УДК 378:004.42

Стрюк А. М.

Криворізький національний університет, Кривий Ріг, Україна

“ADVANCED COURSE ON SOFTWARE ENGINEERING” ЯК ПЕРША МОДЕЛЬ ПІДГОТОВКИ ФАХІВЦІВ З ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

DOI: 10.14308/ite000702

Проєктування мобільно орієнтованого середовища професійно-практичної підготовки потребує визначення стабільної (фундаментальної) та мобільної (технологічної) складових його змістового компоненту та визначення відповідної моделі підготовки фахівця. З метою визначення співвідношення фундаментального та технологічного у змісті підготовки фахівців з інженерії програмного забезпечення (ІПЗ) проведено ретроспективний аналіз першої моделі підготовки фахівців з ІПЗ, розробленої на початку 1970-х років, та встановлено її відповідність сучасному стану розвитку ІПЗ як галузі знань та новим стандартам вищої освіти України за спеціальністю 121 «Інженерія програмного забезпечення». Визначено, що закладена в історично першій програмі підготовки системність та масштабованість у значній мірі відповідають ідеям проєктування еволюційного програмного забезпечення. Аналіз її змісту також надав можливість виявити зв’язки підготовки фахівця з ІПЗ із підготовкою з комп’ютерних наук, комп’ютерної інженерії, кібербезпеки, інформаційних систем і технологій. Встановлено, що фундаментальне ядро підготовки фахівця з ІПЗ повинне забезпечувати досягнення студентами таких провідних результатів навчання: знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення; знати і застосовувати відповідні математичні поняття, методи доменного, системного і об’єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення; застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення. Показано, що формування відповідних компетентностей майбутніх фахівців з ІПЗ необхідно виконувати у навчанні всіх дисциплін професійно-практичної підготовки.

Ключові слова: *інженерія програмного забезпечення, професійна підготовка, програмне забезпечення, модель підготовки фахівця, стандарт вищої освіти.*

Вступ

29 жовтня 2018 року Наказом МОН України № 1166 було затверджено та уведено в дію стандарт вищої освіти за спеціальністю 121 «Інженерія програмного забезпечення» для першого (бакалаврського) рівня вищої освіти (Стандарт 2018). Стандарт вищої освіти містить компетентності, що визначають специфіку підготовки бакалаврів зі спеціальності 121 «Інженерія програмного забезпечення» та програмні результати навчання. Уведенню стандарту передувала тривала робота спільноти вітчизняних фахівців з інженерії програмного забезпечення (ІПЗ), результатом якої стало формування системи загальних та спеціальних компетентностей майбутнього фахівця з ІПЗ.

Стрімкий розвиток моделей, методів та засобів ІПЗ й ІТ-галузі у цілому породжує питання про те, чи є запропонована система компетентностей достатньою для формування сучасного фахівця з ІПЗ, здатного до професійного розвитку та самовдосконалення протягом життя. Позитивна відповідь на це питання сприятиме покращенню позиції України на міжнародному ринку праці в ІТ-галузі, а негативна – консервації поточного стану розвитку технологій та запрограмованому відставанню ІТ-освіти України від інших держав.

Для відповіді на це питання необхідно визначити співвідношення стабільної (фундаментальної) та мобільної (технологічної) складової змісту професійної підготовки фахівців з ІПЗ. Як вказував один із фундаторів ІПЗ Б. Ренделл, «Ті, хто не може згадати минуле, приречені повторювати його. Тому я сподіваюся, що ви ... приділити трохи більше уваги минулому, ... виділивши деякий час на читання або перечитування, наприклад, оригінального звіту НАТО за 1968 рік ..., перш ніж ви зберетесь розробити ще одну нову мову або технологію [програмування]» [7, с. 8]. Саме тому у попередній нашій роботі [6] було представлено аналіз основних етапів розвитку інженерії програмного забезпечення як галузі знань, виокремлено фундаментальні складові підготовки майбутніх інженерів-програмістів та визначено тенденцій розвитку цієї галузі на найближче десятиліття.

Метою статті є аналіз першої моделі підготовки фахівців з інженерії програмного забезпечення, розробленої під керівництвом Ф. Л. Бауера, та встановлення її відповідності із сучасним станом розвитку ІПЗ і новим стандартом вищої освіти за спеціальністю 121 «Інженерія програмного забезпечення».

Перша модель підготовки фахівців з інженерії програмного забезпечення (1972)

За результатами роботи конференцій NATO з інженерії програмного забезпечення 1968 [1] та 1969 [2] рр. міжнародною групою експертів під керівництвом Ф. Л. Бауера наприкінці 1971 – початку 1972 року було розроблено «Поглиблений курс з інженерії програмного забезпечення» (“Advanced Course on Software Engineering”) [3], вперше проведений з 21 лютого по 3 березня 1972 року в Математичному інституті Мюнхенського технічного університету. У передмові до матеріалів цієї програми підготовки з ІПЗ Ф. Л. Бауер вказує, що книга [3] є першим кроком у напрямі концентрації та систематизації відповідних навчальних матеріалів: «при плануванні даного курсу нашою метою було охопити якомога більше аспектів теми [ІПЗ] та сприяти систематизації у цій галузі. ... ми вважаємо важливим показати, як ідеї інженерії програмного забезпечення повинні впливати на комп’ютерні науки та реалізовуватися у програмах підготовки ... для студентів у академічному середовищі» [3, с. 2].

Розроблена програма підготовки з ІПЗ складалась із чотирьох розділів.

Перший розділ – «Вступ [до ІПЗ]» – містив дві лекції. У першій лекції К. У. Мортон (Keith William Morton) на конкретних прикладах описує проблеми, що призвели до кризи програмного забезпечення, та визначає 8 основних напрямів їх подолання при розробці пакетів прикладних програм:

1) застосування проєктного менеджменту: підготовка персоналу за відповідними технологіями програмування, визначення та дотримання стандартів, розподіл робіт, моніторинг виконання та якості робіт;
2) визначення вимог до програмного продукту: його функцій, відповідності потребам користувачів, вимог до операційного середовища;
3) документування: визначення рівнів, методів та засобів автоматизації, контроль якості, розповсюдження та оновлення;
4) проєктування та реалізація;
5) застосування проблемно-орієнтованих мов різного рівня або ієрархії абстрактних машин, найбільш придатних для розв’язання прикладних задач;
6) тестування: генерування тестових даних та використання випробувальних стендів;
7) вимірювання продуктивності: моделювання, застосування засобів вимірювання, моніторингу та оптимізації;
8) обслуговування та вдосконалення.

Визначеним К. У. Мортоном напрямам відповідають наступні програмні результати навчання Стандарту 2018:

1) ПР22 – знати та вміти застосовувати методи та засоби управління проєктами;

2) ПР09 – знати та вміти використовувати методи та засоби збору, формулювання та аналізу вимог до програмного забезпечення та ПР11 – вибирати вихідні дані для проєктування, керуючись формальними методами опису вимог та моделювання;

3), 6) та 7) ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного й об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення, ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення та ПР16 – мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації;

4) ПР10 – проводити передпроєктне обстеження предметної галузі, системний аналіз об'єкта проєктування, ПР12 – застосовувати на практиці ефективні підходи щодо проєктування програмного забезпечення, ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення і низка інших;

5) ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення та ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

Наприкінці лекції К. У. Мортон дає відповідь на запитання, покладене у її назву – «Що може зробити інженер-програміст для користувача комп'ютера?»: вдосконалити комп'ютерні системи, якими послуговується користувач, а також інструменти та методи, які він може використати у своїй роботі [3, с. 4-11].

