001 року. Цей маніфест формулює чотири основні ідеї методології:

- особистісний підхід (особистості та їх взаємодії є важливішими, ніж процеси та інструменти);
- забезпечення працездатності продукту, що розробляється (ПЗ, що ефективно працює є важливішим, ніж наявність детальної документації);
- забезпечення конструктивного діалогу між розробником та замовником (співпраця із замовником є важливішою, ніж жорсткі контрактні зобов’язання);
- гнучкість методів розробки з урахуванням вимог замовника, що динамічно оновлюються (реакція на зміни є важливішою, ніж чітке дотримання початкового плану) [1].

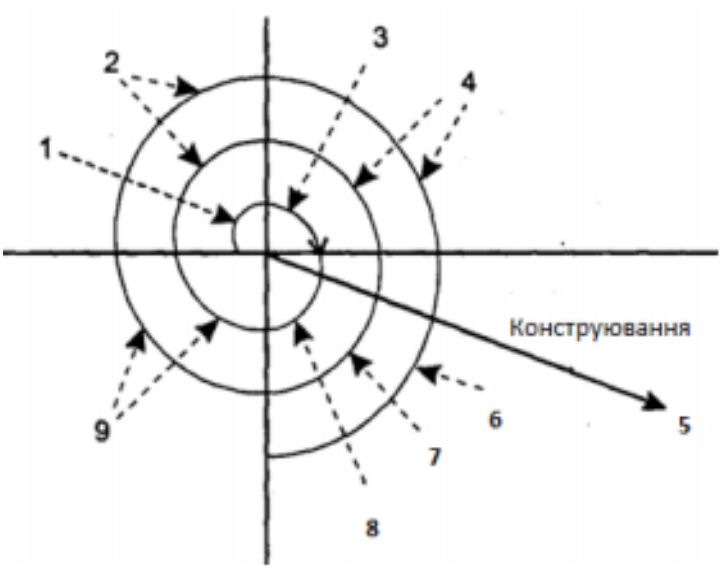
На сьогоднішній день гнучкі методології – основа сучасного світу розробки програмного забезпечення.

Постановка задачі. Задачею дослідження є виконання варіативного аналізу Agile-методологій розробки ПЗ та визначення особливостей застосування Agile-методії під час викладання дисципліни «Групова динаміка та комунікації» для студентів спеціальності 121 – «Інженерія програмного забезпечення».

Результати роботи. У наш час існує декілька Agile-методологій: Scrum, екстремальне програмування, Crystal Clear, Dynamic Systems Development Method, Agile Unified Process, ICONIX і т.д.

Згідно з маніфестом Agile основні принципи, притаманні будь-якій з гнучких методологій розробки ПЗ, наступні:

- задоволення клієнта за рахунок постійного надання програмного забезпечення, що якісно працює;
- дозвіл на внесення змін до вимог навіть наприкінці розробки з метою підвищення конкурентоспроможності кінцевого продукту;
- періодичне надання програмного забезпечення за визначеним планом (щомісяця, щотижня або частіше);
- забезпечення щоденного конструктивного діалогу між замовником і розробником протягом усього терміну виконання проекту;
- мотивація виконавців проекту, забезпечення їх гідними умовами праці, підтримкою та довірою;
- рекомендований метод передачі інформації – особиста розмова («обличчям до обличчя»);
- орієнтація на забезпечення працездатності ПЗ, що розробляється;
- постійне поліпшення технічної майстерності і реалізація зручного дизайну;
- простота реалізації;
- самоорганізація в команді;
- постійна адаптація до зміни обставин і вимог до реалізації ПЗ.



1 – початковий збір вимог і планування проекту;
2 – виконання тієї ж роботи з урахуванням оновлених вимог розробника і замовника ПЗ на кожному витку ітерації;
3 – аналіз ризику на основі початкових умов;
4 – аналіз ризику з урахуванням реакції замовника на кожному витку ітерації, оптимізація процесу програмування шляхом скорочення надлишкових операцій;
5 – перехід до комплексної системи;
6 – проектування початкової моделі системи, розробка початкової версії ПЗ;
7 – конструювання ітераційних версій ПЗ вищого рівня;
8 – розробка кінцевого програмного продукту;
9 – ітераційне оцінювання проекту замовником та тестувальником, діалог між замовником та розробником, розповсюдження проміжних і кінцевої версії ПЗ

Рисунок 1 – Модель розробки ПЗ гнучкими методами

Гнучкі методології розробки ПЗ належать до еволюційної стратегії. Процес створення програмних продуктів засобами Agile-методології можна подати у вигляді спіральної моделі, яка презентує ітеративний підхід у процесі розробки ПЗ (рис.1) [2].

Розглянемо особливості екстремального програмування (XP), функціонально-орієнтованої розробки (FDD) та Scrum-методу.

Екстремальне програмування (Extreme Programming) було запропоноване Кентом Беком у кінці 90-х років минулого століття . Модель процесу програмування акумулює випуск дискретних версій ПЗ, які виходять так часто, наскільки це можливо. Кожна версія обов'язково містить нову функціональність алгоритму розв'язку задачі. На етапі генерації коду використовують наявні стандарти кодування, що дозволяє уніфікувати робочі функції та забезпечити подальше багаторазове використання розроблених і стандартних програмних компонентів у процесі створення ПЗ за ітераційною RAD-моделлю конструювання ПЗ (рис.2) [3].

