Ліцензування програмних продуктів

1. Комп'ютерна програма як об'єкт авторського права
2. Правова охорона програмного забезпечення в Україні
3. Види ліцензій на програмні продукти
4. Моделі відкритості програмного забезпечення
5. Ознаки "ліцензійності" і контрафактності програмного забезпечення
6. Як купити ліцензійне програмне забезпечення

1. Комп'ютерна програма як об'єкт авторського права

Вперше у світі комп'ютерну програму зареєстрували як об'єкт авторського права у листопаді 1961 р. в США. Але американці продовжували дослідження щодо доцільності застосування моделі авторського права до комп'ютерних програм, і лише у 1980 р. внесли відповідне доповнення до свого закону про авторське право.

Цікаво, що першим законом, який включив комп'ютерні програми до переліку об'єктів авторського права, у 1972 р. став Закон про охорону інтелектуальної власності, прийнятий у Філіппінах. У 1985 р. авторсько-правову охорону програм було встановлено у Франції, Німеччині, Японії, Великобританії; у 1987 р. - в Іспанії; а у 1988 р. - у Канаді. В Російській Федерації комп'ютерні програми стали охороняти на підставі спеціального Закону "Про правову охорону електронних обчислювальних машин і баз даних", прийнятого у 1992 р.

В Україні комп'ютерна програма стала користуватись охороною з набранням чинності Законом України "Про авторське право і суміжні права" від 23 грудня 1993 р. № 3792-XII (далі - Закон про авторські права). Причому ст. 5 цього закону визначала комп'ютерну програму як один з видів літературних письмових творів.

На міжнародному рівні питання охорони комп'ютерних програм вперше обговорювала у 1971 р. Консультативна група урядових експертів Всесвітньої організації інтелектуальної власності (далі - ВОІВ). У 1978 р. ВОІВ схвалила "Типові положення про охорону програмного забезпечення обчислювальних машин". Слід зазначити, що найважливіша міжнародна конвенція в сфері авторського права - Бернська конвенція про охорону літературних і художніх творів 1971 р. (далі - Бернська конвенція) - чітко не визначає комп'ютерну програму як об'єкт авторського права. Але таку охорону цим об'єктам надають Угода з торгових аспектів прав інтелектуальної власності (ТРІПС) і Договір ВОІВ з авторського права (ДАП). Причому ці документи визначають, що комп'ютерні програми охороняються як літературні твори відповідно до Бернської конвенції.

У рамках ЄЕС даному питанню також приділялась увага. Так, глава 5 звіту Комісії ЄЕС з авторсько-правової охорони, підготовленого у 1988 р., присвячувалась комп'ютерним програмам. На підставі цього звіту у 1991 р. Рада ЄЕС прийняла відповідну Директиву (директива Ради 91/250/ЄЕС від 14 травня 1991 р.), яка встановила мінімальний перелік норм з авторського права, що мали бути введені до національних законодавств країн-учасниць ЄЕС не пізніше січня 1993 р.

Отже, на сьогоднішній день практично в усьому світі прийнята авторсько-правова модель охорони комп'ютерних програм, хоча на теоретичному рівні суперечки щодо ефективності такої охорони не вщухають. З іншого боку, в

багатьох країнах комп'ютерні програми також визнаються і об'єктами патентної охорони, якщо виконуються всі умови патентоспроможності (новизна, винахідницький рівень, промислова придатність). Як правило, мова йде про ті випадки, коли комп'ютерна програма є частиною технологічного процесу, технічного пристрою тощо й сумісно з ними може бути визнана об'єктом патентної охорони.

2. Правова охорона програмного забезпечення в Україні

Сьогодні українське законодавство визначає комп'ютерні програми виключно як об'єкти авторського права. Так, ст. 8 Закону про авторські права в редакції від 11 липня 2001 р., що діє на даний момент, визначає комп'ютерні програми як окремий об'єкт авторського права; ст. 18 цього самого закону в повній відповідності з міжнародними конвенціями, учасницею яких є Україна, встановлює, що комп'ютерні програми охороняються як літературні твори незалежно від способу чи форми вираження програм. Разом з тим, ч. 3 ст. 6 Закону України "Про Охорону Прав На Винаходи І Корисні Моделі" від 15 грудня 1993 р. з наступними змінами й доповненнями визначає, що комп'ютерні програми не можуть одержати охорону згідно з цим законом. З іншого боку, п. 3 ст. 8 Закону про авторські права стверджує, що передбачена цим законом правова охорона поширюється тільки на форму вираження твору й не поширюється на будь-які ідеї, теорії, принципи, методи, процедури, процеси, системи, способи, концепції, відкриття, навіть якщо вони виражені, описані, пояснені, проілюстровані в творі. Це означає, що в Україні охороняються лише форма виразу програм (по суті, вихідний та об'єктний коди), а їхні структура, алгоритми й ідеї, що лежать в основі програм, не підлягають охороні й можуть вільно використовуватись третіми особами.

