Problematic questions of software requirements engineering training

Andrii M. Striuk[1][0000−0001−9240−1976]

Kryvyi Rih National University,
11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine
andrey.n.stryuk@gmail.com
http://mpz.knu.edu.ua/pro-kafedru/vikladachi/224-andrii-striuk

Abstract. The key problems of training Requirement Engineering and the following ways to overcome the contradiction between the crucial role of Requirement Engineering in industrial software development and insufficient motivation to master it in the process of SE specialists professional training were identified based on a systematic research analysis on the formation of the ability of future software engineering (SE) specialists to identify, classify and formulate software requirements: use of activity and constructivist approaches, game teaching methods in the process of modeling requirements; active involvement of stakeholders in identifying, formulating and verifying requirements at the beginning of the project and evaluating its results at the end; application of mobile technologies for training of geographically distributed work with requirements; implementation of interdisciplinary cross-cutting projects on SE; involvement of students in real projects; stimulating the creation of interdisciplinary and age-old student project teams.

Keywords: software requirements · software engineering training · software engineer competencies

1 Вступ

Перший курс з інженерії програмного забезпечення (ІПЗ), розроблений під керівництвом Ф. Л. Бауера [11, 12], містив лише короткий огляд процесу визначення вимог до програмного продукту: його функцій, відповідності потребам користувачів та вимог до операційного середовища. Як і 50 років тому, визначення вимог до програмної системи є першим кроком розробки, що у значній мірі забезпечує її успішність.

Рекомендації до розробки навчальних програм для бакалаврів ІПЗ визначають компетентність з пошуку компромісів, сутність якої полягає в узгодженні суперечливих цілей проекту, пошуку прийнятних компромісів за обмежень вартості, часу, знань, існуючих систем і організацій: “студенти повинні займатися проблемами, які приводять їх до суперечливих і мінливих вимог. ... Компоненти навчальної програми повинні спрямовувати на

такі проблеми з метою забезпечення високоякісних функціональних і нефункціональних вимог та розробку якісного програмного забезпечення" [10, с. 21].

Інженерія вимог (*requirements engineering*) – це процес виявлення, формалізації та документування вимог, що відбувається під час комунікації із замовником та іншими зацікавленими особами, які зазвичай не володіють методами інженерії програмного забезпечення. Виявлення вимог потребує від фахівця з ІПЗ застосування наступних загальнопрофесійних компетентностей [9]:

- здатність до абстрактного мислення, аналізу та синтезу;
- здатність застосовувати знання у практичних ситуаціях;
- здатність спілкуватися усно та письмово;
- здатність до пошуку, оброблення та аналізу інформації з різних джерел;
- здатність працювати в команді;
- здатність діяти соціально відповідально та свідомо.

Формування та розвиток вказаних компетентностей відбувається у навчанні дисциплін, що не відносяться до професійно зорієнтованих, що знижує увагу студентів до їх вивчення. При цьому інженерія вимог до програмного забезпечення, не пов'язана безпосередньо із діяльністю, що студенти-початківці асоціюють із ІПЗ – створенням програмного коду, – сприймається як неважливий курс.

Розв'язання вказаної проблеми можливе шляхом розробки практико зорієнтованої методики навчання інженерії вимог на основі діяльнісного підходу, спрямованої на подолання *протиріччя між визначальною роллю інженерії вимог у практиці промислової розробки великих програмних проектів та недостатньою мотивацією до її опанування в процесі професійної підготовки фахівців з ІПЗ*.

2 Результати

У рекомендаціях з розробки бакалаврських програм в галузі ІПЗ ("Software Engineering 2014" [10]) вказано, що вимоги відображають реальні потреби користувачів, клієнтів та інших зацікавлених сторін (стейкхолдерів), пов'язаних із системою, що розробляється. Визначення вимог включає виявлення та аналіз потреб стейкхолдерів і створення відповідного опису бажаної поведінки та якостей системи, а також відповідних обмежень та припущень.