Друга лекція першого розділу «Проєктування та конструювання програмних систем», автором якої є Дж. Б. Денніс (Jack Bonnell Dennis), розглядала такі основні поняття, як «комп'ютерні системи», «програмні системи», їх ієрархію, «системне програмне забезпечення» та «прикладне програмне забезпечення», способи опису програмних систем, їх функціональність, правильність, продуктивність та надійність, а також програмні проєкти.

На початку лекції автор викладає власне бачення ІПЗ як застосування принципів, умінь та натхнення для проєктування та розробки програм і програмних систем, зосереджуючись саме на відомих принципах ІПЗ та межах їх застосовності.

Дж. Б. Денніс визначив комп'ютерну систему як об'єднання програмних та апаратних компонентів, що надають певну форму послуги групі «користувачів» [3, с. 13]. Так, для підсистем програмування на прикладі мови Basic автор визначає щонайменше три різні групи комп'ютерних систем та їх «користувачів» (споживачів певних послуг):

1) апаратне забезпечення, «користувачами» якого є розробники операційних систем;
2) апаратне забезпечення та операційна система, «користувачами» яких є розробники їх підсистем;
3) апаратне забезпечення, операційна система та підсистема програмування мовою Basic, послуги яких надаються користувачам цієї мови.

Виходячи за такої єдності, програмні системи автор визначив як програмні та апаратні компоненти, що повинні бути додані до певної базової комп'ютерної системи для реалізації певних функцій. Так, для програмної системи програмування мовою Basic апаратне забезпечення та операційна система утворюють базову комп'ютерну систему.

Ієрархія систем утворюється шляхом їх розширення або визначення нового способу взаємодії (лінгвістичного рівня) з ними через трансляцію або інтерпретацію команд.

Системне програмне забезпечення – набір системних програм, що зазвичай утворюють ієрархію програмних систем з наступними властивостями:

- набір спільно керованих програм;
- ієрархія програмних систем, що визначає один лінгвістичний рівень, що застосовується усіма користувачами набору програм;
- внутрішні лінгвістичні рівні ієрархії приховані від користувача;
- зовнішній лінгвістичний рівень ієрархії є «повним» для реалізації цілей управління;
- основним засобом визначення нових лінгвістичних рівнів є часткова інтерпретація.

Прикладне програмне забезпечення – це прикладні програми або програмні системи, що зазвичай мають наступні властивості:

- програми виражаються у термінах «повного» лінгвістичного рівня;
- програми визначають новий лінгвістичний рівень за допомогою розширення, трансляції та компіляції або їх комбінації;
- лінгвістичний рівень, визначений програмою або програмною системою, не може бути використаний для визначення нових лінгвістичних рівнів;
- клієнтам доступні різні такі програми або програмні системи, що часто реалізують різні способи управління.

Опис програмної системи Дж. Б. Денніс визначає через опис її програмної (за допомогою гарно визначеної мови програмування) складової, апаратної складової (за допомогою моделі, що відображає поведінку апаратної складової у всіх нормальних станах функціонування програмної системи), базової системи та лінгвістичного рівня, на якому повинна бути реалізована програмна система.

Цілі проєктування програмної системи виражаються у її бажаних властивостях: функціональності (як відповідності бажаного результату вхідним даним), правильності (за допомогою структурного стилю програмування, який робить правильність програми самоочевидною, або за допомогою доведення правильності програмних систем або компонентів), продуктивності (ефективності, з якою використовуються ресурси базової системи для досягнення цілей програмної системи) та надійності (здатності програмної системи правильно функціонувати, незважаючи на неполадки складових комп'ютерної системи).

Завершується лекція Дж. Б. Денніса описом типових задач, що виникають при розробці великих програмних проєктів.

Стандарт 2018 визначає наступні програмні результати, відповідні цій лекції:

- ПР03 – знати основні процеси, фази та ітерації життєвого циклу програмного забезпечення;
- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми й основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР10 – проводити передпроєктне обстеження предметної галузі, системний аналіз об'єкта проєктування;
- ПР11 – вибирати вихідні дані для проєктування, керуючись формальними методами опису вимог та моделювання;
- ПР12 – застосовувати на практиці ефективні підходи щодо проєктування програмного забезпечення;
- ПР13 – знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань;

- ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
- ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення;
- ПР22 – знати та вміти застосовувати методи та засоби управління проєктами.

Другий розділ – «Засоби опису [в ІПЗ]» – містив п'ять лекції. У першій лекції «Ієрархії» Г. Гооз (Gerhard Goos) розглядає методологічні засади декомпозиції програмних систем та їх застосування до розробки програмного забезпечення та мов програмування.

Проєктування програмних систем Г. Гооз пропонує розпочинати з опису задачі, що підлягає розв'язанню, та доступної базової систем (у трактуванні Дж. Б. Денніса). Задача може бути формально представлена абстрактною машиною, яка із вхідних даних за допомогою програми для цієї машини виробляє деякі вихідні дані, що розв'язують певну частину проблеми [3, с. 29]. На першому кроці (загальне проєктування системи) шляхом специфікації їх зовнішніх інтерфейсів визначається набір програмних компонентів, кожен із яких розв'язує частину оригінальної задачі, реалізуючи її за допомогою деяких функцій абстрактної машини. На другому кроці (детальне проєктування системи) визначається внутрішня поведінка кожного компонента.

Результати загального проєктування можуть бути подані у вигляді мережі компонентів – орієнтованого графу довільної складності. Наявність неусувних циклів у такому відображає складність процесу проєктування. Принцип структурування систем у вигляді часткового впорядкованого набору шарів (у тому числі деревоподібної або лінійної структури) Г. Гооз називає ієрархічним впорядкуванням та вказує, що його застосування надає можливість розділити систему на компоненти у такий спосіб, що отримується картина їх взаємозв'язків та чітко визначається внесок кожного компонента у розв'язання загальної задачі системи. Кожен шар може бути реалізований певною абстрактною машиною, послідовність яких утворює систему, причому кожен наступний (нижчий) шар надає всі необхідні засоби для реалізації попереднього (більш високого) шару, а останній шар реалізує абстрактну машину, яка тотожна базовій системі. На кожному рівні абстракції визначається мова його програмування, що надає можливість звернутися до всіх функції абстрактної машини цього рівня.

Г. Гооз розглядає ієрархічне проєктування як засіб інженерії розробки та виробництва програмного забезпечення. Основне припущення автора полягає у тому, що операції та структури даних, присутні на рівні A_{i+1}, але відсутні на рівні A_i, не можуть бути використані програмами, що працюють на рівні A_i. Це припущення надає можливість використання ієрархічного впорядкування також як засобу тестування та налагодження [3, с. 44-45].

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР17 – вміти застосовувати методи компонентної розробки програмного забезпечення.

У другій лекції «Характеристики мов: мови програмування як засіб написання системного програмного забезпечення» Г. Гооз з інженерної точки зору розглянув, як мови програмування впливають на процес розробки програми та її властивості.

Так само, як природна мова впливає на мислення людини, що її використовує, мови програмування впливають на їх користувачів, визначаючи: а) концептуальне розуміння того, як задача може бути вирішена комп'ютером, б) спектр задач, які можуть бути розв'язані шляхом програмування, в) набір основних понять, доступних для програмування, г) стиль

програмування (чіткість, надійність, зручність читання та ін.), д) значення понять «переносимість» та «ефективність».

