**Лекція 01. Методи доведення правильності програм. Верифікація і валідація програм**

Сучасні напрями перевірки правильності програм – це формальні специфікації, методи доведення, верифікація і тестування. Формальні методи тісно пов'язані з математичною специфікацією, верифікацією і доведення правильності програм. Ці методи містять математичну символіку, формальну нотацію і апарат виведення. Правила доведення є громіздкими і тому на практиці не часто використовуються рядовими програмістами. Проте з теоретичної точки зору вони слугують розвитку логіки застосування математичного методу індукції в процесі перевірці правильності програм.

**Формальні специфікації** з'явилися у програмуванні в 70-х роках минулого сторіччя. Вони подібні методам програмування і надають засоби, що полегшують опис міркування у математичній нотації про властивості і особливості програм. Під ***специфікацією***розуміють формальний опис функцій і даних програм, якими ці функції оперують.

На формальних специфікаціях базуються методи доведення програм, які були започатковані працями з теорії алгоритмів А.А. Маркова, А.А. Ляпунова, схемами Ю.І.Янова та формальними нотаціями опису взаємодіючих процесів К.А. Хоара і ін. Для перевірки формальної специфікації програми застосовують математичний апарат для опису правильного розв’язку деякої задачі, для якої вона розроблена. Разом з специфікацією розробляються ***додаткові аксіоми***, твердження про опис операторів і умов, так звані попередні умови або передумови, і постумови, що визначають заключні правила одержання правильного результату.

**Доведення програм** проводиться за допомогою *тверджень*, що складаються у формальній мові і слугують для перевірки правильності програми в заданих її точках. Набір тверджень, перед- і постумов використовується для перевірки отриманого результату у деякій точці програми. Якщо твердження відповідає скінченому оператору програми, то за допомогою постумови робиться остаточний висновок про часткову або повну правильність роботи програм.

**Верифікація і валідація –** це методи забезпечення перевірки й аналізу правильності виконання заданих функцій програми відповідно до заданих вимог замовника до них і системи у цілому.

**Тестування** – це метод виявлення помилок у програмній системі шляхом виконання вихідного коду на тестових даних, збирання робочих характеристик у динаміці виконання в конкретному операційному середовищі, виявлення різних помилок, дефектів, відмов і збоїв, викликаних нерегулярними, аномальними ситуаціями або аварійним припиненням роботи системи.

Теоретичні засоби реалізуються як процеси програмування і перевірки правильності програмного продукту. Процеси верифікації, валідації і тестування програмних систем регламентовані стандартом ISO/IEC–12207 з життєвого циклу програмних систем.

Наведені методи доведення, верифікації і тестування при перевірці правильності програм кваліфікуються такими загальними поняттями і діями.

***Доведення та верифікація***коректності (правильності) виконуються за формальною специфікацією програми та за допомогою тверджень, передумов (обмежень вхідних параметрів програми) і постумов (обмежень вихідних параметрів програми), які утворюють незалежну від програми частину механізму її доведення. Ця частина специфікується, як правило, на тій же мові, що і програма. У ній застосовуються математичні операції (диз’юнкції, кон’юнкції, імплікації тощо), квантори існування і загальності та інші.

***Передумова***– це обмеження на сукупність вхідних параметрів і ***постумова*** як обмеження на вихідні параметри. Передумова і постумова задаються предикатами[[1]](#footnote-1), результатом яких є булева величина (true/false). Передумова істинна тоді, коли вхідні параметри входять в область припустимих значень даної функції. Постумова істинна тоді, коли сукупність значень задовольняє вимоги, щодо формального визначення критерію правильності одержання результату.

***Твердження***формулюються на формальній математичній мові у вигляді додаткової доказової частини, що перевіряє правильність виконання програми в початковій, проміжній або кінцевій точках.

***Постумова***– це обмеження з умов про кінцевий результат програми, відповідно до якого формулюється висновок про правильне завершення цієї програми.

