УДК 004.05

# ЕМЕРДЖЕНТНІ ВЛАСТИВОСТІ ЯК НАСЛІДКИ НЕДОСТАТНОСТІ ІНФОРМАЦІЇ У СПЕЦИФІКАЦІЇ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Т. О. Говорущенко**
*Хмельницький національний університет*

***Анотація.*** *Проведено дослідження впливу інформації у специфікації вимог на успішність реалізації програмного проекту та дослідження емерджентних властивостей як наслідків недостатності інформації у специфікації вимог до програмного забезпечення (ПЗ) і аналіз підходів до виявлення емерджентних властивостей, які показали необхідність розроблення методів та технологій виявлення, оцінювання та прогнозування емерджентних властивостей ПЗ.*

***Ключові слова:*** *програмне забезпечення (ПЗ), програмна система, програмний проект, специфікація вимог до програмного забезпечення, емерджентні властивості, достатність інформації.*

## Вступ

Природа програмного забезпечення (ПЗ) істотно змінилась в останні роки. Акцент змістився з виробництва автономних програмних продуктів на виробництво програмних систем як широких систем систем (system-of-systems), інтегрованихз великої кількості компонентів (підсистем) з інтерфейсами взаємодії між ними [1, 2]. Оскільки ПЗ стає дедалі складнішим та масштабнішим, розроблення саме програмних систем відіграватиме все більш важливу роль в діяльності розробника [3].

На рис. 1 представлено матрицю розміру/складності ПЗ, дані якої доводять, що чим складнішим є ПЗ та чим більший воно має розмір, тим вищим є ризик невдачі такого проекту [4, 5]. Цифри в матриці розміру/складності ПЗ означають індекс складності програмного проекту, введений компанією The Standish Group International для статистичних звітів CHAOS Report [4]. Цей індекс складності вимірює вплив факторів складності, властивих IT-проектам, за конкретного сценарію. Даний інструмент поєднує використання факторів складності, визначених за підходом International Project Management Association та використання факторів складності для управління складними IT-проектами [4].

Однією з найбільш вагомих причин незадовільної якості великих програмних проектів дослідники The Standish Group International називають збільшення кількості компонентів (підсистем) та інтерфейсів між ними, а також неконтрольовану складність програмної системи [4, 5].



| | | COMPLEXITY | | | | |
|---|---|---|---|---|---|---|
| | | C1 | C2 | C3 | C4 | C5 |
| SIZE | S1 | 100 | 250 | 400 | 550 | 700 |
| | S2 | 175 | 325 | 475 | 625 | 775 |
| | S3 | 250 | 400 | 550 | 700 | 850 |
| | S4 | 325 | 475 | 625 | 775 | 625 |
| | S5 | 400 | 550 | 700 | 850 | 1000 |

Рис. 1. Матриця розміру/складності ПЗ [4]

Дослідження The Standish Group International (CHAOS reports) [4, 5] доводять, що статистика успішності малих та великих програмних проектів суттєво різна (рис. 2).

CHAOS RESOLUTION BY PROJECT SIZE

| | SUCCESSFUL | CHALLENGED | FAILED |
|---|---|---|---|
| Grand | 2% | 7% | 17% |
| Large | 6% | 17% | 24% |
| Medium | 9% | 26% | 31% |
| Moderate | 21% | 32% | 17% |
| Small | 62% | 16% | 11% |
| TOTAL | 100% | 100% | 100% |

*The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.*

Рис. 2. Статистика успішності малих, помірних, середніх, великих та надвеликих програмних проектів у 2011-2015 роках [4]

Аналіз рис.2 дає можливість зробити висновок, що серед малих проектів 62% є успішними в той час, як серед великих проектів успішними є лише 6%, а серед надвеликих проектів успішними є лише 2%, тобто малі проекти в десятки разів (на порядок) успішніші за великі.

Такий висновок підтверджує й статистика проектів на основі використання функційних точок, як основних одиниць вимірювання розміру ПЗ [6-9] – рис. 3.

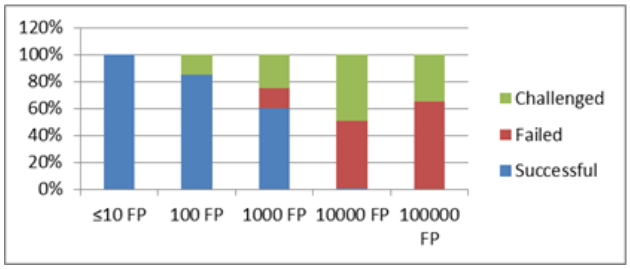


Рис. 3. Статистика успішності програмних проектів обсягом у 10, 100, 1000, 10000, 100000 функційних точок

Серед причин можливих невдач називають [10]: 1) нечітке й неповне формування та формулювання вимог до ПЗ; 2) недостатнє залучення користувачів до роботи над проектом; 3) незадовільне планування; 4) часту зміну вимог і специфікацій; 5) некоректне розуміння або недостатній аналіз проекту.

25-55% всіх помилок програмного проекту складають помилки формування та формулювання вимог і проектування архітектури, причому чим більший обсяг ПЗ, тим більше помилок вноситься саме на ранніх етапах [10]. Аналіз великої кількості програмних проектів, проведений у [11], підтвердив той факт, що головне місце виникнення помилок у ПЗ – це етап формування та формулювання вимог (специфікація), друге місце по внесенню помилок "посідає" етап проектування архітектури. У більшості випадків помилки вказують на проблеми специфікації та проектування архітектури, тобто фактично вже в кінці етапу проектування архітектури можна виявити та усунути левову частку всіх помилок ПЗ.

Є кілька причин, які роблять етап формування та формулювання вимог основним «постачальником» помилок: 1) відсутність специфікації; 2) неповнота або суперечливість інформації специфікації; 3) часті зміни специфікації.

