DOI: https://doi.org/10.32836/2521-6643-2018.2-56.2
УДК 004.519.217

С. О. Яремчук, кандидат технічних наук,
доцент кафедри судових енергетичних
установок і систем Дунайського інституту
Національного університету
"Одеська морська академія"

МЕТРИКИ НАДІЙНОСТІ НА ОСНОВІ ЗАЛЕЖНОСТЕЙ ДЕФЕКТНОСТІ ПРОГРАМНОГО КОДУ ВІД ЙОГО СКЛАДНОСТІ

У роботі виконано аналіз недоліків небагатьох наявних метрик надійності програмних систем та визначено необхідність розробки нових метрик надійності на основі оцінювання складності вихідного коду. На основі відомих метрик складності об'єкто-орієнтованого коду розроблено єдину комплексну оцінку складності. З її використанням запропоновано метрики надійності вихідного коду: співвідношення між дефектними та бездефектними модулями, імовірність виявлення дефектів у модулях, модульна щільність дефектів, локалізація та розподіл дефектів у коді. Запропоновано визначати розроблені метрики надійності для раніше розроблених і верифікованих систем або їх частин, для яких відомі метричні оцінки складності та кількість дефектів. Запропоновані метрики надійності можуть бути використані для планування ресурсів та виконання ефективної верифікації новоствореного коду нових ітерацій, частин, версій, функцій або нових систем конкретного розробника. Метрики надійності можуть бути розраховані для будь-яких систем, подібних до новорозробленої системи за функціональністю, оцінками складності, кваліфікацією розробників, рівнем процесів та методологією розробки.

Ключові слова: *програмна система; програмне забезпечення; надійність; дефект; відмова; вихідний код; складність; метрика.*

В работе выполнен анализ недостатков немногих существующих метрик надежности программных систем и установлена необходимость разработки новых метрик надежности на основе оценки сложности

исходного кода. На основе известных метрик сложности объектно-ориентированного кода разработана единая комплексная оценка сложности. С ее использованием предложены метрики надежности исходного кода: соотношение между дефектными и бездефектными модулями, вероятность обнаружения дефектов в модулях, модульная плотность дефектов, локализация и распределение дефектов в коде. Предложено рассчитывать метрики надежности для ранее разработанных и верифицированных систем или их частей, для которых известны метрические оценки сложности и количество дефектов. Предложенные метрики надежности могут быть использованы для планирования ресурсов и выполнения эффективной верификации вновь разработанного кода новых итераций, частей, версий, функций или новых систем конкретного разработчика. Метрики надежности могут быть рассчитаны для любых систем, подобных новой разработанной системе по функциональности, оценкам сложности, квалификации разработчиков, уровнем процессов и методологии разработки.

Ключевые слова: *программная система; программное обеспечение; надежность; дефект; отказ; исходный код; сложность; метрика.*

The work has analyzed the few existing metrics of the software systems reliability. Standard metrics have been found to have disadvantages. Existing metrics are not sufficient to assess the reliability in a timely manner during the development phase of software systems. It is determined that the complexity of the source code is a major cause of the errors.

The analysis of complexity types for software systems has been performed. It is established need of the additional development of new reliability metrics on the basis of the assessment of the source code complexity.

A study of existing complexity metrics has been conducted. Metrics were chosen that are most informative for reliability evaluation. Based on the selected object-oriented code complexity metrics, a single combined complexity assessment has been developed instead of twenty different scale estimates. This reduces the source data array by twenty times, significantly reduces the hardware load and the processing time. The practical use of the combined complexity assessment greatly simplifies the assessment, visualization and understanding of the properties of the system, as well as facilitates in-depth analysis of its reliability.

Using combined complexity assessment, code reliability metrics are proposed: the ratio between defective and defect-free modules, the probability of the

defects detection in modules, the modular density of defects, the localization and the distribution of defects in the code. The procedure for the designing and the analyzing reliability metrics is described. The proposed reliability metrics are calculated and rendered for various software systems based on their metric data.

The analysis of developed metrics was carried out. Directions of their practical using by specialists of software companies have been established. It is proposed to calculate reliability metrics for previously developed and verified systems or their parts, for which metric assessments of complexity and the defects number are known.

The proposed reliability metrics can be used to plan resources and perform the efficient verification of newly created code of new iterations, parts, versions, functions, or new systems for a particular developer. Reliability metrics can be calculated for any system similar to the newly developed system in terms of the functionality, complexity assessments, the developer qualification, the process level, and the development methodology.

Key words: *software system; software; reliability; defect; fault; source code; complexity; metric.*

Постановка проблеми. Наразі безліч програмних систем (software system, далі система) використовується підприємствами і фізичними особами для різноманітних потреб. Під програмною системою ми розуміємо сукупність взаємопов'язаних підсистем, класів, компонентів або модулів, які взаємодіють між собою з метою виконання вимог користувачів. Зростання потреб користувачів обумовлює підвищення складності систем. Динамічні зміни в предметних областях та законодавстві викликають необхідність змін та розширення функціональності систем. Це призводить до збільшення кількості дефектів в їхньому програмному коді та обумовлює зниження їхньої надійності. Невиявлені розробниками дефекти та їх усунення на етапі експлуатації значно підвищують витрати на супроводження систем. Дослідження, проведені компаніями TRW, Hewlett-Packard та IBM, переконливо доводять, що усунення дефектів на етапі експлуатації в 4–100 разів дорожче, ніж на етапі розробки. Згідно з оцінками Національного інституту стандартів та технологій США, внаслідок низької надійності систем економічні втрати лише цієї країни станолять щороку близько шістдесяти мільярдів доларів. Для підвищення надійності необхідно зменшити кількість не виявле-

них розробниками дефектів, що дозволить скоротити витрати на супроводження та знизити ризики катастрофічних втрат, техногенних аварій та людських жертв, обумовлених низькою надійністю систем. Надійність програмних систем оцінюється за допомогою метрик надійності, які описані в стандартах.

