УДК 316.774
DOI https://doi.org/10.24195/spj1561-1264.2024.3.2

Янушевич Ірина Анатоліївна
кандидат філософських наук,
доцент кафедри філософії, історії та політології
Національного університету «Одеська політехніка»
пр. Шевченка, 1, Одеса, Україна
orcid.org/0000-0002-1753-5111

Кронберг Єлізавета Володимирівна
студентка інституту комп'ютерних систем
Національного університету «Одеська політехніка»
пр. Шевченка, 1, Одеса, Україна

ПРОГРАМУВАННЯ ЯК ФОРМА МОВИ ТА ПРОБЛЕМА РОЗУМІННЯ: ФІЛОСОФСЬКИЙ СЕНС СИНТАКСИСУ ТА СЕМАНТИКИ ПРОГРАМУВАННЯ

Актуальність проблеми. *Програмування тісно пов'язано з розвитком сучасних інформаційних технологій, в сучасному світі воно є корисним вмінням для розуміння інноваційного розвитку суспільства. Зростання популярності програмування як галузі й поширення його впливу на технологічний прогрес і, як наслідок, на буденність людини робить актуальним його розгляд з філософської точки зору, зокрема аналіз синтаксису та семантики програмування та його зв'язок з універсальною формою мислення людини – розумінням, тобто ключовими аспектами, що визначають його філософський сенс. Отже, в статті була поставлена* ***мета*** *– проаналізувати задані аспекти програмування через призму філософських концепцій, що дозволить поглибити знання та вдосконалити розуміння його природи та значення в сучасному світі.*

Методи дослідження. *Відповідно до поставленої мети дослідження та розв'язуваних завдань, у цій роботі використано методи опису, аналізу, пошуку подібності та ілюстрації аналізованих концепцій з подальшим узагальненням, висуванням гіпотез та подальшим їх обґрунтуванням. На заключному етапі дослідження для аналізу аспектів програмування через призму філософських концепцій, використано системний підхід – у тому варіанті, який передбачає параметрична загальна теорія систем.*

Аналіз заданих аспектів в програмуванні дозволив зробити ***висновки****, що програмування, як зовсім молода наука, потребує філософського погляду. Проблеми програмування можуть бути розглянуті як частина філософських проблем, пов'язаних з науково-технічним прогресом, але питання, наприклад, мови програмування далеко виходять за ці рамки. Коли програмування торкається проблеми змісту, перед програмістом постають проблеми розуміння в тому сенсі як його трактує філософія – в залежності від сфери застосування методів інтерпретації, адекватних об'єкту вивчення, зокрема в програмуванні.*

І хоча існуючи в теперешній час в об'єктно-орієнтованому програмуванні концепції, такі як поліморфізм, абстракція, інкапсуляція та успадкування, які складають основу навичок успішного програміста, внесли вагомий вклад в розробку теоретичних підходів в програмуванні, однак у самих підходах до процесу програмування так і не було зроблено спроби систематизації методологічно обґрунтованих способів програмування в рамках єдиної теоретичної концепції, що неможливо здійснити без виходу у філософську площину міркувань.

Йдеться не про те, що не була створена якась особлива концепція програмування, але про те, що поки не запропоновано нетривіальних філософсько-методологічних напрацювань теорії програмування, які б дозволили розглядати процес програмування як більш-менш раціонально контрольовану та загальнозначиму процедуру. А без цього нагальні завдання програмування стають важкими.

Ключові слова**:** *лінгвістика, парадигма, розуміння, теорія мови програмування, Дзен Пайтона.*

Вступ. Дискусії про те, чи можна вважати мови програмування мовами з точки зору лінгвістики, тривають ще з часів зародження комп'ютерних наук. Хоча офіційно мови розробки не розглядаються як мови з, на перший погляд, інтуїтивно зрозумілих причин, ряд філософів, програмістів та лінгвістів виступають за визнання програмування як мови у самому сенсі цього слова, або принаймні «форми мови», зважаючи на відмінності його мов від тих, якими ми користуємось як засобом комунікації.

