Інженерія програмного забезпечення: перші 50 років становлення та розвитку

Андрій Миколайович Стрюк[0000-0001-9240-1976]

Криворізький національний університет,
вул. Віталія Матусевича, 11, м. Кривий Ріг, 50027, Україна
andrey.n.stryuk@gmail.com

Анотація. У статті проаналізовано основні етапи розвитку інженерії програмного забезпечення (ІПЗ). На основі аналізу матеріалів перших конференцій з ІПЗ (1968-1969 рр.) визначено, як криза програмного забезпечення спонукала науковців та практиків об'єднати зусилля для формування інженерного підходу до програмування. Окреслено відмінності професійної підготовки фахівців з ІПЗ. Виокремлено фундаментальні складові підготовки майбутніх інженерів-програмістів. Розглянуто еволюцію підходів до проектування, впровадження, тестування та документування програмного забезпечення. Виокремлено системні наукові, технологічні підходи і методи до проектування та конструювання комп'ютерних програм. 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 створювалась емпіричним способом без його використання. Фактором, що змусив програмістів-практиків звернутися до математичних основ ІПЗ, є зростання складності програмного забезпечення та нездатність емпіричних підходів до його розробки 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. Визначено, що опанування основ комп'ютерних наук (інформатики) є фундаментом професійної підготовки з ІПЗ.

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

Software engineering: first 50 years of formation and development

Andrii M. Striuk[0000-0001-9240-1976]

Kryvyi Rih National University, 11, Vitalii Matusevych St., Kryvyi Rih, 50027, Ukraine
andrey.n.stryuk@gmail.com

Abstract. The article analyzes the main stages of software engineering (SE) development. Based on the analysis of materials from the first SE conferences (1968-1969), it was determined how the software crisis prompted scientists and practitioners to join forces to form an engineering approach to programming. Differences in professional training for SE are identified. The fundamental components of the training of future software engineers are highlighted. The evolution of approaches to the design, implementation, testing and documentation of software is considered. The system scientific, technological approaches and methods for the design and construction of computer programs are highlighted. Analysis of the historical stages of the development of SE showed that despite the universal recognition of the importance of using the mathematical apparatus of logic, automata theory and linguistics when developing software, it was created empirically without its use. The factor that led 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 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. It is determined that mastering the basics of computer science is the foundation of vocational training in SE.

Keywords: software engineering, professional training, software, software system, programming, design, simulation.

1 Вступ

Починаючи з 2012 року, серед пріоритетних напрямів освіти і науки щодо навчання студентів та аспірантів, стажування наукових і науково-педагогічних працівників у провідних закладах вищої освіти та наукових установах за кордоном [31], що відносяться до інформатики та обчислювальної техніки, три – програмна інженерія, програмне забезпечення систем та інженерія програмного забезпечення – відносяться 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 [32]. Ці та низка інших законодавчих ініціатив нашої держави є свідченням нагальної суспільної потреби у компетентних фахівцях з інженерії програмного забезпечення, 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.

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.

2 Виникнення інженерії програмного забезпечення

Перше вживання терміну «програмна інженерія» (software engineering) датується серпнем 1966 року [16], але роком появи інженерії програмного забезпечення (ІПЗ) як галузі вважається 1968 рік, у якому в Німеччині відбулась перша конференція, що мала назву «Software engineering». Головною тематикою конференції 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 [24, с. 13]. Термін «програмна інженерія» (software engineering) був навмисне обраний як провокативний: «це поняття означало, що виробництво програмного забезпечення має базуватися на тому ж типі теоретичних засад та практичних застосувань, що й у традиційних галузях інженерії» [11, с. I; 24, 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, які 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, які 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 думку таких учасників конференції, 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 [24, 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.

3 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 ряд проблемних запитань [24, 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 [24, с. 127]. Інші учасники дискусії підіймали питання практичної підготовки як фахівців, так і викладачів на відповідній спеціальності.

