**Лекція 4.** Конструювання програмного забезпечення як складова програмної інженерії

Термін конструювання програмного забезпечення (software construction) описує детальне створення робочої програмної системи за допомогою комбінації кодування, верифікації (перевірки), модульного тестування (unit testing), інтеграційного тестування і налагодження.

Проектування програмного забезпечення починається, власне, з його конструювання, яке визначає стратегію для його внутрішнього проектування - для етапу програмування. Цей етап виконується без використання мови програмування, але з орієнтацією на певний програмний інструмент розробки ПЗ.

Конструювання ПЗ – створення ПЗ з конструкцій (блоків, операторів, функцій) і його перевірка методами верифікації і тестування. До ***інструментів конструювання ПЗ*** віднесені мови конструювання, програмні методи й інструментальні системи (компілятори, СКБД, генератори звітів, системи керування версіями, конфігурацією, тестуванням й ін.). До ***формальних засобів опису процесу конструювання ПЗ, взаємозв'язків між людиною і комп'ютером з урахуванням середовища оточення*** віднесені структурні діаграми Джексона.

У процесі конструювання програмного виробу здійснюють:

* Функціональну декомпозицію вирішуваного завдання, на основі якої визначається архітектура системи, що представляє задачу;
* Зовнішнє проектування програмного забезпечення , що виражається у формі його зовнішньої взаємодії з користувачем ;
* Проектування бази даних , якщо це необхідно;
* Проектування архітектури програмного забезпечення , тобто визначення безлічі об'єктів або модулів, функціонально пов'язаних з розв'язуваної завданням, включаючи сполучення між ними та вимоги до них.

Конструювання - велика частина процесу розробки ПЗ. Залежно від розміру проекту на конструювання зазвичай іде 30-80% загального часу роботи. Конструювання займає центральне місце в процесі розробки ПЗ. Вимоги до додатка і його архітектура розробляються до етапу конструювання, щоб гарантувати його ефективність. Тестування системи (у строгому сенсі) виконується після конструювання і служить для перевірки його правильності.

Область знань «Конструювання ПЗ (Software Construction)» містить у собі такі розділи:

– зниження складності (Reduction in Complexity),

– попередження відхилень від стилю (Anticipation of Diversity),

– структуризація перевірок (Structuring for Validation),

– використання стандартів (Use of External Standards).

Базову складову професійної діяльності фахівців в галузі програмної інженерії формують вміння та навички конструювання програмного забезпечення, поняття про методи ефективного та оптимального кодування алгоритмів .

Програміст повинен генерувати не просто будь-який код, який працює, а і обов’язково володіти якісним стилем програмування, методами документування, застосовувати методи мінімізації коду, проводити ефективний пошук помилок, зокрема не явних на етапі відладки та вміти якісно тестувати власний програмний продукт.

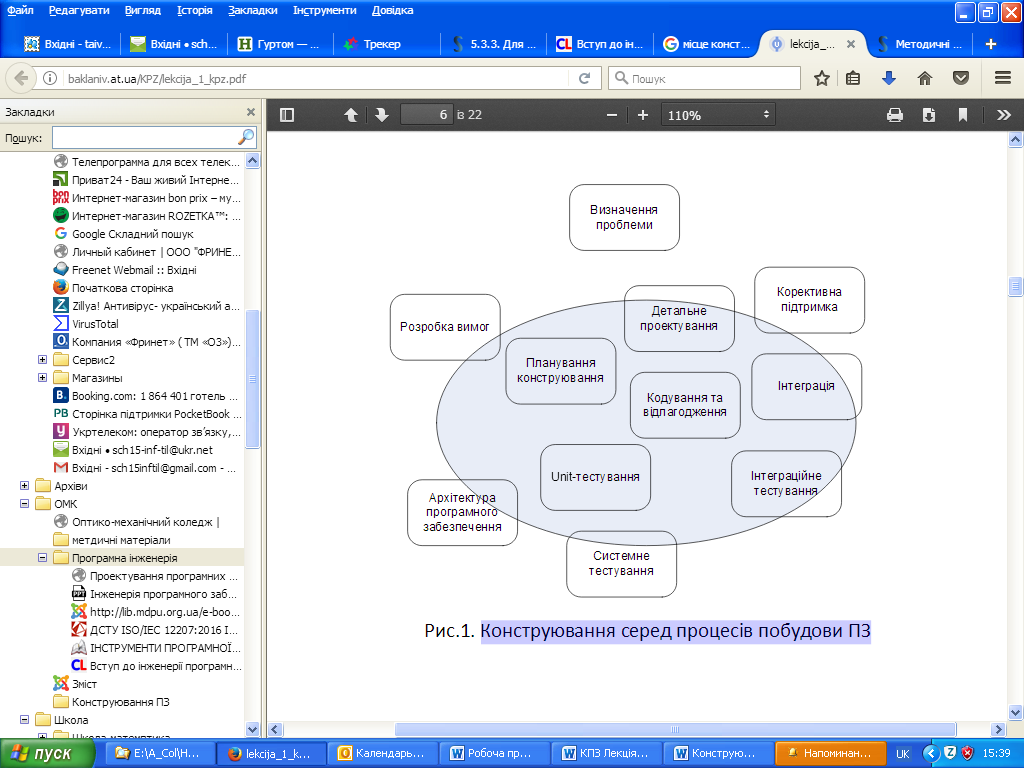
**Основні теоретичні відомості конструювання програмного забезпечення**