Аналіз стандартів щодо метрик надійності програмних систем. Відповідно до ISO/IEC 25010-2011 [1], під надійністю систем розумітимемо комплексу властивість, яка характеризує здатність системи правильно виконувати свої функції упродовж визначеного періоду часу. Згідно з ISO/IEC 9126 [2–4], основними показниками надійності на етапі розробки систем визначено *оцінки кількості дефектів, щільності дефектів та ступінь виявлення дефектів*. Згідно зі стандартами [5–6], під дефектом (defect, fault) розуміється програмна аномалія, некоректне визначення операції, процесу чи даних у програмному забезпеченні (ПЗ), що може викликати не відповідне до специфікації функціонування ПЗ або призвести до відмови. Дефекти виникають у ПЗ внаслідок помилок розробників. Їх визначення наведено в стандартизованих словниках термінів якості ПЗ [7–8].

Метрика "Оцінка кількості дефектів" (assessment of defects number) розраховується за допомогою моделі надійності. Зазвичай важко вибрати найбільш адекватну модель із великої кількості наявних моделей, що значно знижує точність оцінювання цієї метрики. Метрика "Щільність дефектів" (defect density) має суттєві недоліки. Вона не враховує складність вихідного програмного коду системи (source code, далі – код), а також рядки коментарів, порожні та системні рядки в коді, що призводить до перекручень значень цієї метрики. На наш погляд, не обсяг, а складність коду є першоджерелом помилок.

Окрім небагатьох стандартних показників надійності, розробникам необхідні показники кількості дефектних та бездефектних модулів, локалізації дефектів, розподілу дефектів у коді, ймовірності дефектів у модулях, модульної щільності дефектів. Стандарт [8] визначає модуль (компонент) програмних систем як окрему дискретну ідентифіковану структурну одиницю, яка може бути протестована окремо, без чіткого визначення її розмірів.

Своєчасне визначення показників надійності дозволяє розробникам заздалегідь розрахувати, залучити й використати ресурси розробки таким чином, щоб за визначений проміжок часу виявити й усунути якомога більше

дефектів. Для зменшення кількості невиявлених дефектів потрібно визначити показники надійності системи ще до початку виявлення та усунення дефектів. Але зазвичай ці показники стають відомими тільки після завершення цього процесу. На наш погляд, для подолання цієї суперечності необхідно оцінити показники надійності апріорно, до початку верифікації програмного коду систем на основі його складності, виходячи з того, що складність коду є першопричиною дефектів коду.

Згідно з прпцями багатьох дослідників складність, є супутньою властивістю програмних систем. Їхня складність характеризується великою кількістю різноманітних станів, процесів, складових та зв'язків між ними, використанням різних мов програмування і моделей розробки та виконанням великої кількості функцій для розвннєязання багатоцільових задач. Програмним системам властива *статична* складність, що описує зв'язність і структуру підсистем, *динамічна* складність, пов'язана з поведінкою системи у часі, *ієрархічна* складність, *структурна* складність, яка визначається кількістю ієрархічних рівнів, підсистем і компонентів системи, *алгоритмічна* складність, *цикломатична* складність, *обчислювальна* складність, яка оцінюється кількістю операцій для отримання результату і кількістю оброблюваних елементів, часова та просторова складність.

На підставі аналізу стандарту IEEE 1061 [9] зроблено висновок, що основним методом оцінювання складності програмного коду є розрахунок числових показників за відповідними метриками. Згідно з цим стандартом, метрика (metric) – це кількісна (або якісна) оцінка ступеня, в якому система відповідає заданим властивостям. У контексті нашого дослідження метрика складності (далі – метрика) – це математичний вираз числової оцінки одного або кількох аспектів складності програмного коду. Наразі багатьма науковцями та практиками програмної інженерії розроблено близько п'ятдесяти метрик складності. Вибір метрик залежить від систем програмування. За цією ознакою найбільш відомі та необхідні метрики оцінювання складності процедурно-орієнтованого та об'єктно-орієнтованого програмування, які описані в [10]. Метрика WMC (Weighted methods per class) визначає загальну складність методів класу. Метрика DIT (Depth of Inheritance tree) оцінює глибину дерева наслідування як найдовший шлях за ієрархією класів від класу-предка до класу-нащадка. Метрика NOC (Number of children) оцінює кількість класів-нащадків. Метрика CBO (Coupling between object classes) характеризує зчепленість між класами, тобто кількість класів, з якими

пов'язаний вихідний клас. Метрика RFC (Response for a class) визначає кількість методів (даного та інших класів), які викликає даний клас. Метрика LCOM (Lack of cohesion in Methods) визначає, наскільки методи класу не пов'язані між собою через змінні. Саме ці метрики ми використали у нашому дослідженні.

Після проведеного аналітичного огляду джерел [1–12] зроблено наступний висновок. Нині зусиллями дослідників та спеціалістів програмної інженерії розроблено велику кількість метрик оцінювання різноманітних властивостей коду. Однак немає метрик оцінювання дефектності коду залежно від його складності. Водночас складність коду, на наш погляд, є головною причиною помилок. Саме ця обставина обумовлює мету дослідження.

Мета даної статті – розробка та аналіз метрик надійності, які оцінюють дефектність програмного коду залежно від його складності.

Складність коду вимірюється за допомогою відомих метрик, значення яких підраховуються автоматично під час компіляції або інтерпретації програмного коду. Дефектність коду вимірюється як кількість виявлених дефектів у модулях. Значення метрик надійності розраховуються для раніше розроблених і верифікованих систем або їх частин, для яких відомі метричні оцінки складності та кількість дефектів. Розраховані метрики можуть бути використані для планування ресурсів та виконання ефективної верифікації новоствореного коду нових ітерацій, частин, версій, функцій або нових систем конкретного розробника. Метрики надійності можуть бути розраховані для будь-яких систем, подібних до новорозробленої системи за функціональністю, метричними оцінками складності, кваліфікацією розробників, рівнем процесів та методологією розробки.

Виклад основного матеріалу.