В процесі формування та формулювання вимог можуть відбуватись інформаційні втрати через неповне та різне розуміння потреб та контексту інформації – особливо такі втрати суттєві для програмних проектів, які розробляються на стику предметних галузей (наприклад, ПЗ для медицини), коли враховувати потрібно як стандарти щодо розроблення ПЗ, так і стандарти предметної галузі, для якої розробляється ПЗ. Таку кількість стандартів важко імплементувати, а ще важче оцінити ступінь врахування рекомендацій цих стандартів.

Якщо цілі проекту, встановлені на ранніх етапах життєвого циклу, не відповідають потребам користувачів, то ПЗ неможливо визнати упішним, навіть якщо при його розробленні були використані сучасні технології та були задіяні найкваліфікованіші розробники. Отже, успішність реалізації програмного проекту суттєво залежать від специфікації вимог, а також від достатності наявної у ній інформації як наявності всіх інформаційних елементів (даних), визначених стандартом ISO/IEC/IEEE 29148:2011 [12]. Недостатність інформації у специфікації вимог до ПЗ знижує результативність та достовірність роботи над програмним проектом.

### 1. Аналіз аварій взаємодії компонентів програмних систем

Результатом виробництва ПЗ як системи систем з великої кількості компонентів з інтерфейсами взаємодії між ними є той факт, що інциденти та аварії, пов'язані із ПЗ, сьогодні містять новий тип аварії, яка викликана взаємодією компонентів: кожен компонент не містить помилок (задовольняє потреби), але некоректні взаємодії між компонентами призводять до проблем. Аварії взаємодії компонентів стають причинами великих аварій через зростаючу складність програмних систем, яка призводить до неможливості передбачення усіх можливих наслідків взаємодій між компонентами [13-16]. Приклади аварій взаємодії компонентів програмних систем наведено у [3, 13-17].

Просте підвищення якості та надійності окремих компонентів програмних систем не може завадити таким аваріям, тому що їх причиною не є відмови окремих компонентів. Висока якість та надійність компонентів не запобігає таким аваріям. Ці проблеми посилюються тим фактом, що на даний час широко використовуються такі методи інженерного аналізу надійності та безпеки, як FTA та FMEA, які були розроблені для виявлення відмов компонентів, і вони не можуть ефективно використовуватись для запобігання аварій взаємодії компонентів. Проблема, з якою стикаються інженери, полягає в тому, що вони не мають інших, більш підходящих, інструментів для визначення саме системних властивостей ПЗ, а також для прогнозування аварій взаємодії [13-16].

Успішність реалізації програмного проекту може бути низькою і через те, що недостатньо

уваги приділяється питанню відношення до інформації предметної галузі на різних етапах життєвого циклу ПЗ, її достатності, достовірності, уточнення. Деяка інформація аналізується занадто прискіпливо, а деяка – взагалі не враховується. Часто відкидається інформація предметної галузі з малою ймовірністю, а інколи й імовірність її взагалі не оцінюється. Нова інформація може надходити на різних етапах життєвого циклу – як на етапах формування та формулювання вимог і проектування архітектури, так і на етапах реалізації та експлуатації, але нею часто нехтують. Таке нехтування інформацією предметної галузі на всіх етапах життєвого циклу є одним з критичних факторів при розробленні програмного забезпечення [18, 19].

В процесі роботи над програмним проектом важливо оцінити частку інформаційної невизначеності проекту. Причиною виникнення інформаційної невизначеності проекту є низький рівень документування знань, особливо на системному рівні (рис. 4 [20]).

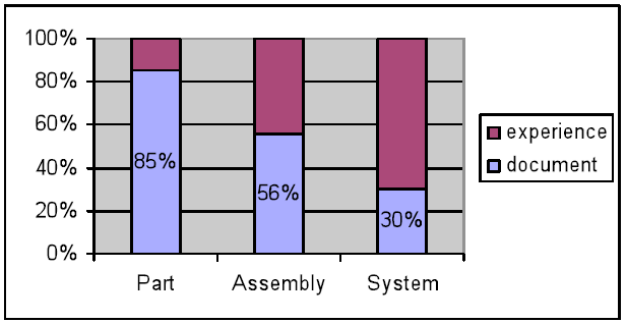


Рис. 4. Документування знань на системному рівні [20]

На рис. 5 зображено ситуацію, що характеризується передчасністю проектних рішень та їх документації до розуміння проектування. Показану область називають «розривом у знаннях» (knowledge gap), наявність якого є практичним результатом і першопричиною багатьох інженерних невдач [21].

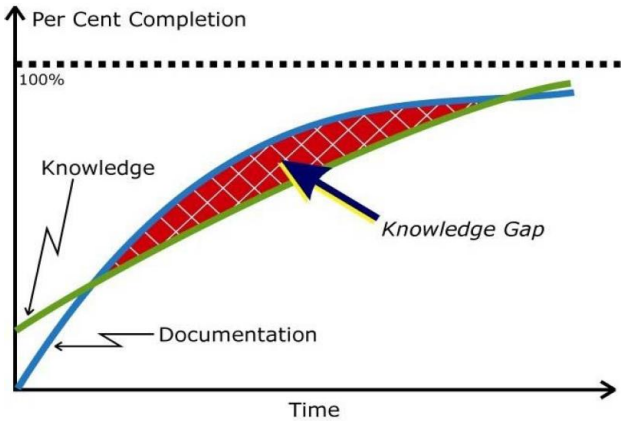


Рис. 5. Розрив у знаннях (Knowledge Gap) [21]

Факт часткового неврахування інформації предметної галузі на різних етапах життєвого циклу ПЗ свідчить про те, що розмір розриву у знаннях не є сталим для програмного проекту – в процесі життєвого циклу він може збільшуватись і зменшуватись, оскільки з'являється нова інформація, яку потрібно враховувати.

Для інженерії ПЗ представлена на рис. 5 точка зору на розрив у знаннях не зовсім відповідає реальності. Часткове врахування інформації предметної галузі протягом життєвого циклу і вплив її вже на готовий продукт призводять до збільшення розміру розриву у знаннях в процесі життєвого циклу, що може стати причиною збоїв та інших проблем з ПЗ. Для безпечного функціювання ПЗ розмір розриву у знаннях бажано зменшувати за рахунок більш повного врахування (зменшення втрат) інформації предметної галузі протягом життєвого циклу програмного проекту [3].

