**УДК [004+519.684]:378**
**Стрюк А. М.**
**Криворізький національний університет, Кривий Ріг, Україна**

# *СТАНОВЛЕННЯ ТА РОЗВИТОК ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЯК ГАЛУЗІ ЗНАНЬ*

DOI: 10.14308/ite000684

*У статті представлено аналіз основних етапів розвитку інженерії програмного забезпечення (ІПЗ) як галузі знань, виокремлено фундаментальні складові підготовки майбутніх інженерів-програмістів, визначено тенденції розвитку цієї галузі на найближче десятиліття. Сучасна ІПЗ базується на трьох групах ключових принципів: основні концепції комп'ютерних наук, пов'язані зі структурами даних, алгоритмами, мовами програмування та їх семантикою, аналізом, обчислювальністю, моделями обчислень тощо; основи інженерії, пов'язані з архітектурою, процесами інженерії, компромісами та витратами, стандартизацією, якістю та гарантіями та інші складові, що забезпечують підхід до проектування та вирішення проблем; соціально-економічні основи, що включають процес створення та еволюції артефактів, а також питання, пов'язані з політикою, ринками, зручністю використання та соціально-економічними впливами; це забезпечує основу для формування інженерних артефактів, що будуть відповідати їхньому призначенню.*

*Сучасна ІПЗ є невід'ємною складовою переважної більшості інновацій у всіх сферах розвитку суспільства, науки та техніки, пропонуючи системні, практичні, економічно вигідні рішення для обчислювальних задач та задач опрацювання інформації. За час розвитку ІПЗ як окремої галузі накопичено значний досвід проектування, впровадження, тестування та документування програмного забезпечення, виокремлено системні наукові, технологічні підходи і методи до проектування та конструювання комп'ютерних програм. У той же час дослідники зазначають, що ІПЗ ще досі не досягла такого рівня сталості, як інші галузі інженерії. Аналіз історичних етапів розвитку ІПЗ показав, що незважаючи на загальне визнання важливості застосування при розробленні програмного забезпечення математичного апарату логіки, теорії автоматів та лінгвістики, вона створювалась емпіричним способом без його використання. Фактором, що змушує програмістів-практиків звернутися до математичних основ ІПЗ, є зростання складності програмного забезпечення і нездатність емпіричних підходів до його розроблення та управління впоратися з нею. У професійній підготовці інженерів-програмістів виділено проблему швидкого застарівання технологічного змісту навчання, розв'язання якої полягає у його фундаменталізації через виокремлення базових основ галузі.*

**Ключові слова:** *інженерія програмного забезпечення; професійна підготовка; програмне забезпечення; програмна система; програмування; проектування; моделювання.*

## Вступ

Починаючи з 2012 року серед пріоритетних напрямів освіти й науки щодо навчання студентів та аспірантів, стажування наукових і науково-педагогічних працівників у провідних вищих навчальних закладах та наукових установах за кордоном [32], що належать до інформатики та обчислювальної техніки, три – програмна інженерія, програмне забезпечення систем та інженерія програмного забезпечення – входять до однієї спеціальності: 121 – Інженерія програмного забезпечення. Крім того, значна частина інших

пріоритетних напрямів (математичне та комп'ютерне моделювання; інформаційно-комунікаційні технології; системи штучного інтелекту; системне програмування та ін.) є дотичними до неї. Технології та засоби розробки програмних продуктів і систем визначено як один із пріоритетних напрямів наукових досліджень і науково-технічних розробок в Україні на період до 2020 року [33]. Ці та низка інших законодавчих ініціатив нашої держави є свідченням нагальної суспільної потреби у компетентних фахівцях з інженерії програмного забезпечення (ІПЗ), підготовлених на основі кращих світових стандартів та передового зарубіжного досвіду і здатних до проектування, апробації, упровадження та комерціалізації інноваційних технологій ІПЗ. Аналіз світового досвіду з підготовки фахівців з ІПЗ доречно почати з ретроспективного огляду еволюції самого поняття «інженерія програмного забезпечення» та основних етапів розвитку цієї галузі.

У роботах низки вітчизняних та зарубіжних авторів розглянуто різні аспекти професійної підготовки майбутніх фахівців з інженерії програмного забезпечення:

– моделювання та проектування професійної підготовки майбутніх фахівців з ІПЗ (Н. Х. Валєєва, Н. К. Нурієв, В. С. Круглик, З. С. Сейдаметова, Л. А. Матвійчук, Р. Д. Ріель (Richard D. Riehle), Х. Рьодер (Holger Röder), М. Дж. Бок (Michael J. Bok), Т. Ч. Лінг (Thong Chee Ling), А. Костер (Alexis Koster), Д. Броман (David Broman), А. Боллін (Andreas Bollin), Ш. Лі (Shu Liu) та ін.);

– компетентності майбутніх фахівців (В. В. Калітіна, М. С. Лежньова, М. М. Міншин, Є. П. Нехожина, В. М. Пелевін, Г. В. Прозорова, І. Г. Фрізен, О. П. Юрковець, С. Гоель (Sanjay Goel) та ін.);

– навчання програмування (В. Є. Жужжалов, Л. О. Кугель, Л. В. Гришко, О. М. Петров, В. П. Коротков, М. В. Слива та ін.);

– ІПЗ як професія (Д. А. Мустафіна, І. Г. Фрізен, М. С. Ємельянова, А. О. Ричкова та ін.);

– методи та підходи до навчання майбутніх інженерів-програмістів (В. П. Агальцов, З. А. П. Зайнал (Dzulaiha Aryanee Putri Zainal), Р. Разалі (Rozilawati Razali), П. А. Манохар (Priyadarshan A. Manohar), Ж. М. Фернандес (João M. Fernandes) та ін.);

– навчання технології розробки програмного забезпечення (Ф. С. Ільясова, Е. Дж. Паттерсон (Andrew Joseph Patterson), А. С. Вільямс (Arrena Sue Williams), П. Дж. Кларк (Peter J. Clarke), Т. Доулінг).

**Метою статті** є аналіз основних етапів розвитку ІПЗ як галузі знань, виокремлення фундаментальних складових підготовки майбутніх інженерів-програмістів та визначення тенденцій розвитку цієї галузі на найближче десятиліття.

**Виникнення інженерії програмного забезпечення**

Р. Кейл-Славік (Reinhard Keil-Slawik) у 1996 році згадував, що термін «програмна інженерія» (software engineering) був навмисне обраний як провокативний для першої конференції 1968 року з ІПЗ, що відбулась у Німеччині: «це поняття означало, що виробництво програмного забезпечення має базуватися на тому ж типі теоретичних засад та практичних застосувань, що й у традиційних галузях інженерії» [12, с. I; 25, с. 13].

Перше вживання цього терміна датується серпнем 1966 року [17], а рішення про проведення конференції було прийняте вже на початку 1967 року. Її головною тематикою було проектування, виробництво та обслуговування програмного забезпечення. Один із головних учасників конференції П. Наур (Peter Naur) зазначав, що робота проектувальників програмного забезпечення схожа на роботу архітекторів та інженерів-будівельників, особливо тих, хто займається проектуванням великих гетерогенних конструкцій, таких як міста та промислові підприємства [25, с. 13].

Походження програмної інженерії учасники конференції пов'язували із кризою програмного забезпечення – терміном, запропонованим головою програмного комітету Ф. Л. Бауером (Friedrich Ludwig "Fritz" Bauer). На його думку, криза полягала у

неможливості застосування «кустарних» (напівінтуїтивних) методів розробки для виробництва великих масштабованих програмних систем: «Існуюче програмне забезпечення розробляється аматорами (незалежно від того, де – в університетах, компаніях чи на виробництві) за допомогою майстерності одинаків (в університетах) або великої кількості працівників («мільйон мавп») на виробництві, є ненадійним і потребує постійного «технічного обслуговування» (причому слово «обслуговування» неправильно використовується для позначення збоїв та відмов, що очікуються від виробника із самого початку), є неохайним, непрозорим та невдосконалюваним (або принаймні занадто вартісним, щоб це зробити). І нарешті, існуюче програмне забезпечення надходить занадто пізно і коштує дорожче, ніж очікувалося, та не виправдовує сподівань, що на нього покладалися» [3]. За результатами роботи конференції у 1971 році Ф. Л. Бауер дав напівжартівливе визначення ІПЗ як частини інформатики, що занадто важка для інформатиків, та більш серйозне – як створення та використання раціональних принципів інженерії для отримання економічного, надійного та ефективно працюючого на реальних комп'ютерах програмного забезпечення [3, с. 530].

На широко обговорюваній схемі, запропонованій на с. 20 звіту про конференцію [25], показано деякі проблеми такого виробництва на шкалі «ресурси – час»: так, найбільші витрати людських та фінансових ресурсів припадають на середню стадію проекту (етапи тестування компонентів та системи у цілому й технічну підтримку). У процесі обговорення один із учасників конференції А. д'Агапеефф (Alex d'Agapeyeff) запропонував перевернуту піраміду програмного забезпечення (рис. 1) та увів поняття проміжного програмного забезпечення (middleware).

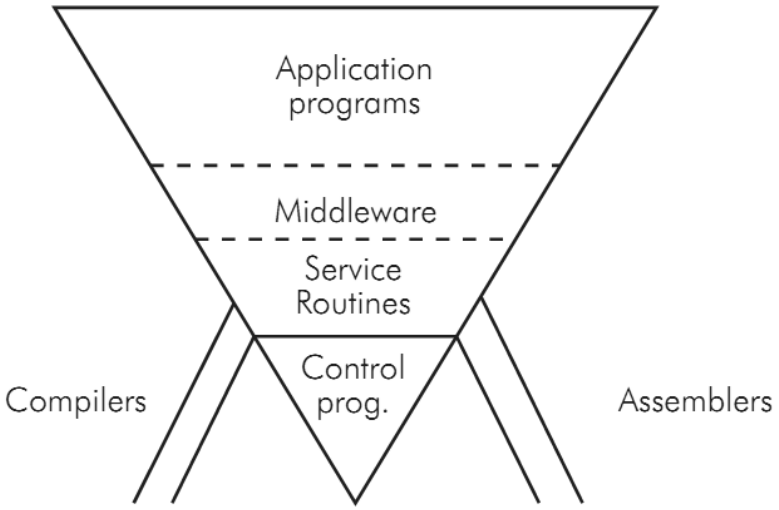


*Рис. 1. Перевернута піраміда д'Агапеeффа.*

На с. 25-30 матеріалів конференції наведена запропонована Дж. Харром (J. A. Harr) загальна схема розробки програми, що розпочинається із проектування:

– специфікація повної програмно-апаратної системи;
– визначення функцій, що виконуватиме програма;
– проектування та документування програми у цілому;
– розділення великої програмної системи на керовані програмні блоки;
– документування програмних інтерфейсів;
– визначення та документування підпрограм;
– детальне проектування, кодування та документування кожного програмного блока;
– паралельно із попереднім – проектування та документування тестів для кожного програмного блока;
– компіляція та «ручна» перевірка кожного програмного блока;

– моделювання кожного програмного блока з використанням тестових методів, запланованих під час проектування програми;

– тестування та оцінювання програмних блоків у системі;

– інтеграція готової програми із системою;

– заключне тестування (під навантаженням) повного програмно-апаратного пакету для визначення відповідності програми проектним вимогам.

Дж. Харр на прикладі системи Electronic Switching Systems (електронного телефонного комутатора) наводить структуру колективу розробників та показує, що у процесі реалізації проекту вона змінюється в залежності від етапу реалізації проекту – якщо на початку головна увага приділяється плануванню системи та виокремленню вимог, то згодом центр уваги зміщується на обслуговування програмного забезпечення. Автор пропонує використовувати розробників програмного забезпечення різного рівня: техніків, бакалаврів, магістрів та докторів філософії з електричної інженерії та математики, налагодивши між ними наступні способи взаємодії та документообігу:

- групові зустрічі зі специфікацій проектування;
- формальні вимоги до проектування програм;
- неформальні пам'ятки з інженерії та програмування;
- дані, що характеризують програму;
- формальна програмна документація.

Вимірювання роботи програміста пропонувалось у таких одиницях, як кількість людино-років та кількість слів програми на одного програміста.

Документування передбачалось для:

- методів та стандартів, що використовувались у підготовці та редагуванні програми: опису програми, блок-схем, програмного коду та змін у ньому;
- підготовки та оновлення керівництва користувача, до якого включались також процедури обслуговування програми.

Дж. Харр вказує, що для розробки програмної системи може бути необхідна розробка додаткових комп'ютерних засобів, таких як макрокомпілятори, асемблери, завантажувачі параметричних компіляторів, симуляторів для тестування апаратного забезпечення та дослідження трафіку тощо.

Останнім кроком є тестування програмного забезпечення та оцінка програм підтримки шляхом симуляції на комп'ютері загального призначення та на самій програмно-апаратній системі (спочатку тестування програмних блоків, далі – програмних функцій і нарешті – комплексне тестування під навантаженням на програмно-апаратній системі).

Р. С. Бартон (Robert Stanley Barton) запропонував називати людину, яка займається проектуванням апаратного та програмного забезпечення, «комп'ютерним інженером» ('a computer engineer') [25, с. 32]. Саме такий підхід використано у вітчизняному класифікаторі спеціальностей, що виокремлює спеціальності 121 Інженерія програмного забезпечення та 123 Комп'ютерна інженерія.

На думку таких учасників конференції, як А. Дж. Перліс (Alan Jay Perlis) та Ф. Л. Бауер, для проектування програмного забезпечення математична підготовка не є необхідною, але її наявність додає програмному проекту елегантності, адже програмні системи є математичними за природою та повинні бути побудовані за рівнями та модулями, що утворюють математичну структуру [25, с. 37]. Основними критеріями проектування були обрані загальні критерії (спільні для різних систем), користувацькі вимоги, надійність та логічна повнота. Серед технологій проектування обговорювались послідовність кроків проектування, структура програмного проекту, забезпечення зворотного зв'язку через моніторинг та моделювання, застосування високорівневих мов програмування тощо.

**Професійна підготовка фахівців з інженерії програмного забезпечення**

Стосовно професійної підготовки фахівців з ІПЗ А. Дж. Перліс та Е. Е. Девід-молодший

(Edward Emil David Jr.) поставили низку проблемних запитань [25, с. 125-126]:

1. Чи можливо працювати інженером-програмістом без формальної освіти з відповідної спеціальності?
2. Чи співпадає ІПЗ із комп'ютерними науками?
3. Як краще підготувати фахівця з ІПЗ: за бакалаврською програмою в університеті, на курсах підвищення кваліфікації або за дворічною програмою підготовки після стандартної шкільної освіти?
4. Чи матимуть фахівці, підготовлені за цими програмами, ріст та перспективи у нашому суспільстві?
5. Чи будуть вони достатньо корисними для фірми, уряду чи університету і чи є їх значення таким, що вони можуть реалізовувати свої таланти в інших видах діяльності, або вони назавжди приречені залишатися програмістами?
6. Які програми підготовки необхідні для фахівців з ІПЗ незалежно від рівня освіти?
7. Чи ІПЗ дійсно відрізняється від того, що ми зараз називаємо системною інженерією?
8. Що спільного мають ІПЗ та комп'ютерна інженерія із традиційною інженерною освітою у Сполучених Штатах Америки або Західної Європи?

Д. Т. Росс (Douglas Taylor Ross), відповідаючи на поставлені запитання, наголошував на необхідності окремої формальної освіти з ІПЗ на рівні бакалавра [25, с. 127]. Інші учасники дискусії порушували питання практичної підготовки як фахівців, так і викладачів на відповідній спеціальності.

Конференція 1969 року [26], що відбулась в Італії, була присвячена технологіям ІПЗ. На відміну від попередньої, професійна підготовка фахівців з ІПЗ стала предметом обговорення на окремій секції конференції.

А. Дж. Перліс, продовжуючи розпочату на попередній конференції дискусію, акцентував увагу на трьох питаннях:

1. Чи існує реальна відмінність між ІПЗ та комп'ютерними науками?
2. Якщо вона існує, то чи потрібне вивчення ІПЗ як окремої дисципліни?
3. За якою формою повинні викладатися університетські курси з ІПЗ?

Обговорення цих та інших питань, включно із тим, чи достатньо усталеними є комп'ютерні науки, щоб них можна було навчати студентів, і чи мають вони певні явні базові принципи.

На перше питання А. Дж. Перліс відповідав ствердно: «Я думаю, що всі ті з нас, хто працює в університетах, добре розуміють сутність Ph. D. програми з комп'ютерних наук: здорова доза логіки, теорії автоматів та обчислень, трохи менша – чисельного аналізу, один-два курси з поглибленого програмування того чи іншого виду, трохи штучного інтелекту, і дещицю ще чогось» [26, с. 61-62]. Ф. Л. Бауер зауважив, що у Німеччині робоча група, що розробляє програми підготовки, назвала відповідний предмет інформатикою ("Informatik"): «Ми очікуємо, що наші студенти самі зроблять вибір, будуть вони фахівцями з комп'ютерних наук чи з інженерії програмного забезпечення» [26, с. 62]. На думку А. Дж. Перліса, для ІПЗ фундаментальними є курси дослідження операцій та управління проектами – настільки ж фундаментальними, як і курс теорії автоматів, та більш фундаментальними, ніж будь-який математичний курс [26, с. 62].