Отже, комп'ютерна програма в Україні є об'єктом авторського права й охороняється як літературний твір. Визначення цього об'єкту містить ст. 1 Закону про авторські права "Визначення термінів": комп'ютерна програма - набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи в будь-якому іншому вигляді, виражених у формі, придатній для зчитування комп'ютером, які приводять його в дію для досягнення певної мети або результату (це поняття охоплює як операційну систему, так і прикладну програму, виражені у вихідному або об'єктному кодах).

На комп'ютерні програми поширюються правила про об'єкти, суб'єкти, зміст і терміни охорони авторських прав. Слід зазначити, що на сьогодні Закон про авторське право встановлює досить довгий строк охорони авторських прав - 70 років з моменту смерті автора або останнього із співавторів. На комп'ютерні програми поширюються всі випадки вільного використання творів, крім вільного використання в особистих цілях.

Особливості режиму охорони саме комп'ютерних програм як особливого виду творів визначає ст. 24 Закону про авторські права. Ця стаття дає користувачу комп'ютерної програми право зробити й зберігати одну архівну або резервну копію правомірно придбаної програми, а також декомпілювати (перевести з об'єктного коду у вихідний текст) й модифікувати програму (внести зміни) винятково для досягнення взаємодії з іншими програмами. При цьому в разі,

якщо подальше використання програми стає неправомірним, архівна або резервна копія має бути знищена. Наприклад, у разі продажу програми третій особі первинний власник втрачає право користуватись програмою і має стерти її зі свого комп'ютеру та знищити архівну (резервну) копію. Що до декомпіляції програм, то практично всі правовласники забороняють такі дії незалежно від їхнього призначення. При цьому не слід забувати про те, що на території України діятиме норма ст. 24 Закону про авторські права щодо декомпіляції, оскільки вона є імперативною.

3. Види ліцензій на програмні продукти

Ліцензія на використання програмного забезпечення (англ. EULA - End User License Agreement) - вид ліцензії, що визначає умови використання виробу, яким є комп'ютерне програмне забезпечення. Ліцензія може надавати дозвіл робити з ним речі, які були б інакше заборонені законом про авторське право. Наприклад, ліцензія на використання програмного забезпечення може дати дозвіл робити копії програмного забезпечення. Власник авторського права може запропонувати ліцензію на використання ПЗ односторонньо, або як частину ліцензійної угоди на використання ПЗ з іншою стороною.

Ліцензія може містити умови. Якщо Ви не дотримуєтесь умов ліцензії, то підпадаєте під дію закону про авторське право.

Ліцензія передбачає ряд обмежень для користувачів програмного продукту: обмеження користування, копіювання, модифікації та розповсюдження, охороняючи, таким чином, авторські права розробника. Проте ліцензування - це не тільки захист авторських прав, а й гарантії для клієнтів. Ліцензування програмного забезпечення дозволяє користувачам отримувати технічну підтримку з боку розробника або офіційних розповсюджувачів програмного продукту, модернізувати програмне забезпечення.

Основним документом, що визначає права та обов'язки користувача програмного забезпечення, є ліцензійна угода.

Види ліцензій на програмне забезпечення:

Freeware - безкоштовні програми. Програми без обмеження на (некомерційне) використання. Охороняються авторським правом. Назву запропоновано у 1982 р. і зареєстрована як товарний знак першим головним редактором PC World і PC Magazine Ендрю Флюгельманом. Правда, слід пам'ятати, що відсутність ціни ще не означає, що виробник дозволяє її вільно поширювати, він може це і забороняти. І буває, що якась програма безкоштовна тільки для домашнього, некомерційного використання, а при використанні її в організаціях потрібно заплатити.