Враховуючи вищесказане, всі наявні знання та інформацію про програмну систему представимо у вигляді діаграми, в якій є сектор, що відображає об'єм недостатньої (невідомої) інформації (розрив у знаннях) – рис. 6.

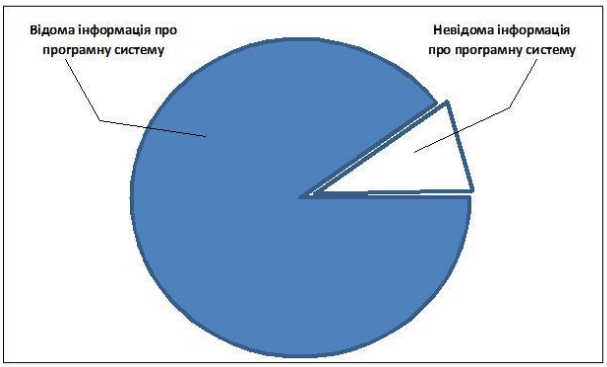


Рис. 6. Поле знань про програмну систему із сектором невідомої інформації

Даний сектор складає неврахована інформація предметної галузі (в тому числі інформація, відсутня у специфікації вимог до ПЗ). Розміри сектору з невідомою інформацією не визначені, оскільки незрозуміло, яка і скільки інформації залишається невідомою. Сектор з невідомою інформацією слід звужувати – за рахунок більш повного врахування інформації предметної галузі, починаючи з ранніх етапів життєвого циклу. Чим меншим буде розмір сектору невідомої інформації, тим якіснішою буде програмна система, тим безпечніше вона працюватиме та тим успішнішою буде її реалізація. Отже, актуальним є підхід до зменшення частки невідомої інформації про програмну систему.

## 2. Емерджентні властивості як наслідки недостатності інформації у специфікації вимог до програмного забезпечення

Часто складним є визначення інформації, яка з'явиться в процесі взаємодії «підсистем – інтерфейсів – даних – зовнішніх впливів», а також майбутніх характеристик розроблюваного ПЗ, через які ця інформація може проявитись.

Відомі методи проектного менеджменту ПЗ слабко враховують системний аспект сучасного ПЗ, а саме недостатньо уваги приділяють емерджентності системи, хоча системний аналіз доводить, що наявність емерджентних властивостей є однією з найважливіших рис системи.

Емерджентність – це нова властивість, яка з'являється в результаті об'єднання елементів у систему. Емерджентність системи породжується її нелінійністю, новими елементами системи або новими зв'язками. Причинами виникнення емерджентних властивостей є: різкий стрибок властивості (присутньої, але мало помітної раніше); біфуркація (втрата стабільності, момент нестійкості, точка, в якій відбувається катастрофа) підсистеми; рекомбінація зв'язків [22]. Такі стрибкоподібні переходи та якісні зміни досліджує теорія катастроф, яка вивчає та прогнозує нестійкість різноманітних систем, а також є корисною при аналізі великих масивів інформації [23]. Чим вища інтенсивність взаємодії елементів системи, чим більше відрізняються властивості системи від властивостей її елементів, тим вищим є системний ефект. Чим вищий рівень системності, тим більшими є переваги у досягненні цілей – здатність системи до вибору прямо пропорційна ступеню її емерджентності [24].

Термін «емерджентні властивості» зустрічається у ряді робіт, присвячених ПЗ, але трактується різними авторами дещо по-різному. Результати аналізу відомих визначень поняття «емерджентні властивості» наведено у роботі [3] та систематизовано у таблиці 1.

Таблиця 1

Аналіз відомих визначень поняття «емерджентні властивості»

| Визначення поняття «емерджентні властивості» | Автор(и) |
|---|---|
| 1 | 2 |
| Емерджентна властивість – це будь-яка характеристика системи, яка не може бути локалізована в одному незалежно діючому компоненті або в невеликій постійній кількості компонентів | Д. Фішер (Fisher) [25], SWEBOK' 2014 [26], О. Дорофі (Dorofee) та інші [1] |
| Властивості цілого, які є результатом взаємодії частин або збирання частин в єдине ціле, є емерджентними властивостями. Емерджентні властивості випливають з комбінації складових частин систем | Ф. Патерсон (Patterson) [21], С. Кайслер (Kaisler) та Г. Мейді (Madey) [27], Дж. Марш (Marsh) [28], К. Джонсон (Johnson) [29], Дж. Хсю (Hsu) та М. Батерфілд (Butterfield) [30] |
| Емерджентні властивості є розбіжністю між передбачуваною та реалізованою конструкцією. Ці властивості не можуть бути відомі апріорі | Я. Елісон (Alison) та Я. Мералі (Merali) [31], К. Цабо (Szabo) та Й. Тео (Teo) [32], П. Кашфі (Kashfi) та інші [33], Дж. Могул (Mogul) [34], Дж. Дайсон (Dyson) [35], К. Руф (Rouff) [36] |

Ряд наведених визначень трактує емерджентні властивості як властивості, які не можуть бути локалізовані в одному компоненті, а є результатом взаємодії компонентів. Деякі визначення вказують на труднощі передбачення емерджентних властивостей або розуміють під цим терміном різницю між передбачуваним і реалізованим дизайном програмної системи, не розкриваючи природи виникнення емерджентних властивостей та їх суті.

*Емерджентними властивостями* вважатимемо такі властивості програмної системи, які з'являються випадковим чином (випали з фокусу уваги) та проявляються в процесі функціонування ПЗ при взаємодії підсистем через інтерфейси і за наявності певних даних та зовнішніх впливів – рис. 7. *Емерджентною поведінкою* вважатимемо непередбачувану розробниками поведінку програмної системи, яка не властива жодній підсистемі окремо, а виникає лише в процесі взаємодії підсистем.