У лекції встановлено відповідність між властивостями мов та властивостями програм, описаних ними, та визначені деякі характеристики «гарних» мов програмування, головною з яких є спонукання програміста до написання правильно (з інженерної точки зору) спроєктованих програм. Особливу увагу автор приділив характеристикам високорівневих мов системного програмування. До засобів підвищення якості інженерії системного програмного забезпечення (створення «гарних» програм) Г. Гооз відносить насамперед засоби структурного програмування (зокрема, модульність), ієрархічне впорядкування, застосування вкладеності та областей видимості, а також паралельне виконання.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

Третя лекція другого розділу «Низькорівневі мови програмування: результати обговорення» є виконаним М. Гріффітсом (M. Griffiths) узагальненням матеріалів круглого столу, в якому взяли участь Ф. Бауер, Г. Гооз, М. Гріффітс та інші автори курсу. Відповідаючи на питання, навіщо створювати нову мову програмування низького рівня, учасники дискусії вказують, що існуючі машинні мови (асемблери) не забезпечують підтримку ефективного розподілу пам'яті та дотримання стилю програмування, що має на меті підвищення надійності програмного забезпечення. До нових низькорівневих мов програмування учасники дискусії ставлять вимоги наявності засобів управління потоком виконання (циклічні та умовні оператори), підтримку модульності (принаймні на рівні процедур із передаванням параметрів), структур даних (з можливістю перетворення типів, індексації, адресної арифметики). При цьому такі мови можуть бути як машинно зорієнтованими, так й проблемно зорієнтованими (такі, як Forth).

Ефективність низькорівневих мов програмування пропонувалось вимірювати за трьома показниками: трудомісткість праці програміста, машинний час та обсяг пам'яті. Стосовно стилю програмування учасники дискусії вказали на життєво важливу роль професійної підготовки майбутніх системних програмістів із використанням доцільних засобів та гарного стилю програмування. Найвдалішим прикладом низькорівневої мови, що відповідає наведеним вимогам, на момент проведення курсу учасники дискусії вважали мову, що використовувалась при розробці операційної системи MULTICS – PL/I, точніше, її ранній діалект EPL, що суттєво вплинув на дизайн мови програмування, що й сьогодні є еталонною реалізацією розробленої у ході дискусії концепції – C.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми й основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

Четверта лекція «Співвідношення між визначенням та реалізацією мови» пов'язана із головним напрямом досліджень М. Гріффітса – розробкою компіляторів: «Неспеціалісти іноді зауважують, що як фахівці з комп'ютерних наук, так й фахівці з інженерії програмного забезпечення марнують свій час на дискусії про мови замість того, щоб займатися

«справжніми» справами. Так, не без того, але слід чітко розуміти, що мова є центром усіх задач інженерії програмного забезпечення. Якщо ми не можемо надати програмісту потужну, гарно визначену, зрозумілу мову з відповідною економічною реалізацією на існуючому комп'ютерному обладнанні, важко від нього очікувати [гарної програмної інженерії]» [3, с. 77]. У лекції обговорюється вплив визначення мови на програми, описані цією мовою, та на методи, що використовуються для виконання цих програм на комп'ютері.

Автор виділяє три точки зору на визначення мови: 1) точку зору користувача, що зосереджується на задачі, а не особливостях опису мови (опис мови як керівництво); 2) точку зору розробника компілятора, який прагне реалізувати усі описані особливості мови, навіть ті, що будуть використовуватись дуже мало (опис мови як святе письмо); 3) точку зору дослідника, для якого опис мови є повним, формальним, математично правильним набором тверджень, які відображають традиційні математичні принципи мінімізації кількості взаємопов'язаних аксіом.

Головне при визначенні мови, на думку автора, – відсутність побічних ефектів. Для цього він виокремлює дві складові семантики: статична семантика – частина семантики мови програмування, що не залежить від виконання програми і визначається під час компіляції (така, як співвідношення між використанням ідентифікатора та його оголошенням); динамічна семантика враховує реальні маніпуляції над об'єктами та їх значеннями під час виконання програми.

При визначенні синтаксису мови провідним є граматичний підхід, зокрема, з використанням LL(1)-граматик для реалізації контекстно-вільних мов. М. Гріффітс на чисельних прикладах демонструє вплив способу визначення мови на її реалізацію, показуючи, що легкість, безпечність та ефективність реалізації мови безпосередньо залежать від її визначення: «саме ця ідея змушує нас припускати, що визначення мови [із самого початку] має передбачати її реалізацію... Тут є паралель із архітектурою комп'ютерної техніки, яка, ймовірно, покращується концептуально кожного разу, коли комп'ютерний інженер співпрацює з інженером-програмістом для кращого використання комп'ютера. Використання, на яке поширюється визначення мови, в першу чергу є її реалізацією. Тому ми вважаємо, що визначення мови повинно бути надано у термінах ідеалізованої реалізації або, принаймні, супроводжуватись «посібником реалізатора» [3, с. 99-100].

У лекції розглядаються наступні способи визначення мови:

- дворівнева граматика (формальна граматика, яка використовується для породження іншої формальної граматики), така, як граматика ван Вейнгаардена для Algol 68 спільно із стилізованим описом природною мовою інтерпретації програми на гіпотетичному комп'ютері;
- віденські визначення (Vienna Definition Language (VDL), сьогодні більш відомі як віденський метод розробки – Vienna Development Method, VDM – набір технологій для моделювання систем, аналізу створених моделей і переходу до деталізованого проєктування та програмування), які використовували операційну (конотативну) семантику – спосіб опису мови за допомогою послідовностей кроків обчислення, дуже тісно пов'язаний з реалізацією системи мовою програмування, оскільки кроки обчислення описуються на мові деякого обчислювача;
- розширювані мови, що складаються з відносно простої, але самодостатньої базової мови, яка сама містить в собі механізми розширення, що надають можливість визначати нові оператори або типи даних.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;

- ПР12 – застосовувати на практиці ефективні підходи щодо проєктування програмного забезпечення;
- ПР13 – знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань.

Остання лекція другого розділу «Конкуренція у програмних системах», написана Дж. Б. Деннісом, присвячена великим модульним програмам, таким як операційні системи, компілятори або системи управління. Через великий розмір таких програмних систем важливо, щоб їх складові були представлені у такий спосіб, щоб описи самих складових не залежали від способу, у який вони з'єднуються, утворюючи систему, а поведінка кожної складової була однозначною і правильно зрозумілою незалежно від ситуації, в якій вона використовується. Для того, щоб це було можливим, всі взаємодії між частинами системи повинні бути через явні точки комунікації, створені проєктувальником кожної частини. Якщо дві частини системи спроєктовані незалежно, тоді в результаті взаємодії між двома частинами таймінг подій в одній частини може бути обмежений лише по відношенню до подій у іншій частині. Поки жодна взаємодія не відбувається, події в двох частинах системи можуть відбуватися одночасно і без визначеного часу взаємодії між ними. Встановлення часових залежностей між незалежними діями в окремих складових системи є загальним джерелом перевизначення. Результатом стає система, яку важко усвідомити та важко змінити, і яка включає непотрібні затримки, які можуть знизити продуктивність. Такі міркування, на думку Дж. Б. Денніса, показують, що поняття паралельності та асинхронної роботи є фундаментальними аспектами програмних систем.

