Вступ до другого видання

У далекі 1990-ті роки ми працювали з компаніями, які мали труднощі у своїх проєктах. І виявилося, що ми щоразу питали у них про одне й те саме: "Може, вам краще спочатку тестувати свої програмні продукти, а потім постачати їх? Чому код створюється лише на машинах розробників та ніхто не питає думку користувачів щодо нього?"

Щоб зекономити час нових клієнтів, ми стали робити нотатки, які врешті-решт перетворилися на цю книгу. І на наш подив, книга, здається, знайшла жвавий відгук у читачів, не втративши своєї популярності протягом минулих двадцяти років.

Але двадцять років є чималим строком для розробки програмного забезпечення. Якщо спробувати перемістити розробника у часі з 1999 року та ввести його до сучасної команди, йому буде досить складно у чужому для нього новому світі. Але й світ 1990-х років рівною мірою чужий сучасному розробникові. Посилання у цьому виданні на літературу з CORBA, CASE, індексованих циклів, були б у найкращому випадку кумедними та, найімовірніше, лише б спантеличували.

В той же час двадцять років, що минули, не справили жодного впливу на те, що називається загальними принципами. Адже могла змінитися технологія, але не люди. Практики та підходи, що були ефективні тоді, залишаються такими й сьогодні. Ці моменти книги й зараз ніврок. І коли настав час випустити це 20-річне ювілейне видання, нам довелося прийняти нелегке рішення. З одного боку, ми могли б продивитися та оновити опис тих технологій, на які ми посилалися у попередньому виданні, і на цьому поставити крапку. А з іншого боку, ми могли б переглянути свої поради з приводу тих норм практики, які ми рекомендували раніше, походячи з досвіду, що накопичився за минулі двадцять років. Але врешті-решт ми зробили і те й інше.

В результаті книга стала чимось на кшталт *корабля Тесея*[1]. Близько однієї третини тем у ній виявилися абсолютно новими, а решта були частково чи повністю переписані. Ми мали намір зробити викладення матеріалу більш зрозумілим, доречним та, як можна сподіватися, таким, що не застаріває.

[1] Якщо з роками поступово замінювати кожну частину корабля, чи залишиться він тим самим кораблем?

Нам довелося прийняти низку складних рішень. Зокрема, ми виключили додаток "Першоджерела", як тому, що було б неможливо зберегти його актуальність, так і тому, що потрібні першоджерела читачеві простіше знайти самостійно. Крім того, ми реорганізували та переписали теми, що стосуються паралелізму, беручи до уваги сучасний достаток апаратного забезпечення для паралельних обчислень та нестачу способів їхньої організації. Нарешті, ми додали матеріал, що відображає зміни у стосунках та середовищах: від руху за гнучку розробку, початку якого ми сприяли, до зростаючого прийняття ідіом функціонального програмування та потреби брати до уваги конфіденційність та безпеку.

Цікаво, однак, що суперечок з приводу змісту цього видання у нас було менше, ніж за написання першого видання цієї книги. Ми обоє вважали, що виявити важливий матеріал цього разу було простіше.

Так чи інакше, це видання побачило світ, тож користуйтесь ним для блага, можливо, прийнявши нові норми практики, а може, вирішивши, що деяка частина матеріалу, який ми пропонуємо, вам не підходить. Але у будь-якому випадку ми сподіваємося, що ви глибше зануритесь до вашого ремесла та відгукнетесь на нашу книгу. А найголовніше, не забудьте весело провести час.

Структура книги

Цю книгу написано як зібрання коротких тем, кожна з яких є самодостатньою та присвячена конкретному предмету. У тексті ви знайдете численні перехресні посилання, що допомагають ввести кожну тему до відповідного контексту. Окремі теми можна читати у довільному порядку, оскільки цю книгу складено таким чином, щоб не читати її від початку і до кінця. Періодично у книзі вам зустрічатимуться врізання під заголовком "Порада *nn*" (наприклад, "Порада 1. Турбуйтесь про своє ремесло"). Ми розглядаємо подібні поради як місця у тексті книги, що звертають на себе увагу. Вони мають самостійне значення та слугують для того, щоб користуватися ними щодня у практичній діяльності.