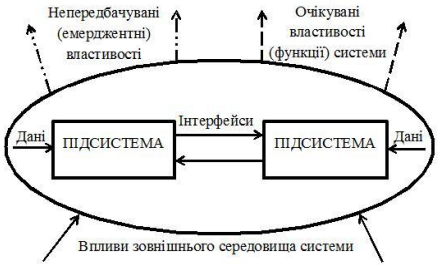


Рис. 7. Природа емерджентних властивостей

Емерджентні властивості можуть як позитивно, так і негативно впливати на якість програмних систем та успішність їх реалізації. Емерджентні властивості ПЗ входять до складу розриву у знаннях.

Деякі з емерджентних властивостей програмних систем відображаються нефункціональними вимогами, оскільки саме нефункціональні вимоги містять деякий ступінь внутрішньої невизначеності (ступінь суб'єктивності) [37], формулюються на системному рівні [37-39] і частково відображають емерджентні властивості програмних систем.

На сьогодні є ряд робіт та підходів, які свідчать про спроби виявлення емерджентних властивостей програмних систем: хронологічний підхід в рамках Software Process Improvement (SPI) [31]; порівняння програмних систем з природніми групами з емерджентною поведінкою – наприклад, з групою перелітних птахів або мурашиними колоніями [27, 28, 30, 36, 40]; підходи та методи, що запозичені з біологічних, фізичних та соціальних систем [25]; використання мультиагентних формальних методів WSCCS, X-Machines [36]; модель UX (UX – user experience) для вимог до ПЗ [33]; використання технології аспект-орієнто-ваного програмування [41]; семантична валідація емерджентних властивостей в імітаційних моделях, заснованих на компонентах [32]; використання сервіс-орієнтованих архітектур (SOA) [25, 34]; на базі інформаційної моделі мовою SysML для визначення вимог та їх трасованості (оперативного контролю) [37]; застосування теорії ігор в якості інваріантного підходу для прогнозування емерджентної поведінки [34]; програма досліджень емерджентної поведінки у складних програмних системах – робота зі спостереженнями системної поведінки та обмірковування їх в зворотному напрямку (дедукція) для виведення бачення, що насправді сталось, якщо виникне непередбачувана поведінка [34]; враховуючи той факт, що емерджентні властивості проявляються лише на системному рівні, то для їх виявлення можна використовувати інтеграційне та системне тестування; з іншого боку, частковим випадком емерджентних властивостей є нефункціональні вимоги, тоді для їх виявлення підійде нефункціональне тестування, зокрема, тестування захищеності, сумісності, надійності та продуктивності (в т.ч. стабільності, навантажувальне, стресове).

Однак, за наявності значної кількості підходів, практичний підхід до ідентифікації та валідації емерджентних властивостей залишається проблемою. Всі наявні дослідження пояснюють природу емерджентності та емерджентних властивостей, але вони не формалізовані і не можуть бути використані в явному вигляді для моделювання і прогнозування емерджентних властивостей ПЗ. Всі розглянуті підходи до виявлення емерджентних властивостей залишаються теоретичними. Відсутня статистика про те, для яких проектів застосовувався той чи інший підхід і який ефект він дав (як завершились ці проекти).

Отже, наразі відсутні підходи, які б дозволили прогнозувати чи передбачати появу емерджентних властивостей ПЗ, тобто на даний час виникнення та прояв емерджентних властивостей є випадковим та непередбачуваним і для розробників, і для користувачів ПЗ. Інженери не мають підходящих інструментів для визначення та прогнозування системних властивостей ПЗ (в тому числі й емерджентних властивостей) та аварій взаємодії. Емерджентні властивості не розглядаються у сучасних підходах до розроблення та оцінювання ПЗ, тому не розроблено дієвих механізмів виявлення проблем інтеграції, які обумовлюють великий відсоток неякісних (провальних та проблемних) програмних проектів [42-44].

Причини ігнорування емерджентних властивостей носять не технологічний, а методологічний характер. Емерджентні властивості бажано виявляти та усувати на ранніх етапах життєвого циклу ПЗ, що було б шляхом до підвищення якості та успішності реалізації програмних систем.

На сьогодні відсутній практичний підхід до ідентифікації емерджентних властивостей ПЗ, але зрозуміло, що увага до вимог на ранніх етапах життєвого циклу ПЗ може бути цінною для прогнозування та виявлення емерджентних властивостей і забезпечить можливість підвищення якості та успішності ПЗ.

**Висновки**

Проведене дослідження впливу інформації у специфікації вимог на успішність реалізації програмних проектів показало, що чиники якості та успішності сучасних програмних систем є менш залежними від написання програмного коду, але суттєво залежать від формування та формулювання вимог і проектування архітектури. Ризиками недостатньо відпрацьованого етапу формування та формулювання вимог є недотримання термінів проектів та фінансові перевитрати, що можуть призвести до закриття проекту, а то й розпаду софтверної компанії внаслідок її фінансової нестабільності. За таких умов актуальним і дуже важливим є аналіз специфікації вимог до ПЗ, можливість «відсікти» програмні проекти з неповною (з недостатньою інформацією) специфікацією.

При розробленні програмних проектів існує розрив у знаннях про характеристики майбутнього ПЗ. Цей розрив з'являється через відсутність знань про те, яка інформація з'явиться в процесі взаємодії «підсистем – інтерфейсів – даних – зовнішніх впливів» і якими характеристиками ПЗ вона проявиться, а також через часткове врахування інформації предметної галузі протягом життєвого циклу ПЗ. Розмір розриву у знаннях не є сталим для програмного проекту. Імові-

рність того, що в процесі життєвого циклу проекту розрив у знаннях зникне, є низькою, а при появі нової інформації предметної галузі може відбутись збільшення розміру розриву у знаннях. Для успішної реалізації та подальшого безпечного функціонування ПЗ, розмір розриву у знаннях бажано зменшувати, враховуючи якнайбільше інформації предметної галузі протягом життєвого циклу ПЗ. Тому потрібні принципово нові підходи, пов'язані із врахуванням інформації предметної галузі в процесі життєвого циклу ПЗ.