Термін КПЗ описує детальне створення робочої програмної системи за допомогою комбінації кодування, верифікації, модульного тестування, інтеграційного тестування і налагодження. Процес конструювання сильно пов'язаний з проектуванням і тестуванням тому конструювання відштовхується від результатів проектування, а тестування означає роботу з результатами конструювання. Ці складові всі пов'язані в один комплекс життєвого циклу ПЗ.

У процесі розробки ПЗ виникають задачі, пов'язані з конструюванням, зокрема, перевірка виконання умов, необхідних для успішного конструювання:

* визначення способів подальшого тестування коду;
* проектування та написання класів та методів;
* створення та присвоєння імен змінних та іменованих констант;
* вибір управляючих структур та організації блоків команд;
* блочне тестування, інтеграційне тестування і відлагодження власного коду;
* взаємний огляд коду та низькорівневих програмних структур членами групи;
* "шліфування" коду шляхом його ретельного форматування та коментування;
* інтеграція програмних компонентів, створених окремо;
* оптимізація коду, яка направлена на підвищення його швидкодії і зниження міри використання ресурсів.

**Конструювання серед процесів побудови ПЗ**



Головними компонентами конструювання є кодування та відлагодження, однак воно включає і детальне проектування, блочне тестування, інтеграційне тестування та інші процеси.

Іноді конструювання називають "кодуванням" або "програмуванням". Кодування в має на увазі механічну трансляцію розробленого плану в команди мови програмування, тоді як конструювання є зовсім не механічним процесом і часто пов’язане з творчістю та аналізом. Сенс слів "конструювання" та "програмування" досить близький.

Дана область знань пов'язана з іншими областями. Найбільш сильний зв'язок існує з проектуванням (Software Design) і тестуванням (Software Testing). Причиною цього є те, що сам по собі процес конструювання програмного забезпечення зачіпає важливі аспекти діяльності з проектування й тестування. Крім того, конструювання відштовхується від результатів проектування, а тестування (у будь-якій своїй формі) передбачає роботу з результатами конструювання. Хоча ряд операцій з проектування детального дизайну може відбуватися до стадії конструювання, великий обсяг такого роду проектних робіт відбувається паралельно з конструюванням або як його частина. Це є сутність зв'язку з областю знань "Проектування програмного забезпечення".

У свою чергу, протягом всієї діяльності з конструювання, інженери використовують модульне і інтеграційне тестування. Таким чином пов'язана дана галузь знань з "Тестуванням програмного забезпечення".

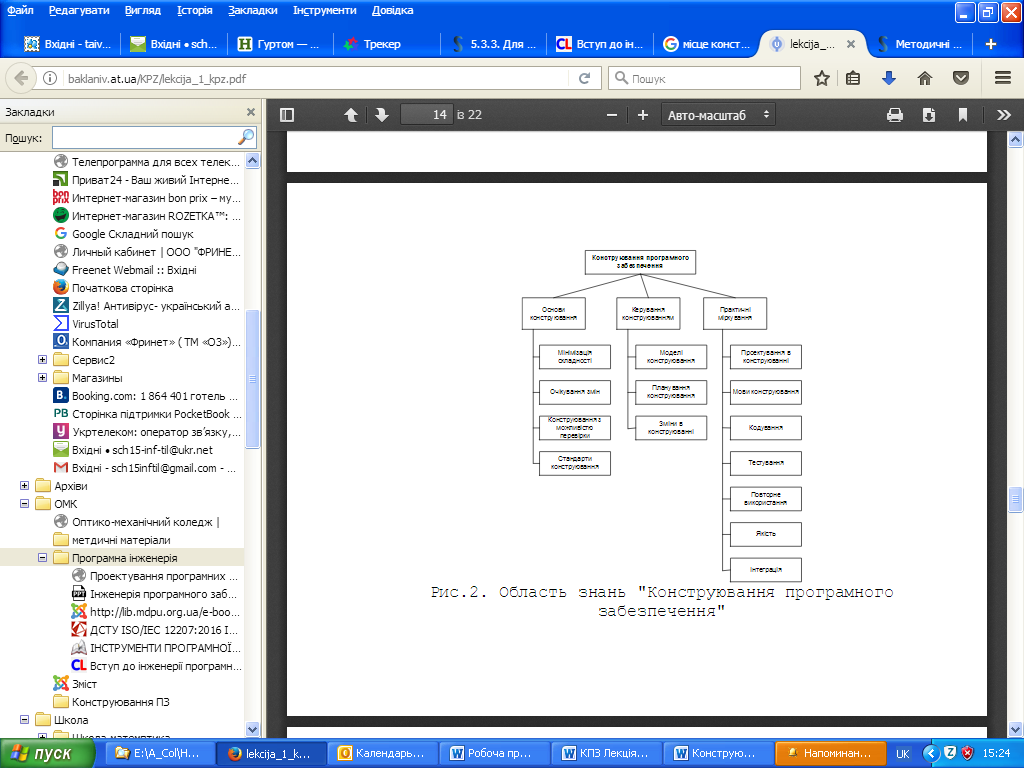
У процесі конструювання звичайно створюється більша частина активів програмного проекту - конфігураційних елементів (configuration items). Тому в реальних проектах просто неможливо розглядати діяльність по конструюванню у відриві від галузі знань "конфігураційного управління" (Software Configuration Management).