Конференція 1969 року [25], 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 їх 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» [25, 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» [25, с. 62]. На думку А. Дж. Перліса, для ІПЗ фундаментальними є курси дослідження операцій та управління проектами – настільки ж фундаментальними, як й курс теорії автоматів, та більш фундаментальними, ніж будь-який математичний курс [25, с. 62].

Певним продовженням цієї дискусії є інша публікація А. Дж. Перліса [17], у якій він підкреслює суспільну значущість професії інженера з 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. ... 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» [17, с. 541].

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, в 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» [25, 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 даємо їм проект. Перший – графічна математична система, другий – система програмування мовою BASIC. Ми даємо 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» [25, с. 62].

Д. Т. Росс вказав на обмеженість та несистемність такого підходу, а Е. Д. Фалкофф (Adin D. Falkoff) – про 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» [25, с. 62-63].

Е. В. Дейкстра (Edsger Wybe Dijkstra) підіймає проблему швидкого застарівання вузькопрофільних знань: «Я вважаю неправильним навчати матеріал, який, як мені відомо, застаріє через кілька років. ... Ви повинні навчити розуміння методу, почуття якості та стилю» [25, 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, які 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 з 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 («інформатики» за Ф. Л. Бауером та «математичної інженерії» за Е. В. Дейкстрою) є фундаментом професійної підготовки з ІПЗ.

4 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» [13, 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'язків.

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*, які 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, які 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 проектування та розробку програмного забезпечення, як й 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 [19] показує, як змінилось трактування ІПЗ – від провокативного терміну 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» [19, 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» [19, 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» [29, 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 в 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» [29, 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) [27, с. 5]. Пізніше до таких засобів додались технології об’єктно-орієнтованого програмування, CASE-системи, універсальні мови програмування, документування та стандартизації: кожен з 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 [8], 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» [8, с. 10]. Серед інших способів розв'язання тривалої (з 1960-тих по 1990-ті рр.) кризи також пропонувались підвищення дисципліни програмістів та їх професіоналізму, застосування формальних методів, процесів і методологій та ін. Ф. Брукс у якості «срібних куль» розглядає також штучний інтелект, експертні системи, автоматичне програмування, графічне програмування, верифікацію програм, інтегровані середовища розробки.

У виступі на конференції [6] та відповідній статті [5] Б. Боем намагається застосувати діалектичній 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)» [5, с. 12]. На рис. 1 показано застосування діалектичного підходу до розвитку ІПЗ з доінженерного періоду (1950-ті рр.) по теперішній час.

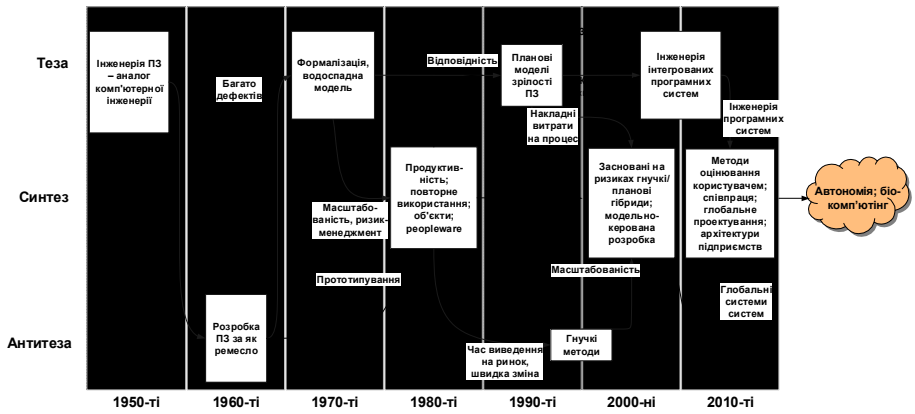

Рис. 1. Діалектичний підхід до еволюції інженерії програмного забезпечення (за Б. Боемом [6])

Для кожного етапу еволюції ІПЗ Б. Боем виокремив принципи, що використовуються донині, та застарілі практики (таблиця 1).