Власне поділ на кодування і створення алгоритмів це вже специфіка: спочатку йде життя, тобто людина спирається на якийсь задум. І ось цей початковий напрямок це питання філософії. Проблема в тому, що елементарними філософськими категоріями: мислення, свідомість, розуміння програміст не оперує. І це дивно, якщо порівняти здатність програміста мислити, наприклад, читаючи статті з функціонального програмування чи алгоритмами пошуку, упереміш із статтями видних європейських філософів, виявиться, що власне навик розуміння (саме як його трактує філософія) у програмістів розвинений не менше, а то й більше. Ось тільки мова програміста дуже багата поки що він розмірковує про патерн-матчинг і жалюгідний і органічний, коли йому треба вийти зі своєї сфери, відірватися від IDE і файлового менеджера.

Аналіз досліджень. Цікавий момент, у 1936 році, англійський філософ неопозитивіст, представник аналітичної філософії Альфред Айєр у своїй роботі «Підстави емперичного знання» виказав думку про те, що основна відмінність «дурної залізяки» від справжньої розумної істоти, яка веде розмову з ним, полягає в тому, що вона не зможе пройти один із емпіричних тестів, які показують наявність або відсутність свідомості у тестованого, здатності до розуміння. І саме цей факт вказує на зв'язок методології програмування з проблемою розуміння як універсальної операції мислення в тому ракурсі, як вона представлена різними напрямками в філсофії.

Мета та завдання. Обгрунтування загальної методології розуміння в програмуванні є тим самим завданням, яке передбачає використання таких спеціальних методів, які б релевантні стосовно цієї галузі наукового знання і адекватні по застосовним до цих галузях знання засобам. На цьому шляху особливої актуальності набуває пошук способів розуміння та пізнання, що дозволяють розкрити зміст конкретного тексту будь-якого змісту (філософського, релігійного, наукового), а також факту, події, комунікативної дії, поведінки тощо. Тим самим традиційна для епістемології та герменевтики проблема розуміння конкретизується залежно від сфери застосування методів інтерпретації, адекватних об'єкту вивчення, зокрема в програмуванні. Це широке завдання обумовлена тим обставиною, що адекватної інтерпретації потребує все те, що має сенс.

При цьому, як правило, поняття «адекватності розуміння» вважається інтуїтивно ясним, практично невідмінним від таких понять, як «релевантність», «еквівалентність». А спроба пов'язати адекватність розуміння з панівними у суспільстві системами «нормативів» зрозумілості (іноді званих «передумовим знанням»), веде до труднощів у вирішенні багатьох практичних завдань, пов'язаних із розумінням текстів різного характеру. Оскільки зрозуміло, що будь-яка норма має відносний характер.

Методи дослідження. У методологічному аспекті норми, на відміну законів, не мають характеру жорстких регуляторів, не бувають істинними чи хибними, не відкриваються, а вводяться людиною, не пояснюють фактів, а вказують орієнтири діяльності, допускають аксеологічну оцінку. Норма не призначена для передбачень. У реальному житті будь-яка норма може порушуватися, її функціонування має характер не внутрішнього, а зовнішнього відношення, тому вона і порушується.

Результати дослідження. Системи норм говорять не про те, як реально діяти суб'єкту, регулюючи об'єкт, а про те, яким об'єкт мав би бути в ідеалі. Норми, прагнучи стати закономірністю, водночас хіба що спрямовані на самоліквідацію. Якщо ж поглянути питання функціонування норм із системної погляду, процес перетворення норми на деяку закономірність, у принципі можливий – у разі, якщо норма, як зовнішнє ставлення, входить у звичку і виявляють тенденцію стати внутрішньої. Це те, що в А.І. Уйомова називається «реляційним колапсом» [1; 133]).

