УДК 004.413

О.В. Кучер, О.В. Крутих, В.В. Сокол, Н.С. Лєсна

Харківський національний університет радіоелектроніки, Харків

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПІДТРИМКИ КУРСОВОГО ПРОЕКТУВАННЯ У ГАЛУЗІ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

В даній роботі проводиться аналіз проблем, що виникають при наближенні процесу виконання курсового проекту у галузі комп'ютерної інженерії до розробки реальних програмних продуктів. Для їх вирішення пропонується використання інформаційної технології, яка дозволяє організовувати виконання курсових проектів на основі методологій розробки програмного забезпечення TSP. Для збільшення ефективності навчання пропонуються шляхи оцінки об'єму та якості коду та можливість їх використання для студентів різних курсів. Також аналізується можливість оцінки щоденного навантаження студента.

Ключові слова: *TSP, курсовий проект, метрики програмного забезпечення, професійна мотивація студентів, оцінка навантаження.*

Вступ

Процес навчання стає все більш складним і менш надійним: навчання у вищих навчальних закладах традиційними методами все частіше не дає задовільних результатів, тобто страждає якість навчань. В оцінці якості навчання можна виділити три основні показники:

а) загальний стан якості навчання в цілому по навчальному закладу;

б) якість навчальної роботи викладачів;

в) якість знань і навичок студентів.

Розглядаючи вищий навчальний заклад як систему управління, необхідно особливо відзначити область перевірки знань і навичок студентів – тобто, зворотний зв'язок блоку підвищення якості знань і навичок студентів. Прийняття рішень в даній області традиційно практично цілком покладається на викладача, незважаючи на можливість часткової автоматизації цього процесу. Цей підхід має серйозні недоліки:

а) суб'єктивізм, що полягає в тому, що різні викладачі можуть по-різному оцінити здібності одного і того ж студента;

б) відсутність широкої шкали оцінювання;

в) «локальність» оцінки, яка має сенс тільки в рамках невеликої групи, що оцінюються;

г) трудомісткість масового тестування.

Організація навчання програмуванню має специфіку. Вона полягає у тому, що знання з програмування мають більше практичну направленість, аніж теоретичну. Найкращим свідченням ступені знань людини з програмування є її уміння складати прикладні програми, а не загальні знання з програмування. Тому під час навчання програмуванню студент виконує велику кількість практичних завдань. Одним з найбільш важливих практичних завдань є курсове проектування.

Курсові проекти виконуються з метою закріплення, поглиблення і узагальнення знань, отриманих студентами за час навчання, та їх застосування до комплексного вирішення конкретного фахового завдання.

У галузі комп'ютерної інженерії він полягає у розробці програмного продукту. Працюючи над курсовим проектом, студент набуває практичних навичок з розробки програмного забезпечення. Отримані навички знадобляться йому під час роботи над реальними проектами.

Для підвищення ефективності навчання програмуванню є важливим збільшити кількість цих навичок. Це можна зробити за допомогою наближення процесу виконання курсового проекту до розробки реальних програмних продуктів.

Для цього необхідно вирішити наступні проблеми:

а) реальні програмний продукт має багато характеристик (якість, об'єм коду), які оцінюються та контролюються під час розробки. Для учбового проекту їх потрібно вимірювати також;

б) не можна однаково оцінювати студентів різних курсів навчання, тому що студенти старших курсів повинні більш пристосовуватись до умов, що склалися в їх професійній галузі, а студенти молодших курсів більше спрямовані на вивчення предметної галузі своєї професії та надбання базових навичок створення програмних продуктів;

в) на об'єктивність оцінювання впливає відсутність автоматизованості процесу перевірки трудомісткості виконаної роботи;

г) робота над реальними проектами ведеться рівномірно, проте вести контроль за рівномірністю роботи усіх студентів для викладача складно.

Таким чином, виникає необхідність у такій організації роботи студента, яка дозволить вирі-

шити розглянуті проблеми. Для цього необхідно створити інформаційну технологію підтримки курсового проектування, яка б дозволяла організовувати виконання курсових проектів на основі реальних методологій розробки програмного забезпечення.

Розгляд існуючих аналогів. Аналіз існуючих технологій підтримки курсового проектування виявив, що переважна більшість з них не пристосована до використання при створенні програмних систем внаслідок різних причин. Основною причиною є те, що кожна з існуючих методологій була розроблена для використання у своїй, специфічній галузі.