Аналіз складності коду та розробка єдиної комплексної оцінки складності. Набори даних систем зі сховища [13] містять числові метричні оцінки складності окремих модулів коду за двадцятьма метриками. Це оцінки обчислювальної складності, складності структури, зв'язності, зчеплення, розмірів модуля та ін. Набори даних містять також інформації про відсутність або наявність кількості дефектів у кожному модулі, що було виявлено в процесі верифікації. Структуру даних можна описати таким вектором метрик (vector of metrics, VM)

$$VM = \begin{Bmatrix} module\_name, wmc, dit, noc, cbo, rfc, lcom, ca, ce, npm, lcom3, loc, \\ dam, moa, mfa, cam, ic, cbm, amc, max\_cc, avg\_cc, bug \end{Bmatrix}.$$

Для окремого модуля системи вектор метричних оцінок (vector of metrics assessment, VMA) має наступний вигляд:

$$VMA = \begin{Bmatrix} \text{module\_}\,name, 35,\ 2,\ 0,\ 16,\ 103,\ 317, 3,\ 16,\ 28,\ 0.813, \\ 865,\ 0.888,\ 0,\ 0.689,\ 0.152,\ 1,\ 10,\ 23.46,\ 3,\ 1.057, 5 \end{Bmatrix}.$$

Для скорочення обсягу даних аналізу доцільно вибрати з них найбільш інформативні показники. Найбільш інформативними для аналізу дефектності є ті показники, які демонструють тісніший зв'язок із кількістю дефектів (показник bug у VM). Для виміру ступеня зв'язку ми використали коефіцієнт парної кореляції Пірсона (correlation coefficient, CC) між метричними оцінками та виявленими дефектами в модулях систем. Для встановлення закономірностей кореляційних зв'язків ми дослідили двадцять наборів даних різноманітних систем зі сховища [13]. В ході дослідження були встановлені додатні та від'ємні, істотні (CC>0,7) та неістотні (CC<0.1) зв'язки між метричними оцінками та кількістю дефектів. Причому від'ємні CC завжди мали неістотні значення. Дослідження CC показало, що доцільно залишити в наборі даних такі метрики, оцінки яких становили CC≈0.5 або CC>0.5. Оцінки з CC<0.5 потрібно виключити з аналізу. Оцінки за цими метриками мають слабкий кореляційний зв'язок із кількістю дефектів. Ці метрики є недостатньо інформативні для аналізу дефектності коду залежно від його складності. При цьому VMA було скорочено з 20 до 5–7 метрик.

Обрані таким чином метрики відображають різноманітні аспекти складності та істотно (в рази та десятки разів) відрізняються за абсолютними величинами. Наприклад, в одній дослідженій системі метричні оцінки модуля становили: DIT – 8, NOC – 29, NPM – 122, WMC – 130, RFC – 391, LOC – 4275, LCOM – 7399 одиниць. Тому виникає необхідність нормалізації оцінок. Нормалізація дозволить привести всі метричні оцінки до близьких числових значень, що надалі дасть змогу отримати на їх основі *єдину комплексну оцінку складності коду*. Найбільш поширені способи: лінійна та нелінійна нормалізація. Їх порівняльний аналіз виявив таке. Нелінійна нормалізація з використанням гіперболічного тангенса має складніший алгоритм розрахунку, проте не дає будь-яких переваг. Тому була виконана лінійна нормалізація метричних оцінок за формулою:

$$X_{ik}^{n} = \frac{X_{ik} - X_{\min i}}{X_{\max i} - X_{\min i}}, \qquad (1)$$

де X_{ik} – значення *i-ї* метричної оцінки для *k-го* модуля,

X_{ik}^{n} – відповідне нормалізоване значення.

Далі всі відібрані метричні оцінки були нормалізовані за формулою (1) і складені в *єдину комплексну оцінку складності коду* (combined assessment of complexity, CA) за формулою

$$CA_k = \sum_{i=1}^{m} X_{ik}^n , \qquad (2)$$

де m – кількість відібраних метричних оцінок.

Чим вищі значення CA для модуля, тим складніше його структура, методи, взаємозв'язки.

Далі постало питання про інформативність значень CA для аналізу дефектності коду. Ми дослідили кореляцію між CA та кількістю дефектів у модулях. Було встановлено, що CC між CA та дефектами був дещо більший (на 0,1), ніж найвищий CC для відібраних метрик. Таким чином, відібрані метричні оцінки були зведені до одного показника CA без зменшення ступеня зв'язку з дефектами і без втрати інформативності CA для аналізу дефектності коду. В результаті описаних дій отримано єдину CA замість двадцяти різномасштабних оцінок і скорочено масив вихідних даних у двадцять разів, що дозволить значно знизити апаратне навантаження і період обробки даних. Практичне використання CA значно спрощує оцінювання, візуалізацію та розуміння властивостей модулів системи, а також сприяє глибокому аналізу надійності модулів та системи загалом.

На основі CA ми запропонували ще два показники для оцінювання властивостей системи. Перший показник – це сумарна оцінка складності системи (summary assessment of system complexity, SA) за формулою

$$SA = \sum_{k=1}^{n} CA_k , \qquad (3)$$

де n – кількість модулів системи.

Наприклад, для досліджуваних систем значення становили 189, 340, 538, 790, 3459, 3825 одиниць. Мінімальне значення відрізнялось від максимального у 18 разів. Тобто системи відрізнялись своїми властивостями у 18 разів. Другий показник – це середня оцінка складності для модулів системи (average assessment of complexity, AA), розрахована за формулою:

$$AA = SA / n \qquad (4)$$

де n – кількість модулів системи.

Практичне значення оцінок за показниками SA й AA полягає в такому. Оцінка SA відображає загальну складність і розмір системи одним числом. Оцінка AA відображає середню складність і розмір модуля системи. Ці оцінки можуть бути використані для порівняння властивостей різних систем та обґрунтування витрат на їх розробку.

Отже, маємо два ключові показники. Це показник складності й показник дефектів у модулях системи. Далі виникають такі запитання. Як чином використати ці дані, щоб отримати показники надійності? Яким чином перетворити нову інформацію на знання? Як використати ці знання для підвищення надійності системи з одночасним зменшенням затрат на її верифікацію? На наш погляд, це можливо за допомогою метрик надійності, запропонованих нами та описаних нижче.