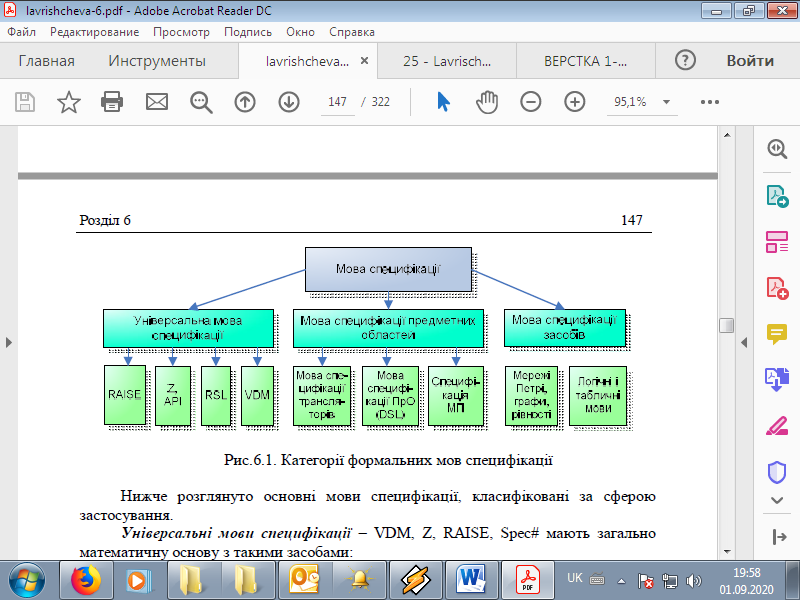
Під ***доведенням***часткової правильності розуміють перевірку виконання програми за допомогою тверджень, які описують те, що повинна одержати ця програма, коли закінчиться її виконання відповідно до умов заключного твердження. Повністю правильною програмою щодо її опису і заданих тверджень буде така програма, яка частково правильна і її виконання завершується при *всіх* даних, що відповідають їй.

Мови формальної специфікації, які використовуються для формального опису властивостей виконання програм шляхом завдання тверджень та перед і постумов, є мовами вищого рівня щодо мов алгоритмічного програмування, як і можуть використовуватися для опису програми, для якої створюється доказ.

У загальному випадку *формальна специфікація програми* – це однозначний специфікований опис програми за допомогою математичних понять, термінів, правил синтаксису і семантики формальної мови.

Опис деякої задачі являє собою сукупність її формальної специфікації та необхідних для доведення аксіом, тверджень, перед- і постумов та інших формалізмів. Всі ці описи при реалізації вимагають не систему програмування з мовою програмування, а спеціальний програмно реалізований математично орієнтований апарат доведення програм, зокрема інтерпретатори або метасистеми.

Основні мови специфікації, класифіковані за сферою застосування таким чином

.

***Універсальні мови специфікації***  (VDM, Z, RAISE, Spec#) мають загально математичну основу з такими засобами:

1) логіки першого порядку, включаючи квантори;

2) арифметичні операції;

3) множини і операції над множинами;

4) описи послідовностей (кортежів, списків) і операції над ними;

5) описи функцій і операцій над ними;

6) описи деревоподібних структур;

7) засоби побудови моделей областей;

8) процедурні засоби мов програмування (оператори присвоювання, циклу, вибору, виходу);

9) операції композиції, аргументами і результатами яких можуть бути функції, вирази, оператори;

10) механізм конструювання нових структур даних.

***Мови специфікації предметних областей (доменів) у програмуванні***:

1) специфікації доменів;

2) описи взаємодій і паралельного виконання;

3) специфікації мов програмування і трансляторів;

4) специфікації баз даних і знань;

5) специфікації пакетів прикладних програм тощо.