Певним продовженням цієї дискусії є інша публікація А. Дж. Перліса [18], у якій він підкреслює суспільну значущість професії інженера із програмного забезпечення та нагальну необхідність розробки відповідних освітніх програм їх підготовки. Спеціалізацією таких інженерів є програмне забезпечення, а саме його проектування, виробництво та обслуговування. До заданих на обговорюваних конференціях проблемних запитань А. Дж. Перліс додає ще декілька:

1. Якщо підготовка фахівців з ІПЗ відбуватиметься в університеті, то на якій кафедрі або факультеті?
2. Чи програма підготовки повинна бути окремою, або вона може бути варіативною

частиною іншої програми?

3. Чи будуть за такою програмою фахівці підготовлені для вирішення системних проблем, що виникнуть у майбутньому?

4. Чому ми говоримо про інженерію, а не про науку?

Автор пропонує розпочинати з магістерської програми, далі поширюючи її на бакалаврат та докторат. «Метою є зосередження на відомих інструментах та їх ефективному використанні, а не на періодах інтенсивних інновацій та відкриттів. Вибір серед існуючих способів проектування є більш важливим, ніж розробка нових. Питання стабільності є більш критичними, ніж питання росту та змін. Визначення обсягу задачі настільки ж важливе, як і відкриття методу. Керівництво командами настільки ж критичне, як і надихання на досягнення. ... На мій погляд, професійна підготовка з інженерії програмного забезпечення є сплавом математики, теорії управління, комп'ютерних наук та практичного досвіду, отриманого при роботі з актуальними програмними системами та пов'язаними проблемами» [18, с. 541].

Окреслюючи програму підготовки, А. Дж. Перліс виокремлює професійні функції інженера, формулюючи перелік із 8 питань, відповіді на які повинен дати підготовлений фахівець:

1. Для поданої задачі на розробку програмного забезпечення (подібної до відомих) та набору комп'ютерів, оцініть комп'ютери та задачу з метою обґрунтованого вибору комп'ютера та оптимального подання програмної системи. Вкажіть критерії оптимальності, визначте необхідність підтримки з боку виробника обладнання, способи забезпечення сумісності, стабільності та природні шляхи змін, зростання та поліпшення.

2. Для поданої задачі на розробку програмного забезпечення визначте раціональний графік його завершення за різних ситуацій із персоналом. Як ви отримуєте, підготуєте або навіть розпізнаєте адекватних програмістів? Яке робоче навантаження слід встановити?

3. Якщо *n* осіб програмують систему, що буде робите *n*+1 тощо?

4. Як протестувати систему? Яку систему ви організуєте для обробки та реагування на навантаження в іншій системі?

5. Як продати систему? Що робить систему корисною? Як скопіювати чужу систему?

6. Як і чого ви навчаєтесь, будуючи систему? Що потрапляє до Вашого набору інструментів після розв'язання задачі на розробку програмного забезпечення?

7. Які є професійні інструменти? Як вони класифікуються? Чи пов'язані вони з різноманіттям обладнання?

8. Як ви об'єднуєте окремі системи у взаємопов'язані при розв'язанні розширених задач?

Дж. Фельдман (Jerome A. Feldman) із власного досвіду наводив приклади того, що фахівці на виробництві не читають літературу, навіть якщо в ній наявні рішення їх виробничих проблем: «Ті, хто опанували освітню програму в галузі комп'ютерних наук, у цьому відношенні є більш ефективними. Ми намагаємось подолати цю проблему в Стенфорді через започаткування програми підготовки з прикладних комп'ютерних наук, спрямованої на підготовку ефективних фахівців для промисловості» [26, с. 62]. Р. М. Макклюр (Robert M. McClure) зазначив, що університети мають значну проблему, в основі якої лежить розмежування між ІПЗ та комп'ютерними науками.

Б. Галлер (Bernard A. Galler) навів приклад курсу ІПЗ: «Ми беремо команду з двох або трьох викладачів і 20-25 студентів та дали їм проект. Перший – графічна математична система, другий – система програмування мовою BASIC. Ми дали їм повну роботу: проектування, вказавши технічні характеристики та розділивши їх на групи по дві-три особи, реалізація, опис власної роботи для інших груп, розробка інтерфейсів, документування та ін. У кожному випадку продукт був практично корисним; за один семестр ви не зможете завершити щось такого масштабу. Цей вид досвіду виходить за межі написання проекту малого класу типу компілятора та надає певний досвід програмної інженерії» [26, с. 62].

Д. Т. Росс вказав на обмеженість та несистемність такого підходу, а Е. Д. Фалкофф (Adin D. Falkoff) – на те, що курс ІПЗ повинен включати також роботу з апаратним забезпеченням. «Крім того, я завжди вважав, що інженерія має справу з питаннями економіки; дисципліна, пов'язана з проблемами комплексного використання ресурсів, повинна містити елементи дослідження операцій та відповідні курси» [26, с. 62-63].

Е. В. Дейкстра (Edsger Wybe Dijkstra) порушує проблему швидкого застарівання вузькопрофільних знань: «Я вважаю неправильним навчати матеріалу, який, як мені відомо, застаріє через кілька років. ... Ви повинні навчити розуміння методу, почуття якості та стилю» [26, с. 65].

Конференції з ІПЗ, проведені під егідою наукового комітету НАТО у 1968 та 1969 рр., у цілому визначили сферу ІПЗ та шляхи підготовки відповідних фахівців у закладах освіти:

1. Методи та засоби ІПЗ застосовуються до великих складних програмних систем, що не можуть бути створені однією особою або невеликим колективом розробників.

2. Попри свою назву, ІПЗ повинна включати питання взаємодії програмної та апаратної складових комп'ютерної системи.

3. Метою ІПЗ є розробка програмних систем із наперед визначеними рівнями якості, надійності та ефективності в умовах обмеження ресурсів (часових, людських, матеріальних, програмно-апаратних тощо). У зв'язку із цим, на відміну від комп'ютерних наук (інформатики), питання дослідження операцій та управління проектами для ІПЗ забезпечення є фундаментальними.

4. Підготовка фахівців з ІПЗ у ЗВО є доцільною на рівні бакалаврату. В процесі підготовки доцільно поєднувати теоретичну та практичну підготовку (на виробництві або із застосуванням запозичених із виробництва методів розробки програмного забезпечення).

5. Суттєвим для підготовки фахівців з ІПЗ є вивчення артефактів, що створюються у процесі програмної інженерії: документації, програмного коду, меморандумів, групових обговорень та ін.

6. У навчанні ІПЗ («технології програмування») із самого початку гостро постала проблема швидкого застарівання технологічного змісту навчання, розв'язання якої полягає у його фундаменталізації через виокремлення базових основ галузі.

Останнє викликало найбільшу дискусію, за результатами якої було визначено, що опанування основ комп'ютерних наук («інформатика» за Ф. Л. Бауером та «математична інженерія» за Е. В. Дейкстрою) є фундаментом професійної підготовки з ІПЗ.

**Етапи розвитку інженерії програмного забезпечення**

У грудні 1969 році в США відбувся третій симпозіум з ІПЗ, перший том матеріалів якого відкривала програмна доповідь Ю. Т. Ту (Julius T. Tou) «Нова професія: інженерія програмного забезпечення» («Software Engineering – A New Profession») [27]. Автор, характеризуючи роль інженерії у розвитку людства, вказує, що вона значною мірою звільнила людину від фізичної праці та рутинних розумових дій, надавши інструменти як для нових наукових відкриттів, так і нових видів творчості. Стрімкий розвиток інженерії, поява нових її галузей змінили зміст інженерної освіти – від утилітарної до науково-теоретичної. Так само як винахід парової машини наприкінці вісімнадцятого сторіття дозволив замінити м'язову силу людей і тварин рушійною силою машин, винахід цифрового комп'ютера після Другої світової війни дозволив замінити багато людських розумових задач, таких як арифметичні обчислення, зберігання даних та ведення обліку, комп'ютерними операціями: «Ми зараз переходимо до стадії, на якій доцільно передбачити заміну деяких вищих розумових задач людини машинами. Це включає в себе здатність розпізнавати шаблони, читати зображення, опрацьовувати дані природною мовою, отримувати інформацію та приймати розумні рішення» [27, с. 2]. Для цього автор пропонує скористатися принципами ІПЗ.

Ю. Т. Ту виділяє три основні етапи розвитку інженерії у XX сторіччі:

1) етап перетворення, передавання та поширення електричної енергії, пов'язаний із розвитком електромеханіки;
2) етап фільтрації та опрацювання сигналів, пов'язаний із розвитком електромеханіки електроніки та індустрії зв'язку;
3) етап опрацювання даних, пов'язаний із розвитком комп'ютерної техніки.

Ускладнення та розвиток комп'ютерної техніки призвели до того, що для ефективного опрацювання даних користувачам потрібні професійні послуги спеціаліста – інженера з програмного забезпечення. Ю. Т. Ту наводить структурну схему, що ілюструє різні види такої діяльності (рис. 2) з використанням апаратного забезпечення, мікропрограм та програмного забезпечення, що стосуються відповідно проектування апаратури для опрацювання даних (використовуючи доступні схеми, чіпи та пристрої з урахуванням швидкості, розміру, ваги, надійності, вимог інтерфейсу тощо), системного програмування (для конкретної архітектури та організації комп'ютера з урахуванням розміру та порядку слів, формату інструкцій, типів регістрів, структури каналів, схеми адресації, ієрархії пам'яті, вимог до зберігання та інтерфейс введення-виведення) і прикладного програмного забезпечення.

Програмні аспекти стосуються, перш за все, системного програмування та проектування інформаційних систем. Серед аспектів проектування для перших – мови програмування, процеси компіляції та компоновки, операційні системи та ін. Проектування інформаційних систем пов'язане із розробкою програмного забезпечення для інформаційних систем із різною прикладною метою, таких як пошук даних, розпізнавання образів, опрацювання зображень, управління процесами. На думку Ю. Т. Ту, ІПЗ повинна охоплювати архітектуру комп'ютерів, системне та прикладне програмування [27, с. 4].

Стосовно ІПЗ автор наголошує, що «для того, щоб досягти значного прогресу у розробці програмного забезпечення, ми повинні мати набагато міцніший науковий фундамент. ... Інакше це стане ремеслом, а не галуззю знань. ...

Цифровий комп'ютер сьогодні нагадує паровий двигун у дев'ятнадцятому столітті. Парова машина широко використовувалась задовго до розробки термодинаміки та статистичної механіки. ... Проте [їх] теоретичні основи дозволили нам розробити більш ефективні парові двигуни та інші машини для перетворення енергії та стимулювали винахід двигуна внутрішнього згоряння, дизельного двигуна, парової турбіни, газової турбіни та навіть реактивного двигуна, які відрізняються від парової машини. Якщо б ми не мали теоретичних основ, ми все ще залишилися на стадії, яка не була надто далеко від віку парового двигуна. ...

Інженерія програмного забезпечення сьогодні має деяку схожість з електроенергетичною інженерією на рубежі нашого століття ... Електроенергетична інженерія займається виробництвом, передачею, розподілом та використанням електроенергії, а інженерія програмного забезпечення займається зберіганням, видобуванням, аналізом, перетворенням та відображенням інформації. Інженер-електроенергетик відповідає за проектування, експлуатацію та технічне обслуговування електростанцій та електричних систем. Інженер-програміст відповідає за проектування, експлуатацію та обслуговування інформаційних систем, до складу яких входять комп'ютерні виробництва, обчислювальні центри та центри опрацювання даних для різних галузей їх застосування. Елементами проектування для електроенергетичної інженерії є схематичні діаграми, електричні схеми, плани тощо. На противагу, елементи проектування для інженерії програмного забезпечення – програми, блок-схеми, мови, компілятори, алгоритми тощо. На початку нашого століття попит на електроенергію був настільки великий, що підприємства-постачальники розпочали програму прискореної експансії та виникла гостра нестача інженерів-електроенергетиків. Це трапляється сьогодні з програмним забезпеченням. Окрім необхідності виробників комп'ютерів та компаній з розробки програмного забезпечення, у всіх основних галузях промисловості, дослідницьких лабораторіях та навчальних закладах

створені обчислювальні центри та інформаційні системи. Їх ріст викликав брак інженерів-програмістів, суттєво більший, ніж дефіцит інженерів-електроенергетиків на початку цього століття. Ситуація ускладнюється тим, що обробка інформації набагато складніша, ніж перетворення та використання електричної енергії» [27, с. 5-6].

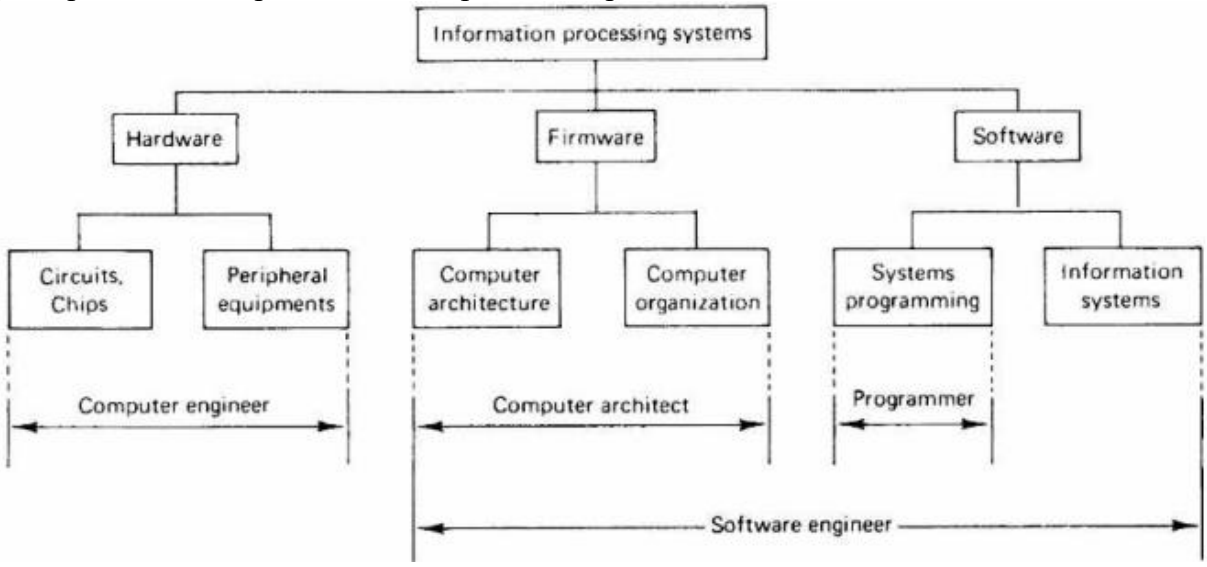


*Рис. 2. Основні види діяльності в інформаційних системах (за [27, с. 4]).*

Ю. Т. Ту оптимістично прогнозує, «що комп'ютер стане інструментом, без якого ніхто не зможе прожити, а інженерія програмного забезпечення стане основною галуззю нашого часу», і завершує доповідь фразою Р. К. Пучинського (Roman Conrad Pucinski) [19]: «Інженерія програмного забезпечення не лише є основним ключем до вирішення проблем науки і техніки, але й має силу для формування нового суспільства, унікального в усій історії людства» [27, с. 6].

Ф. Л. Бауер, характеризуючи проектування та виробництво програмного забезпечення як промислову галузь інженерії, формулює способи боротьби зі складністю великих програмних проектів: розділення на керовані частини з визначенням інтерфейсів між ними, визначення ієрархії компонентів (наприклад, деревовидної), розділення процесу розробки на окремі стадії. «Всі процеси проектування, виробництва та обслуговування повинні автоматизуватися, ... зокрема:

- автоматичне оновлення та контроль якості документації;
- вибіркове розповсюдження інформації серед всіх співробітників проекту;
- моніторинг термінів виконання проекту;
- збір даних для моделювання;
- збір даних для контролю якості;
- автоматичне генерування керівництв користувача та матеріалів із технічного обслуговування» [3].

Значну роль в ІПЗ Ф. Л. Бауер відводить структурному програмуванню – новому (на той час) підходу, запропонованому Е. В. Дейкстрою, ілюструючи його ієрархією абстракцій мов розв'язання задач (від людської до машинної) та уводячи поняття проміжної мови та абстрактної машини, що надають можливість перенесення програмних систем між різними апаратними платформами. Використання технології структурного програмування, на думку Ф. Л. Бауера, надає можливість розробки *мобільного та адаптивного програмного забезпечення* (portable software and adaptable software) на основі компонентного підходу (software components) та генераторів програмного коду («macro generators which allow the specification of new macros»). Подальший розвиток ІПЗ він пов'язує з розробленням відповідних технологій та засобів у співпраці університетських та промислових фахівців.

Одним із перших дослідників, який виокремив професійні функції фахівця з ІПЗ (Software Engineer), була А. С. Вільямс. На рис. 3 відтворено її класифікацію 1976 року

професій, пов'язаних із комп'ютерною технікою.