Проте власними силами норми, навіть які стали закономірностями, що неспроможні створити строгу, наукоподібну концепцію – цього необхідні, передусім, принципи. Прикладні теорії програмування використовують як принципи загальні та необхідні утвердження наук фундаментальних, а останні не обходяться без філософських принципів. Оскільки на певних етапах розвитку суспільства ті чи інші принципи порозуміння порушуються, то нові принципи стають, як правило, передумовою конвенційної норми розуміння, яка в практичному плані відображає зміни у перевазі тих чи інших підходів до цілей та завдань будь-якої творчої діяльності у сфері програмування.

Щоб принцип став більш-менш суворим регулятором норм процесу розуміння в програмуванні, цей принцип мав би стати саме методологічним, тобто супроводжуватись теоретичним обґрунтуванням його застосування. Від цього залежить статус будь-якого наукового знання як загальної теорії, набуття ним інтерсуб'єктивного характеру (отже, за визначенням, набуття статусу загальнозначущого знання).

В іншому випадку норми, що навіть стали закономірністю, залишаються обґрунтованими лише інтуїтивно, а програмуванню уготована доля, як і раніше, постійно балансувати між трьох іпостасей – в якості науки, мистецтва та ремесла. Постійно балансуючи на межі цих трьох сутностей, програмування, однак, не бажає стати жодною з них. Суворі закони логіки з одного боку, широта вибору алгоритмів з іншого, і шаблонні методи вирішення типових завдань з третього боку дозволяють розглядати розробку ПЗ з різних точок зору і робити висновки, що іноді суперечать один одному. Людина ж за своєю природою не любить невизначеності і намагається зрозуміти складне шляхом спрощення. Так і програмування часто розглядають тільки з якогось одного боку, свідомо чи несвідомо не помічаючи інших його властивостей.

Як відомо, згідно з Оккаму: «сутності не слід примножувати без необхідності» [2; 73-102]. Ймовірно, щоб методологія програмування змогла оформитися як наукова концепція, у ній необхідно виділити якісь інваріантні принципи організації обміну інформацією, що неможливо без вирішення низки питань загальнофілософського характеру. Це мають бути такі ідеї, які вказують, нехай навіть у неявному вигляді, на те, що може вважатися загальнозначущим та раціонально контрольованим. Деякі кроки у цьому напрямі робляться у цій роботі.

У розробці мов програмування дивним чином переплітаються простота і неймовірна складність, жорсткі правила і повна свобода, необхідність і необов'язковість. За досвідом роботи багатьом програмістам доводилося не раз спостерігати як складність і простота в розробці програмного забезпечення химерно змінювалися місцями, призводячи до абсолютно непередбачуваних, хоч і не завжди негативних результатів.

Перша основна відмінність мов програмування від звичайних – форма використання. В той час як звичайна мова використовується й в усній, і в писемній формах, усну розмову на мові програмування важко уявити. «Мова не має бути розмовною, щоб вважатися мовою... але це досить незвично, щоб мова існувала виключно в письмовій формі», – пише німецький лінгвіст Томас Мур Девлін в статті «Чи є мови програмування технічно мовою?» [3, 21-23].

Однак якщо формальний аналіз програмування в основному пов'язаний зі структуруванням тексту, з прагматичним завданням програмування і з тими сенсами, які нав'язуються культурою мови, то змістовний аналіз передбачає розгляд усієї палітри мовних значень, що виникають в силу специфіки граматичного устрою, а також з урахуванням контекстуального значення слів.

Другою важливою відмінністю є мета використання. Основним призначенням мови в звичайному розумінні є слугування засобом спілкування між істотами: розповіді, висловлювання думок, попередження про небезпеку тощо. Мови програмування ж використовуються для спілкування з машинами: із визначення, вони являють собою набір інструкцій, які машина має виконувати. Звісно, людська мова теж може містити набір інструкцій, які інша людина має виконати, проте, на відміну від людської, мова розробки не може містити ні прохань, ні побажань, ні почуттів.