*Мови специфікації специфіки доменів* DSL (Domain Specific Language) призначені для формалізованого опису задач в термінах предметної області, що підлягає моделюванню. Ці мови можна підрозділити на зовнішні і внутрішні. Зовнішні мови (наприклад, UML) за рівнем вищі мов програмування і відповідають, наприклад, предметно-орієнтованої мові DSL, яка використовується для подання абстрактних понять і задач предметної області. Їхній опис трансформується до понять деякої внутрішній мові або мови програмування спеціальними генераторами або текстовими редакторами. Внутрішні мови – мови опису специфічних задач обмеженим синтаксисом і семантикою потребують препроцесорів для перебудови цього опису до базової мови програмування.

Специфікації опису взаємодій і паралельного виконання окремих процесів систем предметної області також добре подаються мовами DSL, наприклад, подібно – діаграм UML.

*Мови програмування предметної області*, доповнені засобами і механізмами технологій. ***Метапрограмування*** є ефективним засобом автоматизації специфікацій розроблених програм і в даний час знаходять широке застосування у галузі інформаційних технологій.

Формальні мови специфікації мов програмування спочатку застосовувалися при розробленні трансляторів. Так зазвичай синтаксис мови програмування описувався у формі Бекуса–Наура.[[2]](#footnote-2) Такого типу мови є метамовами.

*Мови специфікації з орієнтацією на засоби програмування* базуються на рівностях і підстановках з операційною семантикою (Лісп, Рефал); логічні мови; мови операцій (АPL) над послідовностями і матрицями; табличні мови; мережі, графи та ін. Мова логіки предикатів використовується для запису передумов і постумов, інваріантів і процесу верифікації (наприклад, Пролог).

Можлива класифікація специфікацій за *способом виконання*:

– виконувана (executable), яка припускають розроблення прототипів систем для досягнення встановленої мети;

– алгебраїчна (algebraic), яка містить у собі механізми опису аксіом і тверджень, які призначені для доведення специфікованих програм;

– сценарна (use case or scenarios), яка дозволяє описувати різні способи можливого застосування системи (UML)

– в обмеженнях (constraints), яку використовують перед- і постумови для опису даних, операцій, інваріантів даних програм, що доводяться.

**Методи доведення правильності програм**

Традиційні методи аналізу ПЗ включають і методи доведення правильності програм. Концепція, покладена в основу цього напрямку, була запропонована в 1960-х рр. в роботах П. Наура і Р. Флойда, в яких сформульована ідея приписування точці програми так званого індуктивного (або проміжного) затвердження і зазначена можливість доказу часткової правильності програми (тобто відповідності один одному її передумови і постумови), побудованого на встановленні узгодженості індуктивних тверджень.

Фундаментальний внесок у теорію верифікації в середині 1970-х рр. внесли Ч. Хоор, який висловив ідеї проведення докази часткової правильності програми у вигляді виведення в деякій логічній системі, і Е. Дейкстра, який ввів поняття «слабка передумова», що дозволяє одночасно як відповідність один одному *передумови* і *постумови,* так і *завершуваності.*

Методи доведення правильності програм принесли певну користь програмування. Було відзначено, що ці методи вказують спосіб міркування про хід виконання програм, дають зручну систему коментування програм і встановлюють взаємозв'язки між конструкціями мов програмування і їх семантикою. Якщо прийняти більш широке тлумачення терміну «аналіз програм», маючи на увазі доказ різноманітних властивостей програм або доведення теорем про програми, то цінність методів аналізу стане більш ясною. Зокрема, можна досліджувати характер зміни вихідних значень програми, кількість операцій при виконанні програми, наявність зациклення, незадіяних ділянок програми. Таким чином, в деяких окремих випадках методи верифікації можуть застосовуватися і для виявлення програмних дефектів, що можуть бути доказані.

Формальне доказ у вигляді виведення в деякій логічній системі цілком надійно, але самі докази виявляються дуже довгими і часто неозорими.

**Базові методи доказу.**

Розглянемо найбільш поширені методи доказу правильності програм.

*Метод Флойда* заснований на визначенні умов для вхідних і вихідних даних і полягає у виборі контрольних точок у програмі, яка доводиться, таким чином, щоб шлях проходження перетинав хоч б одну контрольну точку. Для цих точок формулюються твердження про стан і значення змінних у них (для циклів ці твердження повинні бути істинними при кожному проходженні циклу – інваріанта).