У цьому виданні було включено вправи та завдання там, де це було доречно. Загалом, вправи мають відносно прості рішення, в той час, як завдання припускають багато рішень. Щоб дати уявлення про хід своїх міркувань, ми виділили рішення вправ у окремий додаток до книги, хоча лише небагато з них мають єдине *правильне* рішення. А завдання, що пропонуються, можуть слугувати фундаментом для групових обговорень чи

самостійних робіт на спеціалізованих курсах з програмування. Крім того, наприкінці книги наведено перелік літератури (книг та статей), на яку робляться явні посилання у тексті самої книги.

Що означає ім'я

— Коли я вживаю якесь слово, — майже глузливо проказав Шалам-Балам, — воно означає те, що я хочу — не більше й не менше...

Л'юїс Керролл, *Аліса у Задзеркаллі*,
переклад Валентина Корнієнка

Текст книги рясніє різноманітними професійними термінами: як звичайними словами, трохи викривленими, щоб висловити якийсь особливий технічний сенс, так і жахливими словами, які спеціально обрано для позначення окремих понять з обчислювальної техніки тими спеціалістами у цій галузі, які зневажливо ставляться до мови. За першого вживання терміну ми намагаємося його визначити, або хоча б дати підказку щодо того, що воно може бути таке. Але, на наш погляд, одні терміни так і залишилися незатребуваними, в той час, як інші (наприклад, *об'єкт* та *реляційна база даних*) є настільки розповсюдженими, що розжовувати їхні значення означало б лише наводити на читача нудьгу. Якщо ж вам зустрінеться незнайомий раніше термін, не пропускайте його, а знайдіть час, щоб з'ясувати його визначення в інтернеті чи довіднику з обчислювальної техніки. І якщо ваші пошуки завершаться успішно, повідомте нам електронною поштою, щоб ми додали визначення терміну, яке ви знайшли, до подальшого видання книги.

Попри все сказане вище, ми все ж вирішили помститися спеціалістам з обчислювальної техніки. Зокрема, ми вирішили проігнорувати деякі цілком придатні терміни для позначення певних понять. Чому? А тому, що термінологія, що наразі існує, як правило, обмежується конкретною предметною галуззю чи стадією розробки. Але ж одним з фундаментальних принципів цієї книги є те, що більша частина методик, які ми рекомендуємо, виявляється універсальною. Скажімо, модульність можна застосувати до початкового коду, проєктних рішень, документації та організації команд розробників. Коли ж треба було застосувати звичайний термін у більш широкому контексті, виникала плутанина, оскільки нам не вдава-

лося здолати вантаж змісту цього терміну. І коли відбувалося щось подібне, ми вирішували знехтувати мовними нормами, вигадуючи свої терміни.

Початковий код та інші ресурси

Здебільшого, початковий код, що наведено у цій книзі, взято з початкових файлів, що компілюються, та є вільно доступним для завантаження з нашого вебсайту[2]. Там наведено також посилання на інші корисні ресурси разом з оновленнями книги та новинами про інші розробки програмістів-прагматиків.

Надсилайте нам свої відгуки

Ми будемо раді вашим відгукам на книгу. Пишіть нам на електронну пошту: `ppbook@pragprog.com`.

[2] Див. за адресою: `https://pragprog.com/titles/tpp20/source_code`.

Із вступу до першого видання

Ця книга допоможе вам стати більш досконалим програмістом. Ви можете бути самостійним розробником, членом команди у великому проєкті чи консультантом, який працює з кількома компаніями одразу. Так чи інакше, ця книга допоможе вам у індивідуальному сенсі краще виконувати свою роботу. Вона не носить теоретичного характеру, а зосереджена на практичних питаннях та застосуванні накопиченого досвіду для прийняття більш обґрунтованих рішень. Термін *прагматик* походить від латинського слова *pragmaticus*, що означає "досвідчений у справі" та, у свою чергу, має походження від грецького *πραγματικός*, "придатний для використання". Ця книга про дію.

Програмування — це ремесло. У найпростішому випадку воно зводиться до того, щоб змусити комп'ютер зробити те, що потрібно програмісту чи користувачу його програми. Програміст виступає частково у ролі слухача, порадника, перекладача та диктатора. Він намагається зрозуміти розпливчасті початкові вимоги та знайти спосіб висловити їх таким чином, щоб обчислювальна машина змогла віддати їм належне. Він намагається задокументувати та організувати результати своєї праці таким чином, щоб інші могли їх зрозуміти та спертися на них. Більше того, програміст намагається зробити все це, попри невблаганні строки виконання проєкту. Щодня програміст творить невеличкі дива.

Це непроста річ.