Процедура розробки та аналізу метрик надійності.

Для розробки метрик надійності ми взяли структуровані дані для різних систем зі сховища [13] . Фрагмент даних показано в табл. 1.

Таблиця 1

Фрагмент даних с метричними оцінками та дефектами

	NAME	VERSION	NAME1	WMC	DIT	NOC	CBO	RFC	LCOM	CA	CE	NPM	LOC	BUG
865	xal	2,6	org.apache.xpath.SourceTree	**1**	1	0	1	**2**	0	1	0	1	12	0
866	xal	2,6	org.apache.xalan.xsltc.compile	**11**	5	0	16	**23**	37	3	13	10	141	0
867	xal	2,6	org.apache.xpath.axes.ChildIte	**4**	5	0	5	**9**	6	1	4	3	64	0
868	xal	2,6	org.apache.xml.utils.Trie	**3**	1	0	2	**10**	0	2	1	3	162	1
869	xal	2,6	org.apache.xml.utils.StringBuf	**4**	1	0	7	**9**	0	5	2	3	25	0
870	xal	2,6	org.apache.xalan.xsltc.trax.Tr	**33**	3	0	17	**151**	322	6	16	23	1758	5
871	xal	2,6	org.apache.xalan.xsltc.compile	**4**	3	0	11	**14**	0	0	11	4	44	0
872	xal	2,6	org.apache.xml.utils.Suballoca	**20**	1	0	15	**22**	8	15	0	13	901	1
873	xal	2,6	org.w3c.dom.xpath.XPathExpress	**1**	1	0	0	**1**	0	0	0	1	1	1
874	xal	2,6	org.apache.xml.serializer.Seri	**67**	1	3	12	**111**	1771	3	9	47	958	5
875	xal	2,6	org.apache.xalan.xsltc.cmdline	**1**	4	0	2	**2**	0	1	1	1	5	0
876	xal	2,6	org.apache.xml.serializer.Outp	**6**	1	0	10	**38**	9	6	4	2	388	2
877	xal	2,6	org.apache.xpath.ExpressionNod	**5**	1	0	44	**5**	10	44	0	5	5	0
878	xal	2,6	org.apache.xpath.axes.LocPathI	**52**	4	8	37	**84**	1016	20	20	45	665	1
879	xal	2,6	org.apache.xml.utils.Serializa	**10**	1	0	0	**15**	37	0	0	10	64	0
880	xal	2,6	org.apache.xalan.xsltc.compile	**5**	4	0	14	**10**	4	4	11	4	38	0
881	xal	2,6	org.apache.xalan.xsltc.compile	**2**	4	0	17	**25**	1	1	16	2	137	0
882	xal	2,6	org.apache.xml.serializer.Attr	**6**	2	0	6	**20**	0	6	0	5	124	1
883	xal	2,6	org.apache.xpath.objects.XNode	**7**	5	0	5	**18**	0	1	4	7	115	0
884	xal	2,6	org.apache.xpath.compiler.Psue	**1**	1	0	0	**2**	0	0	0	1	10	0
885	xal	2,6	org.apache.xalan.transformer.T	**36**	1	0	9	**100**	0	1	9	33	914	1

Далі ми нормалізували метричні оцінки модулів за формулою (1) і розрахували CA за формулою (2). Після цього дані були відсортовані за CA та згруповані для кожного інтервалу складності від найменшого до найбільшого значення із кроком 0,2. Запропоновані метрики розраховувались для кожного інтервалу CA та відображались у вигляді графіків або діаграм.

Метрика співвідношення між дефектними та бездефектними модулями.

Перша запропонована метрика обчислюється як співвідношення між дефектними (defect modules, DM) та бездефектними модулями (defect-free modules, DFM), що відображає ступінь дефектності коду системи (Ratio between DM and DFM, **RDM**). Чим вище це співвідношення, тим більше часу потребуватиме верифікація. Для аналізу RDM ми взяли чотири різні системи. Характеристики систем подано в табл. 2. Напівжирним шрифтом позначені максимальні показники.

Таблиця 2

Метрика RDM, показники дефектностіта складності модулів різних систем

Системи	% DM	% DFM	**Метрика RDM**	AA	CA_{max}	SA
Система (a)	22	**78**	0,28	0,56	**6,8**	415
Система (b)	60	40	1,5	0,51	5,1	173
Система (c)	**75**	25	3	**0,67**	5,6	394
Система (d)	99	1	**99**	0,55	4,7	**495**

Графічний вигляд метрики RDM зображено на рис. 1, який відображає залежності DM (область червоного кольору) та DFM (область зеленого кольору) від CA для чотирьох (A, B, C, D) різних систем. На осі абсцис розташовано значення CA, на осі ординат – кількість модулів системи.