"Computing Curricula 2020" [2, с. 120] визначає наступні компетенції з визначення до вимог до програмного забезпечення:

1. Визначення та документування вимог до програмного забезпечення із застосуванням методик виявлення вимог на робочих сесіях із стейкхолдерами, використовуючи фасилітативні навички.
2. Аналіз вимог до програмного забезпечення на відповідність, повноту та здійсненність, вдосконалення документації щодо вимог.

3. Формулювання вимог до програмного забезпечення з використанням стандартних форматів специфікацій та способів опису, що були обрані для проекту; здатність зрозумілим чином описати вимоги неспеціалістам, таким як кінцеві користувачі, інші стейкхолдери або адміністративні менеджери.
4. Здатність перевіряти та підтверджувати вимоги, використовуючи стандартні методики, включаючи перевірку, моделювання, створення прототипів та розробку тестових кейсів.
5. Здатність як члена команди інженерів з вимог дотримуватись процедур управління процесами та продуктами, які були визначені для проекту.

І. Зедельмайєр (Yvonne Sedelmaier) та Д. Ландес (Dieter Landes) [8] вказують, що інженерія вимог є основним компонентом ІПЗ, але навчати їй досить складно. У навчанні ІПЗ, як правило, обмежується невеликими "іграшковими" проектами, які лише частково відображають реальні задачі. Це пов'язано з кількома причинами.

1. На початковій стадії підготовки фахівців із ІПЗ робиться *ухил у навчання програмування*, а не на ідентифікацію, класифікацію та формулювання вимог до програмного забезпечення. Як правило, завдання з програмування невеликі та чітко визначені: студенти розробляють прості програмні компоненти, часто невеликими групами або самостійно, завдання зосереджені на конкретній задачі, що демонструє, наприклад, використання циклів, масивів або пов'язаних з їх обробкою алгоритмів. Тобто завдання з розробки програмного забезпечення в основному зосереджуються на технічних аспектах програмування та конкретних мовах програмування у галузі, що добре знайома студентам.
2. Як наслідок, викладач надає чітко визначені та зрозумілі вимоги з цієї галузі без використання незнайомої термінології чи незвичних понять, що формує у студентів хибне уявлення про те, що не потрібно турбуватися про вимоги, оскільки вони *некоректно узагальнюють свій початковий досвід програмування на реальні проекти розробки програмного забезпечення*: для них не існує такого поняття, як нечіткі вимоги стейкхолдерів, які використовують незнайому термінологію, оскільки студенти ніколи з ними не стикалися. Часто ця інформація не є легкодоступною, до того ж її слід отримати від цілого ряду відповідних стейкхолдерів, яких часто непросто визначити, та які при цьому не співпрацюють. Студенти недооцінюють важливість інженерії вимог, оскільки її методи не сприяють кращому розв'язанню завдань із програмування. Крім того, завдання програмування, як правило, відокремлені та не пов'язані з іншими завданнями. Навіть якщо існує більше одного можливого способу розв'язання завдання, обраний підхід не матиме наслідків для наступних завдань: студентам не потрібно зіставляти переваги та недоліки альтернативних рішень, оскільки вони не постраждають від наслідків помилки, а отже, не має значення, помилкові були вимоги чи ні.
3. Часто студенти навіть не можуть уявити проблеми в ІПЗ, що виникають через помилки при визначенні вимог, та не вірять викладачам-

практикам, уважаючи, що вони їх гіперболізують. Методи визначення вимог їм здаються нудними і марними, оскільки *студенти переважно не знають, навіщо вони їм потрібні*.

4. На подальших стадіях підготовки фахівців із ІПЗ навіть при зміщенні ухилу програмування на ІПЗ, зокрема інженерію вимог, ситуація залишається проблематичною: через обмеження часу *складність реальних проблем навряд чи може бути відтворена в університетській освіті*, через що студенти не сприймають взаємозалежності між вимогами, хибно вважаючи, що складність масштабується лінійно, у той час як зростання кількості вимог експоненційно збільшує взаємозалежність між ними.