Багато людей готові допомогти програмісту. Скажімо, постачальники інструментальних засобів всіляко розхвалюють чудеса, які здатні творити їхні програмні продукти. Знавці методології обіцяють, що їхні методики гарантовано принесуть очікувані результати. І кожен заявляє, що його мова програмування найкраща, а операційна система дає відповіді на всі можливі питання.

Все це, звісно, не відповідає дійсності, адже простих відповідей не буває. Не існує *найкращого* рішення, неважливо, чи це стосується інструментального засобу, мови програмування чи операційної системи. А можуть бути лише такі системи, які є більш придатними за конкретних обставин.

Саме тут і може стати у нагоді прагматизм, який означає, що не слід прив'язуватися до якої-небудь конкретної технології, але треба мати достатньо великі знання та досвід, що дозволяють приймати правильні рі-

шення за конкретних обставин. Знання походять з розуміння основних принципів обчислювальної техніки, а досвід — з широкої низки практичних проєктів. Теорія у поєднанні з практикою складають міцний фундамент для програміста.

Свій підхід програміст адаптує до поточних обставин та оточення. Він оцінює відносну важливість всіх чинників, що впливають на проєкт, та користується своїм досвідом для вироблення підходящих рішень. І робить він це постійно по мірі просування своєї роботи. Програмісти-прагматики доводять свою роботу до кінця і виконують її добре.

Для кого ця книга

Вона призначена для тих, хто прагне стати більш ефективним та продуктивним програмістом. Ви, імовірно, відчуваєте розчарування через те, що не розкриваєте, як вам здається, свій справжній потенціал. А можливо, ви звертаєте увагу на своїх колег, які користуються інструментальними засобами з метою підвищити продуктивність своєї праці. Можливо, у своїй теперішній роботі ви користуєтесь старими технологіями та хочете дізнатися, як прикласти нові ідеї до того, що ви робите.

Ми не претендуємо на те, щоб відповісти на всі (або хоча б на більшість) питань, що виникають у програмістів, а також не претендуємо на те, що наші ідеї вийде застосувати у всіх можливих випадках. Ми можемо лише сказати, що якщо ви оберете наш підхід, то швидко набудете необхідного досвіду, продуктивність вашої праці зросте і ви краще розумітимете весь процес розробки програмного забезпечення. І в кінцевому рахунку ви зможете писати більш якісні програми.

Що означає бути програмістом-прагматиком

Кожен розробник неповторний, відрізняється своїми сильними та слабкими сторонами, симпатіями та антипатіями. З часом кожен розробник створює своє власне середовище, що відображає його чи її індивідуальність тією ж невблаганною мірою, як і його чи її захоплення, одяг чи зачіска. Але якщо ви є програмістом-прагматиком, то вам притаманні багато з перерахованих нижче загальних характеристик.

- **Ви вчасно приймаєте інновації, використовуєте їх та швидко пристосовуєтесь до них.** Ви інстинктивно відчуваєте переваги окремих технологій та методик, і вам подобається пробувати їх. Якщо

вам дають щось нове, ви швидко оволодіваєте ним, додаючи набуте до решти своїх знань. Ваша впевненість народжується з досвідом.

- **Ви допитливі.** Прагнете ставити питання. Наприклад: *як ви це зробили? Чи виникли у вас труднощі з цією бібліотекою? Що таке квантові обчислення, про які я чув? Яким чином реалізуються символічні посилання?* Ставлячи подібні питання, ви, ніби людина, що збирає на перший погляд непотрібні речі, накопичуєте дрібні факти, кожен з яких може вплинути на яке-небудь рішення багато років потому.
- **Мислите критично.** Рідко берете щось на віру, не отримавши спочатку факти, що це підтверджують. Скажімо, якщо колеги говорять: "Тому що треба робити так!" або якщо постачальник обіцяє вирішити всі ваші проблеми, ви інтуїтивно відчуваєте, що на вас очікують складнощі.
- **Ви реаліст.** Ви намагаєтесь зрозуміти таємний характер кожної складнощі, що у вас виникає. Подібний реалізм дає вам чітке розуміння того, у чому саме полягають труднощі та як довго їх доведеться долати. Глибоко розуміючи, що процес *повинен* бути складним чи його завершення *потребуватиме* часу, ви набуваєте необхідної витримки, щоб впоратися з ним.
- **Майстер на всі руки.** Ви старанно намагаєтесь засвоїти широку низку технологій та середовищ, працюючи над тим, щоб бути в курсі нових розробок. І хоча ваша поточна робота може вимагати від вас бути вузьким спеціалістом, ви завжди готові перейти до нових сфер діяльності та прийняти нові виклики.