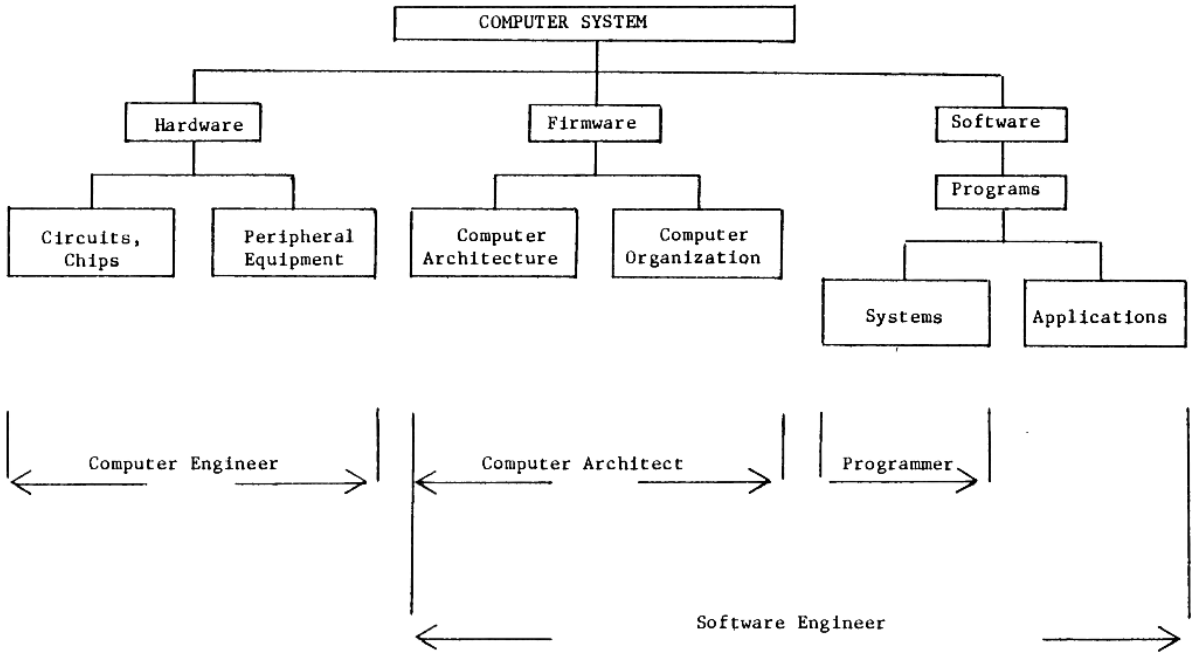


*Рис. 3. Професія Software Engineer за [28, с. 15].*

«Як показано на рисунку [№ 1], професія інженера з програмного забезпечення включає в себе як проектування комп'ютерної архітектури та організації комп'ютера, так і проектування системних та прикладних програм» [28, с. 13-14]. Наведена схема є подібною до більш ранньої Ю. Т. Ту, але містить суттєві уточнення: по-перше, мова йде не про інформаційні системи, а про комп'ютерні системи у цілому; по-друге, А. С. Вільямс не зводить прикладне програмне забезпечення виключно до інформаційних систем. Х. Д. Міллс (Harlan D. Mills) розширює ІПЗ до математичної основи, необхідної для управління комп'ютерами у складних застосуваннях [16].

Х. Д. Міллс вказує, що тривалий час програмне забезпечення, незважаючи на визнання важливості застосування при його розробці математичного апарату логіки, теорії автоматів та лінгвістики, створювалось емпіричним способом без його використання. Фактором, що змушує програмістів-практиків звернутися до математичних основ ІПЗ, є зростання складності програмного забезпечення і нездатність емпіричних підходів до його розробки та управління впоратися з нею [16, с. 1199]. Тому мета, що поставили перед собою Х. Д. Міллс разом з іншими авторами [31], – показати програмістам, як, використовуючи систематичні методи аналізу і синтезу програм, зробити свою роботу більш продуктивною, ознайомити їх із прийомами проектування надійного й ефективного програмного забезпечення. Однією з центральних проблем книги є проблема математичного доведення правильності програм: «доведення є методом експериментальної перевірки правильності програм і грає таку ж роль, як експеримент у фізичній або хімічній лабораторії – тільки тут предметом експериментального дослідження служить об'єкт, відмінний від фізичних предметів або матеріалів» [31, с. 15]. Вирішення цієї проблеми приводить до двох важливих результатів: 1) дозволяє контролювати творчу діяльність – невід'ємний компонент процесу проектування програмного забезпечення та 2) робить можливим застосування математичного апарату доведення правильності до вже створених програмами, як тих, що працюють, так і тих, що потребують налагодження. Так як налагодження – це складний і дорогий процес, його спрощення веде до підвищення і надійності, і продуктивності. Уведення додаткового контролю процесу проектування програмного забезпечення дозволяє приділяти більше уваги питанням його ефективності і створює сприятливі можливості для розробки програмних проектів, що враховують умови реалізації.

«Звичайно, розгляд програмного забезпечення як специфічної галузі математики висуває абсолютно нові вимоги до рівня логічної строгості проектування. Для розробки окремих систем програмного забезпечення можуть залучатися сотні і навіть тисячі фахівців протягом декількох років. При створенні кожного проекту висуваються свої вимоги до документування, організації взаємозв'язку між людьми і управління ходом розробки. Ці вимоги викладаються зазвичай мовою, прийнятою серед програмістів. Останні часто приділяють підвищену увагу деталям, що призводить до затуманення математичної суті. Нерозуміння ж математичного аспекту програмування в свою чергу призводить до формування занадто складної точки зору на програмне забезпечення, заснованої на аналогіях і випадкових уявленнях». Х. Д. Міллс вказує, що роботи Е. В. Дейкстри і Ч. Е. Р. Хоора (Charles Antony Richard Hoare) зіграли значну роль для переосмислення і переоцінки програмного забезпечення та розгляду його як галузі математики: «Як тільки програмування вийшло за межі допустимої складності, відбувся поворот до дисципліни, головною метою якої протягом століть було застосування ефективного структурування з метою подолання складності, що, здавалося, не піддається управлінню. Ця дисципліна, всім нам більш-менш знайома, називається математикою. Якщо ми погодимося із правильністю судження про те, що математичні методи є найбільш ефективним засобом подолання складності, у нас не залишається іншого вибору, як тільки перебудувати область програмування таким чином, щоб стало можливим застосовувати ці методи, бо інших засобів не існує» [11, с. 4.2].

Х. Д. Міллс трактує програму як те, що необхідно для управління комп'ютерним обладнанням (апаратно-центричний підхід), а програмне забезпечення – як гармонійну систему, що забезпечує взаємодію програм, різноманітного апаратного забезпечення та людських ресурсів (системний підхід): «Люди мають такі самі архітектурні характеристики, як апаратне забезпечення, але з суттєво різними параметрами продуктивності в плані зберігання, обробки та надійності, і вони мають дуже різні набори інструкцій. Людина може виконувати логічні операції комп'ютера (у мільйони разів повільніше), але людина також може виконати інструкцію «використати свій здоровий глузд» (у мільйони разів швидше). Наприклад, у системі бронювання авіакомпанії люди із служби обслуговування клієнтів виконують роль критичних аналого-цифрових компонентів у процесі перетворення голосу на цифрові дані у процесі спілкуванням з клієнтами» [16, с. 1200]. На прикладі операційної системи автор уводить поняття абстрактної машини як об'єднання програмного та апаратного забезпечення; у свою чергу, множина таких машин теж може утворювати абстрактну машину, якою будуть керувати її користувачі, які, у свою чергу, можуть бути і її складовими (агентами) – тоді термін «програмне забезпечення» поширюється також на посібники користувача та керівництва оператора (для людей, які відповідають за їх ролі в системі). Виходячи з цього, автор визначає програмне забезпечення як «логічну доктрину гармонійної співпраці людей і машин»: «коротше кажучи, програмне забезпечення визначається як система абстрактних машин, деякі з яких викликають інші абстрактні машини, доки люди та апаратні засоби не будуть досягнуті як найважливіші агенти дій у системі» [16, с. 1201].

Пов'язуючи розвиток програмного забезпечення з опрацюванням даних, Х. Д. Міллс вказує, що у США від великих систем опрацювання даних критично залежать комерційні, державні, громадські установи та інші заклади: «з такою історією короткою дивно, що ці великі системи обробки даних взагалі існують. ... Індустрія опрацювання даних є вражаючим продуктом науки і техніки, але її практика залишає бажати кращого: вона все більше страждає «дитячими хворобами» промисловості, ща не малаа часу для розробки, випробування і вибору продуманої виробничої практики – це суміш великої мудрості та безглуздого фольклору. Це потребує допомоги з боку науки та інженерії. ... Таке відставання ... викликає особливе занепокоєння ... те, що здавалося важливим 10 або 15 років тому, може бути менш важливим внаслідок зміни потреб. Тому критично важливо правильно оцінювати

майбутні тенденції в практиці опрацювання даних, щоб передбачити галузі дослідження, що будуть актуальними тривалий час» [16, с. 1201].

Автор підкреслює не лише важливість програмного забезпечення для опрацювання даних, а й те, що у розвинених країнах опрацювання даних стало важливим національним активом в управлінні та організації промислових ресурсів, що суттєво залежить від якості програмного забезпечення. На його думку, обґрунтованими технологіями ІПЗ є ті, що виникли при розробці компіляторів та операційних систем: «Ідеї, що виникають у них, незмінно знаходять застосування у прикладних системах. Наприклад, формальні граматики та мови, що спочатку використовувались при розробці компіляторів, сьогодні широко використовуються при розробці прикладних інтерфейсів користувача, а способи декомпозиції великих систем, що спочатку використовувались при розробці операційних систем, сьогодні також є корисними у прикладних системах» [16, с. 1202].

Математичною основою подолання складності програмного забезпечення Х. Д. Міллс вважає формалізоване доведення правильності програм за допомогою логіки Хоора та структурного програмування Е. В. Дейкстри: так, на с. 1202-1204 статті [16] у науково-популярному виданні «Science» він наводить аксіоматику послідовних процесів у термінах слідування, розгалуження та умовного повторення, показує зв'язок структурних програм та алгебраїчних функцій, описує способи синхронізації процесів та організації абстрактних машин. Розширений варіант можна знайти у роботі 1972 року «Математичні основи структурного програмування», у передмові до якої Х. Д. Міллс пише, що «ідеї [структурного програмування] є потужним інструментом для мисленнєвого поєднання статичного тексту програми з динамічним процесом її виконання. Це нове співвідношення між програмою та процесом дозволяє досягти нового рівня точності в програмуванні ... від розчарувань, проб та помилок до систематичної, контрольованої за якістю діяльності. Однак для того, щоб запровадити ... таке точне програмування у промислову діяльність, ідеї структурованого програмування повинні бути сформульовані як технічні стандарти. ... Хорошим прикладом технічного стандарту є проектування логічних схем. З основних теорем булевої алгебри відомо, що будь-яка логічна схема, незалежно від рівня складності, може бути побудована з використанням тільки елементів AND, OR і NOT. Наша мета схожа: надати математично обґрунтований, надійний та практичний технічний стандарт для цілей управління [проектуванням програмного забезпечення]. Математичне обґрунтування надано переважно Коррадо Бомом і Джузеппе Якопіні (Італія), які показали, як довести, що відносно прості (структуровані) логіки управління програмою здатні реалізувати будь-які програмні вимоги. Початковий практичний досвід зі структурованим програмуванням вказує на те, що в цьому немає побічних технічних ефектів. ... Коли програмісти вчаться ... правильно писати програми, це надає їм ... нового рівня концентрації, який допомагає уникнути помилок недбалості. Цей психологічний ефект точності має математичний аналог в теорії правильності програм» [15, с. II]. «Таким чином, вимоги реальності полягають у тому, щоб звичайні програмісти з пересічними здібностями змогли навчитися писати програми, які з самого початку не містили б помилок. Знання того, що це можливо, – вже наполовину виграна битва. А вміння писати такі програми – шлях до остаточної перемоги. Набуваючи досвіду в написанні правильних програм, програміст переходить на новий психологічний рівень, що дозволяє подолати укорінену думку про те, що оцінити правильність програм, не вдаючись до експерименту, дуже важко» [31, с. 13-14].

Слід зазначити, що навіть через чверть століття після першої згадки про ІПЗ М. Шоу (Mary Show) вказувала, що ІПЗ «ще не справжня галузь інженерії, але має потенціал стати нею» [23]. Таблиця № 1 із [23, с. 20] показує, що за перші 20 років свого існування ІПЗ пройшла розвиток, порівняний із 200 років традиційних галузей інженерії. Ключовими у переході від «аматорського» програмування до ІПЗ М. Шоу вважає відокремлення Д. Кнутом у 1967 р. алгоритму від програми, уведення Р. В. Флойдом поняття формальної верифікації програм.

Таблиця № 1.

*Характеристика етапів розвитку ІПЗ (за [23])*

| | **1960±5 років: «програмування аби як»** | **1970±5 років: «програмування у малому»** | **1980±5 років: «програмування у великому»** |
|---|---|---|---|
| **Характер задачі** | Невеликі програми | Алгоритми та програми | Інтерфейси, структури систем управління |
| **Подання даних** | Структури і символи | Структури даних і типи | Бази даних тривалого зберігання, символи, а також числа |
| **Способи управління** | Елементарне розуміння про потоки управління | Програми виконуються один раз і завершуються | Набори програм виконуються постійно |
| **Подання специфікацій** | Мнемоніки, точні письмові інструкції | Специфікації простого введення/виведення | Системи зі складними специфікаціями |
| **Простір станів** | Стан погано відрізняється від контролю | Невеликий, простий простір станів | Великий, структурований простір станів |
| **Фокус менеджменту** | Відсутній | Індивідуальні зусилля | Командні зусилля, обслуговування системи протягом всього часу експлуатації |
| **Інструменти, методи** | Асемблери, дампи пам’яті | Мови програмування, компілятори, компонувальники, завантажувачі | Середовища, інтегровані засоби, документи |

У 1976 році Б. Боем (Barry Boehm) визначив ІПЗ як «практичне застосування наукових знань до проектування та конструювання комп’ютерних програм та пов’язаної документації, необхідної для їх розробки, експлуатації та підтримки» [8, с. 1226]. Наукові принципи Б. Боем пропонує застосовувати за 4 напрямами:

- у життєвому циклі програмного забезпечення (рис. 4) це принципи побудови компонентів та деталізованого проектування, практично відсутні для системного проектування та інтеграції, наприклад, алгоритми та теорія автоматів;
- у прикладному програмуванні це деякі принципи для складних програмних систем, практично відсутні для програмного забезпечення, наприклад, дискретні математичні структури;
- в економіці програмного забезпечення це декілька принципів, що застосовуються до економічних систем, таких як алгоритми;
- у професійній підготовці це декілька принципів, що сформульовані для засвоєння техніками-програмістами, таких як структурування коду та базові математичні бібліотеки [8, с. 1239].

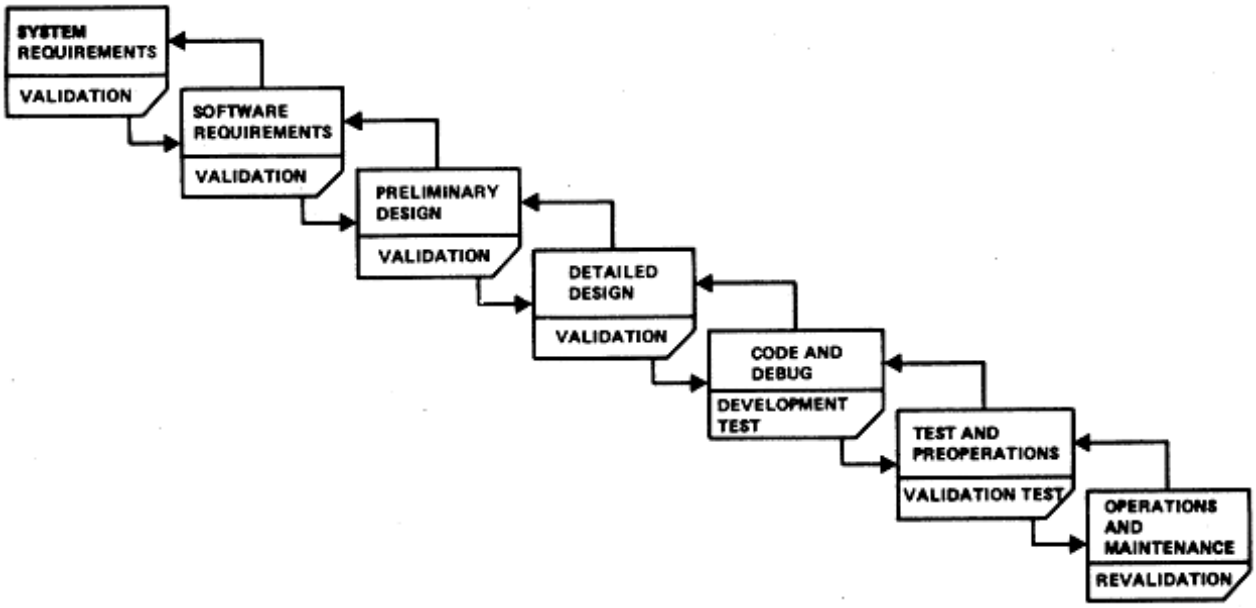


*Рис. 4. Життєвий цикл програмного забезпечення за [8, с. 1227].*

Стаття Б. Боема, опублікована більше 40 років тому, містить всі основні складові сучасної ІПЗ – незважаючи на застарілість прикладів, виокремлені ним наукові принципи довели свою життєздатність. У 1987 році, аналізуючи історичні аспекти ІПЗ [12, с. 9-12], Б. Боем виокремлює три ранні роботи, що мали значний вплив на становлення і розвиток ІПЗ.