Наприклад, система Favorite Subject була розроблена для дисципліни «Обчислювальна техніка», і тому весь навчальний матеріал, що надається системою, та завдання, які можна виконувати з її допомогою, стосуються лише проектування інтегральних схем.

Ще одна технологія, SolidWorks Educational Edition, дозволяє організовувати командну розробку курсових проектів, використовує єдину базу даних, яка зберігає результати роботи студентів.

Проте ця технологія використовує у своїй основі середовище автоматизованої розробки SolidWorks, тому вона пристосована лише для моделювання інженерних розрахунків у машинобудуванні. Область її використання цієї технології обмежується ВНЗ або спеціальностями, пов'язаними з будівництвом, машинобудуванням, промисловим дизайном тощо.

Постановка задачі. Завданнями даної роботи є:

а) створення інформаційної технології, що дозволить організувати роботу студентів та викладачів за реальними методологіями розробки ПЗ та проводити оцінку програмного коду за допомогою метрик;

б) дослідження існуючих метрик оцінювання якості програмного коду;

в) аналіз сильних і слабких сторін кожної метрики;

г) оцінка ефективності застосування конкретних метрик до певних груп студентів;

д) дослідження методів аналізу щоденного навантаження студента;

е) вибір найбільш ефективного методу оцінки щоденного навантаження студента.

Інформаційна технологія. Взаємодію студента та викладача в процесі курсового проектування необхідно організувати на основі реальних методологій розробки програмного забезпечення. За основу можна використати технологію, розглянуту у [1].

У ній процес розробки виконується за методологією Team Software Process.

На рис. 1 зображена схема роботи над курсовим проектом за цією технологією.

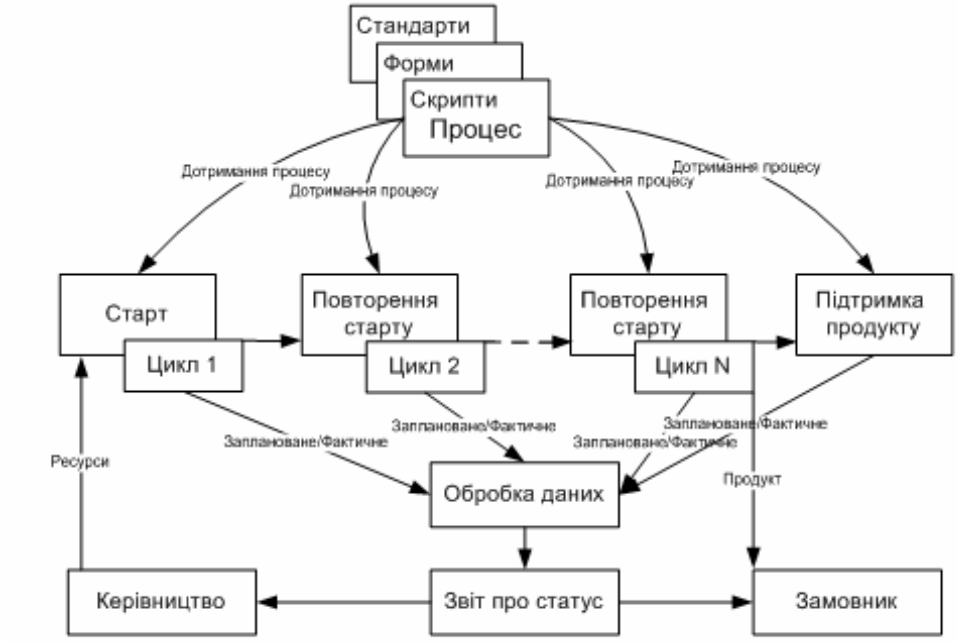

Рис. 1. Схема потоків інформації під час курсового проектування

Проте для вирішення проблеми оцінювання студентів та аналізу навантаження необхідно модифікувати етап збору та обробки даних.

Треба визначити, які саме метрики можна збирати та вимірювати, та як їх використовувати.

Аналіз метрик програмного забезпечення

Метрика програмного забезпечення – це чисельне значення деякої властивості програмного забезпечення або його специфікацій. Оскільки кількісні методи добре зарекомендували себе в інших областях, багато теоретиків і практиків інформатики намагалися перенести цей підхід і в розробку програмного забезпечення.

Кількість рядків коду. Кількість рядків коду – це метрика програмного забезпечення, що використовується для вимірювання його обсягу за допомогою підрахунку кількості рядків у тексті вихідного коду [2].