Таблиця 1. Діалектика етапів розвитку ІПЗ (за Б. Боемом [5])

Період	Категорія діалектики	«Вічні» принципи	Застарілі практики
1950-ті	*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;	*уникайте жорсткого послідовного процесу* – він, як правило, повільніший, і світ стає надто стійким і непередбачуваним для його використання

Період	Категорія діалектики	«Вічні» принципи	Застарілі практики
		подивіться, перш ніж стрибнути – передчасні зобов'язання можуть бути катастрофічними	
1960-ті	*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; *ураховуйте відмінності у програмному забезпеченні*, розробку якого не можна прискорити – зробіть ці відмінності видимими та змістовними для різних зацікавлених сторін	*уникайте ковбойського програмування*
1970-ті	*синтез та антитеза*: формалізація та водоспадні процеси	*усувайте помилки щоякраніше*, а ще краще – запобігайте їх у майбутньому за допомогою глибокого аналізу першопричин; *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 нереалістичнішим для більшості застосувань
1980-ті	*синтез*: продуктивність та масштабованість	*існує багато шляхів підвищення продуктивності*, включаючи кадрове забезпечення, навчання, засоби, повторне використання, вдосконалення процесу, прототипування та інші; *те, що добре для продуктів, добре для процесу*, включаючи архітектуру, повторне використання, компонованість та адаптивність	*скептично ставтесь до «срібних куль»* та панацей
1990-ті	*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; *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*; *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	*уникайте закоханих у ваші гасла* – принцип YAGNI (You Aren't Going to Need It – «Вам це не знадобиться») не завжди є вірним

Період	Категорія діалектики	«Вічні» принципи	Застарілі практики
		втрат	
2010-ті		*реально оцінюйте власні можливості* – деякі системи систем можуть бути занадто великими та складними; *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 програмно 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» [5, с. 25].

5 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») [26]. 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 й 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 сторіччя 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» [26, с. 2]. Для цього автор пропонує скористатися принципами ІПЗ. На думку Ю. Т. Ту, ІПЗ повинна охоплювати архітектуру комп'ютерів, системне та прикладне програмування [26, 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» [26, с. 5]. 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) [18]: «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» [26, с. 6].

Ф. Л. Бауер, характеризуючи проектування та виробництво програмного забезпечення як промислову галузь інженерії, формулює способи боротьби із 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 усіх 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, які 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 розробкою відповідних технологій та засобів у співпраці університетських та промислових фахівців.

Х. Д. Міллс (Harlan D. Mills) розширює ІПЗ до математичної основи, необхідної для управління комп'ютерами у складних застосуваннях [15]. Автор 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 та нездатність емпіричних підходів до його розробки та управління впоратися з нею [15, с. 1199]. Тому мета, яку поставили перед собою Х. Д. Міллс разом з іншими авторами [30], – показати програмістам, як, використовуючи систематичні методи аналізу і синтезу програм, зробити свою роботу більш продуктивною, ознайомити їх з прийомами проектування надійного і ефективного програмного забезпечення. 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 програм, як тих, що працюють, так й 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, які враховують умови реалізації.

Х. Д. Міллс вказує, що роботи Е. В. Дейкстри і Ч. Е. Р. Хоора (Charles Antony Richard Hoare) зіграли значну роль для переосмислення та переоцінки програмного забезпечення і розгляді 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 з 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» [10, 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) [15]. 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 й її складовими (агентами) – тоді термін «програмне забезпечення» поширюється також на посібники користувача та керівництва оператора (для людей, які відповідають 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» [15, с. 1201]. Пов'язуючи розвиток програмного забезпечення з опрацюванням даних, Х. Д. Міллс 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, який суттєво залежить від якості програмного забезпечення.

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 [15] 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» [14, 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» [29, 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» [22]. Таблиця 2 з [22, с. 20] показує, що за перші 20 років свого існування ІПЗ пройшла розвиток, порівняний з 200 роками 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.

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» [7, с. 1226]. Наукові принципи Б. Боем пропонує застосовувати за 4 напрямами:

- у життєвому циклі програмного забезпечення – 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;
- у економіці програмного забезпечення – це декілька принципів, які застосовуються до економічних систем, таких як алгоритми;
- у професійній підготовці це декілька принципів, які сформульовані для засвоєння техніками-програмістами, таких як структурування коду та базові математичні бібліотеки [7, с. 1239].