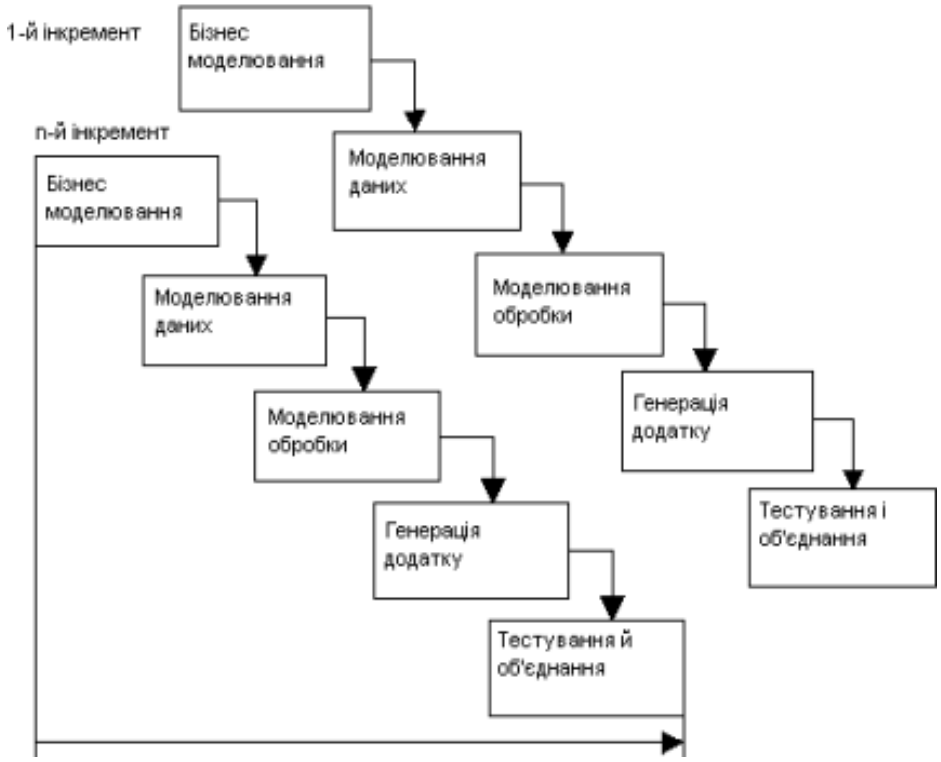


Рисунок 2 – RAD-модель конструювання ПЗ

Метод функціонально-орієнтованої розробки – Feature Driven Development (FDD) – є ітераційним гнучким методом методології Agile. FDD орієнтований на об'єднання популярних методик, спрямованих на забезпечення функціональності програмного продукту, що розробляється. Головним завданням FDD є розробка реальної працездатної версії ПЗ у визначені планом терміни. Модель FDD, наведена на рис.3, характеризує ітераційність та циклічність процесів розробки ПЗ. Розробка моделі починається з високорівневого наскрізного аналізу кола наявних завдань і контексту системи, будується узагальнена модель програмного продукту. Подальша деталізація етапів створення ПЗ виконується за допомогою інкрементних циклів моделі. Кожен інкремент забезпечує реалізацію проміжної версії ПЗ. Ітераційний цикл виконує умови базового інкремента. FDD є зручною для розробки програмних продуктів, умови реалізації яких є строго визначеними замовником і враховуються на початку розробки загальної моделі та деталізуються на кожному інкрементному та ітераційному кроці циклічного блоку реалізації визначених функцій [2].

Scrum є гнучким методом розробки програмного проекту невеликою (до 9 осіб) командою програмістів в умовах схематичної зміни (доповнення, модифікації) вимог до ПЗ. Процес розробки ПЗ є ітераційним і забезпечує можливість доповнення системи вимог на кожному ітераційному кроці моделі (рис.4).

Рисунок 3 – Модель гнучкого методу функціонально-орієнтованої розробки ПЗ

Рисунок 4 – Схема методології Scrum

Для Scrum-методу характерна довгостроковість у розробці кінцевої версії ПЗ за рахунок обумовленої необхідності ітераційного обговорення результатів програмування із замовником на етапі формулювання вимог до програмного продукту. Проте тісний зв'язок розробників із користувачем забезпечує високу надійність та функціональність розроблюваного ПЗ [2].

Висновки. Таким чином, за результатами проведених досліджень виконано варіативний аналіз Agile-методологій розробки ПЗ, за результатами якого можна зробити наступні висновки:

1. Гнучкі методи розробки програмного продукту є альтернативою формальним і великоваговим методологіям. Agile-методологій забезпечують процес більшою ефективністю та зменшують обсяг рутинної роботи, пов'язаної з оформленням супровідної документації.
2. Використання методу екстремального програмування, скрам-методу та функціонально-орієнтованої розробки дозволяє підвищити швидкість й надійність процесів програмування та дотримання визначених термінів виконання проекту.
3. Використання Agile-методології дозволяє забезпечити ефективне виконання вимог замовника у процесі розробки програмного забезпечення за рахунок наявної можливості використання додаткових консультацій із замовником на кожному ітераційному витку моделі створення програмного продукту.

Під час викладання дисципліни «Групова динаміка та комунікації» для студентів спеціальності 121 – «Інженерія програмного забезпечення» слід спиратися на Agile-методологій як основу сучасного світу розробки ПЗ. При цьому особливу увагу треба приділити Scrum-методу як найбільш популярному з існуючих гнучких методологій.

ЛІТЕРАТУРА

1. Маніфест гнучкої розробки програмного забезпечення [Електронний ресурс]: http://www.agilealliance.org.ru.
2. Колодін М.Ю. Гнучкі технології програмування [Електроннийресурс]: http://www.computer.edu.ru/myke/se/index.shtml.
3. Хаф Л. Методологія розробки програмного забезпечення: в 3-х ч. / Хаф Л. Ч.2: Екстремальне програмування [Електронний ресурс]: http://www.lib.csu.ru/dl/bases/prg/kompress/articles/2003_10_XP/index.htm.

Надійшла до редколегії 25.04.2017.