Значна частина проблем якості та успішності реалізації ПЗ зумовлена також недостатньою увагою до його системного аспекту, зокрема, до емерджентних властивостей, які є наслідками недостатності інформації у специфікації вимог до ПЗ. Основним джерелом емерджентних властивостей є взаємодія «підсистем-інтерфейсів-даних-зовнішніх впливів». Емерджентні властивості ПЗ входять до складу розриву у знаннях.

На сьогодні відсутній ефективний практичний підхід до ідентифікації емерджентних властивостей ПЗ, тобто на даний час виникнення та прояв емерджентних властивостей є випадковим та непередбачуваним і для розробників, і для користувачів ПЗ, але саме зменшення кількості емерджентних властивостей ПЗ забезпечить можливість підвищити його якість та успішність реалізації. Найбільш перспективним з усіх розглянутих підходів до виявлення емерджентних властивостей ПЗ є програма досліджень емерджентної поведінки у складних програмних системах автора Дж. Могул (Mogul) [34], але взагалі потрібні принципово нові підходи, пов'язані із врахуванням інформації предметної галузі протягом життєвого циклу ПЗ.

Перспективою для подальших досліджень автора є розроблення методів та технологій виявлення, оцінювання та прогнозування емерджентних властивостей ПЗ, які базуватимуться на врахуванні інформації предметної галузі в процесі життєвого циклу ПЗ.

**Список використаної літератури**

1. Dorofee, A. A Systemic approach for assessing software supply-chain risk [Text] / A. Dorofee, C. Woody, C. Alberts, R. Creel, R. J. Ellison // The 44-th Hawaii International Conference on System Sciences, January 04-07, 2011: Proceedings. – Honolulu (USA), 2011. – Pp. 1–8.

2. Ghadge, A. A systems approach for modelling supply chain risks [Text] / A. Ghadge, S. Dani, M. Chester, R. Kalawsky // Supply Chain Management: An International Journal – 2013. – Vol. 18 (5). – Pp. 523–538.

3. Pomorova, O. The way to detection of software emergent properties [Text] / O. Pomorova, T. Hovorushchenko // The 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, September 24-26, 2015: Proceedings. – Vol. 2. – Warsaw, 2015. – Pp. 779–784.

4. Hastie, S. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch [Electronic resource] / S. Hastie, S. Wojewoda. – Access mode: http://www.infoq.com/articles/standish-cha.os-2015

5. A Look at 25 Years of Software Projects. What Can We Learn? [Electronic resource]. – Access mode: https://speedandfunction.com/look-25-years-software-projects-can-learn/.

6. Maedche, A. Software for people: fundamentals, trends and best practices (Management for professionals) [Text] / A. Maedche, A. Botzenhardt, L. Neer. – Berlin: Springer-Verlag Berlin Heidelberg, 2012. – 293 p.

7. Futrell, R. T. Quality Software Project Management [Text] / R. T. Futrell, D. F. Shafer, L. I. Shafer. – Hoboken (USA): Prentice Hall, 2002. – 1680 p.

8. Sommerville, I. Software Engineering [Text] / I. Sommerville. – London: Pearson, 2015. – 816 p.

9. Вендров, А. М. Проектирование программного обеспечения экономических информационных систем: учебник [Текст] / А. М. Вендров. – Москва: Финансы и статистика, 2006. – 544 с.

10. McConnell, S. Code complete [Text] / S. McConnell. – Microsoft Press, 2013. – 896 p.

11. Jones, C. The economics of software quality [Text] / C.Jones, O.Bonsignour. – Boston: Pearson Education, 2012. – 588 p.

12. Systems and software engineering. Life cycle processes. Requirements engineering: ISO/IEC/IEEE 29148:2011. – [Introduced 01.12.2011]. – Geneva (Switzerland): ISO, 2011. – 28 p. – (International standard).

13. Levenson, N. G. Engineering a safer world: systems thinking applied to safety [Text] / N. G. Levenson. – MIT Press, 2012. – 560 pp.

14. Levenson, N. G. Systemic factors in software-related spacecraft accidents [Text] / N. G. Levenson // AIAA Space 2001 Conference and Exposition, August 28-30, 2001: Proceedings. – Albuquerque (USA), 2001. – Pp. 1–11.

15. Levenson, N. Software challenges in achieving space safety [Text] / N. Levenson // Journal of the British Interplanetary Society. – 2009. – Vol. 62. – Pp. 265–272.

16. Ishimatsu, T. Hazard analysis of complex spacecraft using systems-theoretic process analysis [Text] / T. Ishimatsu, N. G. Levenson, J. P. Thomas, C. H. Fleming, M. Katahira, Yu. Miyamoto, R. Ujiie, H. Nakao, N. Hoshino // Journal of Spacecraft and Rockets. – 2014. – Vol. 51. – No. 2. – Pp. 509–522.

17. Hovorushchenko, T. The software emergent properties and them reflection in the non-functional

requirements and quality models [Text] / T. Hovorushchenko // 10-th International Conference on Computer Science and Information Technologies, September 14-17, 2015: Proceedings. – Lviv (Ukraine), 2015. – Pp. 146–153.

18. Munch, J. Perspectives on the future of software engineering [Text] / J. Munch, K. Schmid. – Berlin: Springer-Verlag Berlin Heidelberg, 2013. – 365 p.

19. Diaz, V. G. Handbook of research on innovations in systems and software engineering [Text] / V. G. Diaz, J. M. Lovelle, B. C. Garcia-Bustelo. – Hershey (USA): IGI Global, 2015. – 745 p.

20. Maier, R. Knowledge management systems. Information and communication technologies for knowledge management [Text] / R. Maier. – Berlin: Springer Science & Business Media, 2013. – 635 p.

21. Patterson, Jr. F. G. Life cycles for system acquisition [Text] / Jr. F. G. Patterson // Encyclopedia of Life Support Systems, Systems Engineering and Management for Sustainable Development. – Paris, 2004. – Pp. 82–110.