Таблиця 2. Характеристика етапів розвитку ІПЗ (за [22])

	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

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 [11, 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., який 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, що брали участь у розробці програмного забезпечення SAGE. Вона включала в себе

асемблер, бібліотеку та систему керування збірками, ряд корисних утиліт, а також засоби тестування та налагодження. Система SAGE успішно задовольняла технічним специфікаціям 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 у описі процесу проектування великої програмної системи, 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), які 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), які 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 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, яке імітує відповідні частини програмної системи. Кожен тест, виконаний на цій фазі, документується у наборі специфікацій тесту, який деталізує використовуване середовище та отримані результати;

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).

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 у інтегральні схеми тощо), призвело до появи підходу «спочатку закодуй, а потім вже перевіряй та виправляй», який і призвів до обговорюваної вище кризи програмного забезпечення.

Так, вже у процесі реалізації SAGE програмна складова системи стала більш значущою, ніж апаратна: Б. Боем вказує, що «навіть у SAGE все більш домінуючими ставали 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 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 відноситься 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.

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. Для реалізації цього етапу У. У. Ройс пропонує:
   a. розпочати процес проектування саме із проектувальниками програми, а не аналітиками або програмістами;
   b. 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;
   c. написати зрозумілий, інформативний та актуальний оглядовий документ, з якого кожен учасник проекту зможе отримати елементарне розуміння проектованої системи;

2. *забезпечення актуальності та повноти документації* – 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» [20, с. 5];
3. *«подвійне» проектування* – «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. ... 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]» [20, с. 7];
4. *планування, управління та моніторинг тестування програмного забезпечення*. 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» [20, с. 8].

На рис. 2 показано деталізовану «водоспадну» модель. 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.

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 розробки, експлуатації та обслуговування програмного забезпечення; тобто застосування інженерії до програмного забезпечення [1].

Е. Бьйоргер (Egon Boerger) [11, 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» [11, с. 15].

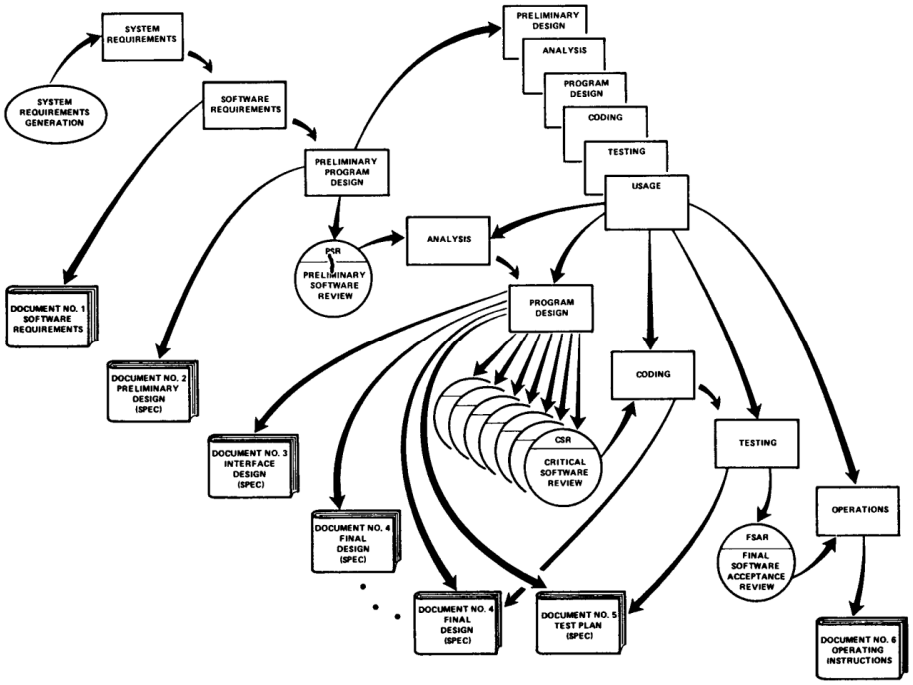

Рис. 2. Модель розробки програмного забезпечення (за У. У. Ройсом)