Враховуючи обмежений час підготовки фахівців із ІПЗ, досить складно створити для студентів умови, які відображають реальні проблеми інженерії вимог: через відсутність реальних замовників студенти не можуть уявити собі складність та взаємозв'язок між вимогами в рамках великого проекту з ІПЗ.

С. Ухбі (Sofia Ouhbi), А. Ідрі (Ali Idri), Х. Л. Фернандес-Алеман (José Luis Fernández-Alemán), А. Товаль (Ambrosio Toval) у [6] формулюють рекомендації для викладачів з формування здатності ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення:

1. Навчати визначати обсяг проблеми та уникати загальних і розмитих специфікацій. Для цього викладачам доцільно враховувати індивідуальні особливості студентів, які можна визначити, зокрема, анкетуванням. Це надає можливість викладачам формувати команди, які демонструють кращі результати. Дослідження [1] показали значну позитивна кореляцію між фактором екстраверсії особистості та якістю програмного продукту, включаючи задоволення вимог.
2. Навчати обирати та використовувати доцільні інструменти інженерії вимог: студенти повинні знати можливості сучасних засобів та вміти обирати найкращий інструмент для проекту відповідно до його потреб – виокремлення вимог, аналіз вимог, специфікація вимог, перевірка та підтвердження вимог, управління вимогами.
3. Сприяти діяльності з аналізу та моделювання вимог на додаток до управління вимогами та впроваджувати концепцію прототипування в навчання: завдяки прототипуванню робоча модель програмного продукту може бути створена до реалізації остаточного продукту, а представлені стейкхолдерам прототипи дуже ефективні для уточнення вимог.
4. Залучати студентів до реальних проектів, щоб надати їм можливість здобути достатні знання та навички. Доцільно також запрошувати практиків для презентації реальних проектів та накопиченого досвіду.
5. Формувати здібності, навички та стратегії, необхідні для узгодження інженерії вимог з сучасними умовами географічно розподіленої (глобальної) розробки програмного забезпечення [3].
6. Навчати студентів підходів до розв'язання проблем, методик та засобів розробки. За лекційно-лабораторної форми організації навчання інженер-

рії вимог студенти не розуміють значення діяльності з визначення вимог та їх вплив на успіх чи невдачу проектів. Викладачам рекомендується використовувати альтернативні форми організації навчання, наприклад, ігрові.

7. Використовувати мобільні пристрої як засоби навчання у мобільному середовищі навчання: студенти можуть спільно використовувати віртуальну дошку, електронні підручники та обмінюватися даними через мережеве середовище для активної участі в обговореннях курсу. Ці пристрої сприяють активізації навчально-пізнавальної діяльності студентів.

Л. Міх (Luisa Mich) [5] вказує на протиріччя між важливістю діяльності з виокремлення вимог для промислових проектів та недостатньою мотивацією студентів до навчання виокремлення вимог, пов'язана з їх відсутнім або малим досвідом роботи у галузі та відповідним ігнорування бізнес-вимог, зосередженням на опануванні засобів моделювання вимог з ухилом у деталізацію замість попереднього глобального огляду проекту, аналізом вимог до реалізації "іграшкових" проектів замість промислових, незалученням стейкхолдерів на етапах постановки проблеми та оцінювання результатів роботи студентів тощо.

Для подолання виокремлених протиріч Л. Міх пропонує використовувати CASE-засоби для підтримки моделювання за допомогою UML та надає рекомендації щодо зменшення ризиків моделювання вимог поспіхом через прагнення студентів розпочинати моделювання вимог, навіть якщо бізнес-аналіз та виявлення вимог ще тільки розпочаті. Дослідником розроблено шаблон студентського проекту, що спрямований на:

– демонстрацію ролі комп'ютерних систем для вирішення бізнес-задач або розробки бізнес-стратегій;
– інтеграцію організаційних питань в аналіз проблеми для відповіді на запитання виду "хто повинен співпрацювати для збору даних, необхідних для розробки системи?";
– розуміння ролі аналізу вимог у процесі розробки системи, включаючи контрактні наслідки;
– виявлення та управління суперечливими вимогами;
– використання UML з самого першого кроку моделювання вимог, також для бізнес-процесів та суб'єктів;
– документування вимог як специфікації проекту та їх підтвердження.