Так як конструювання неможливе без використання відповідного інструментарію важливу роль у конструюванні грає область знань "Інструменти і методи програмної інженерії" (Software Engineering Tools and Methods).

Безумовно, питання забезпечення якості значимі для всіх галузей знань і етапів життєвого циклу. У той же час, код є основним результуючим елементом програмного проекту. Таким чином, присутній зв'язок обговорюваних питань з областю знань "Якість програмного забезпечення" (Software Quality).

З пов'язаних дисциплін програмної інженерії (Related Disciplines of Software Engineering) найбільш тісний і природний зв'язок даної галузі знань існує з комп'ютерними науками (computer science). Саме в них, звичайно, розглядаються питання побудови та використання алгоритмів і практик кодування. Нарешті, конструювання стосується і управління проектами (project management), причому, в тій мірі, наскільки діяльність з управління конструюванням важлива для досягнення. З іншого боку, з видів діяльності, що проходять в процесі розробки програмного забезпечення, до конструювання не відносяться: керування процесом розробки, виробки вимог, розробка високорівневої архітектури програми, проектування інтерфейсу користувача, тестування системи і її супроводження – для кожного з цих пунктів є своя наука.

**Фундаментальні основи конструювання програмного забезпечення**



Фундаментальні ***основи конструювання*** програмного забезпечення включають:

* Мінімізація складності
* Очікування змін
* Конструювання з можливістю перевірки
* Стандарти у конструюванні

Перші три концепції застосовуються не тільки до конструювання, але й проектування, і лежать в основі сучасних методологій управління життєвим циклом програмних систем.

***Зниження складності*** – це мінімізація, зменшення і **локалізація** складності конструювання. ***Мінімізація складності*** – це використання у процесі конструювання простих елементів, а також рекомендацій стандартів. Зменшення складності в конструюванні ПЗ досягається шляхом створення простого коду, що легко читається і спрощує тестування, використання низки специфічних технік кодування і підтримкою практик, спрямованих на забезпечення якості в конструюванні.

***Локалізація складності*** – призначена для внесення змін, пов'язаних з виявленими помилками в коді, або коли джерелом помилок є середовище, у якому виконується код.

Потреба у зменшенні складності впливає на всі аспекти конструювання і особливо критична для процесів верифікації (перевірки) і тестування результатів конструювання, тобто самих програмних систем.

Зменшення складності у конструюванні програмного забезпечення досягається іноді на шкоду прагненню зробити його ідеальним (наприклад, з точки зору гнучкості або слідування тим чи іншим уявленням про красу, витонченість коду, вправність тих чи інших прийомів тощо). Це означає "лише" надання більшої значимості читаності коду, простоті тестування, прийнятного рівня продуктивності та задоволенню заданих критеріїв, замість постійного вдосконалення коду, не оглядаючись на терміни, функціональність і інші характеристики та обмеження проекту.

Більшість програмних систем змінюються з плином часу. ***Очікування змін*** є однією з рушійних сил конструювання програмного забезпечення. Програмне забезпечення не є ізольованим від зовнішнього оточення (як системного, так і з точки зору галузі діяльності, для автоматизації задач і проблем якого воно застосовується). Більш того, програмні системи є частиною середовища, що змінюється, і повинні змінюватися разом з нею, а, іноді, і бути джерелом змін самого середовища. Очікування змін підтримується рядом технік кодування.

***Конструювання з можливістю для перевірки*** припускає, що побудова програмних систем повинна вестися таким чином, щоб сама програмна система допомагала вести пошук причин збоїв, будучи прозорою для застосування різних методів перевірки (і, до речі, внесення необхідних змін), як на стадії незалежного тестування (наприклад, інженерами-тестувальниками), так і в процесі операційної діяльності - експлуатації, коли особливо важлива можливість швидкого виявлення та виправлення помилок що виникають.