Як правило, цей показник використовується для прогнозу трудовитрат на розробку конкретної програми, або для оцінки продуктивності праці вже після того, як програма написана.

Традиційно вважається, що має сенс порівнювати розміри проектів лише з точністю до порядку. Існує два основних методи розрахунку цієї метрики: підрахунок фізичних і логічних рядків. Фізичними рядками вважаються всі непорожні рядки текстового файлу. Порожні рядки враховуються в тому випадку, якщо в деякій секції їх кількість не перевищує 25%. В іншому випадку, порожні рядки, що перевищують поріг у 25% не враховуються. Вимірюючи логічні рядки коду, робиться спроба порахувати кількість власне операторів у програмі, але, зрозуміло, їх визначення залежить від конкретної мови програмування. Наприклад, найпростіший спосіб порахувати кількість логічних рядків коду у Сі-подібних мовах полягає в підрахунку числа точок з комою, якими закінчуються оператори.

Фізичні рядки коду інтуїтивно зрозуміліші і їх простіше рахувати. Однак, результати підрахунку істотним чином залежать від правил оформлення та форматування вихідного коду. Логічні рядки коду менш чуттєві до цього. Результати, одержані на основі підрахунку кількості рядків коду, бувають часто суперечливими, особливо коли їх застосовують некоректно. Експерименти багаторазово підтвердили той факт, що дана метрика добре корелює з трудовитратами – програми з великою кількістю рядків коду потребують більше часу на розробку. Тому застосування цієї метрики в процесі оцінки трудовитрат є виправданим. Однак, кореляція з функціональністю вже не настільки явна. Досвідчені програмісти, як правило, вважають за краще писати менше коду, досягаючи того ж самого результату. І якщо при оцінці продуктивності досить великої команди різниця в класі розробників може нівелюватися, то застосування цієї метрики для оцінки продуктивності індивідуума представляється неадекватним.

Порівняно недавно з'явився ще один аспект даної проблеми – різниця між програмним кодом, написаним розробником власноруч, і згенерованим автоматично. Сучасні засоби розробки досить часто надають можливість автоматично створювати великі обсяги коду всього лише кількома кліками миші. Найбільш яскравим представником даних систем є засоби візуальної розробки графічного інтерфейсу користувача.

Цикломатична складність. Цикломатична складність програми – структурна (або топологічна) міра складності програм, що використовується для вимірювання якості програмного забезпечення, заснована на методах статичного аналізу коду. Вона дорівнює збільшеному на одиницю цикломатичному числу графа програми.

При обчисленні цикломатичної складності використовується граф потоку управління програми: вузли графа відповідають неподільним групам команд програми і орієнтованим ребрам, кожне з яких сполучає два вузли і відповідає двом командам, друга з яких може бути виконана одразу після першої.

Цикломатична складність може також бути застосована для окремих функцій, модулів, методів або класів у межах програми. Для цього здійснюється тестування кожного лінійного незалежного маршруту в межах програми; в цьому випадку, кількість тестів має дорівнювати цикломатичній складності програми.

Цикломатична складність частини програмного коду – кількість лінійно незалежних маршрутів в межах програмного коду. Наприклад, якщо вихідний код не містить ніяких точок рішень, таких, як оператор IF або цикли FOR, то складність дорівнює одиниці, оскільки є тільки єдиний маршрут. Якщо код має єдиний оператор IF, що містить просту умову, то має бути два маршрути: один через оператор IF з оцінкою TRUE, і один – як FALSE.

Математично, цикломатична складність структурного програмування визначається за допомогою посилань від базового блоку програми на орієнтований граф і за допомогою ребер між двома основними блоками, якщо управління може переходити з першого на другий граф управління потоком програми.

Тоді складність визначається за формулою:

$$M = E - N + 2P, \qquad (1)$$

де M – цикломатична складність; E – кількість ребер у графі; N – кількість вузлів у графі; P – кількість компонентів зв'язності.

У іншому формулюванні використовується граф, в якому кожна точка виходу з'єднана з точкою входу. У цьому випадку вважається, що граф є сильнозв’язаним і цикломатична складність про-

грами дорівнює цикломатичному числу цього графа, яке визначається за формулою:

$$M = E - N + P. \quad (2)$$

Останнє може розглядатися як обчислення числа лінійно незалежних циклів, які існують в графі, тобто тих циклів, які не містять у собі інших циклів. Треба враховувати, що кожна точка виходу з циклу стає точкою входу в нього, тобто є принаймні одна ітерація циклу для кожної точки виходу.