Л. Міх вказує, що завдяки розробленому шаблону проекти стають все більше пов'язаними з реальними організаціями чи компаніями. Спочатку компанії брали незначну участь у проектах, шляхом первинної репрезентації компанії її представником та заключної презентації результатів студентами. Пізніше проводилось все більше інтерв'ю із стейкхолдерами, що представляли реальні організації.

А. Госвамі (Anurag Goswami), Г. С. Валіа (Gursimran Singh Walia) [4] наголошують, що розробка вимог є однією з найперших і найважливіших

фаз життєвого циклу розробки програмного забезпечення. Це критична фаза, коли вимоги до програм збираються від різних зацікавлених сторін (як технічних, так і нетехнічних) та описуються природною мовою в офіційному документі, відомому як специфікація вимог до програмного забезпечення. Через неоднозначність, неточність та невизначеність природньої мови під час розробки специфікації часто припускаються помилок. Таким чином, варто зосередитись на виявленні та усуненні невідповідностей на ранніх стадіях життєвого циклу розробки програмного забезпечення, щоб уникнути зайвих зусиль та витрат на переробку програмного продукту на пізніх стадіях. Для цього дослідники пропонують використовувати методи оцінки (інспектування, обстеження) програмного забезпечення, коли кваліфіковані фахівці переглядають документацію, код та інші артефакти проекту, щоб виявити проблеми та повідомити про них. Така інспекція є систематичним методом детального вивчення програмних артефактів. Дослідження підтвердило переваги перевірки артефактів, розроблених на різних етапах розробки програмного забезпечення (наприклад, вимог, дизайну, коду, інтерфейсів). Основними етапами інспекції є: а) відбір кваліфікованих інспекторів; б) індивідуальний огляд з метою виявлення проблем; в) засідання команди для систематизації проблем та г) подальші дії з їх усунення.

І. Озкая (Ipek Ozkaya), О. Акін (Ömer Akin), Дж. Е. Томайко (James E. Tomayko) [7] пропонують 2 напівсеместрові міні-курси для студентів, що навчаються на архітектурно-будівельних спеціальностях.

Метою першого міні-курсу "Моделювання вимог до програмного забезпечення" є огляд методів моделювання вимог до програмного забезпечення та демонстрація за допомогою моделювання вимог стратегій вирішення задач з розробки програмного забезпечення. Моделювання вимог допомагає інженерам (не лише з ІПЗ) краще зрозуміти поставлену перед ними задачу, розкриває сутність взаємозв'язків, що її характеризують. Дослідники підкреслюють, що це суттєво інший підхід, ніж розгляд функціональних можливостей кінцевого продукту. Для конкретних інженерних задач, таких як побудова геометричного подання інформаційних моделей, проектування комунікаційних програм для мобільних пристроїв або побудова моделей даних для різних етапів проектування, інженеру необхідно не лише виокремити вимоги до програмного забезпечення, що розробляється, але й розуміти вимоги інженерної галузі, для якої воно розробляється. Іншими словами, очікування та потреби клієнта повинні систематично моделюватися для кращого проектування програмних рішень.

Зміст першого міні-курсу цілеспрямовано зосереджено на розробці програмного забезпечення для автоматизованого архітектурного та інженерного проектування. В ньому пропонується вивчення декількох методів виявлення вимог і способів отримання основної інформації, що допоможе в розробці програмного забезпечення. Застосування обговорюваних методів визначення вимог до програмного проекту повинно відбуватися за участі стейкхолдерів: деяких реальних зацікавлених сторін, як правило – замовників. Після завершення першого міні-курсу студенти набувають здатності

розробляти прототип проектованої системи та специфікацію вимог до програмного забезпечення.