Кожна точка розглядається для індуктивного затвердження того, що формула залишається істинною при поверненні програми в цю точку, і залежить не тільки від вхідних і вихідних даних, але і від значень проміжних змінних. На основі індуктивних тверджень[[3]](#footnote-3) і умов на аргументи програми створюються твердження з умовами перевірки правильності цієї програми в окремих її точках. Для кожного шляху програми між двома точками встановлюється перевірка на відповідність умов правильності і визначається істинність цих умов при успішному завершенні програми на даних, що задовольняють вхідним умовам.

*Метод Хоара* – це вдосконалений метод Флойда, заснований на аксіоматичному описі семантики мови програмування початкових програм. Кожна аксіома описує зміну значень змінних за допомогою операторів цієї мови. Формалізація операторів переходу і викликів процедур забезпечується за допомогою правил виводу, що містять індуктивні вислови для кожної точки і функції початкової програми.

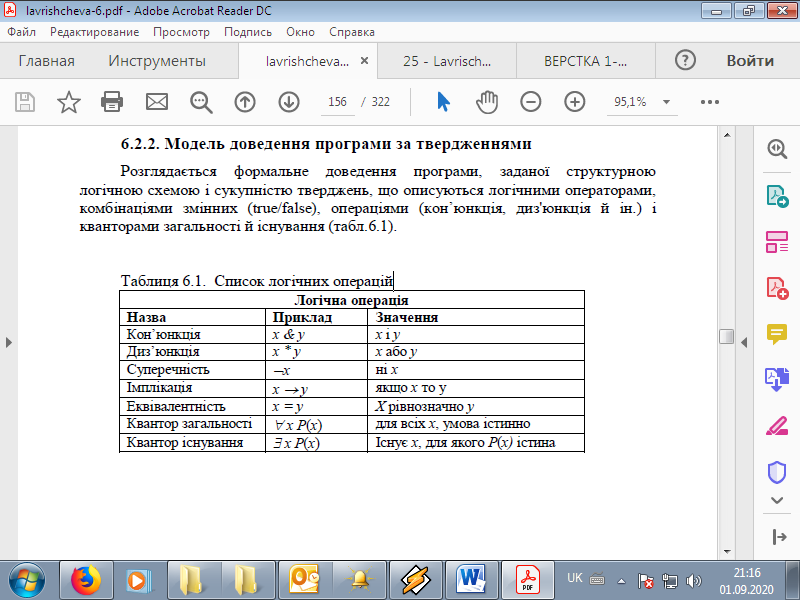
*Метод Маккарті* полягає у структурній перевірці функцій, що працюють над структурними типами даних, структур даних і шляхів переходу під час символьного виконання програм. Ця техніка включає моделювання виконання коду з використанням символів для змінних даних. Тестова програма має вхідний стан, дані і умови її виконання.

Виконувана програма розглядається як серія змін станів. Саме останній стан програми вважається вихідним станом і, якщо він одержаний, програма вважається правильною. Даний метод забезпечує високу якість початкового коду.

*Метод Дейкстри* пропонує два підходи до доказу правильності програм. Перший підхід заснований на моделі обчислень, що оперує з історіями результатів обчислень програми, аналізом шляхів проходження і правил оброблення великого об'єму інформації. Другий підхід базується на формальному дослідженні тексту програми за допомогою предикатів першого порядку[[4]](#footnote-4). У процесі виконання програма одержує деякий стан, який запам'ятовується для подальших порівнянь.

Основу методу складає математична індукція, абстрактний опис програми і її обчислення. За допомогою цього методу можна довести істинність деякого припущення *Р(n)* залежно від параметра *n* для всіх n ≥ n*0*, і тим самим довести випадок *Р (n0)*. Виходячи з істинності *Р(n)* для будь-якого значення *n*, доводиться *Р(n+1)*, що достатньо для доказу істинності *Р(n)* для всіх *n* ≥ *n0.*

Для формального доведення програми, заданої структурною логічною схемою і сукупністю тверджень, що описуються логічними операторами, комбінаціями змінних (true/false), операціями (кон’юнкція, диз'юнкція й ін.) і кванторами загальності й існування використовується такий список логічних операцій.