З іншого боку, якщо розглядати призначення мови як комунікація в цілому, а не лише між живими істотами, програмування цілком може вважатися мовою, хоч і не такою, якою

спілкуються між собою люди, проте такою, якою спілкується людина з машиною. Тоді стає зрозумілою й відсутність усної форми мов розробки – комп'ютер, якщо не розглядати його в контексті штучного інтелекту, не здатен перекласти гучність, тембр голосу чи інтонацію на набір інструкцій.

На користь визнання програмування як форми мови грає також інтерпретація його аспектів як розділів мовознавства, зокрема синтаксис і семантику.

Розділ синтаксису в розмовній мові вивчає граматичну будову словосполучень та речень, синтаксичні властивості окремих слів як частин мови та їх зв'язки у складі висловлювань. Синтаксис в мові програмування містить сукупність комбінацій символів та слів, таких, в яких компілятор чи інтерпретатор мови розробки впізнає набір команд і може перекласти їх на машинну мову. Як в людській мові, так і в коді синтаксис визначає правила та структуру об'єднання слів у зрозумілі висловлювання чи інструкції.

Очевидно, різні національні мови мають різний синтаксис: родинні мови, наприклад, українська та польська, мають деякий обсяг спільних елементів синтаксису, а віддалені мови, наприклад, українська та японська, й зовсім не схожі між собою, й часто така різниця пояснюється різними способами мислення та різними філософськими переконаннями. Таке саме спостереження властиве й мовам програмування: наприклад, C та C++, хоч і написані різними розробниками, але є настільки спорідненими за синтаксисом, що їх часто вивчають разом, в той час як Java і Python не тільки різняться правилами написання, а й керуються абсолютно різними принципами написання коду, тобто різні за структурою.

Класичний структуралізм зорієнтований, в цілому, на те, що аналіз структур і пошук серед них найбільш фундаментальних, зрештою, забезпечує правильне трактування тексту, включаючи його концептуальну інтерпретацію. Дотримуючись цієї філософської та культурологічної методологічної концепції, гуманітарні науки мали б основною своєю проблемою вважати пошук ізоструктурності (ізоморфізму) розуміння, а основним методологічним напрямком діяльності – рух типу: структура → концепт. У значній частині випадків це справді важливе завдання програмування, однак сам пошук структур тексту завжди обумовлений наявними концептами дослідника, і залежить від того, в якому соціокультурному середовищі він перебуває. Проблема розуміння не зводиться лише до аналізу структур [4; 584–588].

Розділ семантики ж в розмовній мові вивчає значення слів та словосполучень. В контексті програмування значення слова практично не змінюється: семантика – це значення слів, операторів та конструкцій. Навіть якщо в процесі говоріння чи написання синтаксис дотримано правильно, це не є гарантією того, що даний набір речень чи інструкцій мають фактичний сенс, тому в розмові чи в програмуванні важливо не тільки володіти словарним запасом, а й розуміти контекст вживання слів чи операторів.

Порядок дій, які розробники визначають в процесі написання коду для вирішення певної задачі, залежить від методів абстрагування, які використовує програміст, від його способу мислення. «Семантика мови програмування говорить нам багато про процеси абстракції, задіяні в програмуванні комп'ютерів, а також про природу алгоритмів», – пише британський філософ та інженер Грем Уайт у главі «Філософія комп'ютерних мов» [5; 66–70].

Згадуючи про алгоритми, буде доречним припустити, що їх природа й розвиток пов'язані з прагненням людини покращити автоматизацію роботи системи, зокрема збільшити її надійність та скоротити час виконання інструкцій. Ідеальним прикладом є еволюція алгоритму сортування масиву: від квадратичної складності алгоритму бульбашкового сортування до логарифмічної складності так званого швидкого сортування.

Також важливо зазначити, що метод абстрагування не є єдиним в межах групи людей чи навіть однієї людини: їх існує і використовується безліч, і жоден з них не можна назвати абсолютно правильним чи абсолютно неправильним, так само, як не можна назвати погляд людини на будь-яку ситуацію абсолютно правильним чи абсолютно неправильним. Обраний чи обрані методи залежать не тільки від вподобань розробника, а й від парадигми середовища.