Перша з них узагальнює досвід проектування автоматизованої системи управління авіацією та протиповітряною обороною SAGE (Semi-Automatic Ground Environment) [4] – найбільш амбітного проекту інформаційної системи протиповітряної оборони США та Канади 1950-х рр., що об'єднав провідних радарних інженерів, інженерів зв'язку, комп'ютерних інженерів та нових інженерів – з програмного забезпечення. У рамках проекту SAGE була розроблена Lincoln Labs Utility System на допомогу тисячам програмістів, які брали участь у розробці програмного забезпечення SAGE. Вона включала в себе асемблер, бібліотеку та систему керування збірками, низку корисних утиліт, а також засоби тестування та налагодження. Система SAGE успішно задовольняла технічні специфікації приблизно через один рік. Провідний розробник SAGE Г. Д. Бенінгтон (Herbert D. Benington) вказував, що йому було легко виділити той чинник, що призвів до такого успіху: «ми всі були інженерами і були навчені організувати наші зусилля на інженерних засадах». У 1956 році він узагальнив досвід розробки SAGE в описі процесу проектування великої програмної системи (рис. 5), що складається з 9 фаз:

1 – операційний план (operational plan) визначає широкі вимоги до проектування всієї системи керування, що складається з машини, оператора та програмної системи. Цей план повинен бути підготовлений спільно з інженерами комп'ютерних систем та кінцевими користувачами системи;

2 – експлуатаційні специфікації (machine specifications, operational specifications), що точно визначають «передавальну функцію» системи управління. У цьому поданні комп'ютер, його термінальне обладнання та системна програма розглядаються як «чорна скринька»;

3 – програмні специфікації (program specifications) описують реалізацію «чорної скриньки» системною програмою. Ці специфікації організовують програму в підпрограми-компоненти та таблиці, вказують основні канали міжпрограмної взаємодії, а також спільне використання машинного часу та даних кожною підпрограмою;

4 – після завершення операційних та програмних специфікацій готуються детальні специфікації кодування (coding specifications), що визначають «передавальну функцію»

кожного компонента підпрограми у термінах опрацювання глобальних та локальних даних;

5 – кожен компонент програмується (coding) за допомогою специфікацій кодування. В ідеалі цей етап має бути простим механічним перекладом; насправді, програмування розкриває невідповідність цих специфікацій кодування (а іноді й операційних специфікацій);

6 – після програмування кожна підпрограма окремо тестується на відповідність параметрів (parameter testing). На цьому етапі тестування виконується у середовищі, що імітує відповідні частини програмної системи. Кожен тест, виконаний на цій фазі, документується у наборі специфікацій тесту, що деталізує використовуване середовище та отримані результати. На рис. 5 пунктирна лінія вказує на те, що при тестуванні параметрів слід керуватися специфікаціями кодування, а не кодованою програмою;

7 – після завершення тестування параметрів підпрограм поступово компонується та перевіряється вся система (assembly testing), використовуючи спочатку модельні, а потім реальні дані;

8 – після завершення збирання програми вона перевіряється в його операційному середовищі у стресовому режимі (shakedown);

9 – програма готова до роботи та оцінки (evaluation).

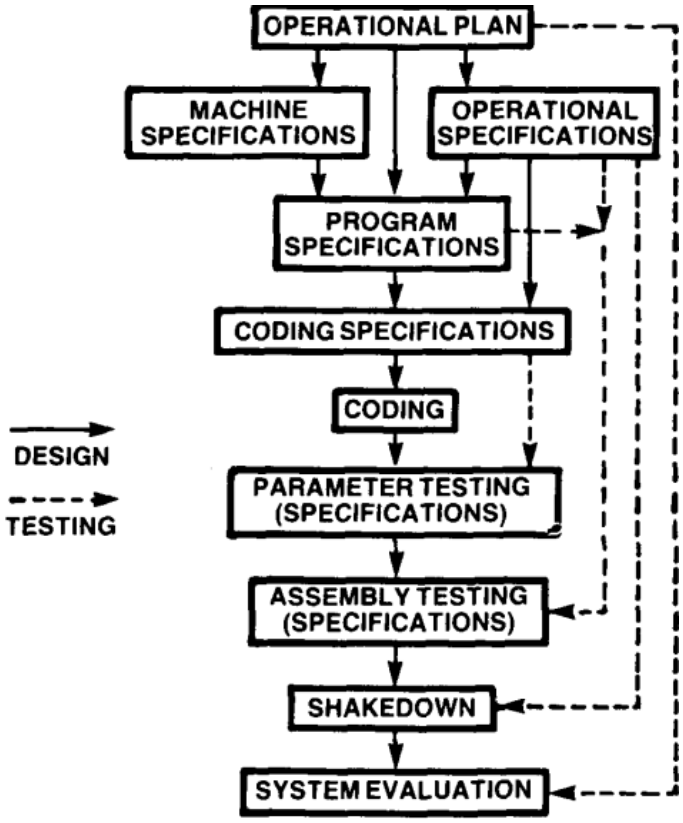


*Рис. 5. Модель проектування системи SAGE.*

Модель Г. Д. Бенінгтона явно не передбачає повернень до попередніх фаз (хоча у статті й згадується про необхідність перегляду навіть першої фази за результатами реалізації специфікацій кодування під час програмування), тому надалі подібні моделі назвали «водоспадними».

Різке зменшення вартості машинного часу наприкінці 1950-х рр., пов’язане насамперед зі зміною апаратної складової (зменшення розміру та енергоспоживання транзисторів, збільшення часу їх неперервної роботи, об’єднання в інтегральні схеми тощо), призвело до появи підходу «спочатку закодуй, а потім вже перевіряй та виправляй», що й спричинив обговорювану вище кризу програмного забезпечення.

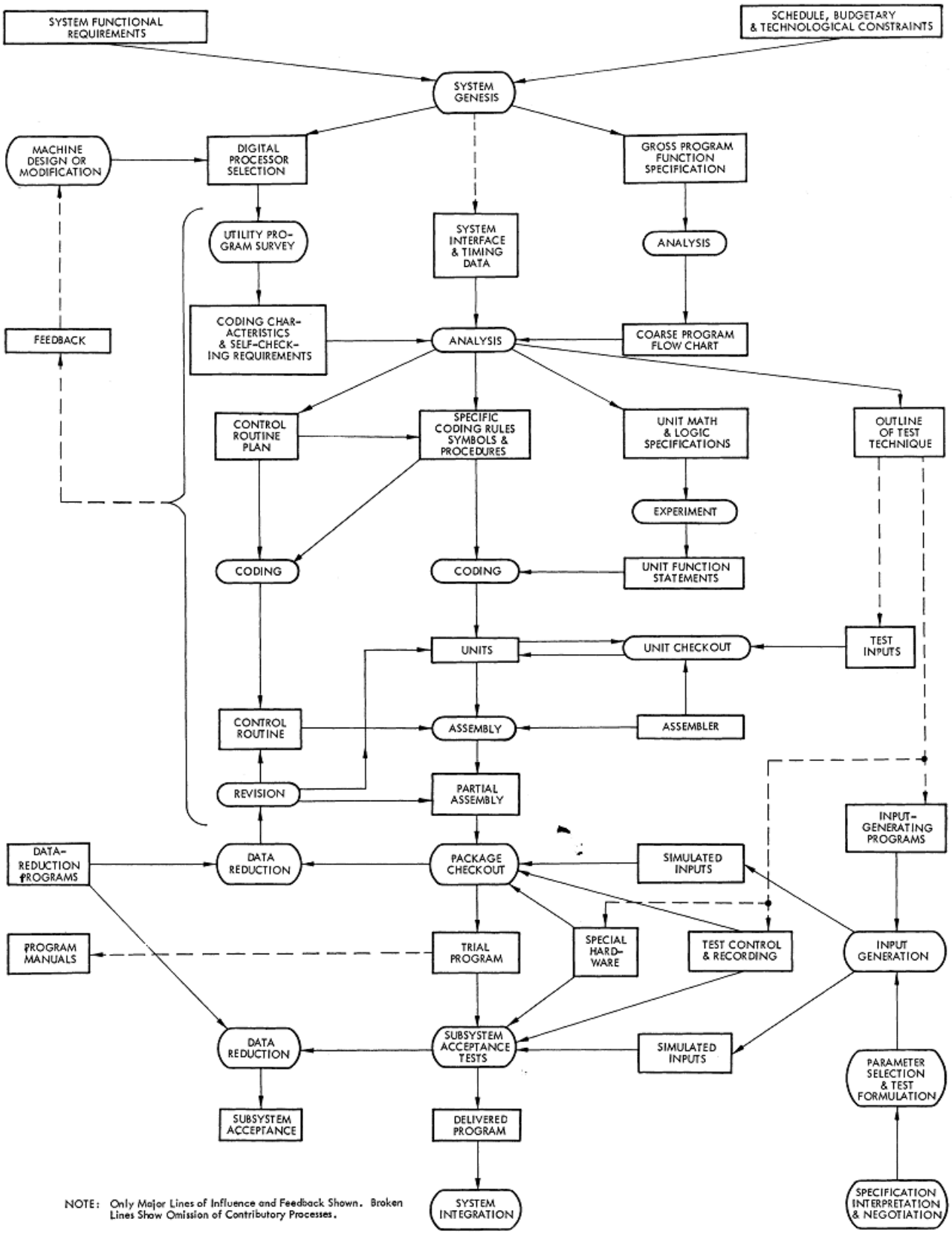


*Рис. 6. Блок-схема типового процесу розробки програми реального часу (за [13, с. 111]).*

Так, уже під час реалізації SAGE програмна складова системи стала більш значущою, ніж апаратна: Б. Боем вказує, що «навіть у SAGE ставали все більш домінувати адресовані психологам питання людино-машинної взаємодії, ніж питання, адресовані радарним інженерам... Швидке розширення попиту на програмне забезпечення перевищило

пропозицію інженерів та математиків. Програма SAGE почала наймати та навчати розробки програмного забезпечення фахівців з гуманітарних, соціальних наук, іноземних мов та мистецтва. Для таких неінженерних фахівців ... й був набагато комфортніший підхід «кодуй та виправляй». Вони часто були дуже креативними, але їх виправлення часто призвело до важкого в обслуговуванні спагетті-коду. ... Суттєвою в цьому відношенні була «хакерська культура» ..., а частими рольовими моделями були «програмісти-ковбої»...» [5, с. 13-14].

Подальшим узагальненням досвіду проектування SAGE та інших оборонних систем, що потребують реагування у реальному часі (зокрема, протиракетною системою Plato), є стаття 1961 року У. Хоз'є (W. A. Hosier). На рис. 6 наведено його варіант моделі проектування програмного забезпечення. На відміну від моделі Г. Д. Бенінгтона, У. Хоз'є явно показав, що майже на всіх етапах проектування у результаті зворотного зв'язку (feedback) є можливими повернення до початку проектування (включно із вибором апаратних засобів).

У. У. Ройс (Winston Walker Royce) за класифікацією Б. Боема належить до фахівців-емпіриків – так, він не пропонує жодних наукових принципів ІПЗ, а описує виключно власний досвід розробки систем високої складності – програмних систем для планування, управління та післяпольотного аналізу місій космічних апаратів.

У. У. Ройс виділяє два етапи проектування, спільні для всіх програмних проектів незалежно від їх розміру та складності: етап аналізу та етап кодування (рис. 7), і одразу ж зауважує, що для великих програмних систем проект, що складається лише з цих двох етапів, приречений на невдачу.

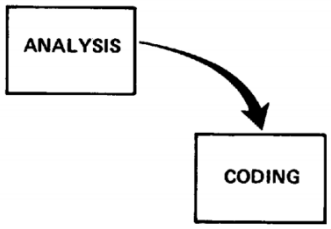


*Рис. 7. Етапи проектування невеликої комп'ютерної програми для внутрішніх потреб (за [21, с. 1]).*

На рис. 8 показано розгорнутий варіант попередньої схеми: «етапи аналізу та кодування залишаються на рисунку, але їм передують два рівня аналізу вимог, розділені етапом розробки програми, а потім етап тестування. Ці доповнення розглядаються окремо від аналізу та кодування, оскільки вони суттєво відрізняються за виконанням. Вони повинні бути заплановані та укомплектовані по-різному для найкращого використання програмних ресурсів» [21, с. 1].

На рис. 9 зображено ітеративний зв'язок між послідовними етапами розробки цієї схеми (подібний до життєвого циклу програмної системи Б. Боема). Упорядкування етапів базується на наступній концепції: після виконання чергового етапу та деталізації проекту можливе повернення до попереднього або перехід на наступний етапи, рідше – на більш віддалені, що може стати причиною проблем, пов'язаних із перевищенням бюджету проекту.

Для зменшення ризиків, пов'язаних із застосуванням ітеративної схеми проектування, У. У. Ройс пропонує внести до неї п'ять доповнень:

1) *етап попереднього проектування програми* між етапом визначення програмних вимог та етапом аналізу (рис. 10). Для реалізації цього етапу У. У. Ройс пропонує:

а) розпочати процес проектування саме із проектувальниками програми, а не аналітиками або програмістами;

б) розробити, визначити та розподілити режими обробки даних навіть за ризику помилок: способи опрацювання, функції, структуру бази даних, розподіляти час виконання, інтерфейси та режими роботи з операційною системою, описати обробку вводу та виводу та визначити попередні операційні процедури;

в) написати зрозумілий, інформативний та актуальний оглядовий документ, з якого

кожен учасник проекту зможе отримати елементарне розуміння проектованої системи;

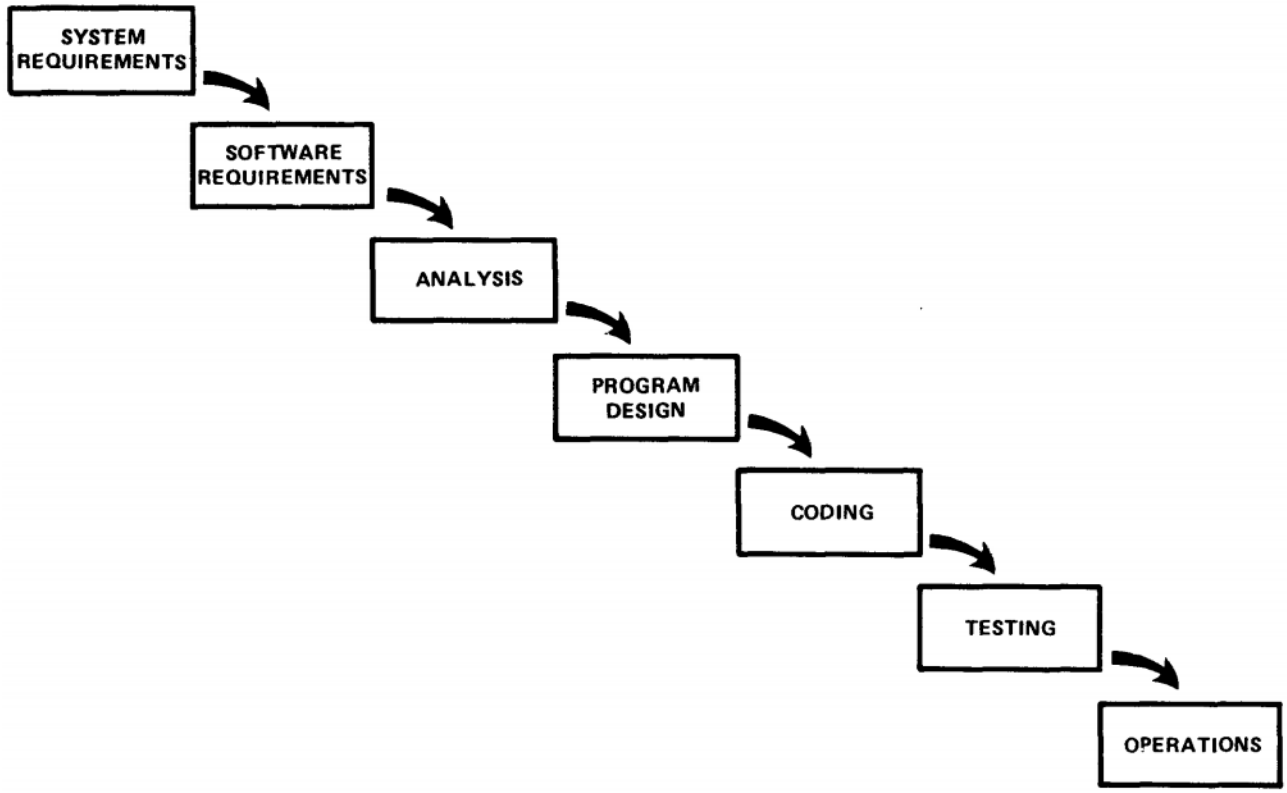


*Рис. 8. Етапи розробки великої комп'ютерної програми, замовленої клієнтом (за [21, с. 2]).*