Shareware - умовно-безкоштовні програми. Клас комерційних програм з безкоштовним періодом використання. Вимагають оплати для повнофункціонального використання. Термін запропонований вперше у 1983 р. Бобом Уоллесом, одним з перших співробітників Microsoft. Різновид вперше застосовано у 1982 р. Джимом Кнопф, співробітником IBM. При такій моделі розповсюдження пропонується спочатку випробувати програму в дії, а потім оплатити її. Щоправда, "умовність" може лежати в дуже широких межах, від простого нагадування про необхідність заплатити за програму при кожному

запуску до обмеженого терміну роботи і навіть блокування в неоплаченій версії найважливіших функцій, що робить неможливим використання програми за прямим призначенням.

Trial, trialware - пробне (оцінне) програмне забезпечення. Обмежено часом використання або кількісними характеристиками, а іноді й функціоналом. Як бачите, термін перетинається з Shareware. Схоже на crippleware для класу умовно-безкоштовних програм.

Demo, demoware - демонстраційні програми. Мають велику кількість обмежень. Основна мета - не пробне використання, а демонстрація можливостей. Помітно більш обмежене порівняно з trialware.

Adware - рекламно-орієнтовані програми. Без обмежень функціональності, але з примусовим показом реклами, яка може підвантажуватися через Інтернет без відома користувача. Зазвичай включають модуль фонового завантаження реклами, що таїть у собі небезпеку несанкціонованого дистанційного контролю комп'ютера. Антивірусні програми часто класифікують даний механізм як "троянського коня".

Nagware, begware - основним обмеженням використання є примусове вікно діалогу, де повідомляється про те, що версія незареєстрована. Після оплати дане обмеження знімається. Ускладнює використання програми в пакетному режимі при автоматичній обробці інформації. Вперше даний різновид застосовано у 1983 р. Бобом Уоллесом.

Public domain - вільні програми. Без обмежень на модифікацію та використання. Не охороняються авторським правом.

Donateware, donationware - авторські програми. Для необов'язкової реєстрації програми потрібно сплатити пожертвування автору. Цей різновид вперше застосовано Ендрю Флюгельманом.

Open source - відкриті програми з вихідними текстами. Можуть накладатися обмеження на модифікацію та використання в комерційних цілях.

Linkware - автор програми просить вказувати посилання на сайті користувача, (якщо є) на свій сайт.

Registerware - для отримання та/або використання програми потрібно надати інформацію про себе (заповнити анкету).

Guiltware - різновид nagware. У програмі міститься явна згадка, що автор не отримав за неї грошей. Може й не передбачати реєстрації.

Crippleware - ключові можливості програми видалені. Немає обмежень на час використання. Після оплати надається повнофункціональна версія.

Abandonware - позаринкові програми. Як правило, це колишні комерційні програми, які з ряду причин перестають постачати на ринок. Їх поширює зазвичай власник авторських прав на безкоштовній основі, але з жорстким зобов'язанням заборони продавати і навіть без права подальшого безкоштовного тиражування.

Orphanware - різновид abandonware, коли автора не можна розшукати.

Cardware, postcardware - як компенсацію за надання програми автор просить надіслати йому листівку або електронний лист зі словами подяки. Ці листи використовуються авторами для реклами своїх робіт.

Liteware - "полегшений" варіант відповідної комерційної версії. Не обмежена часом використання, але обмежена функціоналом.

Hostageware - програми з функціональними, тимчасовими і кількісними обмеженнями. Розблоковуються після оплати.

Beerware - право на використання програми, а також отримання вихідних текстів в обмін на гроші, на які автор зможе купити собі пива. Цей різновид shareware вперше введено Джоном Брістором у 1987 р.

Careware, charityware - стягується збір на благодійні цілі, або безпосередньо автору, або за вказаною адресою.

Requestware - автор просить користувача щось зробити в обмін на використання програми (надіслати листівку або електронного листа з подякою, внести пожертви на благодійні цілі тощо). Різновиди: postcardware, careware.

Betaware - попередня (тестова) бета-версія комерційного або некомерційного ПЗ. Можна використовувати безкоштовно, але часто обмежується періодом тестування.

Commercial - комерційне програмне забезпечення, яке продається за гроші й захищено різними законами.

CDware - ПЗ на компакт-дисках, яке розповсюджується в рекламних цілях.

Spyware - програми-шпигуни. Несанкціоновано збирають інформацію про комп'ютер користувача і його дії. Нерідко маскуються під adware. Крім використання антивірусних програм найбільш ефективний спосіб боротьби - установка брандмауерів.