**Верифікація та валідація** **програм**

Верифікація та валідація ( *verification and validation* - V & *V)* призначені для аналізу, перевірки правильності виконання та відповідності ПЗ специфікаціям і вимогам замовника. Дані методи перевірки правильності програм і систем відповідно означають:

* **верифікаці**я - це перевірка правильності створення системи відповідно до її специфікації;
* **валідація** - це перевірка правильності виконання заданих вимог до системи.

Верифікація допомагає зробити висновок про коректність створеної системи після завершення її проектування і розробки. Валідація дозволяє встановити здійсненність заданих вимог і включає в себе ряд дій для отримання правильних програм і систем, а саме:

* планування процедур перевірки і контролю проектних рішень і вимог;
* забезпечення рівня автоматизації проектування програм CASE- засобами;
* перевірка правильності функціонування програм методами тестування на наборах цільових тестів;
* адаптація продукту до операційному середовищі та ін.

Валідація виконує ці дії шляхом перегляду і інспекції специфікацій і результатів проектування на етапах ЖЦ для підтвердження того, що є коректна реалізація початкових вимог і виконані задані умови і обмеження. До завдань перевірки та затвердження входять перевірки повноти, несуперечності і однозначності специфікації вимог і правильності виконання функцій системи.

Верифікації та валідації піддаються:

* основні компоненти системи;
* інтерфейси компонентів (програмні, технічні та інформаційні) і взаємодії об'єктів (протоколи і повідомлення), що забезпечують виконання системи в розподілених середовищах;
* засоби доступу до БД і файлів (транзакції і повідомлення) і перевірка засобів захисту від несанкціонованого доступу до даних різних користувачів;
* документація до ПЗ і до системи в цілому;
* тести, тестові процедури і вхідні дані.

Іншими словами, основними систематичними методами правильності програм є:

* *верифікація* компонентів програмної системи (ПС) та валідація специфікації вимог;
* *інспектування ПС* для встановлення відповідності програми заданим специфікаціями;
* *тестування* вихідного коду ПС на тестових даних в конкретній операційному середовищі для виявлення помилок і дефектів, викликаних різними недоробками, аномальними ситуаціями, збоями устаткування або аварійним припиненням роботи системи.

Стандарти ISO / IEC 3918-99 і 12207 включають в себе процеси верифікації та валідації. Для них визначено цілі, завдання та дії по перевірці правильності створюваного продукту (включаючи робочі, проміжні продукти) на етапах ЖЦ і відповідності його вимогам.

Основне завдання процесів верифікації та валідації полягає в тому, щоб *перевірити і підтвердити*, що кінцевий програмний продукт (ПП) відповідає призначенню і задовольняє вимогам замовника. Ці процеси дозволяють виявити помилки в робочих продуктах етапів ЖЦ, без з'ясування причин їх появи, а також встановити правильність ПП щодо його специфікації.

Ці процеси взаємопов'язані і визначаються одним терміном - «верифікація та валідація» (V & V 7 ).

При верифікації здійснюється:

* перевірка правильності перекладу окремих компонентів у вихідний код, а також описів інтерфейсів шляхом трасування взаємозв'язків компонентів відповідно до заданих вимог замовника;
* аналіз правильності доступу до файлів або БД з урахуванням прийнятих в використовуваних системних засобах процедур маніпулювання даними та передачі результатів;
* перевірка засобів захисту компонентів на відповідність вимогам замовника та проведення їх трасування.

Після перевірки окремих компонентів системи проводяться їх інтеграція, а також верифікація і валідація інтегрованої системи. Систему тестують на безлічі наборів тестів для визначення адекватності і достатності цих наборів для завершення тестування і встановлення правильності системи.