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 (захоплення 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 [22] М. Шоу у [23] суттєво спростила таблицю 1 та доповнила її четвертим етапом розвитку ІПЗ – 1990±5 років: «програмування у світі», яке характеризувалось розподіленими системами з відкритими та змінними специфікаціями, акцентом на взаємодії підсистем та співробітництві незалежних процесів. 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 з проектування великих програмних систем до підтримки програмного забезпечення, що широко використовується та створюється спільнотою.

У 1993 році Ф. Дж. Баклі (Fletcher J. Buckley) писав, що для визначення професії інженера з 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 [9]. На думку автора, ІПЗ разом з комп'ютерною інженерією є частинами системної інженерії.

6 Висновки

Аналіз передумов виділення ІПЗ як окремої галузі знань та основних етапів її розвитку надав можливість 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 терміну «інженерія програмного забезпечення», який 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 з 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 створювалась емпіричним способом без його використання. Фактором, що змушує програмістів-практиків звернутися до математичних основ ІПЗ, є зростання складності програмного забезпечення та 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 та гарантіями та інші складові, що забезпечують підхід до проектування та вирішення проблем; соціально-економічні основи, які 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, які 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. 24765-2010 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Vocabulary. – New York : IEEE, 2010. – VI, 410, [2] p. – DOI : 10.1109/IEEESTD.2010.5733835.
2. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. – New York : IEEE, 1990. – 83 p. – DOI : 10.1109/IEEESTD.1990.101064.
3. Bauer F. L. Software Engineering / Bauer F. L. // Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971 / Editors : C. V. Freiman, J. E. Griffith, J. L. Rosenfeld. – Amsterdam : North-Holland Publishing Company, 1972. – Vol. 1 – Foundations and Systems. – P. 530-538.
4. Benington H. D. Production of Large Computer Programs / H. D. Benington // Proceedings of Symposium on advanced programming methods for digital computers : Washington,

D.C., June 28, 29, 1956 / US Navy Mathematical Computing Advisory Panel, US Office of Naval Research. – [Washington] : Office of Naval Research, Dept. of the Navy, [1956?]. – P. 15-28. – (ONR symposium report, ACR-15)

