УДК 004.056

Зоріна Т.І. (СНУ ім.В.Даля)
к.т.н. **Петров А.О.** (СНУ ім.В.Даля)
Сергієня К.В. (СНУ ім.В.Даля)

ЖИТТЄВИЙ ЦИКЛ ТА НАДІЙНІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті розглянуто основи надійності програмних систем та програмного коду, досліджено зв'язки між показниками економічності, ефективності та надійності на основі розробки відповідних методик. Проаналізовано організаційні процеси життєвого циклу програмного продукту, що застосовується організацією для встановлення і реалізації базової структури управління інформаційною безпекою. Виявлено причини невірного функціонування інформаційної системи – наявність в ній так званих вірусних програм, програм призначених для надмірного викривлення результатів розрахунків, видалення файлів, створення умов для ненормального функціонування інформаційної системи.

Ключові слова: інформаційна безпека, управління, програмне забезпечення, надійність, відмовостійкість, захищеність.

Вступ. Життєвий цикл програмного забезпечення (продукта) починається з визначення технічного завдання його розробки і закінчується припиненням його використання.

Дії, які виконуються протягом життєвого циклу програмного продукту є наступними: п'ять основних процесів, вісім процесів з підтримки та чотири організаційні процеси.

До складу основних процесів життєвого циклу входять п'ять процесів, що обслуговують основних учасників протягом життєвого циклу програмного забезпечення. Основні учасники – це суб'єкти, які ініціюють, розробляють, проводять експлуатацію чи супровід програмного продукту. До основних учасників належать замовник, постачальник, розробник, оператор та супроводжувач програмного продукту. До основних процесів належать такі процеси:

–*замовлення* – визначає дії замовника-організації, що замовляє систему, програмний продукт або програмну послугу;

–*постачання* – визначає дії постачальника-організації, що надає систему, програмний продукт або програмну послугу;

–*розроблення* – визначає дії розробника-організації, що визначає та проектує програмний продукт;

–*експлуатації* – визначає дії оператора-організації, що надає послугу з експлуатації інформаційної (автоматизованої) системи в її існуючому середовищі для її користувачів;

–*супроводу* – визначає дії супроводжувача-організації, що надає послугу з супроводу програмного продукту, тобто, керує внесенням змін до програмного продукту для підтримання його в належному та працездатному стані. Цей процес передбачає перенесення та вилучення програмного продукту.

Процес підтримки застосовується та виконується в рамках процесу відповідно до його потреб. До складу процесів підтримки життєвого циклу програмного продукту входять процеси:

–*документування* – визначає дії щодо реєстрації інформації, отриманої в процесі життєвого циклу;

–*конфігурування* – визначає дії щодо керування конфігурацією програмного продукту;

–*забезпечення якості* – визначає дії щодо набуття об'єктивної впевненості в тому, що програмні продукти та процеси відповідають заданим для них вимогам та дотримуються встановлених для них планів.

Як методи забезпечення якості можна використовувати процеси спільного перегляду, аудиту, верифікації та валідації;

–*верифікації* – визначає дії замовника, постачальника або незалежного учасника щодо проведення верифікації програмного продукту з різним ступенем глибини залежно від програмного продукту;

– *валідації* – визначає дії замовника, постачальника або незалежного учасника щодо проведення валідації програмного продукту, одержаного в рамках програмного проекту;

– *спільного перегляду* – визначає дії щодо проведення оцінки стану та результатів певної дії. Цей процес може бути застосований будь-якими двома учасниками, один з яких (учасник, що здійснює перегляд) здійснює перегляд дій іншого учасника (учасника, дії якого переглядаються) в рамках спільного обговорення;

– *аудиту* – визначає дії щодо визначення відповідності вимогам, планам та контракту. Цей процес може бути застосований будь-якими двома учасниками, один з яких (учасник, що перевіряє) проводить аудит програмного продукту або дій іншого учасника (учасник, якого перевіряють);

– *вирішення проблем* – визначає дії щодо аналізу і зняття проблем (включно з невідповідностями) незалежно від їхньої природи та причин, виявлених в процесі розроблення, експлуатації, супроводу чи під час виконання інших процесів.