Може бути показано, що цикломатична складність будь-якої структурованої програми з тільки однією точкою входу і однією точкою виходу еквівалентна кількості точок рішення (умовних циклів), що містяться в цій програмі, плюс один.

Цикломатична складність може бути поширена на програму з численними точками виходу; в цьому випадку вона визначається за формулою:

$$M = \pi - s + 2, \quad (3)$$

де π – число точок рішення в програмі; s – число точок виходу.

Кількість класів. Ця метрика може використовуватися в мовах програмування високого рівня, які відповідають принципам об'єктно-орієнтованого програмування (C++, Java, C# і т.п.).

Кількість методів. Як і метрика кількості класів, може використовуватися в мовах високого рівня. Оптимальну кількість методів (середнє значення на одиницю об'єму програми) важко визначити. Але враховуючи умову можливості підтримки коду, весь код методу повинен приблизно вміщатися на екрані монітора, а методи повинні концентрувати лише частки алгоритму, яких більш ніде нема в програмі.

Час. Ця метрика характеризує час, що був витрачений на розробку програмного забезпечення. Одиницею. вимірювання будемо вважати години.

Індекс підтримуваності. Ця метрика розраховується за формулою:

$$171 - 5,2 * \ln(HoV) - \\ -0,23 * (CC) - 16,2 * \ln(LoC)), \quad (4)$$

де HoV – це об'єм програми за метриками Холстеда; CC – цикломатична складність програми; LoC – кількість логічних рядків коду [3].

Значення індексу коливається від 171 до необмежених негативних чисел. Але, як правило, код, індекс якого прямує до 0, важко підтримувати. Тому різниця між кодом із значенням 0 і негативним значенням не корисна. У результаті зменшення корисності негативних чисел і бажання зберегти метрику як можна більш чіткою, було вирішено розглядати всі нульові показники (або менше 0) як 0 і зробити базовим значенням не 171, а 100. Тоді діапазон буде від 0 до 100. Таким чином, використовується формула:

$$\text{Max}(0, \quad (171 - 5,2 * \ln(HoV) - \\ -0,23 * (CC) - 16,2 * \ln(LoC)) * 100 / 171). \quad (5)$$

Метрики Холстеда були розроблені як засіб визначення кількісної міри складності безпосередньо від операторів і операндів в модулі для оцінки складності програмного модуля прямо з вихідного коду.

Ці метрики були одними з перших, що аналізували складність коду. Вони найбільш часто використовуються в якості метрики підтримуваності. Очевидно, що метрики Холстеда також корисні при розробці програми, щоб оцінити якість коду.

Метрики Холстеда засновані на інтерпретації вихідного коду, як послідовності лексем та класифікації кожної з них в якості оператора або операнда. При цьому підраховується кількість унікальних (різних) операторів (N1), кількість унікальних (різних) операндів (N2), загальна кількість операторів (N1), загальне число операндів (N2).

Кількість унікальних операторів і операндів (N1 і N2), також як і загальне число операторів і операндів (N1 і N2), розраховуються шляхом збору частот кожного оператора і операнда вихідної програми.

Розмір словника n – це сума кількості унікальних операторів і операндів розраховується за формулою:

$$n = n1 + n2. \quad (6)$$

Обсяг програми (V) – це інформаційний зміст програми, виміряний в математичних бітах. Він розраховується за формулою:

$$V = N * \log_2(N), \quad (7)$$

де N – розмір словника.

При застосуванні метрик Холстеда частково компенсуються недоліки, пов'язані з можливістю запису однієї і тієї ж функціональності різною кількістю рядків і операторів.

Професійна мотивація

У системі освіти під професійною мотивацією розуміється сукупність факторів і процесів, які, відбиваючись у свідомості, спонукають і скеровують особистість до вивчення майбутньої професійної діяльності.

Професійна мотивація виступає як внутрішній рушійний чинник розвитку професіоналізму і особистості, так як тільки на основі її високого рівня формування є можливим ефективний розвиток професійної освіченості та культури особис-

тості. При цьому під мотивами професійної діяльності розуміється усвідомлення актуальних потреб особистості, які задовольняються за допомогою виконання навчальних завдань і спонукають її до вивчення майбутньої професійної діяльності. Якщо студент вважає свою професію гідною і значущою для суспільства, це, безумовно, впливає на те, як складається його навчання.