Програмними результатами запропонованого курсу є:

- здатність правильно використовувати основну термінологію щодо виявлення та специфікації вимог до програмного забезпечення;
- здатність обирати різні методи збору даних для виявлення вимог до програмного забезпечення;
- здатність розрізняти типи вимог та класифікувати відповідну інформацію;
- здатність документувати вимоги у різних формах;
- здатність використовувати інформацію про вимоги до програмного забезпечення для покращення проектування;
- здатність оцінити різні методи виявлення вимог та розробити стратегії вибору найбільш доцільних [7, с. 5].

Метою другого міні-курсу "Застосування вимог до програмного забезпечення" є навчання застосування визначених вимог для розв'язання практичних задач. Дослідники вказують, що недостатньо лише визначити вимоги до програмного забезпечення – у проектуванні вимоги, як правило, стосуються контексту, в якому задача має бути вирішена. Різні креслення, нотатки та схеми, які проектувальник виробляє як частину рішення, призначені для задоволення цих вимог. Крок, на якому проектувальник перетворює вимоги на проект, є вирішальним, а помилки, допущені під час опрацювання вимог при проектуванні, стають джерелом невдач багатьох програмних проектів.

Програмними результатами другого міні-курсу є:

- здатність будувати діаграми, зокрема UML, для подання вимог;
- здатність використовувати управління потребами в життєвому циклі розробки програмного забезпечення;
- здатність обирати метод розробки програмного забезпечення, придатний для управління заданими вимогами;
- здатність застосовувати методи валідації, верифікації та відстеження вимог;
- здатність розробляти стратегії для перенесення інформації про потреби до дизайну високого рівня [7, с. 6-7].

Програмними результатами курсу "Моделювання програмного забезпечення", розробленого І. Зедельмайєр та Д. Ландесом [8], є:

- поглиблення розуміння терміну "вимоги" та їх ролі у розробці програмного забезпечення;
- здатність використовувати методи специфікації функціональних та нефункціональних вимог та визначення їх пріоритетів;
- розуміння ролі комунікації з іншими сторонами, залученими до розробки вимог;
- розуміння ролі бізнес-процесів як джерела вимог;

– здатність спільно застосовувати відповідні методи та позначення для визначення вимог для прикладу програмного продукту;
– здатність використовувати методи оцінки складності та вартості програмних систем.

Цілями навчання курсу “Моделювання програмного забезпечення” є:

1) формування у студентів розуміння ролі та важливість вимог для їх майбутньої кар’єри – студенти повинні знати проблеми розробки вимог, визнавати їх важливість та труднощі у їх формуванні; студенти повинні вміти видобувати вимоги з майбутніх користувачів, моделювати бізнес-процеси та створювати документи з вимогами;
2) вдосконалення специфічних комунікативних навичок, які необхідні в розробці вимог – студенти повинні мати можливість проводити зустрічі із клієнтами для визначення вимог, які вони не вигадали самі, а сформулювали у співпраці з реальним замовником, навчатися запитувати клієнтів про інформацію, що може стати основою для вимог, документувати вимоги, розподіляти ролі тощо;
3) посилення саморефлексії, самоорганізації та відповідальності студентів як основи розвитку відповідної компетентності.

Навчальне середовище для такого курсу І. Зедельмайєр та Д. Ландес пропонують будувати на основі конструктивістського підходу, за якого викладачі виступають у ролі тренерів, що створюють умови для набуття студентами індивідуального навчального досвіду [8]. Не виділяючи окрему здатність ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення, дослідники виділяють 4 групи компетентностей, необхідних для формування такої здатності [8]:

1) знання предметної галузі, здатність до абстрагування;
2) чутливість до контексту: здатність до модерації та презентації, проведення зустрічей із замовниками, інтеграції у команду, емпатії, витримка;
3) особистісні компетентності: методи роботи, самоорганізації, розподілу ролей, управління часом, особистісна залученість, цілеспрямованість, здатність до саморефлексії;
4) креативність, розмаїття прийомів.

Основними складниками пропонованого у [8] підходу є широка, активна участь студентів у навчанні та реалістичне інтегроване середовище, що включає написання документа з вимогами до складного проекту та видобування вимог від реальних клієнтів, які виконують подвійну роль – крім джерела вимог, вони також виступають як зовнішні експерти з питань комунікації.