Організаційні процеси життєвого циклу програмного продукту застосовуються організацією для встановлення і реалізації базової структури, до склад якої входять взаємопов'язані процеси життєвого циклу та відповідний персонал, а також для постійного вдосконалення структури та процесів. Їх застосування, як правило, виходить за рамки вдосконалення конкретних проектів і контрактів; проте досвід щодо проектів та контрактів використовується для вдосконалення і ефективності роботи організації. До організаційних процесів входять процеси:

– *керування* – визначає основні дії щодо керування, включно з керуванням проектом, протягом життєвого циклу;

– *створення інфраструктури* – визначає основні дії щодо створення основної структури процесів життєвого циклу;

– *утворення* – визначає базові дії, які організація (якою може бути замовник, постачальник, розробник, оператор, супроводжувач або керівник іншого процесу) виконує з метою створення, вимірювання, контролю та вдосконалення процесів життєвого циклу, які вона проводить;

– *навчання* – визначає дії щодо забезпечення відповідного навчання персоналу [1-10].

Основна частина. Розглянемо надійність програмного забезпечення інформаційних систем. Програми для сучасних інформаційних систем можуть нараховувати значну кількість (сотні, тисячі, десятки та сотні тисяч) простих команд. Під час написання програм можуть по різним причинам появитися помилки. В середовищі програмістів з гумором говорять, що немає програм без помилок, а є програми із не виявленими помилками. Грубі помилки виявляються на стадії відпрацювання програми, але перевірити програму на всіх можливих режимах, як правило, не удається, то нема впевненості, що всі помилки в ній знайдені. При цьому використовується статистичний підхід до аналізу процесу виявлення помилок в програмі. Цей процес може бути охарактеризований функцією

$$\frac{f(t)}{R}, \tag{1}$$

де $f(t)$ – кількість виявлених і виправлених помилок в одиницю часу в програмі, яка має R – кількість команд

$$\frac{f(t)}{R} = \frac{d\varepsilon_n}{dt} = \frac{\varepsilon_n(t+\Delta t) - \varepsilon_n(t)}{\Delta t}, \tag{2}$$

де $\varepsilon_n(t)$ – кількість виявлених і виправлених помилок за час t в розрахунку на одну команду.

Відповідно,

$$\varepsilon_n(t) = \frac{1}{R}\int_0^t f(t)dt\,. \tag{3}$$

Функція $f(t)$ може бути дослідно визначена під час відпрацювання програми шляхом фіксації кількості виявлених помилок. Задача визначення $f(t)$ спрощується, якщо

$$f(t) = \frac{\varepsilon_0}{\tau_0} e^{-\frac{t}{\tau_0}}, \tag{4}$$

де ε_0 і τ_0 – параметри $f(t)$, які визначаються під час відпрацювання.

Тоді

$$\varepsilon_n(\tau) = \frac{1}{R}\int_0^{\tau} f(t)dt = \frac{\varepsilon_0}{R}(1 - e^{-\frac{\tau}{\tau_0}}). \tag{5}$$

При $\tau \to \infty$ $\varepsilon_n(\infty) = \frac{\varepsilon_0}{R}$ або $\varepsilon_0 = R\varepsilon_n(\infty)$. Звідки слідує, що ε_0 – це загальне число помилок в програмі перед початком відпрацювання. Так як процес відпрацювання не може бути тривалим, та в програмі завжди буде залишатися деяка кількість помилок

$$\varepsilon(\tau) = \frac{\varepsilon_0}{K} - \varepsilon_n = \frac{\varepsilon_0}{R} e^{-\frac{\tau}{\tau_0}}, \tag{6}$$