Таким чином, можна визначити процеси верифікації та валідації.

**Процес верифікації.** Мета процесу – переконатися, що кожен програмний продукт (і/або сервіс) проекту відбиває погоджені вимоги до їхньої реалізації. Цей процес ґрунтується:

– на стратегії і критеріях верифікації всіх робочих програмних продуктів на ЖЦ;

– на виконанні дій з верифікації відповідно до стандарту;

– на усуненні недоліків, виявлених у програмних (робочих, проміжних і кінцевих) продуктах;

– на узгодженні результатів верифікації з замовником.

Процес верифікації може проводитися виконавцем програми або іншим співробітником тієї ж організації, або співробітником іншої організації, наприклад, представником замовника. Цей процес містить у собі дії з його впровадження і виконання.

Впровадження процесу полягає у визначенні критичних елементів (процесів і програмних продуктів), що повинні піддаватися верифікації, у виборі виконавця верифікації, інструментальних засобів підтримки процесу верифікації, у складанні плану верифікації і його затвердження. У процесі верифікації виконуються задачі перевірки умов: контракту, процесу, вимог, інтеграції, коду і документації.

Відповідно до плану і вимог замовника перевіряється правильність виконання функцій системи, інтерфейсів і взаємозв'язків компонентів, а також доступ до даних і до засобів захисту.

**Процес валідації.** Мета процесу – переконатися, що специфічні вимоги для програмного продукту виконано, і здійснюється це за допомогою:

– розробленої стратегії і критеріїв перевірки всіх робочих продуктів;

– обговорених дій з проведення валідації;

– демонстрації відповідності розроблених програмних продуктів вимогам замовника і правилам їхнього використання;

– узгодження із замовником отриманих результатів валідації продукту.

Процес валідації може проводитися самим виконавцем або іншою особою, наприклад, замовником, що здійснює дії з впровадженню і проведенню цього процесу за планом, у якому відбиті елементи і задачі перевірки. При цьому використовуються методи, інструментальні засоби і процедури виконання задач процесу для встановлення відповідності тестових вимог і особливостей використання програмних продуктів проекту на правильність реалізації вимог.

На інших процесах ЖЦ виконуються додаткові дії:

– перевірка і контроль проектних рішень за допомогою методик і процедур перегляду ходу розроблення;

– звернення до CASE-систем, що містять у собі процедури перевірки вимог до продукту;

– перегляди й інспекції проміжних результатів на відповідність вимогам для підтвердження того, що ПС має коректну реалізацію вимог і задовольняє умови виконання системи.

Т. Хоаром була запропонована ідея створення міжнародного проекту по формальної верифікації. Вона обговорювалася на симпозіумі з верифікації ПЗ в лютому 2005 р в Каліфорнії. Потім в жовтні цього ж року на конференції IFIP в Цюріху був прийнятий міжнародний проект терміном на 15 літ розробки «цілісного автоматизованого набору інструментів для перевірки коректності ПС».

У ньому сформульовані такі основні завдання:

* розробка єдиної теорії побудови і аналізу програм;
* побудова всеосяжного інтегрованого набору інструментів верифікації для всіх виробничих етапів, включаючи розробку специфікацій і їх перевірку, генерацію тестових прикладів, уточнення, аналіз і верифікацію програм;
* створення сховища формальних специфікацій і верифікованих програмних об'єктів різних видів і типів.

В даному проекті передбачається, що верифікація буде охоплювати всі аспекти створення та перевірки правильності ПЗ і стане панацеєю від усіх бід, пов'язаних з постійним виникненням помилок в створюваних програмах.

Багато формальних методів доказу і верифікації програм специфіковані та пройшли практичну апробацію. Проведена велика робота міжнародного комітету ISO / IEC в рамках стандарту ISO / IEC 12207: 2002 по стандартизації процесів верифікації та валідації ПЗ. Перевірка коректності формальними методами різних об'єктів програмування є перспективною.