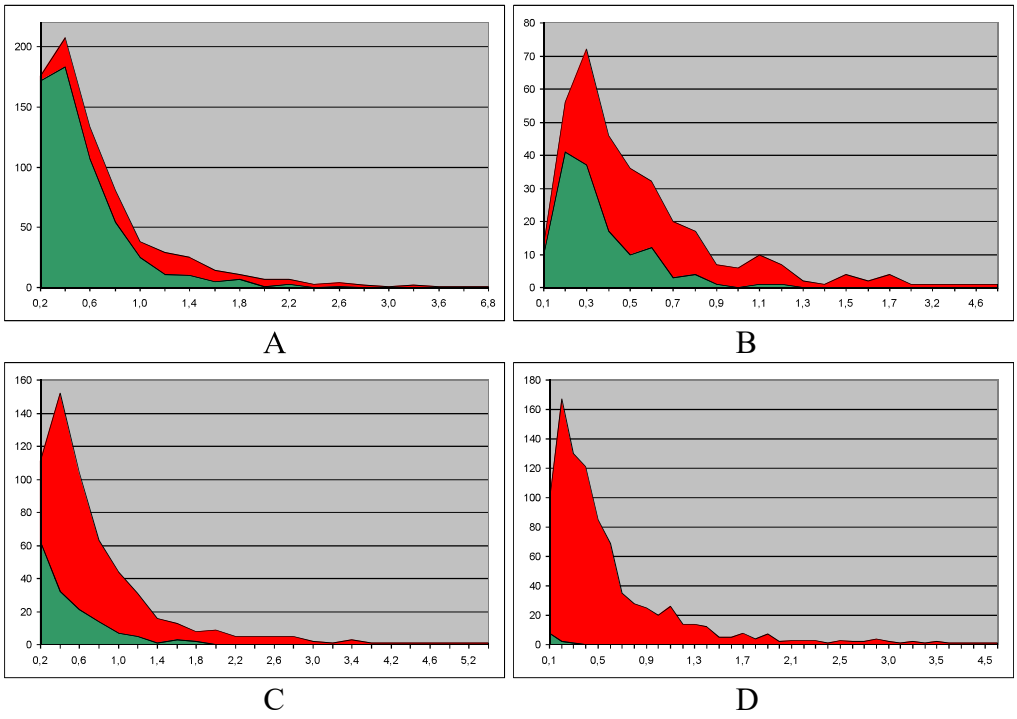


Рис. 1. Графічне зображення метрики RDM для різних систем

На рис. 1 видно, що система A має найнижчий показник дефектності коду. RDM = 0,28. Це показник “чистого коду”. За інших рівних умов верифікація цієї системи забере менше часу і буде дешевшою. SA=415. AA=0,56. Це середні показники. Максимальна CA=6,8. Це найвищий показник серед систем. Окремі модулі цієї системи складніші, ніж модулі інших систем. Щоб уникнути проблем з налагодженням і підтримкою складних модулів, необхідно виконати їх рефакторинг та декомпозицію.

Наступною за рівнем дефектності модулів іде система B. У цієї системи співвідношення між DM та DFM становить відповідно 60% і 40%. Це середні показники. RDM=1,5. SA=173. AA=0,51. Максимальна CA=5,1. Це найнижчі показники серед систем. Ця система складається з найменш складних модулів.

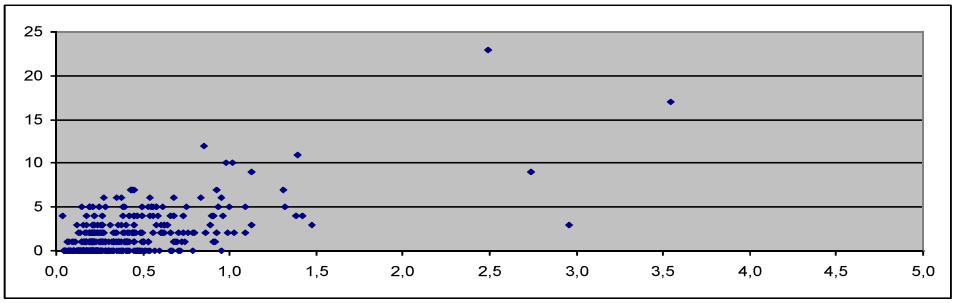
Наступною за рівнем дефектності модулів іде система C. У цієї системи співвідношення між DM та DFM становить відповідно 75 і 25 %. RDM = 3. Це найнижчий показник "сірого коду". SA = 394. Це середній показник. AA = 0,67. Це найвищий показник серед систем. Ця система складається з найбільш складних модулів. Щоб уникнути проблем з їх налагодженням і підтримкою, необхідно виконати рефакторинг і декомпозицію цих модулів.

Система D має найвищий показник дефектних модулів 99%. RDM = 3. Це показник "брудного коду" і низької його надійності. Незалежно від складності та розміру майже всі модулі цієї системи мають дефекти. Менеджерам необхідно виявити причини такого тривожного явища. Верифікація цієї системи буде найбільш тривалою і дорогою серед розглянутих систем. Щоб зменшити витрати, необхідна ретельна інспекція коду до початку його тестування. SA=495. Це найвищий показник серед систем. Ця система сама велика і складна, отже, потребує більше витрат на розробку і верифікацію, ніж системи A, B і C. AA=0,55. Максимальна CA=5,1. Це середні показники серед систем.

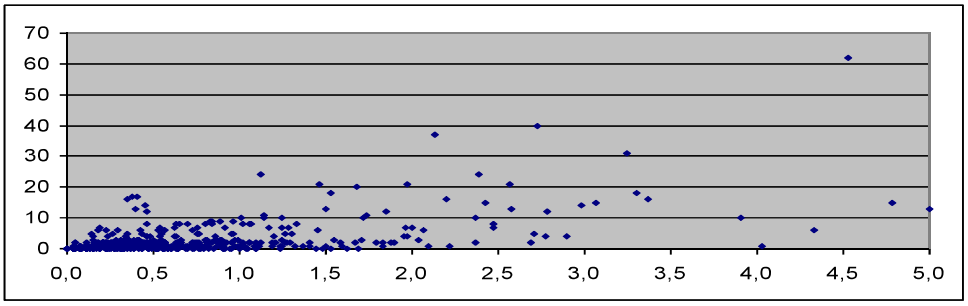
Метрика RDM разом із CA, SA і AA дають фахівцям чітку картину властивостей та надійності розроблюваної системи, дозволяють планувати ресурси і процеси верифікації та рефакторингу.

Для розрахунку CA, SA і AA ми застосовували спеціалізований програмний засіб, розроблений для цілей дослідження. Його простий алгоритм полягає в обчисленні оцінок за формулами (1), (2), (3) і (4). Такий засіб можуть розробити фахівці будь-якої софтверної компанії. У разі паралельної обробки великих обсягів даних програмну реалізацію цього нескладного алгоритму можна розпаралелити на ядра процесора або на робочі вузли апаратного кластера за допомогою системи Spark. Далі ми розглянемо другу запропоновану метрику.