де $\varepsilon(\tau)$ – кількість знайдених помилок в розрахунку на одну команду. Якщо передбачити, що похибки рівномірно розподілені по всій програмі, то ймовірність появи похибки за час Δt буде пропорційна швидкодії δ інформаційної системи (середньому числу змін в одиницю часу команд) і кількості залишених в програмі помилок, тобто

$$P(\tau) = \varepsilon(\tau)\delta\Delta t. \tag{7}$$

Проводячи аналогію між процесом появи помилок і відмов об'єктів $(P(t) = \lambda\Delta t)$ можливо зробити висновок, що інтенсивність похибок $\varepsilon(\tau)$ не залежить від часу t і визначається тільки інтервалом Δt, на якому оцінюється імовірність появи помилки. Звідки, наробіток на «відмову», котре обумовлене появою помилки в програмі буде складати

$$T(\tau) = \frac{1}{\varepsilon(\tau)\delta} = \frac{R}{\varepsilon_0\delta} e^{\frac{\tau}{\tau_0}}. \tag{8}$$

Аналіз зміни $T(\tau)$ може служити основою для вибору часу τ відпрацювання програми, а саме, відпрацювання закінчується, якщо величина $T(\tau)$ становиться достатньо великою.

У випадку, коли вдається оцінити матеріальні втрати C_n від появи помилки в розрахунках, то час τ відпрацювання можливо оцінити кількісно наступним чином. За час T_p роботи програми вона відмовить $\frac{T_p}{T(\tau)}$ разів, що викличе сумарну втрату $C_n \frac{T_p}{T(\tau)}$. Процес відпрацювання програми вимагає затрат часу обчислень та інших затрат пов'язаних з ним. Якщо вартість одного часу відпрацювання позначити C_0, то за час τ таких затрат буде $C_0\tau$. Відповідно, загальна втрата C від помилки і затрат на відпрацювання програми будуть

$$C = \frac{C_n T_p}{T(\tau)} + C_0\tau = \frac{C_n T_p \varepsilon_0 \delta}{R} e^{-\frac{\tau}{\tau_0}} + C_0\tau. \tag{9}$$

Звідки $\frac{dC}{d\tau} = \frac{-C_n N_p \varepsilon_0 \delta}{R\tau_0} e^{-\frac{\tau_M}{\tau_0}} + C_0 = 0$, або $\tau_M = -\tau_0 \ln \frac{C_0 R \tau_0}{C_n T_p \varepsilon_0 \delta}$,

де τ_M – час відпрацювання, котра забезпечить мінімум C.

У цьому випадку, коли необхідно виключити помилку в програмі бажано використовувати «резервування». Одна задача вирішується декількома програмами, кожна з який розробляється незалежними групами програмістів та в основу котрих покладені різні алгоритми, а результати розрахунків програм порівнюються та рахуються вірними, коли вони співпадають. Так як поява помилки в програмах є подією маловірогідною, то поява двох або більше таких подій є подією практично неможливою. Застосування стійких до збоїв програм. Стійкі до збоїв програм одержують, як правило, шляхом багаторазового повторення обчислень на рівні мікро операцій, операцій, команд, модулів програм або всієї програми [7].

Продуктивність інформаційної системи при використанні методу подвійного виконання залежить від числа модулів, на які розбивається програма. Дійсно, велика довжина модулів обумовлює і досить велику ймовірність появи збою. Отже, замість двох необхідно буде три і більше рази повторювати обчислення, через що час вирішення задачі буде збільшуватися. З іншого боку, при малій довжині модулів значна частина часу буде йти на порівняння і запис у пристрої пам'яті результатів обчислень, виконаних на окремих модулях програми[1, 2, 11-15].