Серед технік, спрямованих на досягнення такого результату конструювання:

* Огляд, оцінка коду (code review).
* Модульне тестування (unit-testing).
* Структурування коду для і спільно з застосуванням автоматизованих засобів тестування (automated testing).
* Обмежене застосування складних або важких для розуміння мовних структур.

***Стандарти***, які безпосередньо ***застосовуються при конструюванні***, включають:

* Комунікаційні методи (наприклад, стандарти форматів документів і оформлення вмісту)
* Мови програмування і відповідні стилі кодування (наприклад, Java Language Specification, що є частиною стандартної документації JDK - Java Development Kit і Java Style Guide, що пропонує загальний стиль кодування для мови програмування Java)
* Платформи (наприклад, стандарти програмних інтерфейсів для викликів функцій операційного середовища, такі як прикладні програмні інтерфейси платформи Windows - Win32 API, Application Programming Interface або .NET Framework SDK, Software Development Kit)
* Інструменти (не в термінах середовищ розробки, але можливих засобів конструювання - наприклад, UML як один зі стандартів для визначення нотацій для діаграм, що представляють структура коду і його елементів або деяких аспектів поведінки коду).

*Використання зовнішніх стандартів*. Конструювання залежить від зовнішніх стандартів, пов'язаних з мовами програмування, використовуваним інструментальним забезпеченням, технічними інтерфейсами і взаємним впливом Конструювання програмного забезпечення та інших областей знань програмної інженерії (в тому числі, пов'язаних дисциплін, наприклад, управління проектами). Стандарти створюються різними джерелами, наприклад, консорціумом OMG - Object Management Group (зокрема. Стандарти CORBA[[1]](#footnote-1), UML, ...), міжнародними організаціями з стандартизації такими, як ISO / IEC, IEEE, TMF, ..., виробниками платформ, операційних середовищ і т.д. (Наприклад, Microsoft, Sun Microsystems, CISCO, NOKIA, ...), виробниками інструментів, систем управління базами даних та т.п. (Borland, IBM, Microsoft, Sun, Oracle, ...).

Розуміння цього факту дозволяє визначити достатній і повний набір стандартів, які застосовуються у проектній команді або організації в цілому.

*Використання внутрішніх стандартів*. Певні стандарти, угоди та процедури можуть бути також створені усередині організації або навіть проектної команди. Ці стандарти підтримують координацію між певними видами діяльності, групами операцій, мінімізують складність (у тому числі при взаємодії членів проектної групи та за її межами), можуть бути пов'язані з питаннями очікування та обробки змін, ризиків і питаннями конструювання для перевірки та подальшого тестування. У поєднанні з зовнішніми стандартами, внутрішні стандарти покликані визначити загальні правила гри для всіх членів проектної команди, домовившись про терміни, процедури та інших значущих угодах, незалежно від ступеня формалізації процесів конструювання, зокрема, і процесів життєвого циклу, в загальному випадку.

**Управління конструюванням (Managing Construction)**

***Моделі конструювання* (Construction Models)** визначають комплекс операцій, які включають послідовність, результати (наприклад, вихідний код і відповідні unit-тести) та інші аспекти, пов'язані із загальним життєвим циклом розробки. У більшості випадків, моделі конструювання визначаються використовуваним стандартом життєвого циклу, застосовуваними методологіями і практиками. Деякі стандарти життєвого циклу, за своєю природою, є орієнтованими на конструювання - наприклад, екстремальне програмування (XP-eXtreme Programming). Деякі розглядають конструювання в нерозривному зв'язку з проектуванням (в частині моделювання), наприклад, RUP (Rational Unified Process).

Створено безліч моделей розробки програмного забезпечення. Ряд із них більшою мірою сфокусований на конструюванні програмного забезпечення, як такому. Деякі моделі є більш лінійними з точки зору конструювання ПЗ. До них належать, наприклад, Водоспадна (waterfall) і поетапна (staged-delivery) моделі життєвого циклу. Ці моделі розглядають конструювання як діяльність, яка починає проводитися тільки після завершення певних обов'язкових до виконання (prerequisite) робіт, що включають детальне визначення вимог, детальний дизайн і детальне планування. Більш лінійні підходи намагаються підкреслити дії, що передують конструювання (тобто вимоги та дизайн) і створити більш чіткий поділ між такими різними типами діяльності. У таких моделях основним змістом конструювання може бути кодування.

Інші моделі більш ітеративний, до них відносяться - еволюційний прототипування, екстремальне програмування та Scrum[[2]](#footnote-2). Ці підходи сходяться до розгляду конструювання як діяльності, яка ведеться одночасно з іншими видами робіт зі створення програмного забезпечення та перетинаючись з ними (взаємозалежність і вплив один на одного), включаючи визначення вимог, проектування і планування. Ці підходи змішують проектування, кодування і тестування, часто розглядаючи їх комбінацію як конструювання. Відповідно, що саме мається на увазі під "конструюванням" залежить певною мірою від своєї моделі життєвого циклу.