22. Попов, В. П. Глобальный эволюционизм и синергетика ноосферы [Текст] / В. П. Попов, И. В. Крайнюченко. – Ростов-на-Дону: Издательство АПСН СКНЦ ВШ, 2003. – 333 с.

23. Hubler, A. System under larg stress: prediction and management of catastrophic failures [Text] / A. Hubler // Complexity. – 2016. – Vol. 21. – Issue 3. – Pp. 9–12.

24. Луценко, Е. В. Существование, несуществование и изменение как эмерджентные свойства систем [Текст] / Е. В. Луценко // Квантовая магия. – 2008. – Т. 5 – Выпуск 1. – С. 1215–1239.

25. Fisher, D. A. An emergent perspective on interoperation in systems of systems [Text] / David A. Fisher // TECHNICAL REPORT CMU/SEI-2006-TR-003, 2006. – 67 p.

26. Software Engineering. Guide to the software engineering body of knowledge (SWEBOK): ISO/IEC TR 19759:2015. – [Introduced 01.10.2015]. – Geneva (Switzerland): ISO, 2015. – 336 p. – (International standard).

27. Kaisler, S. Complex adaptive systems: emergence and self-organization [Electronic resource] / S. Kaisler, G. Madey. – Access mode: http://www3.nd.edu/~gmadey/Activities/CAS-Briefing.pdf.

28. Marsh, G. E. The demystification of emergent behavior [Electronic resource] / G. E. Marsh. – Access mode: https://arxiv.org/ftp/arxiv/papers/0907/0907.1117.pdf.

29. Johnson, Ch. W. What are emergent properties and how do they affect the engineering of complex systems? [Text] / Christopher W. Johnson // Complexity in Design and Engineering. – 2006. – Vol. 91. – Issue 12. – Pp. 1475–1481.

30. Hsu, J. C. Modeling emergent behavior for systems-of-systems [Text] / John C. Hsu, M. Butterfield // INCOSE International Symposium, June 24-28, 2007: Proceedings. – Vol. 17. – Issue 1. – San Diego (USA), 2007. – Pp. 1811–1821.

31. Allison, I. Software process improvement as emergent change: a structurational analysis [Text] / I. Allison, Y. Merali // Journal Information and Software Technology. – 2007. – Vol. 49. – Issue 6. – Pp. 668–681.

32. Szabo, C. Semantic validation of emergent properties in component-based simulation models [Text] / C. Szabo, Y. M. Teo // Ontology, Epistemology, & Teleology for Modelling & Simulation : Philosophical Foundations for Intelligent M&S Applications. – Berlin (Germany), 2013. – Pp. 319–333.

33. Kashfi, P. Models for integrating UX into software engineering practice: an industrial validation [Text] / P. Kashfi, R. Feldt, A. Nilsson, R. Berntsson Svensson // The 6th International Working Conference on Human-Centred Software Engineering, September 16-18, 2014 : Proceedings. – Paderborn (Germany), 2014. – Pp. 259–266.

34. Mogul, J. C. Emergent (mis)behavior vs. complex software systems [Text] / J. C. Mogul //ACM SIGOPS Operating Systems Review. – 2006. – Vol. 40. – № 4. – Pp. 293–304.

35. Dyson, G. B. Darwin among the machines: the evolution of global intelligence [Text] / G. B. Dyson. – New York: Basic Books, 2012. – 304 p.

36. Rouff, C. Properties of a formal method for prediction of emergent behaviours in swarm-based systems [Text] / C. Rouff, M. Hinchey, W. Truszkowski, A. Vanderbilt, J. Rash // The 2-nd International Conference on Software Engineering and Formal Methods, September 28-30, 2004: Proceedings. – Beijing (China), 2004. – Pp. 24–33.

37. Guillerm, R. Safety evaluation and management of complex systems: A system engineering approach [Text] / R. Guillerm, H. Demmou, N. Sadou // Concurrent Engineering: Research and Applications. – 2012. – Vol. 20 (2). – Pp. 149–159.

38. Larsson, M. Predicting quality attributes in component-based software systems [Text] / M. Larsson. – Västerås (Sweden), 2004. – 218 p.

39. Pizzi, N. J. Mapping software metrics to module complexity: a pattern classification approach [Text] / N. J. Pizzi // Journal of Software Engineering and Applications. – 2011. – Vol. 4. – Pp. 426–432.

40. Johnson, St. Essay emergence: the connected lives of ants, brains, cities, and software emergence / St. Johnson. – New York: Scribner, 2002. – 288 p.

41. Alexander, R. T. Challenges of aspect-oriented technology [Text] / Roger T. Alexander, James M. Bieman // Workshop on Software Quality of the 34-th International Conference on Software

Engineering, May 25, 2002: Proccedings. – Orlando, Florida (USA), 2002. – Pp. 1–3.

42. Niebuhr, D. Towards a developement approach for dynamic-integrative systems [Text] / D. Niebuhr, C. Peper, A. Rausch // Workshop on Building Software for Pervasive Computing of the 19-th Conference on Object-Oriented Programming Systems, Languages and Applications, October 25, 2004: Proceedings. – Vancouver (Canada), 2004. – Pp. 134–139.

43. Niebuhr, D. Towards reliable self-integrative IT systems [Text] / D. Niebuhr, C. Peper, A. Rausch // The 10-th International Workshop on Component-Oriented Programming, July 25, 2005: Proceedings. – Glasgow (Scotland), 2005. – Pp. 98–102.

44. Vyatkin, V. Software engineering in industrial automation: State of the art Review [Text] / V. Vyatkin. // IEEE Transactions on Industrial Informatics. – 2013. – Vol. 9. – No. 3. – Pp. 1234–1249.

**References**

1. Dorofee, A., Woody, C., Alberts, C., Creel, R. and Ellison, R. J. (2011), "A Systemic approach for assessing software supply-*chain risk", Proceedings of The 44-th Hawaii International Conference on System Sciences, January 04-07, 2011*, Honolulu (USA), 2011, pp. 1–8.