5. Boehm B. A View of 20th and 21st Century Software Engineering / Barry Boehm // Proceedings of the 28th international conference on Software engineering ICSE'06. Shanghai, China – May 20-28, 2006. – New York : ACM, 2006. – P. 12-29. – DOI : 10.1145/1134285.1134288.
6. Boehm B. A View of 20th and 21st Century Software Engineering : ICSE 2006 Keynote Address [Electronic resource] / Barry Boehm. – May 25, 2006. – 54 p. – Access mode : https://isr.uci.edu/icse-06/program/keynotes/Boehm-Keynote.ppt.
7. Boehm B. W. Software Engineering / Boehm B. W. // IEEE Transactions on Computers. – 1976. – Vol. 25. – Iss. 12. – P. 1226-1241.
8. Brooks F. P. Jr. No Silver Bullet – Essence and Accident in Software Engineering / Frederick. P. Brooks, Jr. // Computer. – 1987. – Vol. 20. – Iss. 4. – P. 10-19. – DOI : 10.1109/MC.1987.1663532.
9. Buckley F. J. Standards: defining software engineering as a profession / Fletcher J. Buckley // Computer. – 1993. – Vol. 26. – Issue 8. – P. 76-78. – DOI : 10.1109/2.223554.
10. Dijkstra E. W. On a Methodology of Design / Dijkstra E. W. // MC-25 Informatica Symposium / Ed. Jacobus Willem Bakker. – Amsterdam : Mathematisch Centrum, 1971. – P. 4.1-4.10. – (Mathematical Centre tracts, 37)
11. History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 – 30, 1996 organized by William Aspray, Reinhard Keil-Slawik and David L. Parnas [Electronic resource] / Editors : Andreas Brennecke, Reinhard Keil-Slawik. – Paderborn, July 1997. – III+57 p. – Access mode : https://www.dagstuhl.de/Reports/96/9635.pdf.
12. Hosier W. A. Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming / W. A. Hosier // IRE Transactions on Engineering Management. – 1961. – Vol. Em-8. – Iss. 2. – P. 99-115. – DOI:10.1109/iret-em.1961.5007599.
13. Mead N. R. Half a Century of Software Engineering Education: the CMU Exemplar / Nancy R. Mead, David Garlan, Mary Shaw // IEEE Software. – 2018. – DOI : 10.1109/MS.2018.290110743.
14. Mills H. D. Mathematical Foundations for Structured Programming [Electronic resource] / Harlan D. Mills. – Gaithersburg : Federal Systems Division, International Business Machines Corporation, 1972. – IV, 62 p. – (The Harlan D. Mills Collection). – Access mode : http://trace.tennessee.edu/utk_harlan/56.
15. Mills H. D. Software Engineering / Harlan D. Mills // Science. – Vol. 195. – Iss. 4283. – 18 March 1977. – P. 1199-2105.
16. Oettinger A. A. Letter to the ACM membership / Anthony A. Oettinger // Communications of the ACM. – 1966. – Vol. 9. – No. 8. – P. 545-546.
17. Perlis A. J. Identifying and developing curricula in software engineering / Alan J. Perlis // Proceedings of the May 14-16, 1969, spring joint computer conference on XX - AFIPS '69 (Spring). Boston, Massachusetts. – New York : ACM, 1969. – P. 540-541. – DOI: 10.1145/1476793.1476877.
18. Pucinski R. C. The Challenge for the 1970s in Information Retrieval / Roman C. Pucinski // Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December, 1969 / Edited by Julius T. Tou. – Volume 2. – New York ; London : Academic Press, 1971. – P. XVII-XX. – DOI : 10.1016/B978-0-12-696202-4.50006-1.

19. Randell B. Fifty Years of Software Engineering - or - The View from Garmisch [Electronic resource] / Brian Randell // arXiv:1805.02742 [cs.SE]. – 2018. – 9 p. – Access mode : https://arxiv.org/abs/1805.02742.
20. Royce W. W. Managing the Development of Large Software Systems: Concepts and Techniques / Dr. Winston W. Royce // Proceedings of IEEE WESCON, Los Angeles, 25-28 August 1970. – P. 1-9.
21. Shaw M. Continuing Prospects for an Engineering Discipline of Software / Mary Shaw // IEEE Software. – 2009. – Vol. 26. – Issue 6. – P. 64-67. – DOI : 10.1109/ms.2009.172.
22. Shaw M. Prospects for an Engineering Discipline of Software / Mary Shaw // IEEE Software. – 1990. – Vol. 7. – No. 6. – P. 15-24.
23. Shaw M. Three Patterns that help explain the development of Software Engineering / Mary Shaw // History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 – 30, 1996 organized by William Aspray, Reinhard Keil-Slawik and David L. Parnas [Electronic resource] / Editors : Andreas Brennecke, Reinhard Keil-Slawik. – Paderborn, July 1997. – P. 52-56. – Access mode : https://www.dagstuhl.de/Reports/96/9635.pdf.
24. Software Engineering : Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968 / Editors : Peter Naur and Brian Randell. – Brussels : Scientific Affairs Division, NATO, January 1969. – 231 p.
25. Software Engineering Techniques : Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969 / Editors : J. N. Buxton and B. Randell. – Brussels : Scientific Affairs Division, NATO, April 1970. – 164 p.
26. Tou J. T. Software Engineering – A New Profession / Julius T. Tou // Software Engineering COINS III : Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December, 1969 / Edited by Julius T. Tou. – Volume 1. – New York ; London : Academic Press, 1970. – P. 1-6. – DOI : 10.1016/B978-0-12-395495-4.50009-X.
27. Williams A. S. Software engineering: tools of the profession : Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science / Arrena Sue Williams ; Naval Postgraduate School. – Monterey, 1976. – 92 p.
28. Брукс Ф. П. мл. Как проектируются и создаются программные комплексы / Ф. П. Брукс мл. – М. : Наука, 1979. – 152 с. – (Библиотечка программиста)
29. Дзержинский Ф. Я. Что такое программная инженерия / Дзержинский Ф. Я., Райков Л. Д. // Программирование. – 1990. – № 2. – С. 67-79.
30. Лингер Р. Теория и практика структурного программирования / Р. Лингер, Х. Миллс, Б. Уитт. – М. : Мир, 1982. – 406 с.
31. 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 [Електронний ресурс] : Наказ № 315, Перелік / МОНмолодьспорт України. – 20.03.2012. – Режим доступу : http://zakon2.rada.gov.ua/laws/show/z0500-1.
32. Про затвердження переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок на період до 2020 року [Електронний ресурс] : Постанова № 942, Перелік / Кабінет Міністрів України. – 07.09.2011. – Режим доступу : http://zakon5.rada.gov.ua/laws/show/942-2011-%D0%BF.