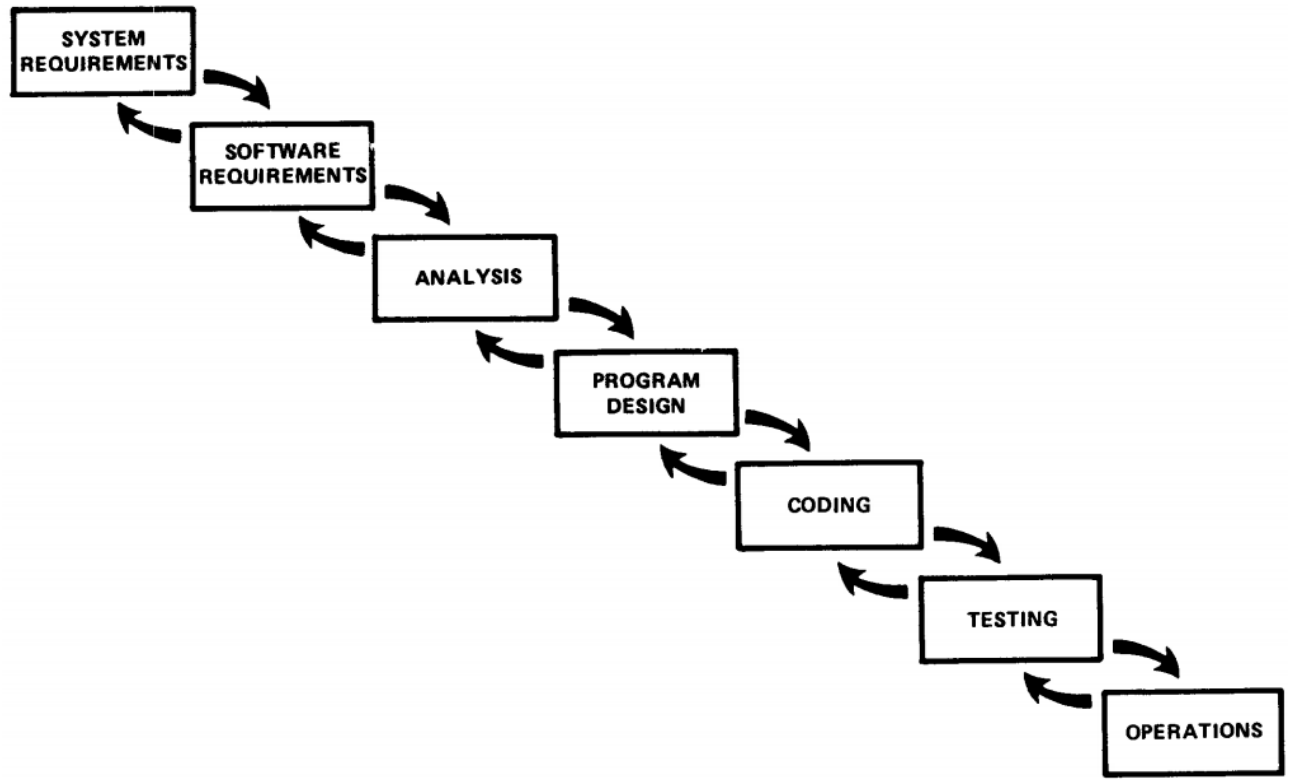


*Рис. 9. Схема ітеративного процесу проектування програмного забезпечення (за [21, с. 3]).*

2) *забезпечення актуальності та повноти документації* (рис. 11) – за У. У. Ройсом, якомога більше: «Першим правилом управління розробкою програмного забезпечення є

безжальне виконання вимог до документації. ... перший крок полягає у вивченні стану документації. Якщо з документацію серйозна проблема, моя перша рекомендація проста: замінити менеджмент проекту; зупинити всі дії, не пов'язані з документацією; привести документацію до відповідних стандартів. Управління програмним проектом просто неможливе без дуже високого ступеня документування» [21, с. 5];

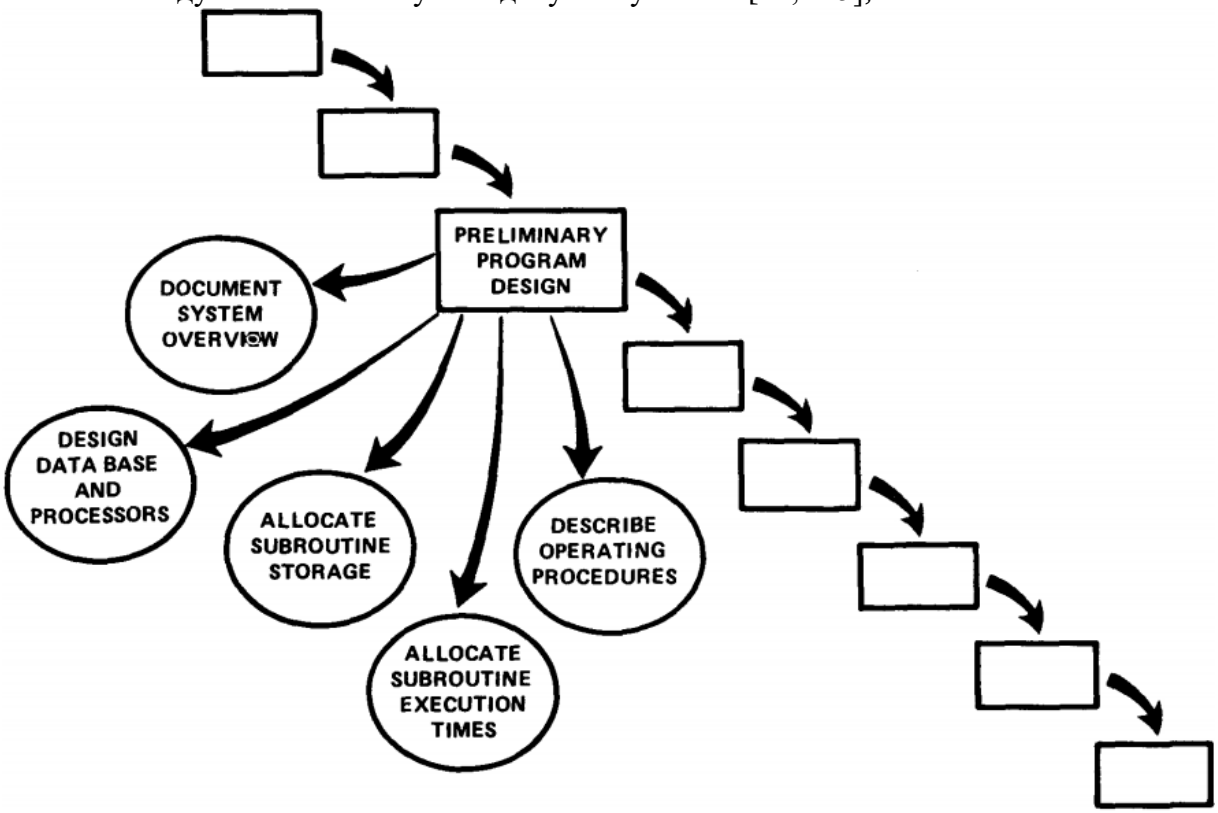


*Рис. 10. Перший додатковий етап: забезпечення завершення попереднього проектування програми до початку етапу аналізу (за [21, с. 4]).*

3) *«подвійне» проектування* (рис. 12) – «після документації другий найважливіший критерій успіху полягає у новизні програмного продукту. Якщо програмне забезпечення розробляється вперше, його остаточна версія, яка поставляється клієнту, повинна бути другою версією. ... На рис. [12] показано, як це можна змоделювати. Зауважимо, що це просто процес, виконаний у мініатюрі, масштабі, відносно невеликому по відношенню до загальних зусиль. ... За допомогою моделювання [керівник проекту] може, принаймні, виконати експериментальну перевірку деяких ключових гіпотез та [занадто оптимістичних людських очікувань]» [21, с. 7];

4) *планування, управління та моніторинг тестування програмного забезпечення* (рис. 13). Як зауважує У. У. Ройс, більш раннє звернення до процедур, пов'язаних із тестуванням, необхідно тому, що етап тестування є одним із останніх етапів проектування, а тому може стати «вузьким місцем» проекту, подолання проблем на якому потребуватиме додаткових ресурсів;

5) *залучення замовника* – «у зв'язку із тим, проектування програмного забезпечення має широкі можливості для інтерпретації навіть після попередніх погоджень, важливо залучати замовника на формальній основі. ... Рис. [14] вказує на три пункти, ... де розуміння, судження та зобов'язання замовника сприятимуть підвищенню ефективності розробки» [21, с. 8].

На рис. 15 показано «водоспадну» модель із деталізованими на рис. 10-14 доповненнями. Складність цієї емпіричної моделі, незважаючи на її спрощеність, вже є досить велика. Ураховуючи, що вона спрямована на подолання складності програмного забезпечення, можна зробити висновок про те, що однією із головних причин кризи проектування програмного забезпечення наприкінці 1960-х рр. була перевага емпіричних

підходів над науковими, про що в ретроспективі й писав Б. Боем, характеризуючи ранні «водоспадні» моделі.

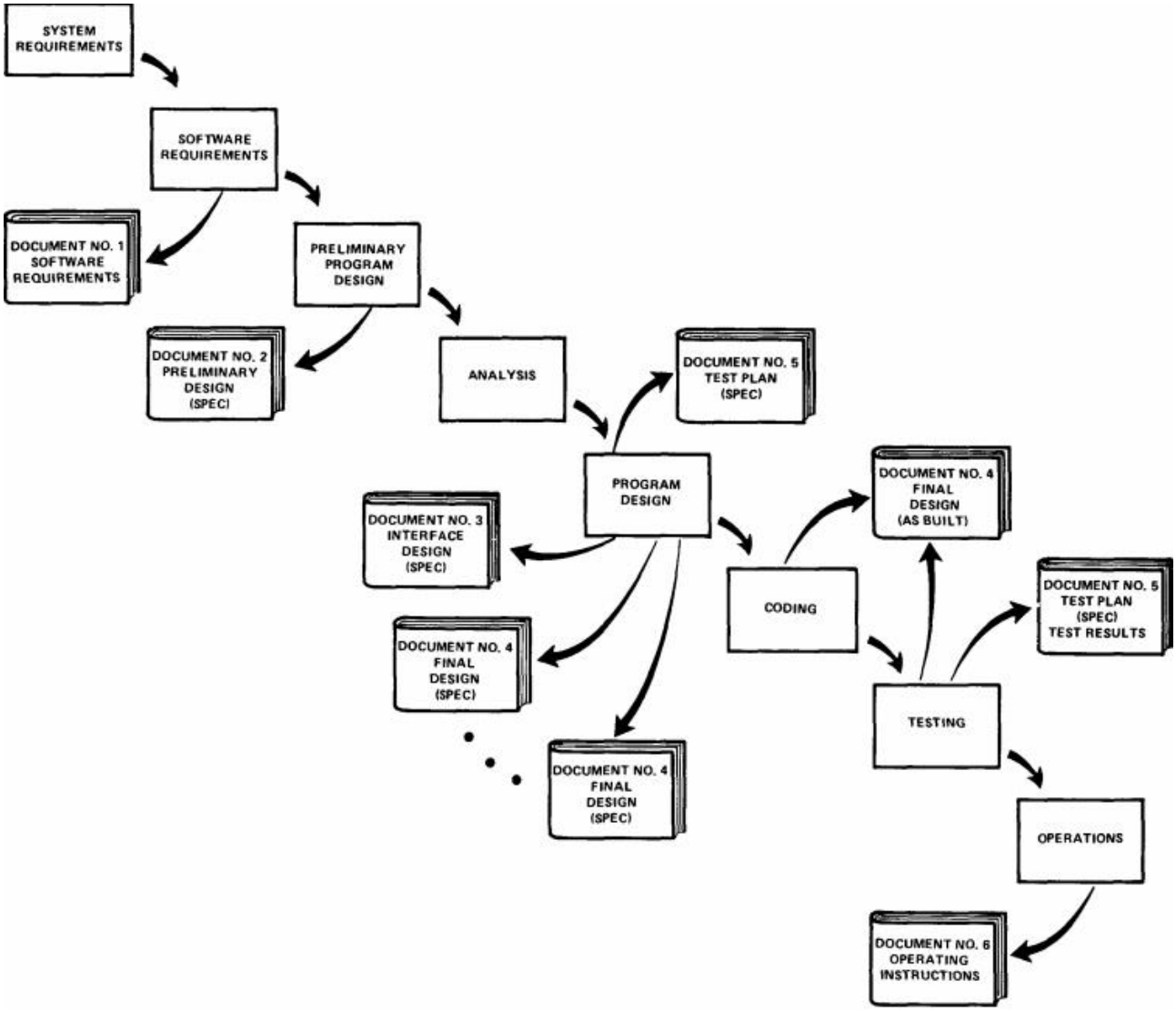


*Рис. 11. Друге доповнення: забезпечення актуальності та повноти документації за 6 основними групами документів (за [21, с. 6]).*

У 1990 році IEEE визначав ІПЗ як [2, с. 67]:

(1) застосування систематичного, дисциплінованого, кількісного підходу до розробки, експлуатації та обслуговування програмного забезпечення; тобто застосування інженерії до програмного забезпечення;

(2) вивчення підходів до (1).

У визначенні IEEE 2010 року емпіричний підхід відійшов на друге місце у порівнянні із науковим:

(1) систематичне застосування наукових та технологічних знань, методів, досвіду проектування, впровадження, тестування та документування програмного забезпечення;

(2) застосування систематичного, дисциплінованого, кількісного підходу до розроблення, експлуатації та обслуговування програмного забезпечення; тобто застосування інженерії до програмного забезпечення [1].

Е. Бьйоргер (Egon Boerger) [12, с. 12-17], розробник одного з таких наукових методів – формального методу машин абстрактних станів (ASM – Abstract State Machines) – вказує, що метод ASM є подальшим узагальненням багаторічної теорії та практики структурного програмування, що розв'язує проблему базової моделі, яка є фундаментальною для ІПЗ. «Базова модель системи *S*, що сама по собі є дуже часто не чітко математично визначеною

системою, – це математична модель, яка формалізує основні інтуїтивні уявлення, концепції та операції *S* без кодування таким чином, що модель може бути розпізнана системним експертом «шляхом інспектування» і у такий спосіб підтверджено, що вона є правильним, адекватним та точним поданням *S*. Гарні базові моделі відіграють вирішальну роль для доказу правильності специфікацій складних систем у цілому» [12, с. 15].