В ході професійного становлення під час навчання у ВНЗ відбуваються зміни мотиваційної сфери студентів та переосмислення основних цінностей. Достатньо високі показники професійної мотивації у студентів п'ятого курсу свідчать про високий ступінь психологічної готовності до майбутньої професійної діяльності.

Вибір метрик мотивації для студентів першого курсу. Показники професійних і навчальних цінностей виступають в ролі мотивів і керують навчальною діяльністю студентів першого курсу. Але рівень професійної освіти у студентів ще початковий. Це пізнавальний етап життя. Знання, які здобуваються на першому курсі, є узагальненими, абстрактними, вони створюють основу для подальшого, більш спеціалізованого розвитку студентів.

Виходячи з цих обставин, найбільш вдалим вибором буде комбінування метрик кількості класів та кількості методів, тому що на першому курсі студенти вчаться основам об'єктно-орієнтованого програмування, основними компонентами якого є об'єкти (екземпляри класів) та їх поведінка, яка обумовлена наявністю методів. Також на першому курсі студенти повинні вивчити ази правильного написання коду.

Для уникання складності коду треба вміти проводити декомпозицію класів (тобто розділяти код на логічні одиниці, які містяться в окремих класах) та розбивати код методу на менші частки, обсяг яких, за рекомендаціями написання «зручного» коду, не повинен перевищувати розміру екрану монітора.

Вибір метрик мотивації для студентів другого курсу. Студенти другого курсу вже мають навички створення програмних продуктів, але перебувають все ще на начальному освітньо-кваліфікаційному рівні, тому потребують закріплення навичок трансформації об'єктів реального світу в об'єктну модель програм, які створюють. Але, все ж таки, початковий рівень створення програм у них вже є.

Тому необхідно вже з ранніх етапів навчання привчати студентів до командної розробки, яка включає процеси підтримки вже існуючого коду та створення нового, який в майбутньому буде легко підтримувати, він буде зрозумілим і написаним за правилами.

Виходячи з цього, для студентів другого курсу більше підходять такі метрики, як кількість класів та індекс підтримуваності програмного коду.

Вибір метрик мотивації для студентів третього курсу. Студенти третього курсу вже більше готові до майбутньої професійної діяльності, ніж студенти попередніх курсів. Але не слід забувати і про те, що росте рівень складності курсових проектів. Алгоритмічна база реалізації програмних продуктів ускладнюється, а вимоги до майбутнього спеціаліста зростають. Це обумовлено тим, що в реальній професійній діяльності при створенні програмних продуктів ключову роль грають два основних чинника: час, витрачений на розробку проекту, та вартість функціональної одиниці програми.

Тому треба велику увагу приділяти умовам командної розробки, а саме створенню коду, який буде в подальшому легко підтримувати, часовим обмеженням на розробку, складності кінцевого продукту.

Виходячи з цих умов, для студентів третього курсу були вибрані такі метрики, як підтримуваність програмного коду, час розробки та цикломатична складність.

Вибір метрик мотивації для студентів четвертого курсу. З кожним наступним курсом студенти потребують умов, наближених до тих, в яких вони будуть працювати в подальшому. Цей фактор є ключовим в процесі підготовки спеціалістів технічного навчального закладу. Виходячи з того, що в реальних умовах основною метрикою продуктивності є час, що був затрачений на розробку, для студентів четвертого курсу було обрано ті ж метрики, що й для студентів третього (а саме: підтримуваність програмного коду, час розробки та цикломатична складність), але більш вагомою метрикою вже буде саме час розробки програми.

Вибір метрик мотивації для студентів п'ятого курсу. Студенти п'ятого курсу вже є фахівцями, майже готовими до самостійної професійної діяльності. Також, психологічно студент налаштований вже на роботу і на ті умови, що існують в процесі створення реальних комерційних програм. На рівні з цим, на п'ятому курсі рівень складності курсових проектів найбільший із усіх попередніх і об'єм коду буде великим. Тому доцільно буде використати наступні метрики: підтримуваність програмного коду, час розробки та кількість логічних рядків коду.