У лекції розглянуто моделі систем, представлених як набори одночасно діючих підсистем, що взаємодіють одна з одною через певні комунікаційні механізми. Автор показує, що, якщо взаємодія між підсистемами підпорядковується певним природним умовам, то детермінованість підсистем гарантує детермінованість всієї системи (здатність давати при різних запусках за однакових вхідних даних однакові результати). Опис таких систем виконується за допомогою мереж Петрі: лекція завершується прикладом їх застосування до систем із паралельних процесів, які взаємодіють за допомогою семафорів, використовуючи примітиви синхронізації P та V Е. В. Дейкстри.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення.

Третій розділ – «Технології [ІПЗ]» – містив чотири лекції. У першій лекції «Модульність» Дж. Б. Денніс поглиблює розпочатий у попередній лекції розгляд концепції модульності, тісно пов'язаної зі структурним програмуванням, як властивості комп'ютерних систем: «Комп'ютерна систем є модульною, якщо лінгвістичний рівень, що її визначає, ... пов'язаний із класом об'єктів, що репрезентують складові програми ... – програмними модулями. Лінгвістичний рівень повинен забезпечити спосіб об'єднання програмних модулів у більші програмні модулі, не вимагаючи внесення змін до жодного компонента модулів. Крім того, значення модуля програми має бути незалежним від контексту, в якому він використовується» [3, с. 130].

Дж. Б. Денніс розглядає модульність і в контексті попередньої лекції (конкуренція з комунікацією), так й у процедурному контексті, коли один модуль викликає інший із певним вхідним набором даних, врешті решт отримуючи результат. У лекції розглядаються питання, що виникають при побудові програмних модулів, які вимагають можливості створювати, розширювати та змінювати структуровані дані. Зроблено висновок, що для досягнення модульності комп'ютерна система повинна визначити лінгвістичний рівень, який забезпечує відповідне базове подання для структурованих даних: 1) будь-яка структура даних може стати

складовою іншої; 2) будь-яка структура даних може бути передана (за посиланням) у або прийнята з програмного модуля як фактичний параметр; 3) програмний модуль може будувати структури даних довільної складності. На такому рівні пам'ять повинна адресуватися не елементами наперед визначеної довжини, а структурами даних. Головним обмеженням на розмір модуля автор вважає обсяг основної пам'яті комп'ютерної системи, а способом його подолання – її віртуалізацію.

Значна підтримка модульності може бути з боку правильно спроєктованої операційної системи – так автор наводить характеристики операційної системи Multics, що сприяють реалізації обговорюваної концепції:

1. Великий віртуальний адресний простір (приблизно 230 елементів) для кожного користувача.
2. Доступ до всієї інформації користувача здійснюється через його віртуальний адресний простір. Для окремих видів даних, таких як файли, не передбачено окремий механізм доступу.
3. Будь-яка активація процедури вимагає обсягу робочого простору, обмеженого лише кількістю вільних сегментів у адресному просторі користувача.
4. Будь-яка процедура може бути розподілена між багатьма процесами без необхідності її копіювання.
5. Кожна процедура, написана на стандартних для Multics мовах (Fortran, PL/I та ін.), може бути багаторазово активізована через рекурсію або паралельне виконання.
6. Спільне цільове представлення використовується компіляторами з двох основних мов – PL/I та Fortran.

Ці досягнення є основним внеском Multics у спрощення розробки та впровадження великих програмних систем. Вони стали можливими завдяки побудові програмного забезпечення Multics на машині, спеціально організованої для реалізації великого обсягу віртуальної пам'яті та спільного доступу до сегментів даних та коду [3, с. 161].

В останній частині лекції неформально представлені семантичні поняття лінгвістичного рівня (загальна базова мова), які можуть слугувати для загального подання програмних модулів, виражених різними мовами програмування.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення;
- ПР17 – вміти застосовувати методи компонентної розробки програмного забезпечення.

Друга лекція «Мобільність та адаптивність» П. С. Пула (Peter Cyril Poole) та У. М. Уейта (William M. Waite) розпочинається із визначення мобільності як рівня легкості, з якою програма може бути перенесена з одного середовища в інше, та адаптивності як рівня легкості, з якою програма може бути змінена для пристосування до відмінних від попередніх уявлень користувача та системних обмежень. Головну відмінність між цими концепціями автори вбачають у тому, що адаптивність зосереджується на змінах у структурі алгоритму, у той час як мобільність – на змінах у програмному оточенні.

Основним способом підвищення програмної мобільності є використання високорівневих мов за умови, що:

- базові операції та типи даних, необхідні для розв'язання задачі, доступні в обраній мові;
- обрана мова має стандарте визначення, реалізації якого широко поширені;

– мовні конструкції, доступні у локальних діалектах, але відсутні у стандарті, застосовуються з обережністю [3, с. 187].

Інший спосіб – застосування абстрактних машин, на яких працюватиме мобільна програма. При цьому компіляція з мови високого рівня спочатку виконується на мову абстрактної машини (такої, як UNCOL – Universal Computer Oriented Language), яка повинна обов'язково підтримувати принаймні цілі числа та цілочисельну арифметику, операції порівняння та відношення, символьне уведення та виведення, а також дійсні числа та дійсну арифметику, рядки (з'єднання, вибір підрядка, лексикографічне порівняння), введення та виведення образів пам'яті, мітки та засоби передавання управління, оголошення, масиви та записи, умовні та циклічні оператори, процедури та блоки. Наприкінці лекції авторами побудовано ієрархію абстрактних машин, що забезпечують мобільність та адаптивність й нівелюють окремі недоліки UNCOL.

Відповідні лекції програмні результати навчання за Стандартом 2018:

– ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
– ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
– ПР13 – знати і застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань;
– ПР15 – мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

На початку третьої лекції «Налагодження та тестування» П. С. Пул вказує, що, доведення правильності програми формальними методами (такими, як VDM) є занадто довгим шляхом для знаходження помилок, тому в циклі виробництва програмного забезпечення гарний інженер-програміст повинен планувати фазу тестування та налагодження [3, с. 279] (при цьому тестування в жодному випадку не показує відсутність помилок – лише їх наявність). Важливу роль у цьому відіграють стандарти документування програмного забезпечення, розглянуті у окремій лекції, та «читабельність» програмного коду – зокрема, наявність змістовних коментарів є корисним для налагодження та незамінним для підтримки та покращення програмного забезпечення. Програмний код повинен супроводжуватися еталонними тестами та даними, які мають змінюватися разом із кодом.

П. С. Пул увів концепцію «охоронного коду» (guard code) – використання логічних виразів для обмеження варіантів обчислень [3, с. 287], який він пропонує активно використовувати спільно із «принципом виключної підозрілості», який полягає у тому, що модуль не повинен використовувати будь-які дані, передані йому через інтерфейс, без попередньої перевірки їх відповідності специфікації інтерфейсу.

Основними методиками налагодження на той час були аналіз посмертних дампів, налагоджувальне виведення, пряме та зворотне трасування частини програмного коду, компіляція із включенням даних для налагодження, документування коду (переформатування тексту з метою виділення структурних елементів коду, створення таблиць зі змістом усіх процедур, міток та оголошень, створення індексу викликів процедур, переходу за мітками та посилань на змінні, візуалізація потоків управління та ін.). Перспективним засобом автор вважає інтерактивне налагодження, кероване користувачем (online debugging).