2. Ghadge, A., Dani, S., Chester, M. and Kalawsky R. (2013), "A systems approach for modelling supply chain risks", *Supply Chain Management: An International Journal*, Vol. 18 (5), pp. 523–538.

3. Pomorova, O. and Hovorushchenko, T. (2015), "The way to detection of software emergent properties", *Proceedings of The 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, September 24-26, 2015*, Vol. 2, Warsaw, pp. 779–784.

4. Hastie, S. and Wojewoda S. (2015), "Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch", available at: http://www.infoq.com/articles/standish-chaos-2015

5. Speed&Function (2017), "A Look at 25 Years of Software Projects. What Can We Learn?", available at: https://speedandfunction.com/look-25-years-software-projects-can-learn/

6. Maedche, A., Botzenhardt, A., and Neer, L. (2012), *Software for people: fundamentals, trends and best practices (Management for professionals)*, Springer-Verlag Berlin Heidelberg, Berlin.

7. Futrell, R. T., Shafer, D. F., Shafer, L. I. (2002), *Quality Software Project Management*, Prentice Hall, Hoboken (USA).

8. Sommerville, I. (2015), *Software Engineering*, Pearson, London.

9. Vendrov, A. M. (2006), *Designing the Software for Economic Information Systems: Tutorial [Proektirovanie programmnogo obespecheniya ekonomicheskih informatsionnih sistem]*, Finance and Statistics, Moscow.

10. McConnell, S. (2013), *Code complete*, Microsoft Press, Redmond.

11. Jones, C. and Bonsignour, O. (2012), *The economics of software quality*, Pearson Education, Boston.

12. ISO/IEC/IEEE 29148:2011, Systems and software engineering. Life cycle processes. Requirements engineering.

13. Levenson, N. G. (2012), *Engineering a safer world: systems thinking applied to safety*, MIT Press, Massachusets.

14. Levenson, N. G. (2001), "Systemic factors in software-related spacecraft accidents", *Proceedings of AIAA Space 2001 Conference and Exposition, August 28-30, 2001*. Albuquerque (USA), pp. 1–11.

15. Levenson, N. (2009), "Software challenges in achieving space safety", *Journal of the British Interplanetary Society*, Vol. 62, pp. 265–272.

16. Ishimatsu, T., Levenson, N. G., Thomas, J. P., Fleming, C. H., Katahira, M., Miyamoto, Yu., Ujiie, R., Nakao, H. and Hoshino, N. (2014), "Hazard analysis of complex spacecraft using systems-theoretic process analysis", *Journal of Spacecraft and Rockets*, Vol. 51 No. 2, pp. 509–522.

17. Hovorushchenko, T. (2015), "The software emergent properties and them reflection in the non-functional requirements and quality *models", Proceedings of 10-th International Conference on Computer Science and Information Technologies, September 14-17, 2015*, Lviv (Ukraine), pp. 146–153.

18. Munch, J. and Schmid, K. (2013), *Perspectives on the future of software engineering*, Springer-Verlag Berlin Heidelberg, Berlin.

19. Diaz, V. G., Lovelle, J. M. and Garcia-Bustelo B. C. (2015), *Handbook of research on innovations in systems and software engineering*, IGI Global,Hershey (USA).

20. Maier, R. (2013), *Knowledge management systems. Information and communication technologies for knowledge management*, Springer Science & Business Media, Berlin.

21. Patterson, Jr. F. G. (2004), "Life cycles for system acquisition", *Encyclopedia of Life Support Systems, Systems Engineering and Management for Sustainable Development*, pp. 82–110.

22. Popov, V. P and Kraynuchenko, I. V. (2003), *Global Evolutionism and Synergy of the Noosphere [Globalnyi evolutsionism i sinergetika noosfery]*, Publishing House APMS NCSCHS, Rostov-na-Donu (Russia).

23. Hubler, A. (2016), "System under large stress: prediction and management of catastrophic failures", *Complexity*, Vol. 21, Issue 3, pp. 9–12.

24. Lutsenko, E. V. (2008), "Existence, non-existence and change as the emergent properties of

systems" [Sushchestvovanie, nesushchestvovanie i izmenenie kak emerdjentnye svoistva sistem], *Quantum magic*, Vol. 5, Issue 1, pp. 1215–1239.

25. Fisher, D. A. (2006), "An emergent perspective on interoperation in systems of systems", *TECHNICAL REPORT CMU/SEI-2006-TR-003*, 67 p.

26. ISO/IEC TR 19759:2015, Software Engineering. Guide to the software engineering body of knowledge (SWEBOK).

27. Kaisler, S. and Madey, G. (2009), "Complex adaptive systems: emergence and self-organization", available at: http://www3.nd.edu/~gmadey/Activities/CAS-Briefing.pdf.

28. Marsh, G. E. (2009), "The demystification of emergent behavior", available at: https://arxiv.org/ftp/arxiv/papers/0907/0907.1117.pdf.

29. Johnson, Ch. W. (2006), "What are emergent properties and how do they affect the engineering of complex systems?" *Complexity in Design and Engineering*, Vol. 91, Issue 12, pp. 1475–1481.

30. Hsu, J. C. and Butterfield, M. (2007), "Modeling emergent behavior for systems-of-systems", *Proceedings of INCOSE International Symposium, June 24-28, 2007*, Vol. 17, Issue 1. San Diego (USA), pp. 1811–1821.

31. Allison, I. and Merali, Y. (2007), "Software process improvement as emergent change: a structurational analysis", *Journal Information and Software Technology*, Vol. 49, Issue 6, pp. 668–681.

32. Szabo, C. and Teo, Y. M. (2013), "Semantic validation of emergent properties in component-based simulation models", *Ontology, Epistemology, & Teleology for Modelling & Simulation : Philosophical Foundations for Intelligent M&S Applications (edited by Tolk A.)*, pp. 319–333.