4. Моделі відкритості програмного забезпечення

Розглянемо проблему охорони та використання ПЗ із урахуванням факту наявності різних видів ліцензій на нього, які кардинально змінюють підходи до специфікації і захисту авторських прав, і як наслідок - його використання - закрите і відкрите (вільне) ПЗ. Ці два типи ліцензій на ПЗ визначають модель використання, організаційну модель розробки (якість і технічні параметри програм), структуру витрат на різних етапах життєвого циклу програм, і, зрештою, на модель ринку ПЗ загалом.

Існує дві моделі відкритості ПЗ:

Закрите ПЗ (Proprietary software) - автор (або інший власник) утримує за собою низку прав, а користувач - лише обмежене право використання ПЗ. Зокрема, заборонено або закрито доступ до коду, заборонено внесення будь-яких змін, використання більш ніж на одному комп'ютері, тиражування та розповсюдження, перепродаж, копіювання тощо.

Відкрите (вільне) ПЗ (Free/Open Source Software) - базовий набір майнових прав передається ("ліцензується") власникові кожного екземпляра програми. Користувач ПЗ отримує право та можливість використання програми для різних цілей, доступ до програмного коду, можливість копіювання (тиражування) і публічного поширення копій програми, а також можливість зміни і вільного поширення як оригінальної програми, так і зміненої; дозволено будь-які дослідження механізмів функціонування програми та можливість використання механізмів (принципів) функціонування і будь-яких довільних частин коду

програми для створення інших програм та/або адаптації до потреб користувача без додаткової згоди автора (або іншого власника), обов'язкових грошових відрахувань тощо.

Спираючись на аналіз ліцензій і враховуючи їх особливості, спробуємо дати коротку характеристику цих двох видів ПЗ.

Для закритого ПЗ характерні такі основні несприятливі для користувача моменти, спричинені насамперед умовами закритого коду, за допомогою яких власники ПЗ впливають на ринок комп'ютерних програм:

- Більшість закритих програм постачають користувачам без вихідного коду, тобто у формі, що допускає тільки експлуатацію програми, але не її вивчення, модифікацію.
- Приховування вихідного коду спричиняє зростання асиметрії інформації про товар або послугу, зокрема про якість програмування. Власник, який приховує код, одержує переваги, пов'язані з якістю пропонованого користувачам ПЗ. Так, у випадку виявлення помилки вартість внутрішнього супроводу екземпляра програми на майбутніх етапах експлуатації може значно збільшитися.
- Приховування вихідного коду спричиняє також монополізацію послуг із супроводу програм (виправлення помилок, розширення функціональності, інтеграція з іншими програмами і новим обладнанням).

Право внесення змін належить до виняткових майнових прав автора, однак, що стосується випущених в обіг екземплярів програм, це право може бути обмежено (ст. 24 Закону України "Про авторське право і суміжні права"). Все ж користувач, не маючи доступу до вихідного коду, але володіючи таким правом, мусить подолати серйозні технічні труднощі, щоб скористатися ним. У результаті на ринку надання таких послуг власник ПЗ може мати або має монополію.

- Серйозним ексцесом закритого ПЗ є тенденція до "пропрієтаризації" ("антистандартизації") інтерфейсів.

Інтерфейс - це спосіб взаємодії програм між собою, з обладнанням і з кінцевим користувачем (визначає взаємодію програми із системою, формат збереження або передачі даних, протокол (послідовність кроків) передачі даних, інтерфейс користувача (сукупність команд, якими користувач керує програмою, і повідомлень, що відображають її стан або результати роботи).

Володіючи винятковим доступом до вихідного коду більш як однієї програми, власник може спонукати інших розробників ПЗ взаємодіяти з його нестандартним інтерфейсом або навмисне відхиляється від стандартів і таким чином створювати "власні інтерфейси". Розвиток ринку ПЗ, як правило, супроводжується стандартизацією інтерфейсів. Стандартизація інтерфейсів, що формує конкурентний ринок сумісних і взаємозамінних програм, вигідна для кінцевого користувача і, зрештою, для галузі загалом, однак може бути невигідна для компаній-монополістів, які домінують у деяких сегментах ринку ПЗ.

Зазначене варто розуміти саме як ексцеси закритого ПЗ, як закономірні, логічні, але не обов'язкові наслідки його закритості, які можна скорегувати виробленням відповідних угод, наприклад, які передбачають обов'язкове постачання вихідного коду програми.