У зв'язку з цим виникає задача знаходження оптимального числа модулів, на яке варто розбивати програму і при якому час T_P вирішення задачі буде мінімальним. Введемо позначення: T – час вирішення задачі за однократне виконання програми; t – тривалість обчислень на одному модулі; $p(t)$ – ймовірність відсутності збою за час t. Тоді відношення $\frac{T}{t}$ буде визначати число модулів, на яке розбивається програма. Визначимо ймовірності дво-, три-, або навіть, i-кратного повторення виконання якого-небудь модуля програми. Якщо збої – незалежні події, то ймовірність того, що даний модуль програми буде виконуватись двічі, дорівнює ймовірності відсутності збою при першому і другому виконаннях, тобто

$$p_2(t) = p_1^2(t). \tag{10}$$

У подальшому ймовірність $p_1(t)$ при фіксованому t буде позначатись як p_1. Аналогічно, g дорівнює ймовірності того, що в одному із двох попередніх обчислень відбувся збій, а в третьому обчисленні отримано правильний результат, тобто

$$p_3 = 2p_1^2(1-p_1) = 2p_1^2 q, \tag{11}$$

де $q = 1 - p$. В загальному випадку p_3 дорівнює ймовірності того, що в i-му і одному з попередніх обчисленнях збої були відсутні, а в інших минулих збої були, тобто

$$p_3 = (i-1)p_1^2 q^{i-2}. \tag{12}$$

Отже, середнє число обчислень буде дорівнювати

$$A = \sum_{i=2}^{\infty} p_i = \sum_{i=2}^{\infty} i(i-1)p_1^2 q^{i-2}. \tag{13}$$

Легко показати, що $\frac{A}{p_1^2} = \frac{2}{(1-q)^3}$. Звідси маємо $A = \frac{2}{p_1}$.

Таким чином, витрати часу на обчислення складатимуть $\frac{2T}{p_1}$. Час T_3, необхідний для виконання операцій порівняння і запису проміжних обчислень у запам'ятовуючому пристрої, залежить від типу запам'ятовуючому пристрої, що використовується, кількості k проміжних результатів і числа кроків $\frac{T}{t}$ програми, тобто

$$T_3 = \frac{T}{t} f\left(k, \frac{T}{t}\right), \tag{14}$$

де $f\left(k, \frac{T}{t}\right)$ – середній час виконання операцій порівняння і звернення до пристрою пам'яті для запису результатів одного модуля програми. Якщо вважати, що $f\left(k, \frac{T}{t}\right) = const = a$, то

$$T_p = \frac{2T}{p_1(t)} + \frac{T_a}{t} = T\left(\frac{2}{p_1 t} + \frac{a}{t}\right). \tag{15}$$

Для деяких типів інформаційних систем експериментально встановлено, що

$$p(t) = e^{-\lambda \cdot t}, \tag{16}$$

де λ – інтенсивність збоїв. В цьому випадку T_P приймає мінімальне значення для t, яке можна визначати з рівняння

$$\frac{dT_P}{dt} = 2\lambda e^{\lambda \cdot t} - \frac{a}{t^2} = 0. \tag{17}$$

Отже значення T_P, можна визначити оптимальну довжину дільниці програми і відповідне їй число $\frac{T}{t}$ дільниць, при якому T_P буде мінімальним.

Причиною невірного функціонування інформаційної системи може бути наявність в ній так званих вірусних програм, програм призначених для надмірного викривлення результатів розрахунків, видалення файлів, створення умов для ненормального функціонування інформаційної системи.

У відповідності з кодифікатором злочинів інформаційних систем Генерального секретаріату Інтерполу, віруси відносяться до категорії QD – зміна даних інформаційних систем, всередині якої вони класифікуються наступним чином:

– QDL – логічна бомба;
– QDT – троянський кінь;
– QDV – вірус інформаційної системи;
– QDW – черв'як інформаційної системи;
– QDZ – інші види зміни даних.

Логічна бомба – здійснює тайне вставлення в програму набору команд, яке повинно спрацювати лише одного разу, але при визначених умовах.

Троянський кінь – здійснює введення до чужої програми таких команд, які дозволяють здійснити інші, не плановані власником програми функції, але одночасно зберегти і попередню працездатність.