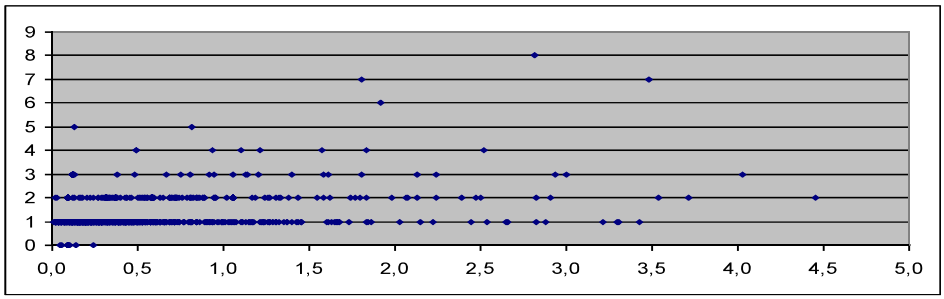
Метрика локалізації дефектів у модулях. Графічна метрика локалізації дефектів у модулях (Defects Localizing in modules, DLM) показує скупчення дефектів у модулях системи для кожного інтервалу складності від найменшого до найбільшого значення CA з кроком 0,2. Діаграми чотирьох різних систем зображено на рис. 2. Системи на рис. 2 не повторюють системи на рис.1. На діаграмах рис. 2 на осі абсцис розташовано значення CA, на осі ординат – кількість дефектів у модулях системи. Крапки на діаграмах відображають дефектні ($y > 0$) та бездефектні ($y = 0$) модулі системи.



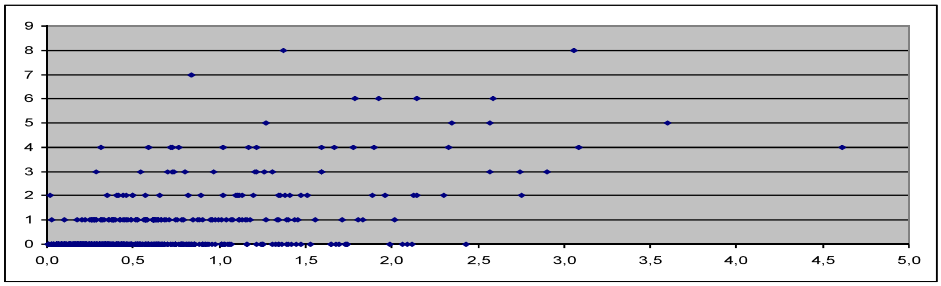
A



B



C



D

Рис. 2. Графічне зображення метрики DLM для різних систем

У системі A основна маса дефектів (найчастіше від 1 до 5) локалізована в модулях із CA<1,5. Це невелика локалізація дефектів, що спрощує і здешевлює верифікацію. Максимальна кількість дефектів в одному модулі понад 20. Система має значну кількість бездефектних модулів.

У системі B дефекти (найчастіше від 1 до 10) більше розсіяні за кодом. У цієї системи дефекти локалізовані в модулях 0<CA<3, і локалізація дефектів більше, ніж у системи A. Верифікація системи B потребує більше часу, зусиль і коштів, ніж верифікація системи A за інших рівних умов. Максимальна кількість дефектів в одному модулі більше 60. Система має значну кількість бездефектних модулів.

Локалізація дефектів у модулях систем C і D схожа, вони більші, ніж у попередніх систем. Верифікація цих систем забере більше часу, ніж попередніх. Максимальна кількість дефектів в одному модулі однакова для цих систем. Відмінність систем C і D полягає в тому, що в системі D багато бездефектних модулів, а в системі C бездефектних модулів тільки 4. Всі інші модулі містять дефекти, найчастіше 1 або 2. Система C має найбільшу локалізацію дефектів.

Якщо ранжувати системи за розміром локалізації дефектів, то найменша локалізація спостерігається у системи A. Далі йде система B, за нею система D. Найбільшу локалізацію дефектів має система C. Якщо ранжувати системи за дефектністю складних модулів, то найменш дефектною буде система A. За 2,5 < CA < 5 система A має всього чотири дефектних модулі, система D має 10 дефектних модулів, системи B і C мають понад 20 дефектних модулів.

Проведений аналіз дозволяє прогнозувати таке. За інших рівних умов найменш тривалою і витратною буде верифікація системи A через найменшу локалізацію дефектів. Унаслідок найбільшої локалізації дефектів і відсутності бездефектних модулів верифікація системи C буде найбільш тривалою і трудомісткою. Найбільшу кількість дефектів буде виявлено в системі B. Цей прогноз, складений на основі аналізу метрики LDM, був підтверджений фактичними даними. Для розглянутих систем загальна кількість дефектів становила: для системи A – 632 дефектів, для системи B – 1596 дефектів, для системи C – 1213 дефектів, для системи D – 338 дефектів.

Практичне значення метрики DLM полягає в такому. Діаграму локалізації дефектів системи, подібної до розроблюваної, можна використовувати

для планування ресурсів верифікації та спрямування зусиль фахівців на модулі з більшою локалізацією дефектів.

Метрика процентного розподілу дефектів у коді. Метрика процентного розподілу дефектів у коді (Defects Percentage Distributing, DPD) обчислювалась як процентне співвідношення кількості дефектних модулів до загальної кількості модулів для кожного інтервалу складності за CA від найменшого до найбільшого значення із кроком 0,2. Графічне зображення метрики для двох систем зображено на рис. 3. На осі абсцис розташовано значення CA, на осі ординат – процент дефектів від їх загальної кількості.

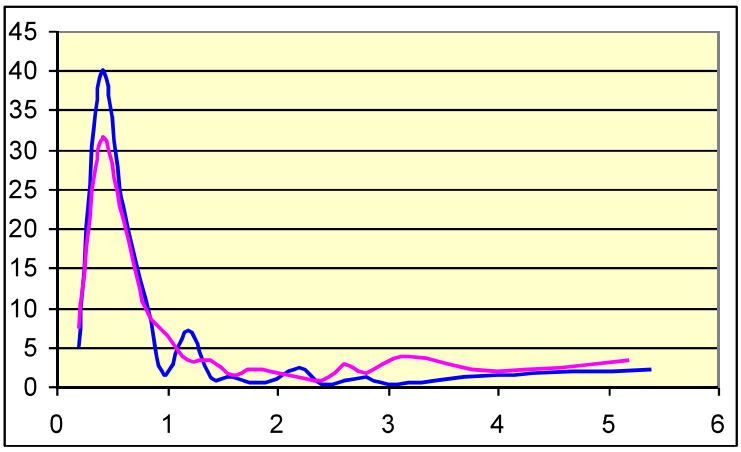


Рис. 3. Графічне зображення метрики DPD для двох систем

Графіки показують, що основна частина дефектів у цих системах перебуває в модулях з оцінкою CA<1. Практичний аспект метрики DPD полягає в можливості використання її для спрямування зусиль фахівців на такі модулі нової системи, які містять найбільшу частину дефектів.