Відповідні лекції програмні результати навчання за Стандартом 2018:

– ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
– ПР19 – знати та вміти застосовувати методи верифікації та валідації програмного забезпечення;

– ПР20 – знати підходи щодо оцінки та забезпечення якості програмного забезпечення.

В останній лекції розділу «Надійність» Д. Цикрітзіс (Dionysios C. Tsichritzis) розглядає питання проєктування та конструювання надійного програмного забезпечення. Автор розпочинає огляд із цитати Е. Дейкстри «Тестування може показати наявність помилок, але не їх відсутність» та Б. Ренделла «Надійність не є додатковою властивістю [ПЗ]», наголошуючи на основній ідеї ІПЗ – надати засоби, які нададуть можливість особі середніх здібностей створювати якісне ПЗ [3, с. 319-320].

Д. Цикрітзис постулює низку положень, що впливають на надійність розробки ПЗ:

1) різні мови програмування спонукають програмістів до різних типів помилок, які він називає характеристичними;
2) компілятор повинен виявляти семантичні невідповідності у текст програм;
3) стиль програмування, зокрема:
   – використання доцільних імен змінних та структурування програмного коду;
   – використання таких методів, як перехресна перевірка, перевірка діапазонів значень змінних, перевірка логічності змін даних, дотримання унікальності імен, поетапне опрацювання даних, включення до результатів кодів помилок;
4) вплив захисту (керованого програмного оточення з чітко визначеними правилами та обмеженнями) шляхом діагностування помилок як порушення захисту та ізоляції помилок у модулях, в яких вони відбулись;
5) доведення правильності програми неформальними (за допомогою уведення перевірок значень вхідних, проміжних та вихідних змінних) та формальними (наприклад, за допомогою числення предикатів першого порядку) методами;
6) проєктування програми, спрямоване на підвищення надійності:
   – структурування програми з метою полегшення її тестування;
   – неформальна перевірка логічної правильності деяких частин програми;
   – застосування засобів синхронізації та взаємодії процесів;
   – використання ієрархії абстрактних машин;
7) забезпечення надійності під час роботи програми шляхом:
   – забезпечення цілісності даних з використанням часткових або повних дампів пам’яті;
   – дублювання ключових даних у різних сховищах;
   – урахування можливості апаратних відмов;
   – наявність можливостей відновлення роботи програми для мінімізації впливу відмов.

Обговорюючи способи захисту системи у цілому, процесів та даних користувачів, Д. Цикрітзіс уводить поняття домену (активної та потенційно агресивної сутності) та об’єкту (пасивної та потенційно уразливої сутності), наводячи приклади їх реалізації через процеси та файли. На рівні абстракції віртуальної машини порушення надійності – це порушення користувачем роботи власної віртуальної машини або віртуальних машин інших користувачів. Для опису цього процесу автор використовує матричний апарат розподілу ресурсів між паралельними процесами в умовах нестачі ресурсів. Серед способів, що використовуються донині, автор пропонує випадково згенеровані імена об’єктів синхронізації (magnum – магічні числа). У лекції детально обговорюються механізми захисту даних у файлових системах, що мають бути реалізовані в ядрі операційної системи.

Поняття безпеки даних (information security, інформаційної безпеки або кібербезпеки) автор трактує як управління доступом до привілейованих даних, що зберігаються у великих масштабованих банках даних, за трьома категоріями [3, с. 357]:

1) приватність даних включає правові та етичні питання контролю особистістю доступу до даних;
2) конфіденційність даних включає правила доступу до них;
3) безпека даних включає засоби забезпечення конфіденційності.

Д. Цикрітзіс розрізняє захист та безпеку даних у такий спосіб: захист стосується лише контролю доступу до даних в операційній системі без урахування природи даних, а для задач інформаційної безпеки розглядається інформаційна система у цілому, а не лише система, що працює на комп'ютері. Активними елементами інформаційної системи є люди, а природа та зміст даних ураховуються при наданні доступу до них. Саме тому приклади задач інформаційної безпеки автор розглядає не на файлових системах, а на системах управління базами даних.

До важливих задач інформаційної безпеки автор відносить:

а) на рівні входу в систему: ідентифікацію користувачів, їх аутентифікацію, моніторинг використання ресурсів;
б) на рівні файлової системи: визначення доступних файлів, ідентифікацію користувачів, парольно керовані властивості файлів, криптографічний захист;
в) на рівні захисту даних: відокремлення даних різних користувачів, цілісність даних, резервне копіювання, надійне видалення захищених даних;
г) обмеження використання (зокрема, примусовий вихід із системи за нестандартних дій).

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного й об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР08 – вміти розробляти людино-машинний інтерфейс;
- ПР11 – вибирати вихідні дані для проєктування, керуючись формальними методами опису вимог та моделювання;
- ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
- ПР18 – знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних;
- ПР19 – знати та вміти застосовувати методи верифікації та валідації програмного забезпечення;
- ПР20 – знати підходи щодо оцінки та забезпечення якості програмного забезпечення;
- ПР21 – знати, аналізувати, вибирати, кваліфіковано застосовувати засоби забезпечення інформаційної безпеки (зокрема кібербезпеки) і цілісності даних відповідно до розв'язуваних прикладних завдань та створюваних програмних систем.

Четвертий розділ – «Практика [ІПЗ]» – був найбільшим в усьому курсі та включав шість лекцій. У першій лекції «Управління [програмними] проєктами» Д. Цикрітзіс формулює загальну мету управління програмними проєктами: «виробництво бажаного продукту за визначених цілей проєктування, специфікацій та доступних ресурсів» [3, с. 375]. Для досягнення цієї мети автор пропонує спочатку налаштувати комунікації розробників (через використання спільної документації, часте неформальне спілкування та регулярні зустрічі малих груп розробників), організувати розробників (шляхом виокремлення особи, що має системний погляд на проєкт, залучення розробників до інших складових проєкту та зменшення формалізму управління) та розставити контрольні точки.

Серед засобів розробки програмних проєктів Д. Цикрітзіс особливу увагу приділяє засобам моделювання інформаційних систем (так, для опису паралельних процесів таким засобом є мережі Петрі) та системам автоматизованого проєктування програмних систем, деякі з яких (такі як Project LOGOS) суттєво випередили свій час. Головна різниця між малими та великими програмними проєктами, на його думку – кількість рівнів управління: їх повинно бути щонайменше два. Наприкінці лекції обговорюються питання зменшення часу реалізації програмних проєктів та вимоги до менеджерів великих проєктів.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР03 – знати основні процеси, фази та ітерації життєвого циклу програмного забезпечення;
- ПР04 – знати і застосовувати професійні стандарти і інші нормативно-правові документи в галузі інженерії програмного забезпечення;
- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР11 – вибирати вихідні дані для проєктування, керуючись формальними методами опису вимог та моделювання;
- ПР12 – застосовувати на практиці ефективні підходи щодо проєктування програмного забезпечення;
- ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
- ПР16 – мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації;
- ПР18 – знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних;
- ПР22 – знати та вміти застосовувати методи та засоби управління проєктами;
- ПР23 – вміти документувати та презентувати результати розробки програмного забезпечення.

У другій лекції розділу – «Документація» – Г. Гооз ставить 8 запитань, на які мають дати відповідь різні види документації (табл. 1) [3, с. 386].

Таблиця №1.