Було запропоноване створення репозитарію, який є сховищем програм, специфікацій та інструментів, що застосовуються при розробках і випробуваннях, оцінках готових компонентів, інструментів і заготовок методів. На нього покладаються такі загальні завдання:

* накопичення верифікованих специфікацій, методів докази, програмних об'єктів і реалізацій кодів для складних застосувань;
* накопичення всіляких методів верифікації, їх оформлення у вигляді, придатному для пошуку і вибору реалізованої теоретичної ідеї для подальшого застосування;
* розробка стандартних форм для завдання і обміну формальними специфікаціями різних об'єктів програмування, а також інструментів і готових систем;
* розробка механізмів інтероперабельності[[5]](#footnote-5) та взаємодії для перенесення готових верифікованих продуктів з репозитария в нові розподілені і мережеві середовища для створення нових ПС.

Даний проект передбачається розвивати протягом 50 років. Більш ранні проекти ставили подібні цілі: поліпшення якості ПЗ, формалізація сервісних моделей, зниження складності за рахунок використання ПІК, створення налагоджувального інструментарію для візуальної діагностики помилок і їх усунення та ін. Однак докорінної зміни в програмуванні не відбулося ні в сенсі візуальної налагодження, ні в досягненні високої якості ПЗ. Процес розвитку триває.

Новий міжнародний проект по верифікації ПЗ вимагає від його учасників не лише знань теоретичних аспектів специфікації програм, але і високої кваліфікації програмістів для його реалізації в найближчі роки.

**Підхід до валідації сценарію вимог**

До процесу створення програм належить опис вимог мовою UML за допомогою сценаріїв і діючих виконавців – акторів як зовнішніх сутностей щодо системи. Вимоги потрібно перевіряти до їхньої перебудови у програмні елементи. Сценарій після трансформації – це послідовність взаємодій між одним або декількома акторами і системою, у якій актор виконує мету сценарію при взаємодії з нею. У моделі вимог сценарій задає кілька альтернативних подій, заданих мовою діаграм UML. Вони розділяються на функціональні (системні) і внутрішні, як визначальне поводження системи. На основі опису сценарію вимоги перевіряються шляхом валідації для виявлення помилок у поданні сценарних вимог. Ця перевірка відбувається ітераційною і складається з наступних кроків:

1. Формалізований опис вимог у вигляді сценаріїв;

2. Створення моделі вимог;

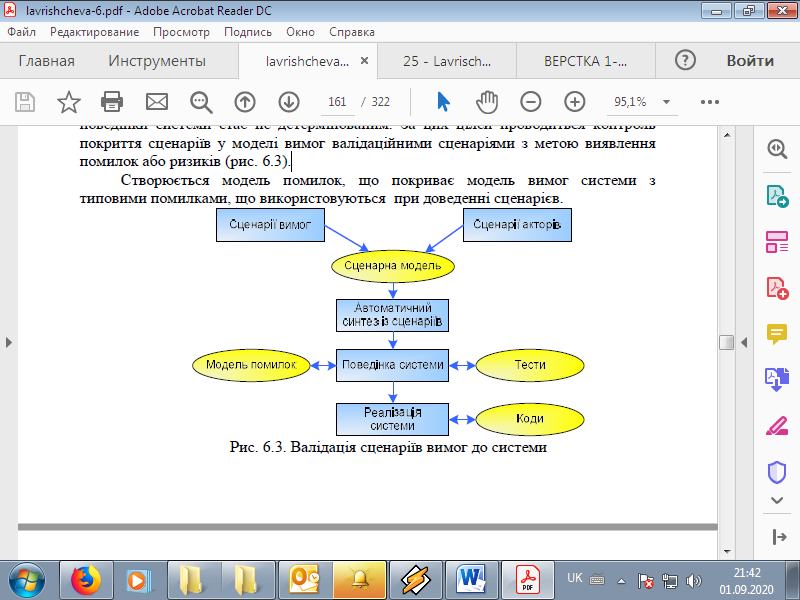
3. Створення спеціальних сценаріїв для валідації вимог;

4. Застосування валідаційних сценаріїв у моделі вимог;

5. Оцінювання результатів поводження моделі вимог;

6. Перевірка умов завершення процесу валідації і при виявленні яких-небудь неточностей повторення кроків, починаючи з п. 2.