Проблема "піратства" - це ще один ексцес, тісно пов'язаний із існуванням закритого ПЗ. Це серйозний руйнівний фактор функціонування ринку ПЗ загалом. Посилення державної боротьби з піратством в Україні призвело до сплеску попиту на відкрите ПЗ, яке через розподіл виняткових майнових прав припиняє можливість зловживань.

Переваги відкритого ПЗ порівняно із закритим ПЗ найяскравіше виявляються в ситуації масової експлуатації програм цього типу.

Наявність численних вже готових вільних програм. Замовлення на доопрацювання: модифікацію, адаптацію, локалізацію, документування існуючої програми, пакета або системи значно дешевше, ніж замовлення розробки "з нуля".

Право введення в цивільний обіг додаткових екземплярів програм. Наявність цього права є важливим, оскільки дає змогу уникнути проблем, пов'язаних із ліцензуванням, і, як наслідок, користувач отримує право виготовляти та використовувати додаткові екземпляри програми.

Наприклад, зміст ліцензії "GNU General Public License" (далі - GNU GPL) [46] (один із видів ліцензії відкритого - вільного - ПЗ) свідчить, що користувач (ліцензіат), який придбав екземпляр програми, здобуває право на відтворення необмеженої кількості екземплярів програми або твору, що є похідним від програми, поширювати екземпляри програми на будь-якому носії, стягувати плату за передачу екземпляра програми, можна також за плату надавати послуги гарантійної підтримки програми. Програма, що вводиться в цивільний обіг під ліцензією GNU GPL, обов'язково супроводжується повним вихідним текстом чи пропозицією надати будь-якому користувачу за винагороду, яка не перевищує вартість виготовлення копії, повну копію вихідного тексту програми.

Ліцензія "GNU General Public License" була розроблена благодійним фондом Free Software Foundation (далі - FSF), що сприяє розробці ПЗ із відкритим вихідним текстом. Усім авторам, які підтримують ідеї FSF, було рекомендовано включити в розроблені програми повідомлення про свої авторські права і про те, що користувач одержує доступ до програми на умовах ліцензії GNU GPL, а також її текст. Тобто кожен користувач разом із екземпляром програми одержує за ліцензією GNU GPL майнове право на неї, не виплачуючи за це правовласникові ніякої винагороди. Завдяки свободі відтворення, поширення і модифікації відкрите ПЗ називають також вільним (free software).

Ліцензія GNU GPL - це варіант реалізації особливого порядку укладання авторського договору з масовим користувачем. Можливість його застосування передбачена в ст. 32 Закону України "Про авторське право і суміжні права". За своєю правовою природою ліцензія GNU GPL - договір приєднання за ст. 634 ЦК України: "Договором приєднання є договір, умови якого встановлені однією із сторін у формулярах або інших стандартних формах, який може бути укладений лише шляхом приєднання другої сторони до запропонованого договору в цілому. Друга сторона не може запропонувати свої умови договору".

На відміну від відкритого ПЗ, введення в цивільний обіг додаткових екземплярів закритого ПЗ дозволено за умови отримання додаткових ліцензій на кожен примірник комп'ютерної програми. Ліцензія на закрите ПЗ неподільна і не допускає одночасного її використання на декількох ЕОМ.

Однак користувач закритого ПЗ, який правомірно володіє примірником комп'ютерної програми, має право без згоди автора або іншої особи, яка має авторське право на цю програму, виготовити одну копію комп'ютерної програми за умови, що ця копія призначена тільки для архівних цілей або для заміни правомірно придбаного примірника у випадку, якщо оригінал комп'ютерної програми буде втраченим, знищеним або стане непридатним для використання. При цьому копія комп'ютерної програми не може бути використана з іншою метою (ч. 2 ст. 24 Закону України "Про авторське право і суміжні права").

Не зважаючи на вимоги ст. 24 Закону України "Про авторське право і суміжні права", можливі ситуації, коли право на виготовлення резервної копії заборонено договором. Тоді виникає питання про правомірність такої заборони.

Право на модифікацію і доступ до вихідних текстів. Наявність цього права дає змогу уникнути "зв'язування" кінцевого користувача в плані вибору контрагента при замовленні або перезамовленні послуг. Будь-яку послугу із виправлення, адаптації, модифікації програми можна замовляти на конкурентному ринку.