Види документації (за Г. Гоозом)

Номер	Питання	Керівництво користувача	Концептуальний опис	Проєктна документація	Документація по продукту
1	Як використовувати програму?	■			
2	Який стан проєкту?		■	■	■
3	Які загальні характеристики проєкту?		■	■	
4	Які моделі використовуються для поділу програми на складники й взаємодії різних модулів?		■	■	
5	Які базові моделі використовуються для модулів?		■	■	
6	Які у програмі потік управління та потік даних?			■	
7	Детальний опис даних				
8	Що означають повідомлення про помилки?	■			

Короткі, але вичерпні відповіді на ці питання є однією із умов успішності проєкту – «якщо програмісти чітко не зрозуміють, як їх робота пов'язана із роботою інших, вони приречені на невдачу через зроблені невірні припущення» [3, с. 387].

Створення керівництва користувача («швидкого старту», довідника та керівництва оператора) розпочинається першим із завершується останнім. «Швидкий старт» (introductory manual) не лише описує стандартні способи використання програми, слугуючи своєрідною «куховарською книгою» програми, а й є основою для її реклами та продажу.

Концептуальний опис розробляється під час виконання проєкту як його загальний опис, проєктна документація описує поточний стан проєкту на фазі проєктування та є основою для фази конструювання, а документація по продукту (включно із текстом програми) описує поточний стан проєкту на фазах конструювання та обслуговування. Включення до останньої тексту програми необхідно для перехресних посилань на інтерфейсні модулі, дані та алгоритми.

Підтримка документації в актуальному стані – важлива задача, для розв'язання якої Г. Гооз пропонує використовувати розподілені системи текстового документування з підтримкою часових міток.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР04 – знати і застосовувати професійні стандарти і інші нормативно-правові документи в галузі інженерії програмного забезпечення;
- ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
- ПР16 – мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації;
- ПР23 – вміти документувати та презентувати результати розробки програмного забезпечення.
- ПР24 – вміти проводити розрахунок економічної ефективності програмних систем.

Третя лекція – «Прогнозування продуктивності» – є однією з найбільших у розділі. Її автор Р. М. Грем (Robert M. Graham) був одним із перших фахівців у галузі кібербезпеки – так, у проєкті Multics він відповідав за механізми безпеки, динамічного компонування та інші ключові компоненти ядра операційної системи.

На прикладі проєктування та розробки операційних систем автор пропонує критерії та методи оцінки продуктивності програмних систем, а також аналізує вплив цих оцінок на процеси розробки та впровадження програмного забезпечення. Продуктивність у цьому контексті автор розглядає як ефективність використання системних ресурсів при реалізації цілей програмного забезпечення. Автор підкреслює важливість моделювання систем для оцінки їх ефективності та продуктивності. Модель програмної системи, на думку автора, в тій чи іншій мірі відображає відносини між змінними цієї системи. Складність моделі безпосередньо пов'язана зі складністю самої системи та способами її використання. Для простих систем модель буде настільки простою, що може існувати лише в уяві проєктувальника. Але зі зростанням складності виникає потреба у відображені моделі системи якимось точним і формальним способом. Р. М. Грем наголошує, що повна, детальна характеристика системи, наприклад, її програмний код, теж фактично є такою моделлю. Але така модель, як правило, не є корисною, оскільки містить велику кількість надлишкової інформації і не чітко демонструє зв'язки між основними змінними: «модель має бути абстракцією, що містить лише значущі змінні та відносини, і бути простішою, ніж система, що моделюється» [3, с. 404].

Концептуально модель є функцією або набором функцій, в яких системні параметри, що використовуються для характеристики продуктивності системи, виражаються як функції основних змінних системи. Автор підкреслює, що «продуктивність системи є скоріше не сталою, а функцією або декількома функціями, значення яких залежать від вхідних даних. Для

того, щоб охарактеризувати продуктивність даної системи, ми повинні виразити ці функції, і саме це подання є моделлю системи» [3, с. 404].

Таким чином, важливою складовою професійної підготовки фахівців з інженерії програмного забезпечення є оволодіння методами та засобами формального опису моделей програмних систем.

Автор виділяє два базових типи моделей: аналітичні та логічні. Аналітична модель не відображає структуру системи. Фактично – це набір математичних рівнянь, які виражають співвідношення, що існують між основними системними змінними та параметрами продуктивності. Ці рівняння потім розв'язуються відносно залежних змінних, тобто для параметрів продуктивності. Після розв'язання цих рівнянь продуктивність системи повністю відома, оскільки графіки параметрів продуктивності можна побудувати з отриманих математичних виразів. Логічна модель, навпаки, в першу чергу відображає структуру системи, що моделюється. З такої моделі важно отримати точні вирази для оцінки продуктивності. Однак логічна модель часто включає математичні рівняння, які виражають деякі співвідношення між змінними.

Як приклад реалізації аналітичної моделі Р. М. Грем наводить опис алгоритмів планування процесорного часу в операційних системах. Автор підкреслює стохастичний характер такої моделі, але, передбачаючи певні діапазони значень змінних у системі та ймовірності цих значень, ми можемо робити оцінку очікуваної продуктивності системи.

Однією з найпростіших та найбільш наочних логічних моделей системи, на думку автора, є її подання у вигляді направленого графа. Прикладами таких графів є блок-схеми алгоритмів. У загальному випадку вузлами такого графу є певні стани системи (значення змінних), а дугами – процеси або дії, що необхідні для переходу від одного стану до іншого. Оцінивши часові або інші ресурси, необхідні для таких переходів, ми отримуємо можливість оцінювати та вимірювати продуктивність системи.

Окрему увагу Р. М. Грем приділяє імітаційним моделям як найбільш загальному та гнучкому способу оцінки продуктивності програмних систем і демонструє використання цих моделей в проєктуванні операційних систем.

До проблем, що виникають при розробці моделей продуктивності, Р. М. Грем відносить адекватність моделі, характеристику задач або бажаних властивостей системи та інтерпретацію результатів моделювання. Для опису моделей автор пропонує застосовувати спеціалізовані мови моделювання, які мають містить можливості опису класів та їх атрибутів, активностей та подій, а також підтримувати черги, різні розподіли ймовірностей та засоби для збору і аналізу даних. Фундаментальність такого підходу підкреслюється тим, що з 4 мов моделювання загального призначення, які згадує автор – GPSS, SIMSCRIPT, SIMULA та CSL – лише CSL на даний момент відноситься до застарілих. Simula 67 став основою для розробки C++ та Java, а останні версії aGPSS (1.30) та SIMSCRIPT III (Release 5.0) взагалі датуються 2019 роком.

Мови моделювання спеціального призначення – це мови моделювання операційних систем, як існуючих, так й тих, що ще не створені. Системи моделювання для цих мов містять відомості про апаратне забезпечення та мовні конструкції для його опису. Саме прикладами використання таких мов й завершується третя лекція розділу.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного й об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
- ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення;
- ПР10 – проводити передпроєктне обстеження предметної галузі, системний аналіз об'єкта проєктування;

- ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення;
- ПР20 – знати підходи щодо оцінки та забезпечення якості програмного забезпечення.

Автора четвертої лекція «Вимірювання продуктивності» К. К. Готліб (Calvin Carl "Kelly" Gotlieb) можна вважати канадським Лебедєвим – він був у складі першої групи в країні, яка в 1940-х рр. будувала комп'ютери та надавала комп'ютерні послуги, а у 1950 році створив перший університетський курс інформатики. Поступово його інтереси зміщувались у бік соціально-економічних наслідків застосування ІКТ, що зумовило його авторство й глави, присвяченої економіці інженерії програмного забезпечення.