Ми залишили розгляд більшості основних характеристик наостанок. Вони притаманні всім програмістам-прагматикам і є настільки простими, що їх можна сформулювати у вигляді окремих порад, як показано нижче.

Порада 1 Турбуйтесь про своє ремесло

Ми вважаємо, що розробляти нове програмне забезпечення немає сенсу, якщо не потурбуватися про те, щоб робити це як слід.

Порада 2 Думайте! Про свою роботу

Щоб стати програмістом-прагматиком, ми закликаємо вас думати про те, що ви робите, у той час, коли ви це робите. Це не одноразова ревізія поточних норм практики, а безперервне критичне оцінювання кожного рішення, що приймається, яку здійснюють щодня та у кожному проєкті. Ніколи не рухайтесь на автопілоті. Постійно думайте, критично оцінюючи свою роботу у реальному часі. Старий девіз корпорації IBM "ДУМАЙ!" (THINK!) є мантрою для програміста-прагматика.

Якщо така робота здасться вам складною, значить, ви мислите *реалістично*. Для цього знадобиться трохи вашого дорогоцінного часу, якого, імовірно, і без того не вистачає. Але винагородою за старанність буде те, що ви активно залучатиметесь до своєї улюбленої справи, відчуваючи, що цілком володієте низкою предметів, що неухильно розширюється, а також отримуєте задоволення від постійного вдосконалення. А в довгостроковій перспективі витрачений вами час окупиться сповна, коли ви й ваша команда працюватимете більш ефективно, щоб писати код, який легко супроводжується, та проводити менше часу на нарадах.

Окремі прагматики та великі команди

Дехто вважає, що індивідуальності немає місця у великих командах чи складних проєктах. "Програмне забезпечення є технічною дисципліною, — говорять вони, — яка руйнується, якщо окремі члени команди приймають рішення самостійно". Ми категорично не згодні з цим.

У побудові програмного забезпечення *повинна* бути присутньою технічна складова, але це зовсім не заважає проявляти індивідуальну майстерність. Розглянемо у якості прикладу великі кафедральні собори, побудовані в Європі за часів Середньовіччя. Зведення кожного з них потребувало чималих витрат, які вимірюються у тисячах людино-годин протягом багатьох десятиліть. Засвоєні уроки передавалися від одного покоління будівельників до іншого, яке вдосконалювало техніку будівництва своїми індивідуальними досягненнями. Але ж теслярі, каменярі, різники та склодуви були ремісниками, що втілювали технічні вимоги у щось цілісне, таке, що виходило за межі виключно механічного боку будівництва. *Навіть ті, хто лише обробляє камінь, повинні завжди уявляти собі цілі собори.*

У всій структурі проєкту завжди знайдеться місце для індивідуальності та майстерності. І це особливо справедливо для поточного стану розробки програмного забезпечення. За сто років наші способи розробки можуть здаватися настільки ж архаїчними, як методи середньовічних будівель-

ників, що зводили кафедральні собори, виглядали для сучасних інженерів-будівельників, хоча наша майстерність буде шануватися так само.

Це безперервний процес

> Турист, що одного разу відвідав Ітонський коледж в Англії, спитав садівника, як тому вдається підтримувати галявину у ідеальному стані.
>
> Дуже просто, — відповів садівник. — Треба лише змахувати росу щоранку, стригти траву щодня та стягувати її один раз на тиждень.
>
> І це все? — спитав турист.
>
> Так, це все! — відповів садівник. — Робіть це протягом п'ятисот років, і у вас теж вийде чудова галявина.

Прекрасні галявини не потребують великого догляду, як, втім, і чудові програмісти. Консультанти з керування люблять додавати слівце *кайзен* до своїх бесід. Термін *кайзен* японською означає безперервний процес внесення багатьох дрібних вдосконалень. Це стало однією з головних причин для докорінних змін у продуктивності та якості виготовлення японських товарів і широко розповсюдилося по всьому світу. Кайзен застосовується і на індивідуальному рівні, де окремі особистості щодня працюють над вдосконаленням власних навичок, поповнюючи свій арсенал новими інструментальними засобами. Але, на відміну від ітонських галявин, вони починають помічати результати буквально за лічені дні. А з роками вони вражаються тому, наскільки розквітнув їхній особистий досвід і розвинулися їхні індивідуальні навички.