References (translated and transliterated)

1. 24765-2010 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Vocabulary. IEEE, New York (2010). doi:10.1109/IEEESTD.2010.5733835
2. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. IEEE, New York (1990). doi:10.1109/IEEESTD.1990.101064
3. Bauer, F.L.: Software Engineering. In: Freiman, C.V., Griffith, J.E., Rosenfeld, J.L. (eds.) Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971, vol. 1 – Foundations and Systems, pp. 530–538. North-Holland Publishing Company, Amsterdam (1972)
4. Benington, H.D.: Production of Large Computer Programs. In: Proceedings of Symposium on advanced programming methods for digital computers, Washington, D.C., June 28, 29, 1956, pp. 15–28. Office of Naval Research, Dept. of the Navy, Washington (1956)
5. Boehm, B.: A View of 20th and 21st Century Software Engineering. In: Proceedings of the 28th international conference on Software engineering ICSE'06. Shanghai, China – May 20-28, 2006, pp. 12–29. ACM, New York (2006). doi:10.1145/1134285.1134288
6. Boehm, B.: A View of 20th and 21st Century Software Engineering: ICSE 2006 Keynote Address. https://isr.uci.edu/icse-06/program/keynotes/Boehm-Keynote.ppt (2006). Accessed 11 Jun 2018
7. Boehm, B.W.: Software Engineering. IEEE Transactions on Computers. **25**(12), 1226–1241 (1976)
8. Brooks, F.P.Jr.: No Silver Bullet – Essence and Accident in Software Engineering. Computer. **20**(4), 10–19 (1987). doi:10.1109/MC.1987.1663532
9. Buckley, F.J.: Standards: defining software engineering as a profession. Computer. **26**(8), 76–78 (1993). doi:10.1109/2.223554
10. Dijkstra, E.W.: On a Methodology of Design. In: Bakker, J.W. (ed.) MC-25 Informatica Symposium, pp. 4.1–4.10. Mathematisch Centrum, Amsterdam (1971)
11. Brennecke, A., Keil-Slawik, R. (eds.) History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 – 30, 1996 organized by William Aspray, Reinhard Keil-Slawik and David L. Parnas. https://www.dagstuhl.de/Reports/96/9635.pdf (1997). Accessed 6 May 2018
12. Hosier, W.A.: Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming. IRE Transactions on Engineering Management. **Em-8**(2), 99–115 (1961). doi:10.1109/iret-em.1961.5007599
13. Mead, N.R., Garlan, D., Shaw, M.: Half a Century of Software Engineering Education: the CMU Exemplar. IEEE Software (2018). doi:10.1109/MS.2018.290110743
14. Mills, H.D.: Mathematical Foundations for Structured Programming. Federal Systems Division, International Business Machines Corporation, Gaithersburg. http://trace.tennessee.edu/utk_harlan/56 (1972)
15. Mills, H.D.: Software Engineering. Science. **195**(4283), 1199–2105 (1977)
16. Oettinger, A.A.: Letter to the ACM membership. Communications of the ACM. **9**(8). 545–546 (1966)
17. Perlis, A.J.: Identifying and developing curricula in software engineering. In: Proceedings of the May 14-16, 1969, spring joint computer conference on XX - AFIPS '69 (Spring). Boston, Massachusetts, pp. 540–541. ACM, New York (1969). doi:10.1145/1476793.1476877
18. Pucinski, R.C.: The Challenge for the 1970s in Information Retrieval. In: Tou, J.T. (ed.) Software Engineering COINS III: Proceedings of the Third Symposium on Computer and