Метрика імовірності виявлення дефектів. Метрика імовірності виявлення дефектів (одного або кількох) у модулі (Probability of Defects Detection, PDD) обчислювалась як імовірність виявлення дефектів у модулях для кожного інтервалу складності за CA від найменшого до найбільшого значення із кроком 0,2. Графічне зображення метрики для двох систем зображено на рис. 4. На осі абсцис розташовано значення CA, на осі ординат – імовірність виявлення дефектів.

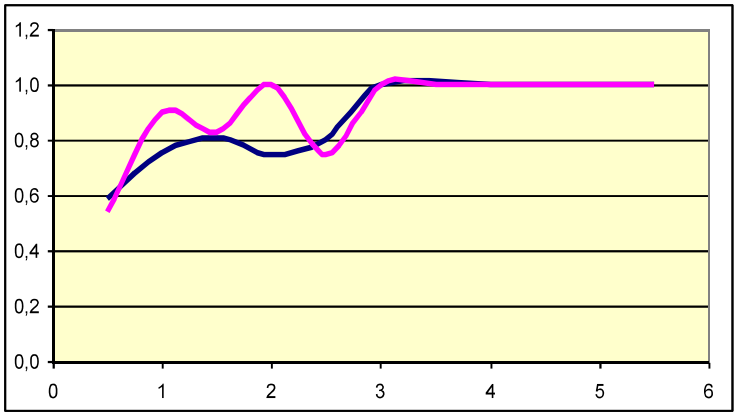


Рис. 4. Графічне зображення метрики PDD для двох систем

Початкова координата залежності на осі Y у цих систем близька і становить приблизно 0,6. Це означає, що з кожних десяти модулів мінімальної складності шість модулів містять дефекти. Обидві криві містять прямий відрізок (y=1) за CA=3. Це та частина коду, в якій всі модулі містять один або кілька дефектів. Практичний аспект метрики полягає в такому. Аналіз DPD подібної системи дозволяє фахівцям виявляти модулі з DPD = 1 в новій системі для обов'язкової їх верифікації.

Метрика модульної щільності дефектів (Modular Defect Density, MDD) обчислювалась як кількість дефектів в одному модулі системи для кожного інтервалу складності за CA від її найменшого до найбільшого значення із кроком 0,2. Після оцінювання MDD ми зобразили залежність значень MDD від CA на рис. 5. На осі абсцис розташовано значення CA, на осі ординат – значення MDD.

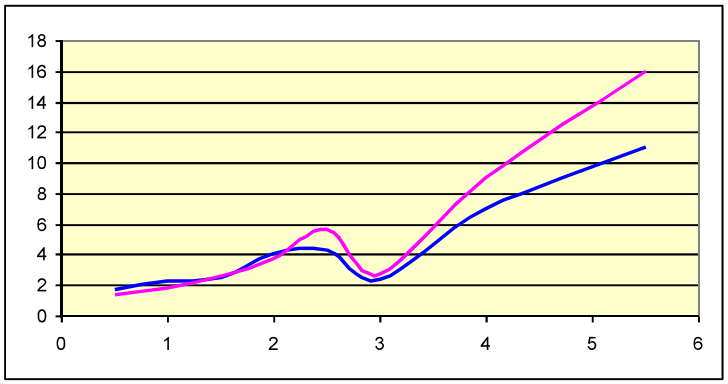


Рис. 5. Графічне зображення метрики MDD для двох систем

На рис. 5 ми бачимо зростання нелінійної залежності, близької до експоненціальної. Практичний аспект метрики полягає в такому. Знання та аналіз MDD дозволяє фахівцям спрямувати зусилля верифікації на модулі з найбільшою кількістю дефектів у них. Наприклад, для цих систем це модулі, для яких 4<CA<5,5. Але слід розуміти, що зазвичай таких модулів небагато, і кількість виявлених у них дефектів становитиме невелику частину від їх загальної кількості.

Висновки з данного дослідження і перспективи подальших розвідок у даному напрямку. В роботі виконано аналіз наявних метрик надійності програмних систем. Виявлені недоліки цих метрик. Установлена необхідність розробки нових метрик надійності на основі залежностей дефектності вихідного програмного коду від його складності. Для оцінювання складності запропоновані показники: комплексна оцінка складності коду модуля, середня оцінка складності модулів системи та сумарна оцінка складності системи.

Описана процедура розробки метрик надійності. На основі комплексних оцінок складності коду модулів і кількості дефектів у модулях запропоновано п'ять метрик надійності коду:

1. Метрика співвідношення між дефектними та бездефектними модулями (Ratio between DM and DFM, RDM);
2. Метрика локалізації дефектів у модулях (Defects Localizing in Modules, DLM);
3. Метрика процентного розподілу дефектів у коді (Defects Percentage distributing, DPD);
4. Метрика імовірності виявлення дефектів у модулях (Probability of Defects Detection, PDD);
5. Метрика модульної щільності дефектів (Modular Defect Density, MDD).

Значення метрик надійності розраховуються для раніше розроблених і верифікованих систем або їхніх частин, для яких відомі метричні оцінки складності та кількість дефектів.