Проблема внесення до закритого ПЗ змін (модифікації) із метою забезпечення його функціонування на технічних засобах користувача, а також виправлення очевидних помилок допускається відповідно до ч. 1 ст. 24 Закону України "Про авторське право і суміжні права". Але особливо необхідно звернути увагу, на те, що застосування цієї норми можливе лише у випадку, якщо інше не передбачено угодою, тобто в угоді може бути заборонено вчинення таких дій.

Що стосується відкритого ПЗ, то відповідно до ліцензії GNU GPL, ліцензіат має право модифікувати свій екземпляр або екземпляри програми цілком, або будь-яку її частину. Ліцензіат має право також виготовляти і поширювати екземпляри такого продукту, похідного від програми, або змінені екземпляри.

Вважаємо за необхідне звернути увагу на питання декомпілювання комп'ютерної програми (ч. 3 ст. 24 Закону України "Про авторське право і суміжні права"). Декомпілювання - це технічний прийом, що включає перетворення комп'ютерної програми з об'єктного коду у вихідний текст для одержання інформації, необхідної для досягнення її взаємодії із незалежно розробленою комп'ютерною програмою. Право на декомпілювання комп'ютерної програми має особливо важливе значення для користувачів закритого ПЗ, бо згідно загального правила ПЗ надається тільки в об'єктивному коді при збереженні комерційної таємниці на вихідний текст. Що ж стосується відкритого ПЗ, то завдяки вільному доступу до вихідних текстів, користувач може не відчувати потреби в декомпілюванні ПЗ взагалі.

Можливість переносу до іншого програмного або апаратного середовища, тобто модифікація, право на яку передбачають вільні ліцензії і технічну можливість якої забезпечує доступність вихідного коду, - це перенесення програми в інше програмне (під іншу операційну систему) або апаратне (на устаткування іншого типу) середовище. Більшість вільних програм доступні більш як для однієї операційної системи або апаратної платформи. Така

можливість важлива, оскільки вона зменшує залежність кінцевого користувача від раніше прийнятих рішень у частині апаратного і ПЗ.

2002 р. в Україні була розгорнута активна діяльність із "легалізації" ПЗ, зокрема того, що використовується у державних установах і державному секторі господарства. Необхідно пам'ятати, що основним одержувачем прибутку від легалізації є закордонні корпорації, виробники закритого ПЗ (наприклад, американська компанія Microsoft). Збільшуючи валовий національний продукт цих країн, ми ризикуємо стати ще менш конкурентоспроможними не тільки в економічному, а й у політичному плані.

Розробники українського законопроекту "Про використання Відкритих стандартів даних та Вільного програмного забезпечення в державних установах і державному секторі господарства" пропонують широке запровадження, при підтримці держави, використання ПЗ із вільними кодами, що забезпечує масову доступність ліцензійного ПЗ (на відміну від нинішньої ситуації, коли ліцензійне ПЗ від монополістичного постачальника доступне або добре забезпеченим верствам населення, або воно контрафактне). Законопроект створює умови, що гарантують права авторів як "відкритого", так і "пропрієтарного" (закритого) ПЗ, зменшуючи мотивацію для неліцензійного використання ПЗ.

5. Ознаки "ліцензійності" й контрафактності програмного забезпечення

На практиці порушення прав на програмне забезпечення може здійснюватись у декілька способів. Найбільш поширеними з них є такі: відтворення й розповсюдження програмного забезпечення на дискетах та компакт-дисках; установка "піратського" програмного забезпечення на комп'ютер, що продається; відтворення й розповсюдження програмного забезпечення через Інтернет.

Що до першого виду порушень, то в такому разі виявити порушення авторських прав, як правило, найлегше. Фальсифікований носій та упаковка мають інший вигляд, ніж оригінальні. Часто "піратське" програмне забезпечення продається на "самописних" дисках (CD-R), тоді як правовласники випускають в обіг лише диски, виготовлені промисловим способом. Упаковка фальсифікованих дисків низької якості, а може бути й простою ксерокопією оригінальної упаковки. На одному "піратському" диску може бути записана збірка програм, та ще й різних правовласників, що свідчить про безумовну контрафактність продукту. Але трапляються й такі "піратські" диски, які дуже схожі на оригінальні: співпадають образ диску, поліграфія, коробка. Для того, щоб відрізнити "ліцензійну" продукцію від дуже схожої "піратської" правовласники застосовують такі спеціальні захисні засоби, як голографічні наклейки, сертифікати й ліцензії, виготовлені на спеціальному папері. Отже, для вирішення питання про контрафактність програмного забезпечення, яке розповсюджується на компакт-дисках і дискетах, порівнюється зовнішній вигляд продуктів, що викликають підозру, з оригінальними примірниками правовласника. Ще однією ознакою контрафактності є ціна продуктів.