Дослідження методів аналізу навантаження студента в часі

Для реальних проектів важливо вміння працювати за графіком. Зрив термінів розробки про-

грамного забезпечення є, нажаль, звичним явищем в практиці створення програмних продуктів. Це явище шкідливо для всіх учасників процесу. Причинами зриву термінів є велика кількість факторів. Одним з них є неправильне розподілення навантаження під час розробки і, як наслідок, невідповідність виконання роботи зазначеному плану [4].

При роботі над курсовим проектом студент сам планує процес розробки та стежить за виконанням плану. З різних причин цей процес не завжди є правильним та ефективним. Через це погіршується якість виконання курсового проекту. Тому однією з складових оцінки якості курсового проектування повинна бути оцінка ефективності розподілення навантаження у часі.

Курсове проектування виконується в обмежений період часу. Зважаючи на це, план розробки обов'язково пов'язаний з часом. Кожна задача у плані (окрім значень метрик) має час початку та час завершення. Таким чином, заплановані та реальні значення метрик залежать від часу. Під час розробки програмного забезпечення показники, що характеризують ступінь завершеності системи, можуть змінюватися у будь-який момент часу. Тому функція метрики від часу є неперервною. Проте план розробки містить кінцеву кількість ітерацій та задач, для яких визначаються значення показників завершеності розробки. Тому функція залежності запланованих значень метрики від часу є дискретною.

Контроль за виконанням плану також здійснюється не постійно. Є логічним перевіряти значення показників завершеності розробки періодично. Таким чином, функція реальних показників також є дискретною.

Задачу аналізу ефективності розподілення навантаження з математичної точки зору можна розглядати таким чином. Заплановані та реальні показники вибраної метрики можна представити у вигляді дискретних функцій однієї змінної. Аргументом такої функції виступає час (контрольні точки), а значенням є показник, що вимірюється. Мірою якості розподілення часу є число, що характеризує ступінь близькості двох функцій. Необхідно провести аналіз математичних методів, що дозволяють розрахувати ступінь розбіжності двох функцій та визначити метод, що найбільш відповідає умовам конкретної задачі.

Метод порівняння довжин функцій. Сенс даного методу полягає у розрахунку для кожної функції її довжини. Мірою їх близькості є різниця між цими довжинами. Проте вірний результат за цим методом можна отримати, якщо функція зміни запланованих показників, що аналізується, є лінійна. Це є серйозним недоліком даного методу.

Хаусдорфова відстань. Хаусдорф у своїй книзі дав визначення відстані між двома підмножинами метричного простору.

Нехай (M, p) – метричний простір, де p(a, b) – відстань між двома елементами $a, b \in M$.

Нехай підмножина $A \subset M$.

Позначимо через U(ε, A) множину усіх точок x $x \in M$, що задовольняють умову $p(x, A) \le \varepsilon$, де $\varepsilon \ge 0$.

Нижня межа h(A, B) чисел ε таких, що $U(\varepsilon, A) \supset B$ та $U(\varepsilon, A) \supset A$ називається хаусдорфовою відстанню.

Особливістю хаусдорфової відстані є те, що вона залежить від відстаней кожної точки однієї функції до кожної точки іншої функції. Тому цей метод неможливо використовувати для оцінки навантаження студента.

Міра похибки функцій при рівномірному приближенні. Цей метод заснований на виборі максимального відхилення однієї функції від іншої [4].

Нехай існують дві функції – f(x) та φ(x). Q – область визначення даних функцій.

Тоді міру похибки μ(f, φ) можна розрахувати за формулою:

$$\mu(f, \varphi) = \sup_{t \in Q} |f(t) - \mu(t)|. \qquad (8)$$

Ця міра містить у собі вади при використанні її для аналізу навантаження студента у часі. По-перше, він не враховує, яка саме з функцій є більшою, f(x) чи φ(x). Окрім цього, метод не є чуттєвим до відхилень, що є меншими за максимальне.

Критерій близькості при апроксимації функцій методом найменших квадратів. Ще один з алгоритмів апроксимації, у якому використовується міра похибки, є алгоритм найменших квадратів [4]. Для кожного значення аргументу x_i розраховується різниця Δy_i між значеннями двох функцій. Критерій близкості розраховується як

$$\sigma = \sum_{i=1}^{n} (\Delta y_i)^2 .$$

Необхідно враховувати, що критерій близькості при апроксимації функцій методом найменших квадратів використовується при побудові моделі випадкових процесів. Проте прогрес у розробці програмного забезпечення не можна характеризувати як випадковий.