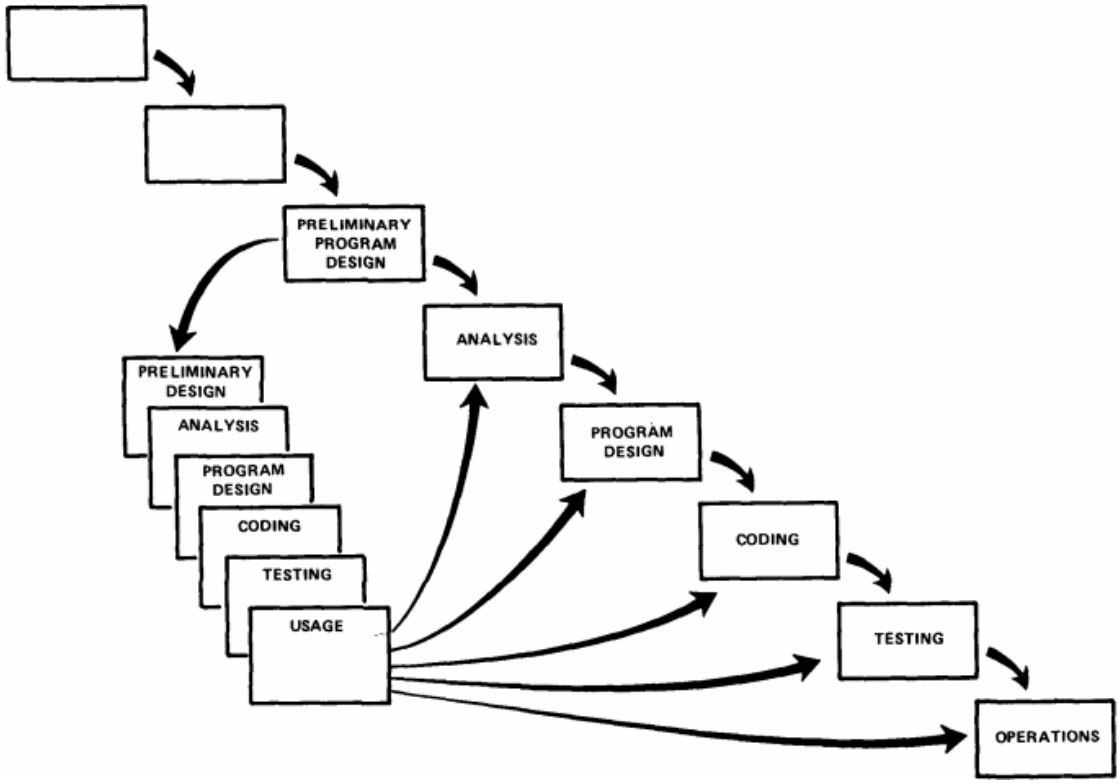


*Рис. 12. Третє доповнення: спроба зробити роботу двічі – перший результат забезпечує робочу модель кінцевого продукту (за [21, с. 7]).*

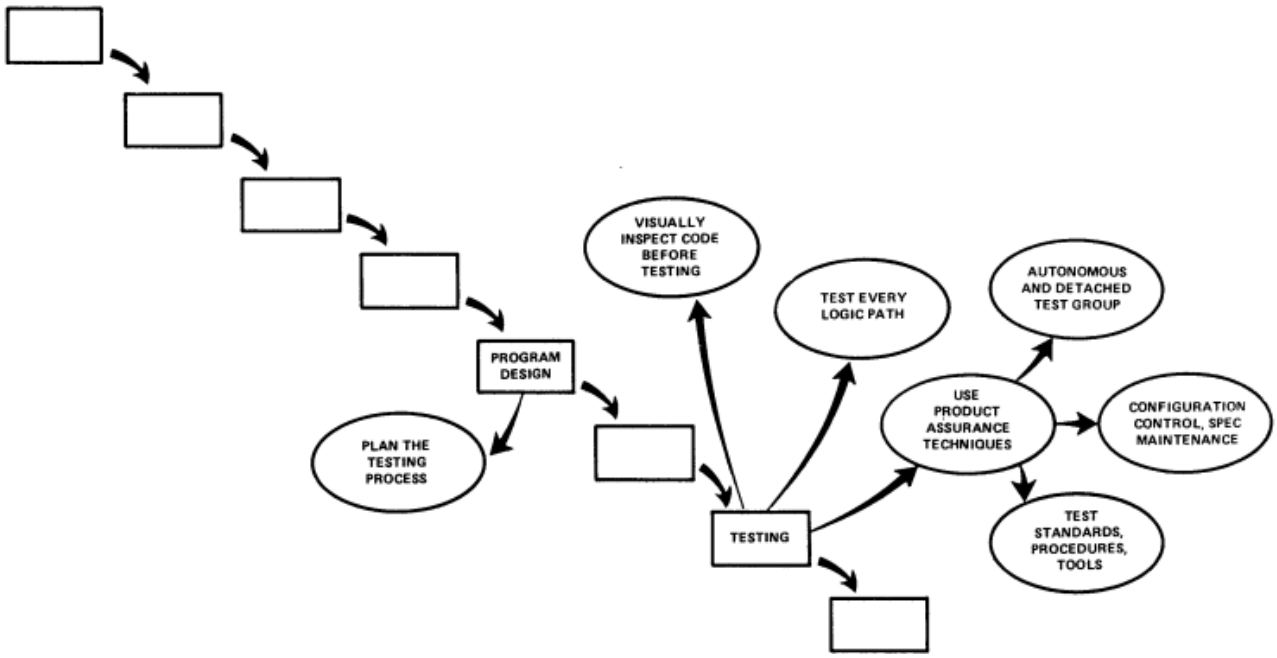


*Рис. 13. Планування, управління та моніторинг тестування програмного забезпечення (за [21, с. 9]).*

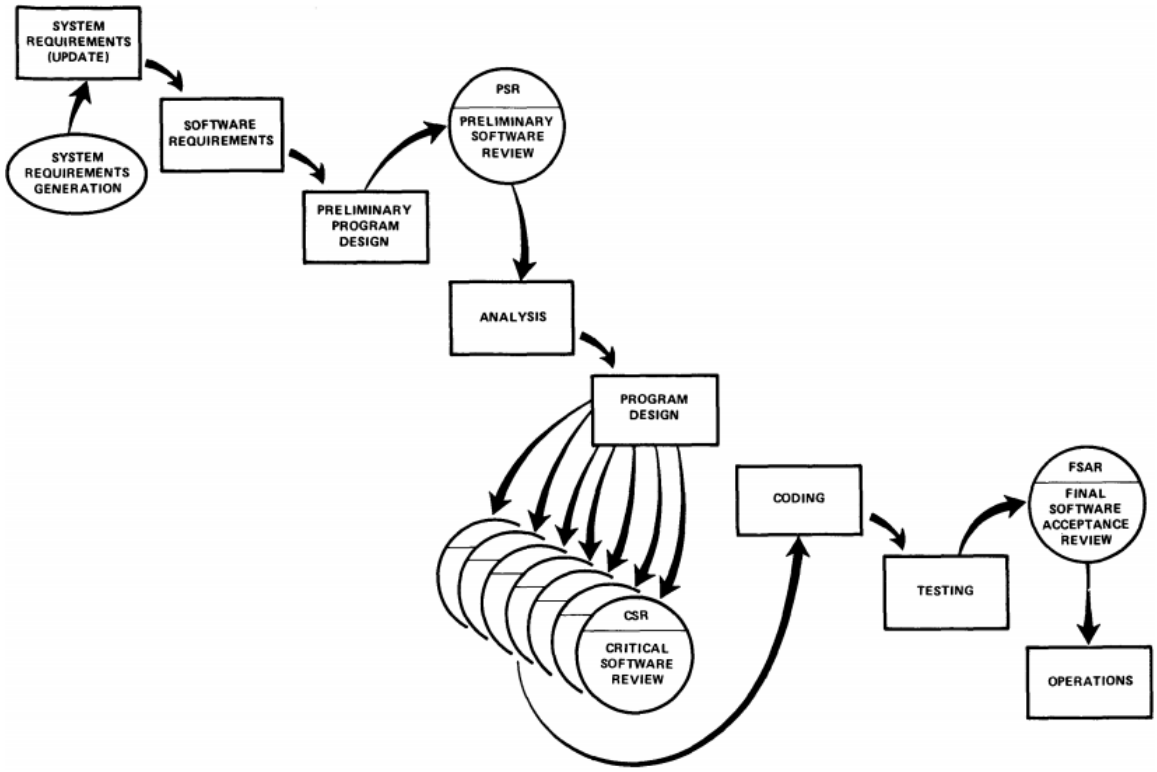


*Рис. 14. П'яте доповнення: залучення замовника – його участь повинна бути формальною, глибокою та продуктивною (за [21, с. 9]).*

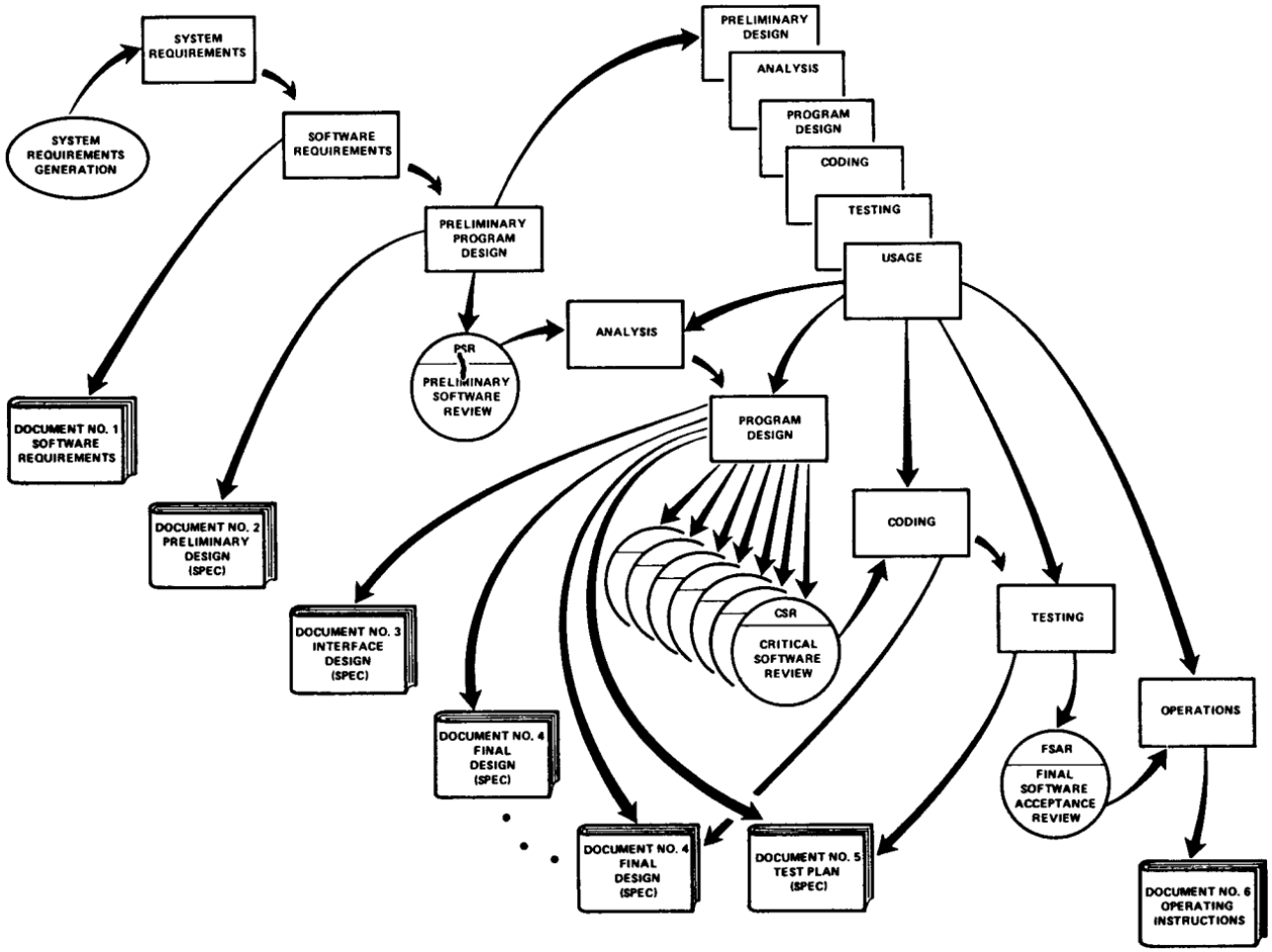


*Рис. 15. Модель розробки програмного забезпечення (за У. У. Ройсом).*

Сьогодні метод ASM є практичним та науково обґрунтованим методом системної інженерії, що дозволяє подолати розрив між двома боками проектування програмного забезпечення:

– людське розуміння і формулювання реальних проблем (охоплення вимог шляхом точного моделювання високого рівня на рівні абстракції, визначеною заданою галуззю застосування);

– розгортання їх алгоритмічних рішень машинами, що виконують код, на змінних платформах (визначення конструктивних рішень, деталей системи та реалізації).

Моделі ASM можуть бути перевірені як математично (через механізми логічного виведення), так й емпірично (через виконання тестів), що робить їх містком між теоретичними комп'ютерними науками та емпіричними методами ІПЗ.

Через 6 років після статті [23] М. Шоу у [24] суттєво спростила таблицю № 1 та доповнила її четвертим етапом розвитку ІПЗ – 1990±5 років: «програмування у світі», що характеризувалось розподіленими системами з відкритими та змінними специфікаціями, акцентом на взаємодії підсистем та співробітництві незалежних процесів. На рис. 16 показано зміни, що відбувались у технологіях програмування на виокремлених М. Шоу етапах розвитку ІПЗ.

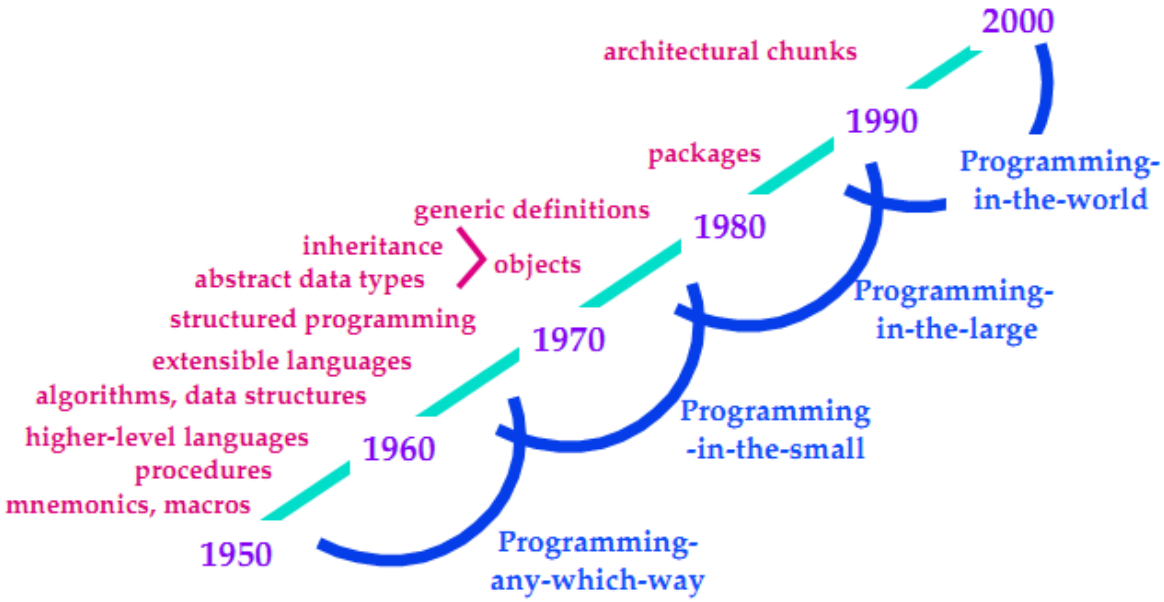


*Рис. 16. Взаємозв'язок конструкцій мов програмування та етапів розвитку ІПЗ (за [24, с. 54]).*

З рис. 16 видно, що прогрес у технологіях програмування на перших трьох етапах розвитку ІПЗ поступово уповільнювався, а четвертий етап характеризується скоріше особливостями програмної архітектури, ніж розвитком технологій програмування. Через 19 років після статті [23] М. Шоу констатувала, що розвиток ІПЗ після третього етапу змінився: об'єктно-орієнтовані технології розпочали витісняти класичні технології структурного програмування, а розвиток модельного опису окремих галузей досяг рівня, що надав можливість створювати за моделями генератори програмного коду. Інші фактори змін – бурхливий розвиток Інтернет-технологій. Це розширило галузь ІПЗ із проектування великих програмних систем до підтримки програмного забезпечення, що широко використовується та створюється спільнотою. У зв'язку з цим нові етапи розвитку, які М. Шоу доповнила перші три (фактично зробивший уведений у [24] четвертий неканонічним), фокусуються на абстракціях більш високого рівня, ніж «програмування у ...»:

– 1990±5 років: «абстрактні архітектури», що керують розробкою програмного забезпечення системи. Архітектура програмного забезпечення є способом структурування програмної системи, абстракції елементів системи на певній фазі її

роботи. Система може складатись із кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру. На цьому етапі вивчались різновиди абстрактних архітектур: компоненти, абстрактні протоколи їх з'єднання та архітектурні стилів, що спрямовують на постійне використання сумісних компонентів і з'єднувачів. Для підтримки цих стилів виникли мови опису абстрактних архітектур, такі як UML.

– 2000±5 років: «демократизація Інтернет» як головного рушія інформації та торгівлі. «Інтернет-додатки разом із недорогими ПК дозволяють звичайним людям стати не тільки споживачами, а й виробниками контенту. Внаслідок неефективності розробки програмного забезпечення зросла кількість людей, які розробляють власне програмне забезпечення або адаптують стороннє чи створюють активний веб-вміст – їх тепер на два порядки більше, ніж професійних програмістів, які є невеликою меншістю» [22, с. 65-66].
– 2010±5 років: «зникнення системних обмежень». М. Шоу, описуючи цей частково майбутній на момент написання статті [22] етап, вказує: «я бачу проблеми, що виникають в результаті глибокої взаємозалежності між дуже складними системами та їхніми користувачами. Проблеми, з якими стикаються програмні інженери, дедалі більше перебувають у складних соціальних контекстах, а межі проблем ускладнюються дедалі складніше. Наприклад, Інтернет – це не просто інфраструктура зв'язку та пошуку, його багатство та життєздатність виникають завдяки інноваційним додаткам, широкому вмісту та безпосередній участі своїх користувачів, включаючи електронну комерцію, соціальні мережі, розваги, довідкову інформацію та – і це дійсно так – спам, віруси та фішинг. Інтернет є прикладом «надшвидкісної» системи, в якій немає можливості централізованого контролю, немає чистих системних обмежень, і немає єдиних вимог для визначення правильності, за винятком вузького сенсу, що інфраструктура веде себе належним чином. Різні незалежні, конкурентоспроможні та творчі користувачі є двигунами його еволюції, і завдання збереження контролю над такою системою виходить за межі технічних програмних обмежень у соціальну, економічну та політичну площину» [22, с. 66].