***Планування конструювання (Construction Planning).***

Вибір методу (методології) конструювання є ключовим аспектом для планування конструкторської діяльності, а також необхідних умов її здійснення, визначаючи порядок відповідних операцій та рівень виконання заданих умов перед тим як почнеться конструювання або складові його дії. Наприклад, модульне тестування в ряді методів є частиною робіт, після написання відповідного функціонального коду, в той час, як ряд гнучких (agile) практик, вимагають написання Unit-тестів до того, як пишеться відповідний код, що вимагає тестування.

Використовуваний підхід до конструювання впливає на можливість зменшення (в ідеалі - мінімізації) складності, готовності до змін і конструюванні з можливістю перевірки.

Планування конструкторської діяльності визначає порядок, в якому створюються компоненти та інші активи цієї галузі знань (фази діяльності), проводяться роботи по забезпеченню якості одержуваного програмного забезпечення, розподіляються \* завдання і відповідні ресурси, в тому числі, визначаються призначення / відображення робіт конкретним інженерам-програмістам, членам проектної групи. Все це, звичайно, відбувається, дотримуючись правил, що визначаються використовуваним методом (методологією, практиками і т.п.).

\* Зауважте - не розподіляють, а розподіляються, маючи на увазі процес, що призводить до забезпечення явного зв'язку між завданням і ресурсами. У нечітко регламентованих та неформальних методах, таких, як екстремальне програмування (XP), члени проектної групи самі приймають на себе відповідальність за вирішення певних завдань, а "володіння" кодом є спільним на основі співробітництва, як одного з ключових принципів роботи проектної команди.

***Зміни в конструюванні та вимірювання в конструюванні (Construction Measurement)***

Більша частина результатів, та й самої діяльності з конструювання програмного забезпечення, може бути виміряна, в тому числі - кількісно. Це і вихідний розроблений код, і модифікований обсяг коду, і ступінь повторного використання, і багато інших характеристик. Ці виміри, або як їх ще прийнято називати - результати аудиту коду і метрики коду, несуть велику користь як для оцінки ризиків і якості (призводять до відповідних операціях зі зниження ризиків та підвищення якості), а також, для керування конструюванням і програмними проектами, в цілому. Вони дозволяють передбачити (наприклад, на основі оцінки результатів попередніх проектів) тривалість робіт, вартість окремих завдань, ймовірність виникнення дефектів проти заданих параметрів прийнятної якості?

Код є одним з найбільш чітко детермінованих активів проекту (поступово такими стають і моделі, що будуються на основі структур метаданих, і тісно пов'язані з кодом - наприклад, UML). Код є і самим носіям необхідної функціональності. Відповідно, застосування вимірювань щодо коду стає тим інструментом, який впливає і на управління і на сам код.

Важливим є ***рефакторинг коду*** як метод його реструктурування, покликаний без зміни змісту (тобто функціональності та функціональної цілісності) забезпечити вирішення завдань мінімізації складності, готовності до змін (гнучкості), прозорості документування і багатьох інших актуальних аспектів конструювання. Застосування вимірювань, зокрема, метрик, дозволяє визначити необхідність внесення таких змін, проведення рефакторинга. Якщо застосовується рефакторинг, але не застосовуються метрики - у переважній більшості випадків, це негативно впливає на проект.

**Практичні міркування (Practical Considerations)**

Конструювання - діяльність, в рамках якої програмне забезпечення наводиться до угоди з довільними (іноді - хаотичними) обмеженнями реального світу, які вимагають від програмного забезпечення точного дотримання їх. Наближаючись до обмежень реального світу, конструювання (ведеться на основі практичних міркувань і технік.

***Проектування в конструюванні (Construction Design)***

Деякі проекти передбачають більший обсяг робіт з проектування саме на стадії конструювання; інші проекти явно виділяють проектну діяльність у формі фази дизайну. Практично завжди на стадії конструювання доводиться займатися і питаннями детального дизайну системи для встановлення обмежень, пов‘язаних з конкретними проблемами, вирішення яких має бути забезпечене програмною системою.

***Мови конструювання*** включають всі форми комунікацій, за допомогою яких людина може задати рішення проблеми, що виконується на комп'ютері.

Найпростіший тип мов конструювання - *конфігураційна* *мова* (configuration language), що дозволяє задавати параметри виконання програмної системи.

*Інструментальний мова* (toolkit language) - мова конструювання з повторно-використовуваних елементів; зазвичай будується як сценарний мова (script), що здійснюється у відповідному середовищі.

*Мова програмування* (programming language) - найбільш гнучкий тип мов конструювання. Містить мінімальний обсяг інформації про конкретні областях програми, але процесі розробки, вимагаючи найбільше (порівняно з іншими типами мов конструювання) зусиль на вивчення і напрацювання досвіду для ефективного застосування при вирішенні конкретних завдань.