Вірус інформаційної системи – це спеціально написана програма, котра може «приписувати» себе до інших програм (тобто «заражати» їх), розмножуватися і народжувати нові віруси для виконання різних небажаних дій в інформаційній системі.

Черв'як інформаційної системи – представляє собою спеціальну самостійно розповсюджуючи програму, що здійснює зміни даних або програм інформаційної системи, без права на це, шляхом передачі, впровадження або розповсюдження за допомогою мережі інформаційних систем [13].

Висновки. Доля помилок або зависань інформаційної системи за рахунок вірусів складає приблизно від 10 до 30%. Відомо більш 10.000 вірусів і близько 100 антивірусних програм, призначених для боротьби з ними. Існують віруси (самошифруючі, поліморфні віруси і макровіруси тощо), здатні протидіяти противірусним програмам. Один із різновидів таких вірусів «поселення» в противірусній програмі. Зазвичай антивірусна програма видає сигнал про своє власне зараження, якщо таке зараження здійснюється. Час необхідний для вилікування від вірусу складає в середньому від 15 до 30 хв. Самий небезпечний вірус є вірус, який знаходиться у виконавчому файлі. В основному віруси «працюють» коректно і не викликають зависань інформаційної системи. Але серед них попадаються такі, котрі повністю стирають системні області жорсткого диску або підкаталоги інформаційних масивів. У 90% випадків віруси впроваджуються в інформаційні системи через мережі. Причому локальні мережі самі по собі не є розповсюджувачами вірусів. Але користувачі, котрі працюють із магнітними носіями, котрі вражені вірусом, доставляють багато клопоту такій мережі.

Ознакою зараження інформаційної системи вірусом є:

– зростання помилок і зависання інформаційної системи;

– сповільнення загрузки програми;

– неполадки (різні сповільнення і похибки) при роботи принтера;

– мигання лампочки дисковода, коли не повинні проходити операції читання/запису;

– зміна розмірів програм, що виконуються, зменшення основної доступної пам'яті.

Самими короткими є руйнуючі віруси, їх довжина не перевищує 20 кілобайт. Самі довгі віруси досягають 100 кілобайт і більше. В останній час багато клопоту доставляють користувачам макровіруси.

Якість противірусної програми визначається по наступним характеристикам, наведеним у порядку зменшення їх важливості:

– надійність і зручність роботи (відсутність технічних проблем, вимагаючи від користувача спеціальної підготовки);

– кількість знайдених вірусів всіх типів, можливість перевірки файлів документів/таблиць, упакованих файлів, також можливість лікування заражених об'єктів;

– швидкість роботи і різні корисні функції.

Коли користувач має декілька ефективних противірусних програм і користується ними, самим надійним захистом від вірусів є профілактика зараження:

– регулярне створення резервних копій (наприклад, один раз в неділю – повне, кожен день – часткове копіювання). Наявність незаражених копій дозволить просто переписати «хворий» файл, наявність заражених, але не порчених копій дозволить відновити файли після видалення вірусу;

– створення резервних копій інсталяційних магнітних носіїв інформації перед установкою нового програмного забезпечення (при встановлені програми на заражену інформаційну систему вихідний магнітний носій інформації може заразитися під час інсталяції);

– перевірка по E-mail файлів, які пересилаються, на наявність вірусів;

– застосування захищених від запису магнітних носіїв інформації під час копіювання файлів на жорсткий диск. Це попередить проникнення вірусу на магнітний носій і послідуюче зараження інших інформаційних систем;

– перевірка магнітних носіїв перед загрузкою з них файлів;

– постійне використання резидентної частини противірусної програми, котра слідкує за всім підозрілим при роботі інформаційної системи.

ЛІТЕРАТУРА:

1. Азарсков В. Н. Надежность систем управления и автоматики / В. Н. Азарсков, В. П. Стрельников – К.: НАУ. – 2004. – 164 с.