Якщо в філософії парадигма – це сукупність загальноприйнятих філософських основ, то в програмуванні парадигма представляє собою сукупність загальноприйнятих концепцій організації коду. Двома основоположними парадигмами є процедурна, що розглядає програму як сукупність функцій, та об'єктно-орієнтовна (скорочено ООП), що розглядає програму як сукупність об'єктів. Як згадувалося раніше, мови Java і Python різняться як синтаксисом, так і реалізованою парадигмою: в Java програма розбивається на сукупність класів, об'єкти яких взаємодіють між собою, а в Python, хоч і є поняття класу, програма зазвичай пишеться як набір функцій, які викликаються в певному порядку при певних умовах. Два абсолютно різні підходи до написання коду, але обидва мають однакове право на існування – як і в філософії нормальним є одночасний розвиток різних і навіть протилежних точок зору.

Багато філософських складових програмування – синтаксис, семантика, методи абстрагування, парадигми, алгоритми та інші характеристики мов – досліджуються окремою галуззю інформатики, яка називається «теорія мов програмування». Моментом народження цієї теорії вважається заснування в 1930-х роках американським математиком Алонзом Чорчем системи лямбда-обчислень, також відомих як «анонімні функції». Згідно із однойменним записом в Стенфордській Енциклопедії Філософії, вони стали «рогом достатку логіки та математики» завдяки їх «виразності і гнучкості» [6].

В тематиці філософських аспектів програмування, особливо відзначається одна праця, яка, хоч і не є частиною теорії мов програмування, але зробила значний внесок у розвиток філософії програмування і вже протягом двадцяти п'яти років є актуальним об'єктом дискусій серед програмістів та філософів. «Дзен Python-а» – це невеличкий збірник з дев'ятнадцяти «Python-ічних», як назвав їх сам автор, принципів написання коду на Python-і, доданих у список розсилки Python-а і пізніше опублікованих у публічний домен американським розробником Тімом Пітерсом. Звучать вони наступним чином: «Красиве краще, ніж потворне. Явне краще, ніж неявне. Просте краще, ніж складне. Складне краще, ніж заплутане. Плоске краще, ніж вкладене. Рідке краще, ніж щільне. Читабельність має значення. Особливі випадки не настільки особливі, щоб порушувати правила. Хоча практичність перемагає правильність. Помилки ніколи не повинні проходити мовчки. Якщо явно не замовчувати. Зіткнувшись з двозначністю, відмовтеся від спокуси вгадувати. Має бути один – і бажано лише один – очевидний спосіб це зробити. Хоча спочатку цей шлях може бути неочевидним, якщо ви не голландець (Тім Пітерс має відчуття гумору). Зараз краще, ніж ніколи. Хоча ніколи краще, ніж «прямо» зараз. Якщо реалізацію важко пояснити, це погана ідея. Якщо реалізацію легко пояснити, це може бути хорошою ідеєю. Простори імен – це чудова ідея, – давайте робити їх більше!» [7, 46-64]. Хоча останній принцип більше пов'язаний із технічною складовою організації коду, перші вісімнадцять принципів можна трактувати й за межами програмування на мові Python, і мова йде не лише про інші мови розробки, а й про сфери життя людини.