Існує три основних види нотацій використовуються при визначенні мов програмування:

• лінгвістична

• формальна

• візуальна

Лінгвістичні нотації характеризуються, зокрема, використанням рядків тексту, що містять спеціалізовані "слова", які представляють складні програмні конструкції, і комбіновані в шаблони, що нагадують пропозиції, побудовані відповідно до визначеного синтаксисом. У разі коректного використання таких нотацій, кожна отримується рядок має суворої смисловим навантаженням (семантикою), що забезпечує інтуїтивне розуміння того, що відбуватиметься коли буде виконуватися програмне забезпечення, побудоване з використанням такої мови конструювання.

Формальні нотації (наприклад, нота́ція Бе́куса—Нау́ра) є менш інтуїтивними, ніж лінгвістичні, і часто базуються на точних формальних (математичних) визначеннях. Формальні нотації конструкцій і формальні методи є ядром практично всіх форм системного програмування, точніше - поведінки систем в часі. Такі нотації забезпечують найбільшу готовність одержуваного коду до тестування, що набагато важливіше, ніж просто можливість відображення на звичайний людський мову. Формальні конструкції також використовують точний метод визначення комбінацій застосовуваних символів, що дозволяє уникнути неоднозначностей, властивих конструкціям природних мов.

Візуальні нотації найменш пов'язані з текстово-орієнтованими підходами, припускаючи безпосередню інтерпретацію візуальних конструкцій в процесі виконання описуваної логіки. При цьому логіка у візуальних нотаціях задається розташуванням відповідних візуальних сутностей, відповідальних за ті чи інші операції і структури даних.

Використання візуальних конструкцій обмежено складністю візуального представлення складних висловів та тверджень тільки за рахунок переміщення візуальних сутностей на діаграмі (візуальному представленні). Однак, візуальна нотація може грати роль досить потужного інструменту, коли застосовується в тих завданнях програмування, де необхідна побудова інтерфейсу для програм, чия логіка. Деталізоване поведінка визначено раніше.

Сьогоднішні роботи (і їх стан) в області архітектур і додатків, керованих моделями, в першу чергу - OMG MDA (Model-Driven Architecture www.omg.org / mda) / UML (Unified Modeling Language www.omg.org / uml), Microsoft DSL (Domain-Specific Language), спрямовані на те, щоб використовувати ту чи іншу візуальну нотацію, що базується на мета-моделях, як інструмент, що застосовується і для визначення функціональної логіки системи. В термінах, пропонованих SWEBOK, вони відносяться саме до візуальних нотаціям, так як припускають однозначну інтерпретацію візуального представлення у вигляді тексту і навпаки. Крім того, історично ці нотації визначалися спочатку як нотації візуального представлення функціональності і вже надалі ці візуальні уявлення були відображені на рівні відповідних мета-моделей (хоча це більшою мірою вірно для UML). Інша область стандартів, спрямованих на застосування візуальних нотацій для опису функціональності - Business Process Management Notation (BPMN - www.omg.org / bpmn) і пов'язаний з нею мову Business Process Execution Language, побудований на базі XML. Таким чином, район обгрунтованого застосування візуальних нотацій для конструювання програмних систем якісно розширитися і, не виключено, ми станемо свідками de facto-формування нової категорії нотацій, угод і змішаних типів мовних засобів, призначених для конструювання програмного забезпечення як природного продовження проектування.

***Кодування* (Coding).** Практика конструювання програмного забезпечення показує активне застосування таких міркувань і технік:

• техніки створення легко зрозумілого вихідного коду на основі використання угод про іменуванні, форматування і структурування коду;

• використання класів, що перераховуються типів, змінних, іменованих констант та інших сутностей виразів;

• організація вихідного тексту (у вирази, шаблони, класи, пакети / модулі та інші структури);

• використання структур управління;

• обробка помилкових умов і виняткових ситуацій;

• запобігання можливих проломів в безпеці (наприклад, переповнення буфера або вихід за межі індексу в масиві);

• використання ресурсів на основі застосування механізмів виключення (з розгляду) та порядку доступу до паралельно використовуваних ресурсів (наприклад, на основі блокування даних, використання потоків і їх синхронізації і т.п.);

• документування коду;

• тонка "настройка" коду;

• рефакторинг

***Тестування в конструюванні (Construction Testing).***

При конструюванні використовуються дві форми тестування, проведеного інженерами, безпосередньо створюють вихідний код:

• модульне тестування (unit testing)

• інтеграційне тестування (integration testing)

Головна мета тестування в конструюванні зменшити часовий розрив між моментом прояви помилок, що є в коді, і моментом їх виявлення. У багатьох випадках, тестування в конструюванні проводиться після того, як код написаний. У ряді випадків, тести (що зазначалося раніше, на прикладі XP) пишуться до того, як створюється код. Тестування в конструюванні зазвичай не включає системного тестування, тестування навантаження, usability-тестування (оцінки прозорості використання) та інших видів тестової діяльності.