Information Sciences held in Miami beach, Florida, December, 1969, vol. 2, pp. XVII-XX. Academic Press, New York, London (1971). doi:10.1016/B978-0-12-696202-4.50006-1
19. Randell, B.: Fifty Years of Software Engineering - or - The View from Garmisch. arXiv:1805.02742 [cs.SE]. https://arxiv.org/abs/1805.02742 (2018). Accessed 23 Jun 2018
20. Royce, W.W.: Managing the Development of Large Software Systems: Concepts and Techniques. In: Proceedings of IEEE WESCON, Los Angeles, 25-28 August 1970, pp. 1–9 (1970)
21. Shaw, M.: Continuing Prospects for an Engineering Discipline of Software. IEEE Software. **26**(6), 64–67 (2009). doi:10.1109/ms.2009.172
22. Shaw, M.: Prospects for an Engineering Discipline of Software. IEEE Software. **7**(6), 15–24 (1990)
23. Shaw, M. Three Patterns that help explain the development of Software Engineering. In: Brennecke, A., Keil-Slawik, R. (eds.) History of Software Engineering : Position Papers for Dagstuhl Seminar 9635 on August 26 – 30, 1996 organized by William Aspray, Reinhard Keil-Slawik and David L. Parnas, pp. 52–56. https://www.dagstuhl.de/Reports/96/9635.pdf (1997). Accessed 6 May 2018
24. Naur, P., Randell, B. (eds.): Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968. Scientific Affairs Division, NATO, Brussels (1969)
25. Buxton, J.N., Randell, B. (eds).: Software Engineering Techniques: Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969. Scientific Affairs Division, NATO, Brussels (1970)
26. Tou, J.T.: Software Engineering – A New Profession. In: Tou, J.T. (ed.) Software Engineering COINS III: Proceedings of the Third Symposium on Computer and Information Sciences held in Miami beach, Florida, December, 1969, vol. 1, pp. 1–6. Academic Press, New York, London (1970). doi:10.1016/B978-0-12-395495-4.50009-X
27. Williams, A.S.: 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 (1976)
28. Brooks, F.P.Jr.: The Mythical Man Month: Essays on Software Engineering. Addison-Wesley Publishing Company, Reading (1975)
29. Dzerjinskiy, F.Ya., Raykov, L.D.: Chto takoe programmnaia inzheneriia (What is software engineering). Programmirovanie. 2, 67–79 (1990)
30. Linger, R.C., Mills, H.D., Witt, B.I. Structured Programming: Theory and Practice. The Systems programming series. Addison-Wesley Publishing Company, Reading (1979)
31. Pro zatverdzhennia Pereliku priorytetnykh napriamiv osvity i nauky shchodo navchannia studentiv ta aspirantiv, stazhuvannia naukovykh i naukovo-pedahohichnykh pratsivnykiv u providnykh vyshchykh navchalnykh zakladakh ta naukovykh ustanovakh za kordonom u 2012 rotsi (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). http://zakon2.rada.gov.ua/laws/show/z0500-1 (2012). Accessed 21 Mar 2012
32. Pro zatverdzhennia pereliku priorytetnykh tematychnykh napriamiv naukovykh doslidzhen i naukovo-tekhnichnykh rozrobok na period do 2020 roku (On approval of the list of priority thematic areas of scientific research and scientific and technical developments for the period until 2020). http://zakon5.rada.gov.ua/laws/show/942-2011-%D0%BF (2011). Accessed 21 Mar 2012