Та, як кажуть, популярність породжує конкуренцію. Багато програмістів намагалися написати свій власний дзен для інших мов програмування, як на аматорському рівні у вигляді публікацій в інтернеті, так і на професійному – на конференціях із супроводженням науково-філософських тез і доводів. Однією з популярних таких робіт є «Дзен Рубі» американського розробника Еріка Пірса, що виступає як протилежність «Дзену Python-а». Кожен його принцип суперечить відповідному принципу дзену Тіма Пітерса: «Краса в очах того, що дивиться. Неявний є кращим, ніж явний. Просте є нудним. Складне є цікавим. Доручіть деталі комусь іншому. Якщо можливо, зробіть це однорядково. Читабельність іноді приємна. Особливі випадки всюди; правила не можуть охопити їх усіх. Якщо сумніваєтеся, використовуйте «мавпячий патч». Помилки слід придушувати. Якщо не ввімкнено «плаксивий нуль». Якщо сумніваєтеся, зробіть припущення щодо того, чого хотів користувач. Має бути багато – бажано десятки – неочевидних способів зробити це. Те, що очевидно для вас, може бути зовсім неінтуїтивним для когось іншого. Краще зараз, ніж пізніше. І краще пізніше, ніж ніколи. Якщо проект має недоліки, поясніть причину в документах реалізації. Якщо дизайн хороший, не турбуйтесь про реалізацію документів. Простори імен абсолютно непотрібні – давайте зробимо

все глобальним!» Уявлення цих двох розробників про правила написання програм є різними, та обидві думки мають сенс у контекстах як програмування, так і філософії. Свобода думки є важливим елементом розумового й духовного розвитку людини, філософії, програмування й інших сфер життя.

Висновки. Отже, робота програміста в значній мірі є творчою, оскільки часто обмежена лише конкретним завданням, тоді як сама реалізація і саме бачення підходів розробки залежить від розуміння виконавця. Технічне завдання навіть для великих проектів іноді складаються надто поверхово, що в результаті призводить до того, що розробник сам повинен фантазувати і застосовувати всі «творчі куточки» свого мозку для проектування системи.

Разом з тим, надмірне зловживання творчістю призводить до того, що в результаті з'являються зовсім дивні компоненти системи: починаючи від класів з ієрархією мови програмування, закінченої складними методами, про реальну потребу яких ніхто навіть не замислювався, і шаленим «копіпастом» коду. Хоча, саме в цьому і є елементи творчості, а саме в сенсі не дотримання стандартів і правил при написанні коду, а в імпровізації при виконанні завдання.

Але в тій мірі, в якій програміст дійсно описує реальний стан справ, використовує інваріантні методи програмування (адже насправді неможливо написати чітко структурований правильний код, не обмежуючи себе ні в чому і не дотримуючись усталених правил), програмування є наукою.

З цього випливає висновок, що філософський сенс програмування можна трактувати по-різному – від ідеї простоти й швидкості алгоритмів до звеличення творчого погляду на різні аспекти. Галузь програмування стрімко розвивається, створюючи нові й вдосконалюючи старі свої аспекти; еволюціонує й філософія, впливаючи на методи абстрагування програмістів. Дисципліни гнучкі за своєю природою й пов’язані в контексті взаємовдосконалення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Uyomov A., Saraeva I., Tsofnas A. Ogólna teoria systemów dla humanistów. – Wydawnictwo Universitas Redeviva, 2001. – С. 133. – 276 с.

2. Оккам У. Избранные произведения: Пер. с лат. – К.: Эдиториал УРСС, 2002. – С. 73–102.

3. Thomas Moore Devlin – «Are Programming Languages Technically A Language?», 2022 https://www.babbel.com/en/magazine/are-programming-languages-language

4. Потапова І. Умберт Еко про теорію перекладу: автобіографічний аспект //Філософські пошуки. – Львів-Одеса, 2004. – С. 584–588. – 739с.

5. Graham White – «The Philosophy of Computer Languages» («The Blackwell Guide to the Philosophy of Computing and Information», pp. 237–247), 2004 https://sidoli.w.waseda.jp/White_2004_Philosophy_Computer_Language.pdf

6. Stanford Encyclopedia of Philosophy – «The Lambda Calculus», 2023 https://plato.stanford.edu/entries/lambda-calculus/

7. Tim Peters – «The Python Way» Python-list thread, 1999 https://mail.python.org/pipermail/python-list/1999-June/001951.html; Python Enhancement Proposals, 2004 https://peps.python.org/pep-0020/

REFERENCES

1. Ujomov, A.I., Sarajeva I.V., Zofnas A.J. (2001). Obczaja teorija sistem dlja gumanitarijev. [The General system theory for humanitarian].Warsaw: Wydawnictwo Universitas Redeviva [in Poland]

2. Okkam, W. (2002). Izbrannyje proizvedenija. [Selected works]. Kyiv: Editorial URSS [in Ukrainian].

3. Moore Devlin Th. (2022) [Are Programming Languages Technically A Language?]. https://www.babbel.com/en/magazine/are-programming-languages-language [in English].