IEEE опублікував два стандарти, присвячених даній темі:

• IEEE Std 829-1998: "IEEE Standard for Software Test Documentation"

• IEEE Std 1008-1987: "IEEE Standard for Software Unit Testing"

**Повторне використання коду** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *code reuse*) — методологія проектування комп'ютерних та інших систем, що полягає в тому, що система (комп'ютерна програма, програмний модуль) частково або повністю повинна складатися з частин, написаних раніше компонентів і/або частин іншої системи, і ці компоненти повинні застосовуватися більше одного разу (якщо не в рамках одного проекту, то хоча б різних). Повторне використання — основна методологія, яка застосовується для скорочення трудовитрат при розробці складних систем.

Найпоширеніший випадок повторного використання коду — бібліотеки програм. Бібліотеки надають загальну достатньо універсальну функціональність, яка покриває обрану предметну область. Приклади: бібліотека функцій для роботи з комплексними числами, бібліотека функцій для роботи з 3D-графікою, бібліотека для використання протоколу TCP/IP, бібліотека для роботи з базами даних. Розробники нової програми можуть використовувати існуючі бібліотеки для вирішення своїх завдань і не «винаходити велосипеди».

***Якість*** програмного забезпечення – це відповідність реальним вимогам, явним і неявним.

З точки зору замовника або користувача продукту: *Якість – це придатність до використання***.** Чи робить даний продукт те, що мені потрібно? Чи спрощує роботу? Чи можу я його використати так, як мені потрібно?

Точка зору розробника: *Якість – це відповідність специфікованим і забраним вимогам.* Чи робить даний продукт усе те, що вказано у вимогах?

Проблема в тому, що специфіковані і забрані вимоги – це зазвичай частина реальних вимог і очікувань замовника.

Стандарт **ISO 9126** пропонує використовувати для опису внутрішнього та зовнішнього якості ПЗ багаторівневу модель. На верхньому рівні виділено 6 основних характеристик якості ПЗ. Кожна характеристика описується за допомогою кількох вхідних у неї атрибутів. Для кожного атрибута визначається набір метрик, що дозволяють його оцінити. Множина характеристик і атрибутів якості згідно з ISO 9126.

Визначення цих характеристик і атрибутів за стандартом ISO 9126:2001:

* **Функціональність (functionality)**. Здатність ПЗ в певних умовах вирішувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно вирішує.
  + *Функціональна придатність* (suitability). Здатність вирішувати потрібний набір задач.
  + *Точність* (accuracy). Здатність видавати потрібні результати.
  + *Здатність до взаємодії* (interoperability). Здатність взаємодіяти з потрібним набором інших систем.
  + *Відповідність стандартам і правилам* (compliance). Відповідність ПЗ наявним індустріальним стандартам, нормативним і законодавчим актам, іншим регулюючим нормам.
  + *Захищеність* (security). Здатність запобігати неавторизованому, тобто без вказівки особи, що намагається його здійснити, і недозволеному доступу до даних і програм.
* **Надійність (reliability)**. Здатність ПЗ підтримувати визначену працездатність у заданих умовах.
  + *Зрілість, завершеність* (maturity). Величина, зворотна частоті відмов ПЗ. Звичайно виміряється середнім часом роботи без збоїв і величиною, зворотною імовірності виникнення відмови за даний період часу.
  + *Стійкість до відмов* (fault tolerance). Здатність підтримувати заданий рівень працездатності при відмовах і порушеннях правил взаємодії з середовищем.
  + *Здатність до відновлення* (recoverability). Здатність відновлювати визначений рівень працездатності та цілісність даних після відмови, необхідні для цього час і ресурси.
  + *Відповідність стандартам надійності* (reliability compliance). Цей атрибут доданий в 2001 році.
* **Зручність використання (usability) або практичність**. Здатність ПЗ бути зручним у навчанні та використанні, а також привабливим для користувачів.
  + *Зрозумілість* (understandability). Показник, зворотний до зусиль, які затрачаються користувачами на сприйняття основних понять ПЗ та усвідомлення їх застосовності для розв'язання своїх задач.
  + *Зручність навчання* (learnability). Показник, зворотний зусиллям, затрачуваним користувачами на навчання роботі з ПЗ.
  + *Зручність роботи* (operability). Показник, зворотний зусиллям, що вживається користувачами для розв'язання своїх задач за допомогою ПЗ.
  + *Привабливість* (attractiveness). Здатність ПЗ бути привабливим для користувачів. Цей атрибут доданий в 2001 році.
  + Відповідність стандартам зручності використання (usability compliance). Цей атрибут доданий в 2001 році.
* **Продуктивність (efficiency) або ефективність**. Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділюваного для цього ресурсам. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до затрачуваних на це ресурсів усіх типів.
  + *Часова ефективність* (time behaviour). Здатність ПЗ видавати очікувані результати, а також забезпечувати передачу необхідного об'єму даних за відведений час.
  + *Ефективність використання ресурсів* (resource utilisation). Здатність вирішувати потрібні задачі з використанням визначених об'ємів ресурсів визначених видів. Маються на увазі такі ресурси, як оперативна й довгострокова пам'ять, мережні з'єднання, пристрої вводу та виводу та ін.
  + *Відповідність стандартам продуктивності* (efficiency compliance). Цей атрибут доданий в 2001 році.
* **Зручність супроводу (maintainability)**. Зручність проведення всіх видів діяльності, пов'язаних із супроводом програм.
  + *Аналізованість* (analyzability) або зручність проведення аналізу. Зручність проведення аналізу помилок, дефектів і недоліків, а також зручність аналізу необхідності змін і їх можливих наслідків.
  + *Зручність внесення змін* (changeability). Показник, зворотний трудовитратам на виконання необхідних змін.
  + *Стабільність* (stability). Показник, зворотний ризику виникнення несподіваних ефектів при внесенні необхідних змін.
  + *Зручність перевірки* (testability). Показник, зворотний трудовитратам на проведення тестування і інших видів перевірки того, що внесені зміни привели до потрібних результатів.
  + *Відповідність стандартам зручності супроводу* (maintainability compliance). Цей атрибут доданий в 2001 році.
* **Переносимість (portability)**. Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні й програмні аспекти оточення.
  + *Адаптованість* (adaptability). Здатність ПЗ пристосовуватися різним оточенням без проведення для цього дій, крім заздалегідь передбачених.
  + *Зручність установки* (installability). Здатність ПЗ бути встановленим або розгорнутим у визначеному оточенні.
  + *Здатність до співіснування* (coexistence). Здатність ПЗ співіснувати з іншими програмами у загальному оточенні, ділячи з ними ресурси.
  + *Зручність заміни* (replaceability) іншого ПЗ даним. Можливість застосування даного ПЗ замість інших програмних систем для вирішення тих же задач у певному оточенні.
  + *Відповідність стандартам переносимості* (portability compliance). Цей атрибут доданий в 2001 році.