При виконанні сценаріїв можуть виникнути помилкові ситуації, за яких поведінки системи стає не детермінованим. За цих цілей проводиться контроль покриття сценаріїв у моделі вимог валідаційними сценаріями з метою виявлення помилок або ризиків (див рис. нижче).



Складова частина валідації вимог за сценаріями – визначення класів еквівалентності вхідних і вихідних даних для валідації і синтезу сценаріїв. Вхідна інформація для синтезу сценаріїв – сценарна модель, що задається мовою взаємодії.

Інформація використовується при генерації додаткових сценаріїв з метою поліпшення процесу валідації, автоматичного синтезу сценаріїв моделі й отримання моделі поведінки системи під керуванням актора.

Модель перевіряється за допомогою тестів і моделі помилок, що в цілому дозволяє знайти неповноту вихідних вимог або суперечності у вимогах.

***Контрольні запитання для самоперевірки***.

1. Визначте та охарактеризуйте напрями перевірки правильності програм.
2. В який галузі важливим може бути використання формальних методів доведення правильності програм?
3. В чому полягає доведення правильності програм?
4. Визначте найбільш поширені методи доказу правильності програм. На чому базуються ці методи?
5. Надайте визначення верифікації ПЗ. Які особи виконують цей процес?
6. Надайте визначення валідації ПЗ. Які особи виконують цей процес?
7. Які стандарти застосовуються до процесів верифікації та валідації ПЗ?
8. В чому полягає процес валідації вимог до ПП?

**Рекомендована література**

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
5. Проектування інформаційних систем: Посібник // За редакцією Пономаренка В.С. – К.: Видавничий центр "Академія". 2002. ­ 488 с. URL: <http://www.dut.edu.ua/uploads/l_874_10304054.pdf>.

***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. **Предикат** (*n*-арний чи *n*-місний) — це функціяз областю значень {0,1}визначена на *n*-ому декартовому степені множини *M*. Таким чином, він характеризує кожну *n*-ку елементів *M* як «істинну», чи як «хибну». В теорії множин, дека́ртів добу́ток (*прями́й добу́ток*) двох множин X та Y — це множина усіх можливих впорядкованих пар, у яких перша компонента належить множині *X*, а друга — множині *Y*. [↑](#footnote-ref-1)
2. Нотація Бекуса-Наура (БНФ) є набором «продукцій», кожна з яких відповідає зразку:

   <символ> ::= <вираз, що містить символи>

   де вираз, що містить символи це послідовність символів або послідовності символів, розділених вертикальною рискою |, що повністю перелічують можливий вибір символ з лівої частини формули.

   Далі,

   * < — лівий обмежувач виразу
   * > — правий обмежувач виразу
   * ::= — *визначене як*
   * | — *або*

   Ці чотири символи є символами мета-мови, вони не визначені у мові, котру описують. Решта описаних символів належать до «абетки» описуваної мови. [↑](#footnote-ref-2)
3. Індуктивним називають умовивід, в формі якого відбувається емпіричне узагальнення, коли на підставі повторюваності ознаки у явищ певного класу роблять висновок про їх приналежність усім явищам даного класу [↑](#footnote-ref-3)
4. **Предикат першого порядку** часто називають атомарною формулою (атомом), що означає неподільність предиката на підформули, які еквівалентні по своїй семантиці другому предикату. Номер **порядку** (перший) означає, що в якості термів не можна використовувати інші предикати. [↑](#footnote-ref-4)
5. **Інтероперабельність** (англ. *interoperability* — здатність до взаємодії) означає можливість створення систем з довільних неоднорідних, розподілених компонентів на базі уніфікованих інтерфейсів або протоколів [↑](#footnote-ref-5)