4. Potapova I. V. (2004). [Umberto Eko about the theory of translation: biographical aspect]. Lviv: Filosofski poshyki [in Ukrainian].

5. White G. (2004). [The Philosophy of Computer Languages]. The Blackwell Guide to the Philosophy of Computing and Information https://sidoli.w.waseda.jp/White_2004_Philosophy_Computer_Language.pdf [in English].

6. Stanford Encyclopedia of Philosophy (2004). [The American dictionary]. New York: The Lambda Calculus https://plato.stanford.edu/entries/lambda-calculus/ [in English].

7. Peters Tim (1999). [«The Python Way» Python-list thread]. https://mail.python.org/pipermail/python-list/1999-June/001951.html; Python Enhancement Proposals, 2004 https://peps.python.org/pep-0020/References [in English].

Yanushevich Iryna Anatolyivna
Candidate of Philosophical Sciences,
Associate Professor at the Department of Philosophy, History and Political Science
Odessa Polytechnic National University
1, Shevchenko Ave., Odesa, Ukraine
orcid.org/0000-0002-1753-5111

Kronberg Elizaveta Volodymyrivna
Student at the Institute of Computer Systems
Odessa Polytechnic National University
1, Shevchenko Ave., Odesa, Ukraine

PROGRAMMING AS A FORM OF LANGUAGE AND PROBLEM OF UNDERSTANDING: THE PHILOSOPHICAL MEANING OF SYNTAX AND SEMANTICS IN PROGRAMMING

The relevance of the article *programming is closely related to the development of modern information technologies, in today's world it is a useful skill for understanding the innovative development of society. The growing popularity of programming as a field and the spread of its influence on technological progress and, as a result, on the everyday life of a person, makes it relevant to consider it from a philosophical point of view, in particular, the analysis of the syntax and semantics of programming as key aspects that determine its philosophical meaning. So, the goal of the article was to analyze the mentioned aspects of programming through the prism of philosophical concepts, which will allow to deepen knowledge and improve understanding of its nature and significance in the modern world.*

Investigation methods. *According to the established research and development of the tasks, in this article has been developed methods of describing, analyzing, searching for similarities and illustrating the analyzed concepts with further references, putting forward hypotheses with their subsequent justification. At the final stage of research, to analyze aspects of programming through the prism of philosophical concepts, a systemic approach is chosen – this is the version conveyed by the parametrical theory of systems.*

The analysis of the given aspects in programming made it possible to draw ***conclusions*** *that programming, as a very young science, needs a philosophical view. Programming problems can be considered as a part of philosophical problems related to scientific and technological progress, but questions, for example, programming languages go far beyond this framework. When programming touches on the problem of content, the programmer faces problems of understanding in the sense that it is interpreted by philosophy – depending on the field of application of interpretation methods adequate to the object of study, in particular in programming.*

And although existing in the present time in object-oriented programming concepts such as polymorphism, abstraction, encapsulation and inheritance, which form the basis of the skills of a successful programmer, have made a significant contribution to the development of theoretical approaches in programming, but in the very approaches to the programming process, so and no attempt was made to systematize methodologically justified methods of programming within the framework of a single theoretical concept, which is impossible to do without going into the philosophical plane of reasoning.

It is not that no special concept of programming has been created, but that no non-trivial philosophical and methodological development of the theory of programming has yet been proposed, which would allow us to consider the programming process as a more or less rationally controlled and generally meaningful procedure. And without it, urgent programming tasks become difficult.

Key words: *linguistics, paradigm, programming language theory, understanding, Zen of Python.*