Запропоновані метрики розраховано та візуалізовано для ряда систем. Проведеной аналіз метрик та встановлено напрями їх практичного використання фахівцями софтверних компаній. Метрики надійності можуть бути використані для планування ресурсів та виконання ефективної верифікації новоствореного коду нових ітерацій, частин, версій, функцій або нових систем конкретного розробника. Метрики надійності можуть бути використані для будь-яких систем, подібних до новорозробленої системи за функціональністю, метричними оцінками складності, кваліфікацією розробників, рівнем процесів та методологією розробки.

Надалі необхідно дослідити метрики надійності для подібних систем. На основі метрик складності та надійності доцільно розробити нові моделі та методи підвищення надійності програмних систем.

Список використаних джерел:

1. ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model / ISO: офіційний сайт. URL: https://www.iso.org/standard/22749.html

2. ISO/IEC TR 9126-2:2003 Software engineering – Product quality – Part 2: External metrics / ISO: офіційний сайт. URL: https://www.iso.org/standard/22750.html

3. ISO/IEC TR 9126-3:2003 Software engineering – Product quality – Part 3: Internal metrics / ISO: офіційний сайт. URL: https://www.iso.org/standard/22891.html

4. ISO/IEC 25010:2011 Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models / ISO: офіційний сайт. URL: https://www.iso.org/standard/35733.html

5. IEC 61513:2001 Nuclear power plants – Instrumentation and control systems important for safety – General requirements for systems / IEC Webstore International Electrotechnical Commission : офіційний сайт. URL: https://webstore.iec.ch/publication/19812

6. IEC 60880-2010 Nuclear power plants. Instrumentation and control systems important for safety. Software aspects for computer-based systems performing category A functions / Електронний фонд правової і нормативно-технічної документації: офіційний сайт. URL: http://docs.cntd.ru/document/gost-r-mek-60880-2010

7. IEEE 982.1-2005 – IEEE Standard Dictionary of Measures of the Software Aspects of Dependability / IEEE Xplore Digital Library: офіційний сайт. URL: https://ieeexplore.ieee.org/document/1634994

8. IEEE 610.12-1990 – IEEE Standard Glossary of Software Engineering Terminology / IEEE Xplore Digital Library: офіційний сайт. URL: https://ieeexplore.ieee.org/document/159342

9. IEEE 1061-1998 – IEEE Standard for a Software Quality Metrics Methodology / IEEE Xplore Digital Library: офіційний сайт. URL: https://ieeexplore.ieee.org/document/749159

10. *Chidamber S., Kemerer C.* A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*. 1994. № 20. P. 476–493.

11. *Табунщик Г. В., Кудерметов Р. К., Брагіна Т. І.* Інженерія якості програмного забезпечення: навчальний посібник. Запоріжжя: ЗНТУ. 2013. 180 с.

12. *Маєвський Д. А.* Проблеми забезпечення надійності при експлуатації динамічних інформаційних систем. *Системи обробки інформації.* 2012. № 7. С. 99–103.

13. Tera-PROMISE Home. The PROMISE Repository of empirical software engineering data / Міжнародна наукова конференція PROMISE: офіційний сайт. URL: http://openscience.us/repo/defect/ck/

References:

1. International Organization for Standardization/International Electrotechnical Commission (2018), ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model, ISO official site, available at: https://www.iso.org/standard/22749.html

2. International Organization for Standardization/International Electrotechnical Commission (2018), ISO/IEC TR 9126-2:2003 Software engineering – Product quality –

Part 2: External metrics, ISO official site, available at: https://www.iso.org/standard/227509.html

3. International Organization for Standardization/International Electrotechnical Commission (2018), ISO/IEC TR 9126-3:2003 Software engineering – Product quality – Part 3: Internal metrics, ISO official site, available at: https://www.iso.org/ standard/22891.html

4. International Organization for Standardization/International Electrotechnical Commission (2018), ISO/IEC 25010:2011 Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, ISO official site, available at: https://www.iso.org/standard/35733.html

5. International Electrotechnical Commission (2018), IEC 61513:2001 Nuclear power plants – Instrumentation and control systems important for safety – General requirements for systems / IEC Webstore International Electrotechnical Commission, official site, available at: https://webstore.iec.ch/publication/19812

6. International Electrotechnical Commission (2018), IEC 60880-2010 Nuclear power plants. Instrumentation and control systems important for safety. Software aspects for computer-based systems performing category A functions / Electronic Legal and Regulatory Documentation Fund, official site, available at: http://docs.cntd.ru/document/gost-r-mek-60880-2010

7. IEEE Xplore Digital Library (2018), 982.1-2005 – IEEE Standard Dictionary of Measures of the Software Aspects of Dependability / Institute of Electrical and Electronics Engineers, official site, available at: https://ieeexplore.ieee.org/document/1634994

8. IEEE Xplore Digital Library (2018), 610.12-1990 – IEEE Standard Glossary of Software Engineering Terminology / Institute of Electrical and Electronics Engineers, official site, available at: https://ieeexplore.ieee.org/document/159342

9. IEEE Xplore Digital Library (2018), 1061-1998 – IEEE Standard for a Software Quality Metrics Methodology / Institute of Electrical and Electronics Engineers, official site, available at: https://ieeexplore.ieee.org/document/749159

10. Chidamber S. A Metrics Suite for Object-Oriented Design [Text] / S. Chidamber, C. Kemerer // IEEE Transactions on Software Engineering – 1994. – № 20, P. 476-493.

11. Tabunshhy`k G.V., Kudermetov R. K. and Bragina T. I. (2013), Inzheneriya yakosti programnogo zabezpechennya: navchal`ny`j posibny`k [Engineering of the software quality: Manual], Press Zaporizhzhya, ZNTU, 180 p. [Ukraine]

12. Mayevs`ky`j, D. A. (2012), "*Problemy` zabezpechennya nadijnosti pry` ekspluataciyi dy`namichny`x informacijny`x sy`stem*" ["Reliability assurance problems during operation of dynamic information systems"], journal *Sy`stemy` obrobky` informaciyi* [Information processing systems], vol. 7, pp. 99-103 [Ukraine]

13. Tera-PROMISE Home. The PROMISE Repository of empirical software engineering data (2018), International scientific conference PROMISE, official site, available at: http://openscience.us/repo/defect/ck/