К. К. Готліб вказує, що вимірювання продуктивності потрібно при:

а) встановленні нової обчислювальної системи;
б) зміні конфігурації або «тюнінгу» системи для поліпшення її продуктивності;
в) порівняння систем для визначення технологічних удосконалень, економічного ефекту масштабу та співвідношення вартості й переваг.

До методів вимірювання продуктивності автор відносить:

1. Встановлення показників якості (figures of merit) на основі рейтингу (ваги) компонентів системи, зокрема – емпіричного закону Гроша 1953 року, сформульованого Г. Р. Дж. Грошем (Herbert Reuben John Grosch): «отримання додаткової економії є лише квадратним коренем від збільшення швидкості – тобто, щоб зробити обчислення в 10 разів дешевше, ви повинні зробити їх в 100 разів швидше» [4, с. 310]. К. К. Готліб вказує, що саме вартість має бути загальним показником продуктивності, формулюючи закон Гроша у вигляді

$$C = K\sqrt{E},$$

де C – вартість, K – стала, E – ефективність, виміряна у швидкості, пропускній здатності тощо. Таким чином, продуктивність є пропорційною квадрату вартості.

Для більш точної оцінки автор пропонує визначати продуктивність системи, пов'язуючи низку атрибутів з кожною характеристикою системи з урахування ваги кожного атрибуту, визначеною методом експертного оцінювання. Загальний показник продуктивності розраховується як зважена сума ознак.

2. Запуск набору «ядерних», «стендових» або синтетичних задач. Під «ядром» (kernel) автор розумів еталонну програму загального призначення, для кожної складової якої були виконані необхідні вимірювання (часу виконання та ін.). Тоді продуктивність двох комп'ютерних систем порівнювалась одна відносно другої шляхом запуску «ядра» на кожній з них. «Стендова» програма (benchmark) – це спеціальна програма, призначена для оцінки продуктивності (тест продуктивності). Якщо в «ядрі» вимірювання продуктивності було побічним ефектом, то для «стендової» програми це є основним призначенням. Синтетичні програми призначені для комплексної перевірки стабільності системи в штатному і у форсованому режимах. На сьогодні всі ці види задач вважаються складовою тесту продуктивності (бенчмаркінгу).

3. Проведення спостережень і вимірювань за допомогою апаратних засобів та моніторів програмного забезпечення на трьох рівнях:

- на рівні задач користувача вимірюється кількість викликів програм, розрахунковий час виконання завдання, витрачений час на етапи виконання завдання, вибрані параметри часу виконання, використання ядра, зчитування та запис, друк, час обертання, обрані пріоритети, вартість, діагностика, що викликається на системному рівні;
- на системному рівні вимірюється розподіл ресурсів, активність каналів і операцій введення-виведення, довжина черг завдань і системних черг, час обслуговування та ін.;

– на апаратному рівні вимірюється трафік і потоки завдань, використання послуг, розподіл ресурсів, дії та втручання оператора, запити користувачів, запити і скарги, статистика витрат і доходів.

4. Аналітичне або імітаційне моделювання систем.

Останній метод є чи не єдиним інструментом, доступним на етапі проєктування програмного забезпечення, коли виконується попередня оцінка продуктивності. Перші три методи частіше використовуються при оцінці існуючих систем і альтернативних конфігурацій.

Відповідні лекції програмні результати навчання за Стандартом 2018:

– ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення;
– ПР19 – знати та вміти застосовувати методи верифікації та валідації програмного забезпечення;
– ПР20 – знати підходи щодо оцінки та забезпечення якості програмного забезпечення.

У п'ятій лекції розділу «Механізми ціноутворення» К. К. Готліб показує, що ціноутворення відіграє важливу роль у розподілі сервісних ресурсів та раціоналізації планування. Рівень цін визначається вартістю, а також міркуваннями політики. У лекції розглядаються різні методи встановлення рівнів цін разом із деякими наслідками та вимогами, які випливають з них.

До основних компонентів вартості автор відносить заробітну плату (фахівцям з управління, експлуатації, застосування, розробки) та додаткові пільги (пенсія, страхування, включно із медичним), обладнання (оплата за покупку або оренду, обслуговування, витрати на зв'язок, офісне обладнання), постачання (носії даних, папір, документація), програмне забезпечення (придбане, орендоване, самостійно розроблене), місце (приміщення, витрати на підготовку, комунальні послуги), накладні витрати (використання послуг з придбання та обслуговування, бібліотеки), різне (подорожі, реклама, керівництва користувача тощо).

Автор аналізує ринок комп'ютерних послуг, що сформувався на той час, і пропонує приклади різних моделей ціноутворення.

Відповідні лекції програмні результати навчання за Стандартом 2018:

– ПР18 – знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних;
– ПР24 – вміти проводити розрахунок економічної ефективності програмних систем.

Х. Й. Хелмс (Hans Jørgen Helms), автор останньої лекції розділу – «Виконання програм у середовищі обчислювального центру», яка завершує курс, був одним із співголів програмного комітету конференції НАТО 1968 року. У 1965-1974 рр. він одним із провідних співробітників першого скандинавського суперкомп'ютерного центру – Northern Europe University Computing Center у Датському технічному університеті. Працюючи з 1974 року у складі Європейської Комісії, він очолював Спільний дослідницький центр Директорату науки, досліджень та розвитку, який залишив у 1995 році в статусі Генерального директора.

Х. Й. Хелмс вказує, що ця лекція відноситься не скільки до інженерії програмного забезпечення, скільки до застосування спроєктованих та розроблених програм для надання різноманітних послуг різним групам користувачів. Тому під середовищем обчислювального центру автор розуміє спільноту людей, що використовують послуги певної комп'ютерної системи: агент з продажу авіаквитків, що використовує систему бронювання місць; друкарка, яка використовує систему редагування тексту; касир банку, що використовує онлайнову системи обліку; менеджер, який використовує інформаційну систему управління; інженер-консультант, який використовує стандартні інженерні програми з терміналу у своєму кабінеті; хімік, який розробляє програми для вирішення власних дослідницьких завдань; студент, який розв'язує вправи з курсу інформатики; програміст, який розробляє програми для замовника та ін. [3, с. 504].

Середовище обчислювального центру – велика розподілена спільнота користувачів, яка отримує доступ до обчислювальних послуг віддалених комп'ютерів через мережні термінали. На основі проведеного аналізу автор вказує напрями оптимізації відповідного програмного забезпечення за різними показниками (час виконання, час обслуговування та ін.) з метою забезпечення задоволення потреб зростаючої кількості користувачів та уникнення потенційних «вузьких місць» (bottle-necks) обслуговування.

Автор наголошує на спільності університетських обчислювальних центрів, що надають послуги за потреби без урахування їх прибутковості, з іншими постачальниками комунальних послуг, таких як поштові або транспортні. Як було показано у [5], саме такі комунальні обчислювальні послуги у подальшому трансформувались у хмарні технології.

Відповідні лекції програмні результати навчання за Стандартом 2018:

- ПР08 – вміти розробляти людино-машинний інтерфейс;
- ПР18 – знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних;
- ПР21 – знати, аналізувати, вибирати, кваліфіковано застосовувати засоби забезпечення інформаційної безпеки (в тому числі кібербезпеки) і цілісності даних відповідно до розв'язуваних прикладних завдань та створюваних програмних систем.