У 1993 році Ф. Дж. Баклі (Fletcher J. Buckley) писав, що для визначення професії інженера із програмного забезпечення необхідно визначити 4 складові: професійну галузь, технічні вимоги до кваліфікації професіонала, необхідні (entry-level) вимоги до професії та набір правил, що утворюють професійну етику [10]. На думку автора, ІПЗ разом із комп'ютерною інженерією є частинами системної інженерії.

**Тенденції розвитку професійної підготовки фахівців з інженерії програмного забезпечення**

У 2018 році в статті, присвяченій 50-тиріччю підготовки фахівців з ІПЗ, Н. Р. Мід (Nancy R. Mead), Д. Гарлан (David Garlan) та М. Шоу представили сучасне трактування ІПЗ як «складової комп'ютерних наук, яка створює практичні, економічно вигідні рішення для обчислювальних задач та задач опрацювання інформації, переважно шляхом застосування наукових знань та розробки програмних систем, що служать людству» [14, с. 1]. На думку авторів, сучасна ІПЗ базується на трьох групах ключових принципів:

1. *Основні концепції комп'ютерних наук*, пов'язані зі структурами даних, алгоритмами, мовами програмування та їх семантикою, аналізом, обчислювальністю, моделями обчислень тощо:

– абстракція надає можливість контролювати складність;

– структурування задач часто робить їх більш доступними; існує ряд загальних структур;

– символічні репрезентації є необхідними та достатніми для вирішення проблем, пов'язаних з інформацією;

– точні моделі підтримують аналіз та прогнозування;

– загальні структури задач призводять до канонічних розв'язань.

2. *Основи інженерії*, пов'язані з архітектурою, процесами інженерії, компромісами та витратами, стандартизацією, якістю та гарантіями та інші складові, що забезпечують підхід до проектування та вирішення проблем, який враховує прагматичні аспекти:

– джерелом інженерної якості є інженерні рішення;

– якість програмного продукту залежить від вірності інженера інженерному артефакту;

– інженерія вимагає узгодження суперечливих обмежень;

– інженерні навички покращуються внаслідок ретельної системної рефлексії досвіду.

3. *Соціально-економічні основи*, що включають процес створення та еволюції артефактів, а також питання, пов'язані з політикою, ринками, зручністю використання та соціально-економічними впливами; це забезпечує основу для формування інженерних артефактів, що будуть відповідати їхньому призначенню:

– обмеження на витрати та час значущі, а не просто можливі;

– технологія покращується експоненціально, на відміну від людських здібностей;

– успішна розробка програмного забезпечення залежить від командної роботи творчих людей;

– цілі бізнесу та політики так само накладають обмеження на проекування та розробку програмного забезпечення, як і технічні міркування;

– функціональність програмного забезпечення часто настільки глибоко вбудована в інституційні, соціальні та організаційні механізми, що необхідні методи спостереження, які матимуть коріння в антропології, соціології, психології та інших необхідних дисциплінах;

– замовники та користувачі, як правило, точно не знають, чого хочуть, і відповідальність розробника полягає в тому, щоб полегшити виявлення вимог.

Таким чином, протягом останніх 50 років предмет ІПЗ залишався незмінними, проте аспекти, що покривались ІПЗ, суттєво розширювались. Так, Б. Ренделл у статті 2018 року [20] показує, як змінилось трактування ІПЗ – від провокативного терміна та неозначуваної, але іменованої галузі діяльності (за М. Хемілтон) до «інженерної дисципліни, у якій зібрані всі аспекти виробництва програмного забезпечення від ранніх етапів специфікації систем до її обслуговування після уведення в експлуатацію» [20, с. 5], проте «немає універсальних методів та методів розробки програмного забезпечення, які є придатними для всіх систем та всіх компаній – навпаки, за останні 50 років склався різноманітний набір методів та інструментів розробки програмного забезпечення» [20, с. 7].

У 1990 році Ф. Я. Дзержинський і Л. Д. Райков на основі аналізу вітчизняної та зарубіжної літератури, зразків програмних засобів та інших джерел, що втілюють у собі сучасні досягнення ІПЗ, пропонують наступні визначення: «інженерія програмного забезпечення – це концепції, методи, інструменти, системи тощо, які призначені для забезпечення якісного та високопродуктивного характеру інженерної діяльності зі створення та використання програмно-інформаційних засобів обчислювальної техніки» [30, с. 67]. Дослідники, спираючись на практичний досвід, відзначають, що низка помилок під час проектування та реалізації програмного забезпечення допускається фахівцями через занадто вузьке трактування історично сформованих підходів до ІПЗ. Таких «вузьких підходів» дослідники виділяють три: «машинобудівний», «адміністративний», «інструментальний».

«Машинобудівний» підхід полягає в занадто буквальній аналогії з поняттям технології машинобудівного виробництва, що передбачає жорстко регламентовану послідовність технологічних операцій, що мають гарантувати отримання продукції з заданою якістю в мінімальній залежності від індивідуальних (наприклад, творчих) можливостей окремих працівників.

«Адміністративний» підхід акцентує на важливості різноманітних організаційно-управлінських заходів з упровадження тих чи інших стандартів, регламентів робіт, типових або базових програм тощо.

«Інструментальний» підхід є найбільш розповсюдженою оманою, на думку дослідників. Його суть полягає в тому, що пошук рішення проблем підвищення продуктивності та якості результатів програмування зводиться до пошуку або розробки тих

чи інших інструментальних засобів від особистих до систем «наскрізної» автоматизації робіт тощо.

«Визначимо засіб ІПЗ як сукупність деяких результатів в галузі ІПЗ, що мають «концептуальний» (методологічний, організаційний тощо), програмний або інший характер і мають наступні відмінності: по-перше, багатократним безпосереднім застосуванням у деякому класі робочих умов; по-друге, достатньо надійною відтворюваністю, при цьому, певного практичного ефекту (реально або імовірно корисного); по-третє, ідентифікованістю – можливістю достатньо об'єктивно і точно виокремити певний засіб від інших при розгляданні його як об'єкту застосування, впровадження, передачі тощо» [30, с. 72].

У 1976 році А. С. Вільямс до професійних засобів ІПЗ, спрямованих на розв'язання кризи програмного забезпечення (причому не лише загроз перевищення термінів та бюджетів при розробці, а й нанесення шкоди і створення загрози для життя та здоров'я людини у процесі експлуатації), відносила технології програмування (низхідне, структурне, модульне) та засоби проектування (таблиці прийняття рішень, організація команди програмістів за хірургічним принципом [29, с. 27-34], текстові редактори) [28, с. 5]. Пізніше до таких засобів додались технології об'єктно-орієнтованого програмування, CASE-системи, універсальні мови програмування, документування та стандартизації: кожен із них при уведенні позиціонувався як «срібна куля» – універсальний засіб, покликаний розв'язати усі проблеми, породжені кризою програмного забезпечення. Ф. Брукс (Frederick Phillips Brooks, Jr.), автор ідеї організації команди програмістів за хірургічним принципом, у статті [9], вказує, що «немає єдиного відкриття ні в технології, ні в методах управління, одне тільки використання якого обіцяло б протягом найближчого десятиліття на порядок підвищити продуктивність, надійність, простоту розробки програмного забезпечення» [9, с. 10]. Серед інших способів розв'язання тривалої (з 1960-их по 1990-ті рр.) кризи також пропонувались підвищення дисципліни програмістів та їх професіоналізму, застосування формальних методів, процесів та методологій та ін. Ф. Брукс у ролі «срібних куль» розглядає також штучний інтелект, експертні системи, автоматичне програмування, графічне програмування, верифікацію програм, інтегровані середовища розробки.

У виступі на конференції [7] та відповідній статті [6] Б. Боем намагається застосувати діалектичний підхід до еволюції ІПЗ: «Філософ Гегель висунув гіпотезу, що поглиблення людського розуміння йде шляхом теза (чому певні речі робляться так, як робляться) – антитеза (надає краще пояснення у тих важливих випадках, які теза нездатна пояснити) – синтез (виявляється, що антитеза відкидає занадто велику частину оригінальної тези, створюється гібрид, який вбирає в себе найкраще з тези та антитези, уникаючи їх недоліків)» [6, с. 12]. На рис. 17 показано застосування діалектичного підходу до розвитку ІПЗ з доінженерного періоду (1950-ті рр.) по теперішній час.

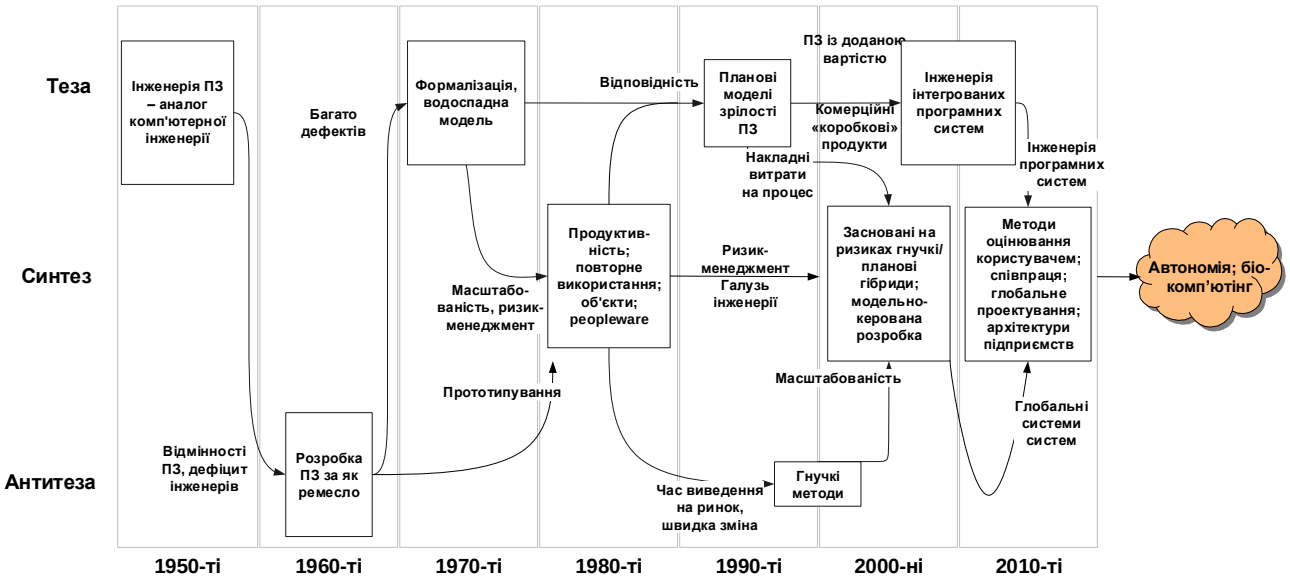

*Рис. 17. Діалектичний підхід до еволюції інженерії програмного забезпечення (за Б. Боемом [7]).*

Для кожного етапу еволюції ІПЗ Б. Боем виокремив принципи, що використовуються донині, та застарілі практики (таблиця № 2).

Таблиця № 2.

*Діалектика етапів розвитку ІПЗ (за Б. Боемом [6])*

| **Період** | **Категорія діалектики** | **«Вічні» принципи** | **Застарілі практики** |
|---|---|---|---|
| 1950-і | *теза*: ІПЗ – аналог комп'ютерної інженерії | *не нехтуйте науками*, адже це перша складова інженерії, яка повинна включати в себе не лише математику та комп'ютерні науки, а також поведінкові науки, економіку та менеджмент разом із використанням наукового методу для вивчення досвіду;<br>*подивіться, перш ніж стрибнути* – передчасні зобов'язання можуть бути катастрофічними | *уникайте жорсткого послідовного процесу* – він, як правило, повільніший, і світ стає надто стійким і непередбачуваним для його використання |
| 1960-і | *антитеза*: розробка програмного забезпечення як ремесло | *думайте нестандартно* – рутинна інженерія ніколи не створить Інтернет або графічний інтерфейс; створюйте цікаві прототипи з низьким ризиком і потенційно високою винагородою;<br>*ураховуйте відмінності у програмному забезпеченні*, розробку якого не можна прискорити – зробіть ці відмінності видимими та змістовними для різних зацікавлених сторін | *уникайте ковбойського програмування* |
| 1970-і | *синтез та антитеза*: формалізація та водоспадні процеси | *усувайте помилки щоякраніше*, а ще краще – запобігайте їх у майбутньому за допомогою глибокого аналізу першопричин;<br>*визначайте цілі системи* – без чіткого спільного бачення, ви, ймовірно, отримаєте хаос та розчарування | *уникайте низхідного проектування та редукціонізму* – комерційні «коробкові» продукти, повторне використання, IKIWISI (I'll know it when I see it – «дізнаюсь, коли побачу»), швидкі зміни та виникаючі вимоги роблять це все нереалістичним для більшості застосувань |

| Період | Категорія діалектики | «Вічні» принципи | Застарілі практики |
|---|---|---|---|
| 1980-і | *синтез*: продуктивність та масштабованість | *існує багато шляхів підвищення продуктивності*, включаючи кадрове забезпечення, навчання, засоби, повторне використання, вдосконалення процесу, прототипування та інші;<br>*те, що добре для продуктів, добре для процесу*, включаючи архітектуру, повторне використання, компонованість та адаптивність | *скептично ставтесь до «срібних куль»* та панацей |
| 1990-і | *антитеза*: паралельні процеси проти послідовних | *час – це гроші*: люди зазвичай інвестують у програмне забезпечення, щоб отримати прибуток, і чим швидше буде встановлено програмне забезпечення, тим швидше надійдуть прибутки – за умови, що воно задовільної якості;<br>*робіть програмне забезпечення, корисне для людей*, адже це друга складова інженерії | *поспішай повільно* – надмірно амбіційні ранні версії, як правило, призводять до неповних та несумісних специфікацій і багатьох переробок |
| 2000-ні | *антитеза та частковий синтез*: гнучкість та цінність | *якщо зміни є швидкими, адаптивність краще за повторюваність*;<br>*розглядайте та задовольняйте інноваційні пропозиції всіх зацікавлених сторін* – якщо критично важливими зацікавленими сторонами знехтувати, вони, як правило, контратакують або відмовляються від участі, що веде до загальних втрат | *уникайте закоханих у ваші гасла* – принцип YAGNI (You Aren't Going to Need It – «Вам це не знадобиться») не завжди є вірним |
| 2010-і | | *реально оцінюйте власні можливості* – деякі системи систем можуть бути занадто великими та складними;<br>*майте стратегію відходу*: керуйте очікуваннями так, що якщо щось піде не так, буде запасний варіант | *не довіряйте усьому, що читаєте* |

Б. Боем вказує, що студенти, які навчатимуться ІПЗ у найближчі 20 років, будуть продуктивними у своїй професії у 2040-ві, 2050-ті та, ймовірно, 2060-ті роки: «Зростання темпів змін продовжує прискорюватися, як і складність програмного насичених системи або систем систем, які потребують проектування. Це створює багато серйозних, але хвилюючих викликів для освіти в галузі програмної інженерії, зокрема: підтримувати курси та програми підготовки у постійно оновлюваному стані; передбачати майбутні тенденції та готувати студентів до них; виконувати моніторинг поточних принципів і практики та відокремлювати «вічні» принципи від застарілих практик; забезпечувати масштабування освітнього досвіду до способів, що застосовуються у великих проектах; брати участь у найсучасніших наукових дослідженнях та практиці з інженерії програмного забезпечення та включати результати у навчальні плани; допомагати студентам навчатися навчатися шляхом аналізу власних досягнень, спрямованих на майбутнє навчальних ігор та вправ, а також участі в наукових

дослідженнях; і навчатися протягом усього життя стільки ж, скільки інженери з програмного забезпечення продовжують займатися своєю професією» [6, с. 25].

**Висновки**

Аналіз передумов виділення ІПЗ як окремої галузі знань та основних етапів розвитку цієї галузі дозволив зробити наступні висновки:

1. На сьогодні ІПЗ є невід'ємною складовою переважної більшості інновацій у всіх сферах розвитку суспільства, науки та техніки, пропонуючи системні, практичні, економічно вигідні рішення для обчислювальних задач та задач опрацювання інформації.

2. За 50 років від першого задокументованого використання терміна «інженерія програмного забезпечення», що втілював у собі загальне побажання того, що розробка програмного забезпечення має бути подібною до інженерії і ґрунтуватись на чітко визначених теоретичних засадах та перевірених методах, накопичено значний досвід проектування, впровадження, тестування та документування програмного забезпечення, виокремлено системні наукові, технологічні підходи і методи до проектування та конструювання комп'ютерних програм. У той же час дослідники зазначають, що ІПЗ ще досі не досягла того рівня сталості, як інші галузі інженерії.