33. Kashfi, P., Feldt, R., Nilsson, A. and Berntsson Svensson, R. (2014), "Models for integrating UX into software engineering practice: an industrial validation", *Proceedings of The 6th International Working Conference on Human-Centred Software Engineering, September 16-18, 2014*, Paderborn (Germany), pp. 259–266.

34. Mogul, J. C. (2006), "Emergent (mis)behavior vs. complex software systems", *ACM SIGOPS Operating Systems Review*, Vol. 40, No. 4, pp. 293–304.

35. Dyson, G. B. (2012), *Darwin among the machines: the evolution of global intelligence*, Basic Books, New York.

36. Rouff, C., Hinchey, M., Truszkowski, W., Vanderbilt, A. and Rash, J. (2004), "Properties of a formal method for prediction of emergent behaviours in swarm-based systems", *Proceedings of The 2-nd International Conference on Software Engineering and Formal Methods, September 28-30, 2004*, Beijing (China), pp. 24–33.

37. Guillerm, R. Demmou, H. and Sadou, N. (2012), "Safety evaluation and management of complex systems: A system engineering approach", *Concurrent Engineering: Research and Applications*, Vol. 20 (2), pp. 149–159.

38. Larsson, M. (2004), *Predicting quality attributes in component-based software systems*, Västerås.

39. Pizzi, N. J. (2011), "Mapping software metrics to module complexity: a pattern classification approach", *Journal of Software Engineering and Applications*, Vol. 4, pp. 426–432.

40. Johnson, St. (2002), *Essay emergence: the connected lives of ants, brains, cities, and software emergence*, Scribner, New York.

41. Alexander, R. T. and Bieman J. M. (2002), "Challenges of aspect-oriented technology", *Proceedings of Workshop on Software Quality of the 34-th International Conference on Software Engineering, May 25, 2002*, Orlando, Florida (USA), pp. 1–3.

42. Niebuhr, D. Peper, C. and Rausch, A. (2004), "Towards a developement approach for dynamic-integrative systems", *Proceedings of Workshop on Building Software for Pervasive Computing of the 19-th Conference on Object-Oriented Programming Systems, Languages and Applications, October 25, 2004*, Vancouver (Canada), pp. 134–139.

43. Niebuhr, D., Peper, C. and Rausch, A. (2005), "Towards reliable self-integrative IT systems", *Proceedings of The 10-th International Workshop on Component-Oriented Programming, July 25, 2005*, Glasgow (Scotland), pp. 98–102.

44. Vyatkin, V. (2013), "Software engineering in industrial automation: State of the art Review", *IEEE Transactions on Industrial Informatics*, Vol. 9, No. 3, pp. 1234–1249.

## EMERGENT PROPERTIES AS CONSEQUENCES OF INSUFFICIENCY OF INFORMATION IN THE SOFTWARE REQUIREMENTS SPECIFICATION

**T. O. Hovorushchenko**
*Khmelnytskyi National University*

***Abstract.*** *The conducted studies of the influence of information in the software requirements specification (SRS) on the success of the software project implementation and the analysis of failures of interaction between the components of the software systems have shown that factors of the quality and*

*success of modern software systems are less dependent on the coding, but essentially depend on the formation and formulation of requirements and on the software architecture design. Under such circumstances, it is actual and very important to analyze the SRS, to "cut" software projects with incomplete (with insufficient information) specification. The research of the emergent properties as a consequence of insufficiency of information in the SRS showed that the emergent properties are such properties of the software system, which appear randomly and manifest in the process of functioning of the software in the interaction of subsystems through interfaces and in the presence of certain data and external influences. Emergent properties of software are part of its knowledge gap. Nowadays, the emergence and manifestation of emergent properties are random and unpredictable both for software developers and for software users. But exactly the reduction of the number of the emergent properties of the software will provide the increase of its quality and success of implementation. The analysis of the approaches to detecting the emergent properties has shown the lack of the effective practical approach to identifying the software's emergent properties and the need to develop methods and technologies for detecting, evaluating and predicting the software's emergent properties. Therefore, the prospect for further research of the author is the development of the methods and technologies for detecting, evaluating and predicting the software's emergent properties, which will be based on the consideration of the subject domain information during the software life cycle.*

***Keywords:*** *software, software system, software project, software requirements specification (SRS), emergent properties, sufficiency of information.*

## ЭМЕРДЖЕНТНЫЕ СВОЙСТВА КАК СЛЕДСТВИЯ НЕДОСТАТОЧНОСТИ ИНФОРМАЦИИ В СПЕЦИФИКАЦИИ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

**Т. А. Говорущенко**
*Хмельницкий национальный университет*

***Аннотация.*** *Проведено исследование влияния информации в спецификации требований на успешность реализации программного проекта, анализ аварий взаимодействия компонентов программных систем, а также исследование эмерджентных свойств как следствий недостаточности информации в спецификации требований к программному обеспечению (ПО) и анализ подходов к идентифкациии эмерджентных свойств, которые показали необходимость разработки методов и технологий идентификации, оценки и прогнозирования эмерджентных свойств ПО.*

***Ключевые слова:*** *программное обеспечение (ПО), программный проект, успешность ПО, начальные этапы жизненного цикла ПО, информационная технология, онтология, интеллектуальный агент.*

Отримано 23.10.2018

**Говорущенко Тетяна Олександрівна**, доктор технічних наук, старший науковий співробітник, доцент, завідувач кафедри комп'ютерної інженерії та системного програмування Хмельницького національного університету. Вул. Інститутська, 11, Хмельницький, Україна, E-mail: tat_yana@ukr.net, t.hovorushchenko@khnu.km.ua, тел. +38-095-11-22-544

**Tetiana Hovorushchenko**, Full Doctor, Senior Researcher, Associate Professor, Head of Computer Engineering and System Programming Department, Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, Ukraine, E-mail: tat_yana@ukr.net, t.hovorushchenko@khnu.km.ua, тел. +38-095-11-22-544

**ORCID ID:** 0000-0002-7942-1857