Висновки

1. Розроблена під керівництвом Ф. Л. Бауера перша модель професійної підготовки з інженерії програмного забезпечення не лише узагальнила наявний на початок 1970-х рр. досвід навчання різних складових інженерії програмного забезпечення, а й систематизувала його у відповідні складові підготовки. Незважаючи на майже п'ятидесятирічний вік навчальних матеріалів курсу, значна їх частина стала фундаментальною основою інженерії програмного забезпечення, заклавши напрями подальшого розвитку відповідної теорії та емпіричного узагальнення.

2. Закладена у запропонованій програмі підготовки системність (усі розділи та глави у них взаємо- та перехресно пов'язані) та масштабованість (від двотижневого інтенсиву до підготовки бакалавра) у значній мірі відповідають ідеї проєктування еволюційного програмного забезпечення. Аналіз змісту курсу не лише показав низку проблем, що стосуються співвідношення стабільної (фундаментальної) та швидкозмінної (технологічної) складових змісту навчання, а й надав можливість виявити зв'язки підготовки фахівця з інженерії програмного забезпечення із підготовкою з комп'ютерних наук, комп'ютерної інженерії, кібербезпеки, інформаційних систем і технологій.

3. Проведений аналіз співвідношення програмних результатів навчання фахівців з ІПЗ за моделлю Ф. Л. Бауера 1972 року та Стандартом 2018 року надав можливість встановити, що:

- здатність аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки повинна формуватися у навчальних дисциплінах, що передують дисциплінам професійно-практичної підготовки фахівця з ІПЗ та розвиватись у процесі професійної підготовки та подальшої професійної діяльності;
- набуття знань кодексів професійної етики, розуміння соціальної значимості та культурних аспекти інженерії програмного забезпечення і дотримання їх у професійній діяльності вимагає окремої цілеспрямованої роботи викладача та студентів як за окремою навчальною дисципліною, так й у процесі професійної підготовки;
- фундаментальне ядро підготовки фахівця з ІПЗ повинне забезпечувати досягнення студентами таких провідних результатів навчання: ПР07 – знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного

забезпечення; ПР05 – знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення; ПР14 – застосовувати на практиці інструментальні програмні засоби доменного аналізу, проєктування, тестування, візуалізації, вимірювань та документування програмного забезпечення. Формування відповідних компетентностей майбутніх фахівців з ІПЗ необхідно виконувати у навчанні всіх дисциплін професійно-практичної підготовки;

– у моделі Ф. Л. Бауера 1972 року недостатню увагу приділено інженерним та соціально-економічним основам професійної діяльності фахівця з ІПЗ, які значною мірою знайшли своє відображення у Стандарті 2018 року.

4. Перспективи подальших досліджень полягають в аналізі співвідношення змісту загальнопрофесійних компетентностей бакалавра з ІПЗ Стандарту 2018 з альтернативними вітчизняними та зарубіжними стандартами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Naur, P. & Randell, B. (Eds.) (1968). *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels : Scientific Affairs Division, NATO.
2. Buxton, J. N. & Randell, B. (Eds.) (1970). *Software Engineering Techniques: Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. Brussels: Scientific Affairs Division, NATO.
3. Bauer, F. L. (Ed.) (1975). *Software Engineering: An Advanced Course*. Berlin, Heidelberg, New York: Springer-Verlag. (Lecture Notes in Computer Science, Vol. 30) (Formerly published 1973 as Lecture Notes in Economics and Mathematical Systems, Vol. 81). DOI: 10.1007/3-540-07168-7
4. Grosch, H. R. J. (1953). High Speed Arithmetic: The Digital Computer as a Research Tool. *Journal of the Optical Society of America*, 43 (4), 306 - 310. DOI : 10.1364/JOSA.43.000306.
5. Маркова, О. М., Семеріков, С. О. & Стрюк, А. М. (2015). Хмарні технології навчання: витоки. *Інформаційні технології і засоби навчання*, 46 (2), 29 - 44. DOI : 10.33407/itlt.v46i2.1234.
6. Стрюк, А. М. (2018). Становлення та розвиток інженерії програмного забезпечення як галузі знань. *Інформаційні технології в освіті*, 37, 101 - 136. DOI : 10.14308/ite000684.
7. Randell, B. (2018). *Fifty Years of Software Engineering - or - The View from Garmisch*. arXiv:1805.02742 [cs.SE]. Retrieved from https://arxiv.org/abs/1805.02742.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Naur, P. & Randell, B. (Eds.) (1968). *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels : Scientific Affairs Division, NATO.
2. Buxton, J. N. & Randell, B. (Eds.) (1970). *Software Engineering Techniques: Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. Brussels : Scientific Affairs Division, NATO.
3. Bauer, F. L. (Ed.) (1975). *Software Engineering: An Advanced Course*. Berlin, Heidelberg, New York : Springer-Verlag. (Lecture Notes in Computer Science, Vol. 30) (Formerly published 1973 as Lecture Notes in Economics and Mathematical Systems, Vol. 81). DOI: 10.1007/3-540-07168-7
4. Grosch, H. R. J. (1953). High Speed Arithmetic: The Digital Computer as a Research Tool. *Journal of the Optical Society of America*, 43 (4), 306 - 310. DOI : 10.1364/JOSA.43.000306.

5. Markova, O. M., Semerikov, S. O. & Striuk, A. M. (2015). The cloud technologies of learning: origin. *Information Technologies and Learning Tools*, 46 (2), 29 - 44. DOI : 10.33407/itlt.v46i2.1234.
6. Striuk, A. M. (2018) Formation and development of software engineering as a knowledge area. *Information technologies in education*, 37, 101 - 136. DOI : 10.14308/ite000684.
7. Randell, B. (2018). *Fifty Years of Software Engineering - or - The View from Garmisch*. arXiv:1805.02742 [cs.SE]. Retrieved from https://arxiv.org/abs/1805.02742.

Стаття надійшла до редакції 25.07.2019
The article was received 25 July 2019.

Andrii Striuk
Kryvyi Rih National University, Kryvyi Rih, Ukraine

“ADVANCED COURSE ON SOFTWARE ENGINEERING” AS THE FIRST MODEL FOR TRAINING OF SOFTWARE ENGINEERS

Designing a mobile-oriented environment for professional and practical training requires determining the stable (fundamental) and mobile (technological) components of its content and determining the appropriate model for specialist training. In order to determine the ratio of fundamental and technological in the content of software engineers’ training, a retrospective analysis of the first model of training software engineers developed in the early 1970s was carried out and its compliance with the current state of software engineering development as a field of knowledge and a new the standard of higher education in Ukraine, specialty 121 “Software Engineering”. It is determined that the consistency and scalability inherent in the historically first training program are largely consistent with the ideas of evolutionary software design. An analysis of its content also provided an opportunity to identify the links between the training for software engineers and training for computer science, computer engineering, cybersecurity, information systems and technologies. It has been established that the fundamental core of software engineers’ training should ensure that students achieve such leading learning outcomes: to know and put into practice the fundamental concepts, paradigms and basic principles of the functioning of language, instrumental and computational tools for software engineering; know and apply the appropriate mathematical concepts, methods of domain, system and object-oriented analysis and mathematical modeling for software development; put into practice the software tools for domain analysis, design, testing, visualization, measurement and documentation of software. It is shown that the formation of the relevant competencies of future software engineers must be carried out in the training of all disciplines of professional and practical training.

Keywords: software engineering, professional training, software, specialist training model, standard of higher education.