3. Метою ІПЗ є розробка програмних систем із наперед визначеними рівнями якості, надійності та ефективності в умовах обмеження ресурсів (часових, людських, матеріальних, програмно-апаратних тощо). Аналіз історичних етапів розвитку ІПЗ показав, що незважаючи на загальне визнання важливості застосування при розробці програмного забезпечення математичного апарату логіки, теорії автоматів та лінгвістики, воно створювалось емпіричним способом без його використання. Фактором, що змушує програмістів-практиків звернутися до математичних основ ІПЗ, є зростання складності програмного забезпечення і нездатність емпіричних підходів до його розробки та управління впоратися з нею.

4. Сучасна ІПЗ базується на трьох групах ключових принципів: основні концепції комп'ютерних наук, пов'язані зі структурами даних, алгоритмами, мовами програмування та їх семантикою, аналізом, обчислювальністю, моделями обчислень тощо; основи інженерії, пов'язані з архітектурою, процесами інженерії, компромісами та витратами, стандартизацією, якістю і гарантіями та інші складові, що забезпечують підхід до проектування та вирішення проблем; соціально-економічні основи, що включають процес створення та еволюції артефактів, а також питання, пов'язані з політикою, ринками, зручністю використання та соціально-економічними впливами; це забезпечує основу для формування інженерних артефактів, що будуть відповідати їхньому призначенню.

5. Суспільна значущість професії інженера з програмного забезпечення породжує нагальну необхідність розробки, уточнення відповідних освітніх програм їх підготовки. У навчанні ІПЗ («технології програмування») із самого початку гостро постала проблема швидкого застарівання технологічного змісту навчання, розв'язання якої полягає у його фундаменталізації через виокремлення базових основ галузі. Визначено, що опанування основ комп'ютерних наук (інформатики) є фундаментом професійної підготовки з ІПЗ.

***СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ***


1. International Organization for Standardization (ISO) (2010). 24765:2010. ISO/IEC/IEEE International Standard. Systems and software engineering -- Vocabulary. DOI : 10.1109/IEEESTD.2010.5733835.
2. International Organization for Standardization (ISO) (1990). 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. DOI : 10.1109/IEEESTD.1990.101064.
3. Bauer, F. L. (1972). Software Engineering. *Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971,* Vol. 1, Foundations and Systems, 530 - 538. Amsterdam: North-Holland Publishing Company.
4. Benington, H. D. (1956). Production of Large Computer Programs. *Proceedings of Symposium on*

*advanced programming methods for digital computers, Washington, D.C., June 28-29,* pp. 15 - 28. US Navy Mathematical Computing Advisory Panel, US Office of Naval Research.

5. Boehm, B. A. (2006). View of 20th and 21st Century Software Engineering. *Proceedings of the 28th international conference on Software engineering ICSE'06. Shanghai, China (May 20-28, 2006),* 12-29. New York : ACM. DOI : 10.1145/1134285.1134288.
6. Boehm, B. A. (2006). View of 20th and 21st Century Software Engineering : ICSE 2006 Keynote Address. Retrieved from https://isr.uci.edu/icse-06/program/keynotes/Boehm-Keynote.ppt.
7. Boehm, B. W. (1976). Software Engineering. *IEEE Transactions on Computers,* 25, (12), 1226 - 1241.
8. Brooks, F. P. Jr. (1987). No Silver Bullet – Essence and Accident in Software Engineering. *Computer*, 20 (4), 10 - 19. DOI : 10.1109/MC.1987.1663532.
9. Buckley, F. J. (1993). Standards: defining software engineering as a profession. *Computer*, 26 (8), 76 - 78. DOI : 10.1109/2.223554.
10. Dijkstra, E. W. (1971). On a Methodology of Design. *MC-25 Informatica Symposium,* 4.1 - 4.10. Amsterdam : Mathematisch Centrum.
11. Brennecke, A. & Keil-Slawik, R. (Eds.) (1996). History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 - 30, 1996. Retrieved from: https://www.dagstuhl.de/Reports/96/9635.pdf.
12. Hosier, W. A. (1961). Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming. *IRE Transactions on Engineering Management*, 8 (2), 99 - 115. DOI:10.1109/iret-em.1961.5007599
13. Mead, N. R., Garlan, D. & Shaw, M. (2018). Half a Century of Software Engineering Education: the CMU Exemplar. *IEEE Software*. DOI : 10.1109/MS.2018.290110743.
14. Mills, H. D. (1972). Mathematical Foundations for Structured Programming. *Gaithersburg : Federal Systems Division, International Business Machines Corporation*, IV, 62. Retrieved from http://trace.tennessee.edu/utk_harlan/56.
15. Mills, H. D. (1977). Software Engineering. *Science*, 195 (4283), 1199 - 2105.
16. Oettinger, A. A. (1966). Letter to the ACM membership. *Communications of the ACM*, 9 (8), 545 - 546.
17. Perlis, A. J. (1969). Identifying and developing curricula in software engineering. *Proceedings of the May 14-16, 1969, spring joint computer conference on XX - AFIPS '69 (Spring),* 540 - 541. Boston, Massachusetts. New York : ACM. DOI: 10.1145/1476793.1476877.
18. Pucinski, R. C. (1969). The Challenge for the 1970s in Information Retrieval. *Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December 1969*, Vol. 2, XVII-XX. London: Academic Press. DOI: 10.1016/B978-0-12-696202-4.50006-1.
19. 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.
20. Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *Proceedings of IEEE WESCON, Los Angeles, 25-28 August 1970*, 1 - 9.
21. Shaw, M. (2009). Continuing Prospects for an Engineering Discipline of Software. *IEEE Software*, 26 (6), 64 - 67. DOI : 10.1109/ms.2009.172.
22. Shaw, M. (1990). Prospects for an Engineering Discipline of Software. *IEEE Software*, 7 (6), 15 - 24.
23. Shaw, M. (1997). Three Patterns that help explain the development of Software Engineering. *History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26-30, 1996,* III+57. Retrieved from https://www.dagstuhl.de/Reports/96/9635.pdf.
24. 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.

25. 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.
26. Tou, J. T. (Ed.) (1970). Software Engineering – A New Profession. *Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December, 1969*, 1, 1 - 6. London : Academic Press. DOI: 10.1016/B978-0-12-395495-4.50009-X.
27. Williams, A. S. (1976). Software engineering: tools of the profession. *Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science*. Naval Postgraduate School. Monterey.
28. Брукс, Ф. П. (1979). Как проектируются и создаются программные комплексы. Москва: Наука.
29. Дзержинский, Ф. Я. & Райков, Л. Д. (1990). Что такое программная инженерия. *Программирование*, 2, 67-79.
30. Лингер, Р., Миллс, Х. & Уитт, Б. (1982). Теория и практика структурного программирования. Москва: Мир.
31. МОН України (2012). *Про затвердження Переліку пріоритетних напрямів освіти і науки щодо навчання студентів та аспірантів, стажування наукових і науково-педагогічних працівників у провідних вищих навчальних закладах та наукових установах за кордоном у 2012 році*: *Наказ № 315*. Відновлено з http://zakon2.rada.gov.ua/laws/show/z0500-1.
32. Кабінет Міністрів України (2011). *Про затвердження переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок на період до 2020 року, Постанова № 942*. Відновлено з http://zakon5.rada.gov.ua/laws/show/942-2011-%D0%BF.

## *REFERENCES (TRANSLATED AND TRANSLITERATED)*

1. International Organization for Standardization (ISO) (2010). 24765:2010. ISO/IEC/IEEE International Standard. Systems and software engineering -- Vocabulary. DOI : 10.1109/IEEESTD.2010.5733835.
2. International Organization for Standardization (ISO) (1990). 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. DOI : 10.1109/IEEESTD.1990.101064.
3. Bauer, F. L. (1972). Software Engineering. *Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971*, Vol. 1, Foundations and Systems, 530 - 538. Amsterdam: North-Holland Publishing Company.
4. Benington, H. D. (1956). Production of Large Computer Programs. *Proceedings of Symposium on advanced programming methods for digital computers, Washington, D.C., June 28-29*, pp. 15 - 28. US Navy Mathematical Computing Advisory Panel, US Office of Naval Research.
5. Boehm, B. A. (2006). View of 20th and 21st Century Software Engineering. *Proceedings of the 28th international conference on Software engineering ICSE'06. Shanghai, China (May 20-28, 2006)*, 12 - 29. New York : ACM. DOI : 10.1145/1134285.1134288.
6. Boehm, B. A. (2006). View of 20th and 21st Century Software Engineering : ICSE 2006 Keynote Address. Retrieved from https://isr.uci.edu/icse-06/program/keynotes/Boehm-Keynote.ppt.
7. Boehm, B. W. (1976). Software Engineering. *IEEE Transactions on Computers*, 25, (12), 1226 - 1241.
8. Brooks, F. P. Jr. (1987). No Silver Bullet – Essence and Accident in Software Engineering. *Computer*, 20 (4), 10 - 19. DOI : 10.1109/MC.1987.1663532.
9. Buckley, F. J. (1993). Standards: defining software engineering as a profession. *Computer*, 26 (8), 76 - 78. DOI : 10.1109/2.223554.
10. Dijkstra, E. W. (1971). On a Methodology of Design. *MC-25 Informatica Symposium*, 4.1 - 4.10. Amsterdam : Mathematisch Centrum.
11. Brennecke, A. & Keil-Slawik, R. (Eds.) (1996). History of Software Engineering : Position Papers for

Dagstuhl Seminar 9635 on August 26 - 30, 1996. Retrieved from: https://www.dagstuhl.de/Reports/96/9635.pdf.

12. Hosier, W. A. (1961). Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming. *IRE Transactions on Engineering Management*, 8 (2), 99 - 115. DOI:10.1109/iret-em.1961.5007599
13. Mead, N. R., Garlan, D. & Shaw, M. (2018). Half a Century of Software Engineering Education: the CMU Exemplar. *IEEE Software*. DOI : 10.1109/MS.2018.290110743.
14. Mills, H. D. (1972). Mathematical Foundations for Structured Programming. *Gaithersburg : Federal Systems Division, International Business Machines Corporation*, IV, 62. Retrieved from http://trace.tennessee.edu/utk_harlan/56.
15. Mills, H. D. (1977). Software Engineering. *Science*, 195 (4283), 1199-2105.
16. Oettinger, A. A. (1966). Letter to the ACM membership. *Communications of the ACM*, 9 (8), 545 - 546.
17. Perlis, A. J. (1969). Identifying and developing curricula in software engineering. *Proceedings of the May 14-16, 1969, spring joint computer conference on XX - AFIPS '69 (Spring)*, 540-541. Boston, Massachusetts. New York : ACM. DOI: 10.1145/1476793.1476877.
18. Pucinski, R. C. (1969). The Challenge for the 1970s in Information Retrieval. *Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December 1969*, Vol. 2, XVII-XX. London: Academic Press. DOI: 10.1016/B978-0-12-696202-4.50006-1.
19. 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.
20. Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *Proceedings of IEEE WESCON, Los Angeles, 25-28 August 1970*, 1-9.
21. Shaw, M. (2009). Continuing Prospects for an Engineering Discipline of Software. *IEEE Software*, 26 (6), 64 - 67. DOI : 10.1109/ms.2009.172.
22. Shaw, M. (1990). Prospects for an Engineering Discipline of Software. *IEEE Software*, 7 (6), 15 - 24.
23. Shaw, M. (1997). Three Patterns that help explain the development of Software Engineering. *History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26-30, 1996*, III+57. Retrieved from https://www.dagstuhl.de/Reports/96/9635.pdf.
24. 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.
25. 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.
26. Tou, J. T. (Ed.) (1970). Software Engineering – A New Profession. *Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December, 1969*, 1, 1 - 6. London : Academic Press. DOI: 10.1016/B978-0-12-395495-4.50009-X.
27. Williams, A. S. (1976). Software engineering: tools of the profession. *Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science*. Naval Postgraduate School. Monterey.
28. Brooks, F. P. (1979). *How software systems are designed and created*. Moscow: Nauka.
29. Dzerjinskiy, F. Ya. & Raykov, L. D. (1990). What is software engineering. *Programmirovanie*, 2, 67 - 79.
30. Linger, R. C., Mills, H.D. & Witt, B.I. (1982). Structured Programming: Theory and Practice. Moscow: Mir.
31. The Cabinet of Ministers of Ukraine (2012). *On Approval of the List of Priority Areas of Education*

*and Science for Students and Postgraduate Students, internship of scientific and scientific-pedagogical workers in leading higher educational establishments and scientific institutions abroad in 2012, Order № 315*. Retireved from http://zakon2.rada.gov.ua/laws/show/z0500-1.

32. The Cabinet of Ministers of Ukraine (2011). *On approval of the list of priority thematic areas of scientific research and scientific and technical developments for the period until 2020, Postanova № 942*. Retireved from http://zakon5.rada.gov.ua/laws/show/942-2011-%D0%BF.

Стаття надійшла до редакції 11.11.2018.
The article was received 11 November 2018.

**Andrii Striuk**
**Kryvyi Rih National University, Kryvyi Rih, Ukraine**

**FORMATION AND DEVELOPMENT OF SOFTWARE ENGINEERING AS A KNOWLEDGE AREA**

The article presents an analysis of the main stages of the development of software engineering (SE) as a branch of knowledge, highlights the fundamental components of the training of future software engineers, identifies trends in the development of this industry for the next decade.

The modern SE is based on three groups of key principles: the basic concepts of computer science, related to data structures, algorithms, programming languages and their semantics, analysis, computational, computational models, etc.; engineering fundamentals related to architecture, engineering processes, trade-offs and costs, standardization, quality and warranties, and other components that provide an approach to design and problem-solving; socio-economic foundations that include the process of creating and evolving artifacts, as well as issues related to politics, markets, user-friendliness and socio-economic impacts; it provides the basis for the formation of engineering artifacts that will fit their purpose.

Modern SE is an integral part of the overwhelming majority of innovations in all areas of the development of society, science and technology, offering systemic, practical, cost-effective solutions for computing tasks and information processing tasks. During the SE development as a separate industry, considerable experience in designing, implementing, testing and documenting software has been accumulated; system scientific, technological approaches and methods for designing and designing computer programs have been highlighted. At the same time, researchers note that SE has not yet reached the level of sustainability as other areas of engineering. Analysis of the historical stages of the development of the SE showed that despite the universal recognition of the importance of using the mathematical apparatus of logic, automata theory and linguistics in software development, it was created empirically without its use. The factor forcing practitioners to turn to the mathematical foundations of an SE is the increasing complexity of software and the inability of empirical approaches to its development and management to cope with it. The professional training of software engineers highlighted the problem of the rapid obsolescence of the technological content of education, the solution of which lies in its fundamentalization through the identification of the basic foundations of the industry.

**Keywords**: software engineering; professional training; software; software system; programming; design; simulation.

**Стрюк А. Н.**
**Криворожский национальный университет, Кривой Рог, Украина**

**СТАНОВЛЕНИЕ И РАЗВИТИЕ ИНЖЕНЕРИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КАК ОТРАСЛИ ЗНАНИЙ**

В статье представлен анализ основных этапов развития инженерии программного обеспечения (ИПО) как отрасли знаний, выделены фундаментальные составляющие подготовки будущих инженеров-программистов, определены тенденции развития этой отрасли на ближайшее десятилетие.

Современная ИПО базируется на трех группах ключевых принципов: основные

концепции компьютерных наук, связанные со структурами данных, алгоритмами, языками программирования и их семантикой, анализом, вычислимостью, моделями вычислений и тому подобное; основы инженерии, связанные с архитектурой, процессами инженерии, компромиссами и расходами, стандартизацией, качеством и гарантиями и другие составляющие, которые обеспечивают подход к проектированию и решение проблем; социально-экономические основы, которые включают процесс создания и эволюции артефактов, а также вопросы, связанные с политикой, рынками, удобством использования и социально-экономическими воздействиями; это обеспечивает основу для формирования инженерных артефактов, которые будут отвечать их назначению.

Современная ИПО является неотъемлемой составляющей подавляющего большинства инноваций во всех сферах развития общества, науки и техники, предлагая системные, практические, экономически выгодные решения для вычислительных задач и задач обработки информации. За время развития ИПО как отдельной отрасли накоплен значительный опыт проектирования, внедрения, тестирования и документирования программного обеспечения, выделены системные научные, технологические подходы и методы к проектированию и конструированию компьютерных программ. В то же время исследователи отмечают, что ИПО до сих пор не достигла того уровня устойчивости, как другие области инженерии. Анализ исторических этапов развития ИПО показал, что несмотря на всеобщее признание важности применения при разработке программного обеспечения математического аппарата логики, теории автоматов и лингвистики, она создавалась эмпирическим способом без его использования. Фактором, заставляющим программистов-практиков обратиться к математическим основам ИПО, является рост сложности программного обеспечения и неспособность эмпирических подходов к его разработке и управлению справиться с ней. В профессиональной подготовке инженеров-программистов выделено проблему быстрого устаревания технологического содержания обучения, решение которой заключается в его фундаментализации через выделение базовых основ отрасли.

**Ключевые слова:** инженерия программного обеспечения; профессиональная подготовка; программное обеспечение; программная система; программирование; проектирование; моделирования.