2. Гнеденко Б.В. Надежность. Энциклопедичексий справочник «Автоматизация производства и промышленная электроника» / Б. В. Гнеденко, Я. Б. Шор. – Т. 2, Советская энциклопедия, 1963, с. 348-353.

3. Иыуду К. А. Надежность, контроль и диагностика вычислительных машин и систем / Иыуду К. А. – М.: Высш. шк., 1989. – 216 с.

4. Коваленко И. Н. Методы расчета высоконадежных систем / И. Н. Коваленко, Н. Ю Кузнецов. – М.: Радио и связь, 1988. – 176 с.

5. Креденцер Б. П. Прогнозирование надежности с временной избыточностью / Креденцер Б. П. – К.: Наукова думка, 1978. – 238 с.

6. Кудрицкий В. Г. Прогнозирование надежности радиоэлектронных устройств / Кудрицкий В. Г. – К.: - «Техніка». – 1973. – 156 с.

7. Липаєв В. В. Качество программного обеспечения / Липаєв В. В. – М.: Финансы и статистика – 1993. – 262 с.

8. Модели и методы оптимизации надежности сложных систем / Под ред. И.Н. Коваленка. – К.: Наукова думка, 1993. – 312 с.

9. Процеси життєвого циклу програмного забезпечення. ДСТУ 3918-1999 (ISO/IEC 12207: 1995) . – К. : Держстандарт України, 2000. VI. 49 с.

10. Пупков К. А. Оценка и планирование эксперимента / К. А. Пупков, Г. А. Костюк .– М.: Машиностроение. – 1977.

11. Сборник переводов «Методы введения избыточности для вычислительных систем». / Под ред. В. С. Пугачева., 1966. – 326 с.

12. Стрельников В. П. Оценка и прогнозирование надежности электронных элементов и систем / В. П. Стрельников, А. В. Федухин– К,: Логос, 2002. – 486 с.

13. Тарасенко В. П. Надійність комп'ютерних систем. / Тарасенко В. П., Маламан А. Ю., Чередниченко Ю. П., Корнейчук В. І. – К.: «Корнійчук», 2007. – 256 с.

14. Хевиленд Р. Инженерная надежность и расчет на долговечность / Хевиленд Р. – М.: Изд-во «Энергия». – 1966.

15. Черкесов Г. Н. Надежность аппаратно-программных комплексов / Черкесов Г. Н. – Санкт-Петербург.: Питер. – 2005. – 479 с.

Рецензент: д.т.н., проф. Хорошко В.О., Національний авіаційний університет

Зорина Т.И., к.т.н. **Петров А.О., Сергиеня К.В.**

ЖИЗНЕННЫЙ ЦИКЛ И НАДЕЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В статье рассмотрены основы надежности программных систем и программного кода, исследованы связи между показателями экономичности, эффективности и надежности на основе разработки соответствующих методик. Проанализированы организационные процессы жизненного цикла программного продукта, применяемого организацией для разработки и реализации базовой структуры управления информационной безопасностью. Выявлены причины неверного функционирования информационной системы - наличие в ней так называемых вирусных программ, программ предназначенных для чрезмерного искажения результатов расчетов, удаление файлов, создание условий для ненормального функционирования информационной системы.

Ключевые слова: информационная безопасность, управление, программное обеспечение, надежность, отказоустойчивость, защищенность.

Zorina T.I., Ph.D. Petrov A.A., Serhiyenya K.V.

LIFE CYCLE AND SOFTWARE RELIABILITY

This article covers the basics of reliability of software systems and software code, the connections between indicators of economy, efficiency and reliability through the development of appropriate techniques. Analyzed the organizational processes of the life cycle of a software product used by the organization to develop and implement the basic structure of information security management. The causes of improper functioning of the information system - the presence in it of so-called virus programs, programs designed to excessive distortion of the results of calculations, delete files, create conditions for the abnormal functioning of the information system.

Keywords: information security, management, software reliability, resiliency, security.