***Інтеграція***  - це сукупність дій, щодо отримання комплексного вирішення низки інформаційних завдань, які спрямовані на отримання синергетичного ефекту. Є декілька складових системної інтеграції у цій сфері:

* *інтеграція програмних застосунків* — діяльність, пов'язана із впровадженням спеціалізованих програмних рішень (галузевих);
* *інтеграція даних* — діяльність, що має на меті оптимальну організацію бази даних, при якій реалізовано всі необхідні взаємозв'язки між елементами даних, але база не містить повторів і зайвих елементів (так як по мірі використання бази даних вона має тенденцію розосереджуватись)
* *мережева інтеграція* — діяльність, пов'язана із створення локальних і глобальних мереж, структурованих кабельних систем, впровадження оптоволоконних технологій передачі даних і т.д.;

*Системна інтеграція* — діяльність, пов'язана із встановленням та налагодженням операційних систем, баз даних, офісних за стосунків, засобів зв'язку, структурованих кабельних систем і активних мережевих пристроїв, пристроїв зберігання даних, підключення до Інтернету, контролю доступу, живлення, системи аварійної сигналізації і т.д.

***Контрольні запитання для самоперевірки***.

1. Чи можна вважати конструювання ПЗ складовою програмної інженерії?
2. В чому полягає конструювання ПЗ?
3. Які процеси побудови ПЗ відносяться до конструювання ПЗ?
4. В чому полягає зниження складності ПЗ?
5. Які стандарти використовуються при конструюванні ПЗ?
6. В чому полягає планування конструювання ПЗ?
7. Визначте основні характеристики якості ПЗ.

**Рекомендована література**

1. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
2. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
3. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

1. **CORBA** (*Common Object Request Broker Architecture* — загальна архітектура брокера об'єктних запитів) — це запропонований [консорціумом OMG](https://uk.wikipedia.org/wiki/Object_Management_Group) [технологічний стандарт](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82) розробки розподілених [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA). Завдання CORBA — інтегрувати [розподілені системи](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D1%96_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8), дати можливість програмам, що написані різними [мовами](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) та працюють у різних вузлах мережі, взаємодіяти одна з одною так само просто, наче вони знаходяться в адресному просторі одного процесу. [↑](#footnote-ref-1)
2. **Scrum** (чит.як *скрам*) — підхід [управління проектами](https://uk.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8) для [гнучкої розробки програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%93%D0%BD%D1%83%D1%87%D0%BA%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F). Scrum чітко робить акцент на якісному контролі процесу розробки. [↑](#footnote-ref-2)