Окрім цього, величина σ розраховується як сума розбіжностей у всіх дискретних точках. Якщо ці точки розподілені нерівномірно за часом, то значення Δy_i матимуть різну вагу. Тому результат не буде точним.

Вибір найбільш ефективного методу оцінки навантаженості студента в часі. Проаналізувавши існуючи методи оцінки близькості функцій, було зроблено висновок, що кожен з них має недоліки при використанні у аналізі навантаження студента у часі. Тому жоден з них неможливо використати без проведення адаптації до умов задачі. Серед розглянутих методів найбільш відповідним є критерій розбіжності при апроксимації методом найменших квадратів.

Як було розглянуто раніше, даний метод має декілька недоліків.

По-перше, при розрахунку суми розбіжностей кожна окрема розбіжність не буде зводитися у квадрат.

По-друге, будуть окремо визначені два значення міри розбіжності – для випадків випередження графіку та відставання від нього. Відставання від графіку характеризуватиме неефективність організації поточної роботи, випередження – помилку при плануванні та аналізу складності програми.

По-третє, контрольні точки можуть бути розподілені у часі нерівномірно. Тому замість дискретних функцій доцільно використовувати неперервні функції $f_1(t)$ та $f_2(t)$, які отримані за допомогою лінійної апроксимації дискретних функції запланованої та реальної зміни метрик коду. При цьому робиться припущення, що між контрольними точками студент працює рівномірно. Такий варіант буде більш точним, тому що процес розробки програмного забезпечення є неперервним. Лише через неможливість постійного вимірювання прогресу доводиться розглядати його як дискретну функцію.

Тому критерій розбіжності буде розраховуватися як інтеграл:

$$\sigma = \int_0^t (f_1(t) - f_2(t))\ dt\ . \qquad (9)$$

Висновки

Розроблена інформаційна технологія дозволяє організовувати виконання курсових проектів на основі реальних методологій розробки програмного забезпечення. Ця технологія дає можливість автоматично проводити аналіз програмного забезпечення за різними характеристиками коду та робити комплексну оцінку на основі цих характеристик. Також існує можливість виконувати аналіз якості розподілення навантаження під час розробки курсового проекту.

Реалізація даної інформаційної технології на практиці дозволить більш ефективно організувати процес виконання курсових проектів з програмування. Технологія дозволяє проводити оцінку як готового проекту, так і проекту, що знаходиться на стадії розробки. Тому використання цієї технології надасть студенту та викладачу можливість краще контролювати хід роботи над курсовим проектуванням.

Список літератури

1. Зозуля В.В. Інформаційна технологія підтримки курсового проектування / В.В Зозуля, О.В. Крутих, І.Н. Келеберда, Н.С. Лєсна // Вісник Херсонського національного технічного університету. – 2010. – №2. – С. 303-308.

2. Дюваль П.М. Неперервна інтеграція / П.М. Дюваль, Э. Гловер. – К.: Вища школа, 2008. – 458 с.

3. Маршалл К. Эффективный тайм-менеджмент / К. Маршалл. – М.: ФАИР-ПРЕСС, 2003. – 336 с.

4. Калинкин Н.Н. Численные методы / Н.Н. Калинкин. – М.: Наука, 1990. – 496 с.

Надійшла до редколегії 17.06.2011

Рецензент: д-р техн. наук, проф. І.В. Гребеннік, Харківський національний університет радіоелектроніки, Харків.

ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ ПОДДЕРЖКИ КУРСОВОГО ПРОЕКТИРОВАНИЯ В ОБЛАСТИ КОМПЬЮТЕРНОЙ ИНЖЕНЕРИИ

А.В. Кучер, А.В. Крутых, В.В. Сокол, Н.С. Лесная

Предлагаются пути оценивания объема и качества кода и возможность их использования для студентов разных курсов. Также анализируется возможность оценивания ежедневной нагрузки студентов.

Ключевые слова: *TSP, курсовой проект, метрики программного обеспечения, профессиональная мотивация студентов, оценка нагрузки.*

INFORMATION TECHNOLOGY OF SUPPORT OF COURSE DESIGNING IN AREA OF THE COMPUTER ENGINEERING

O.V. Kucher, O.V. Krutikh, V.V. Sokol, N.S. Lesna

Proposed the ways of estimating the amount and quality of code and they ability to be used for students from different courses. Also analyzed the possibility of estimating the daily loading of students.

Keywords: *TSP, course project, birth-certificates of software, professional motivation of students, estimation of loading.*