3 Висновки

1. Огляд джерел з проблеми дослідження показав спільність проблем у навчанні інженерії вимог майбутніх фахівців з ІПЗ в Україні та зарубежем:

- нерозуміння студентами важливості інженерії вимог через недостатній досвід роботи із реальними замовниками;
- недостатня увага до процесу виокремлення взаємопов'язаних вимог у комунікації зі стейкхолдерами, що не взаємодіють один з одним;
- ухил у навчання мов та засобів моделювання вимог, за якого їх деталізація передує виявленню;
- висока чіткість вимог до навчальних проектів з ІПЗ у порівнянні з реальними;
- нелінійна залежність складності реалізації програмного проекту від зростання кількості вимог.

2. Шляхами подолання вказаних проблем у навчанні інженерії вимог в процесі підготовки фахівців з ІПЗ є:
   - застосування діяльнісного та конструктивістського підходів, ігрових методів навчання у процесі моделювання вимог;
   - активне залучення стейкхолдерів до виявлення, формулювання та верифікації вимог на початку проекту та оцінки його результатів наприкінці;
   - застосування мобільних технологій для навчання географічно розподіленої роботи з вимогами;
   - уведення міждисциплінарних наскрізних проектів з ІПЗ;
   - залучення студентів до реальних проектів;
   - стимулювання створення міжфахових та різновікових студентських проектних команд.

References

[1] Acuña, S.T., Gómez, M., Juristo, N.: How do personality, team processes and task characteristics relate to job satisfaction and software quality? Information and Software Technology **51**(3), 627–639 (2009), ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2008.08.006

[2] CC2020 Task Force: Computing Curricula 2020: Paradigms for Global Computing Education. A Computing Curricula Series Report (2020), URL https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

[3] Damian, D., Hadwin, A., Al-Ani, B.: Instructional Design and Assessment Strategies for Teaching Global Software Development: A Framework, p. 685–690. Association for Computing Machinery, New York, NY, USA (2006), ISBN 1595933751, URL https://doi.org/10.1145/1134285.1134391

[4] Goswami, A., Walia, G.: Teaching software requirements inspections to software engineering students through practical training and reflection. Computers in Education Journal **16**(4), 2–10 (2016)

[5] Mich, L.: Teaching requirements analysis: A student project framework to bridge the gap between business analysis and software engineering. CEUR Workshop Proceedings **1217**, 20–25 (2014)

[6] Ouhbi, S., Idri, A., Fernández-Alemán, J., Toval, A.: Requirements engineering education: a systematic mapping study. Requirements Engineering **20**(2), 119–138 (2015), https://doi.org/10.1007/s00766-013-0192-5

[7] Ozkaya, I., Ömer Akin, Tomayko, J.E.: Teaching to Think in Software Terms: An Interdisciplinary Graduate Software Requirement Engineering Course for AEC Students, pp. 1–10 (2005), https://doi.org/10.1061/40794(179)8, URL https://ascelibrary.org/doi/abs/10.1061/40794%28179%298

[8] Sedelmaier, Y., Landes, D.: A multi-level didactical approach to build up competencies in requirements engineering. CEUR Workshop Proceedings **1217**, 26–34 (2014)

[9] Semerikov, S., Striuk, A., Striuk, L., Striuk, M., Shalatska, H.: Sustainability in Software Engineering Education: a case of general professional competencies. E3S Web of Conferences **166**, 10036 (2020), https://doi.org/10.1051/e3sconf/202016610036

[10] Software Engineering 2014: Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series (2015), URL https://www.acm.org/binaries/content/assets/education/se2014.pdf

[11] Striuk, A.: "Advanced course on software engineering" as the first model for training of software engineers. Journal of Information Technologies in Education (ITE) (40), 48–67 (Sep 2019), https://doi.org/10.14308/ite000702, URL http://ite.kspu.edu/index.php/ite/article/view/732

[12] Striuk, A., Semerikov, S.: The dawn of software engineering education. CEUR Workshop Proceedings **2546**, 35–57 (2019)