При другому виді порушень та у випадку, коли сам користувач вже встановив програму на свій комп'ютер, виявити порушення авторських прав на програмне забезпечення вже складніше. Але в багатьох випадках простий аналіз програми

дозволяє виявити такі ознаки контрафактності, як наявність спеціальних програм "креків" або певної нестандартної установки програми. Водночас легальність програмного забезпечення, встановленого на жорсткий диск, підтверджують оригінальні упаковки й оформлення дискет і дисків, випущених правовласником, експлуатаційна документація, сертифікати й ліцензії, які входили в пакет при придбанні продукту. Причому характер використання програми не може суперечити умовам, передбаченим у ліцензіях та інших експлуатаційних документах. Так, наприклад, у ліцензії має зазначатись кількість комп'ютерів, на яких можна встановити придбаний продукт. Також користувачі "ліцензійного" програмного забезпечення мають зберігати договори купівлі-продажу продуктів, бланки-замовлення, накладні й документи, що підтверджують оплату придбаних продуктів. Наявність вказаних документів навіть у разі придбання неліцензійних примірників програм свідчитиме про неумисність дій особи, яка здійснила таке придбання, за умови значної схожості контрафактного й оригінального продуктів.

Найскладнішим завданням є виявлення порушень авторських прав на програмне забезпечення шляхом розповсюдження програм по Інтернету. Тут необхідно з'ясовувати безпосередньо в правовласників, чи надавали вони дозвіл на таке розповсюдження, і яким чином користувач може використовувати програму, яку він "скачав" з Інтернету. Так, деякі правовласники розміщують свої продукти (повністю або демо-версії) в Інтернеті й надають право необмеженій кількості осіб використовувати їх, але лише для власних некомерційних цілей.

6. Як купити ліцензійне програмне забезпечення

Перед придбанням ліцензійного програмного забезпечення, якщо немає повного розуміння, як це зробити, варто в першу чергу звернутись до ресурсів Інтернету. Усі розробники й правовласники програмних продуктів мають сайти, на яких можна знайти необхідну інформацію про продукт - його функціональне призначення, можливості, зовнішній вигляд, підтвердження ліцензійності, комплектацію продукту тощо. Крім того, на таких сайтах як правило можна знайти інформацію про продавців продуктів (у разі наявності - сертифіковані партнери, ділери, роздрібна мережа).

Як приклад розглянемо процедуру придбання ліцензійних програмних продуктів компанії Microsoft. Її російськомовний сайт, на якому є й інформація стосовно України, розміщений за адресою: http://www.microsoft.com/ukraine/sam/what/licensing/what.mspx.

Microsoft вважає придбання програмного продукту придбанням ліцензій (прав) на використання цих продуктів. Тому основний принцип для ліцензійних продуктів цієї компанії - на кожний продукт має бути ліцензія. Умови ліцензії фіксуються в ліцензійних угодах, якими комплектуються продукти ("обгорткова ліцензія").

Microsoft здійснює продаж своїх продуктів за різних умов:

1. **"коробкові"** версії продуктів - **Full Package Product (FPP)** - можна придбати в роздрібній мережі. Це продукт, упакований в кольорову коробку, в якій містяться сам носій, ліцензія та документація. Коробкові

версії найкраще підходять для приватних осіб та організацій, яким потрібно небагато ліцензій. Якщо ви плануєте використовувати програмне забезпечення на п'яти або більше комп'ютерах, доречно обрати програму корпоративного ліцензування, яка дає змогу зекономити час і кошти;

2. **"передвстановлені"** версії продуктів - **Original Equipment Manufacture (OEM)** - придбати можна лише разом з новим комп'ютерним обладнанням. У вигляді OEM-версій доступні операційні системи (зокрема, Windows) та певні прикладні програми (наприклад, Office);
3. **"корпоративні"** ліцензії - програми ліцензування для організацій, які передбачають використання продукту в мережі робочих комп'ютерів:

- *Open License* - низка програм ліцензування для організацій, яким потрібно п'ять або більше ліцензій на будь-які прикладні програми. Компанії, які оберуть програму Open License, знайдуть відповідну інформацію з ліцензування на сайті Microsoft Open, до якого надається захищений доступ;
- *Open Business* - забезпечує економію коштів порівняно з придбанням програмного забезпечення у роздріб. Компанія має зробити початкове замовлення на п'ять або більше ліцензій. Щоб набрати необхідний мінімум (п'ять ліцензій), можна скласти набір із будь-яких продуктів Microsoft;
- *Open Business 500+* - дає змогу зекономити ще більше коштів компаніям, які можуть зробити початкове замовлення великого обсягу на продукти однієї або кількох категорій - прикладні програми, системи або сервери (мінімум 500 балів в певній категорії продуктів);
- *Open Value* - дає змогу організаціям, які зроблять початкове замовлення на п'ять або більше ліцензій, скористатися перевагами програми Software Assurance (SA) включеної до вартості ліцензії та отримати безстрокові ліцензії на програмне забезпечення Microsoft, при цьому замовними не мають необхідності сплачувати всю суму на момент придбання. Вартість ліцензій виплачується рівними частинами раз на рік протягом терміну дії угоди. Заощадити додаткові кошти можна, обравши варіант Company-Wide (для всієї компанії);
- *Microsoft Enterprise Agreement*

Enterprise Agreement - програма корпоративного ліцензування для організацій, що мають у власності щонайменше 250 комп'ютерів і прагнуть обрати певні продукти як стандартну платформу та одержати їх за спеціальними цінами. Пропонується два варіанти цієї програми:

- Enterprise Agreement варто обирати організаціям, які мають щонайменше 250 комп'ютерів і планують, уклавши угоду на три роки, вибрати як стандарт продукти платформи Microsoft Platform Enterprise (зокрема, Office Professional, Windows Professional Upgrade та набір клієнтських ліцензій Core CAL).
- Enterprise Subscription Agreement розрахована на організації, які мають щонайменше 250 комп'ютерів та вважають за краще отримати ліцензії на продукти Microsoft на умовах передплати. Ця програма дає змогу обрати

як стандарт один або кілька продуктів платформи Microsoft Platform Enterprise, уклавши угоду на трирічний період.

Програми для навчальних і державних установ

Академічна програма ліцензування - дає змогу установам, що відповідають визначеним у ній умовам, придбавати продукти за спеціальними цінами як у роздрібній торгівлі, так і через мережу продажу корпоративних ліцензій.

Програма ліцензування для державних установ Open License - встановлює спеціальні ціни на програмне забезпечення для малих та середніх державних організацій, які відповідають умовам програми. Ця програма підходить для тих компаній, яким потрібна невелика кількість ліцензій та проста і гнучка модель придбання.

"Державною установою" може вважатися будь-яка організація, правоохоронний орган або інша установа державного чи місцевого рівня, статус якої підтверджується її статутом. Організації, що хочуть купувати програмні продукти за програмою ліцензування для державних установ Open License, мають відповідати певним умовам.

Залежно від потреб у програмному забезпеченні, внутрішніх процедур, виділення коштів та способів придбання можуть бути вибрані, наприклад, ліцензії, куплені в розстрочку (Open Value або Enterprise Agreement) чи придбані по підписці (Open Value Subscription, Enterprise Agreement Subscription), коробкові ліцензії чи OEM-версії програмного забезпечення, передустановлені на нові ПК. У кожному з цих варіантів є свої особливості (детально тут www.microsoft.com/Ukraine/Licensing/volume/government.mspx), та використовуватися вони можуть державними та комерційними організаціями на рівних умовах.

Крім стандартних програм для корпоративних клієнтів, для державних установ пропонується також спеціальна програма ліцензування - Microsoft Open License Government. Схема постачання ліцензій за цією програмою здебільшого подібна до стандартних умов програми Microsoft Open License, проте існують і відмінності, а саме - більш низькі ціни на програмне забезпечення для державних замовників.

При придбанні коробкових та OEM-версій підтвердженням ліцензійності продуктів слугують усі компоненти придбаного пакету (ліцензійна угода, носії, документація, купон реєстраційної картки, сертифікат автентичності), а також чек/інвойс, що підтверджує факт придбання продукту. В разі, якщо пакет укомплектований ліцензією в електронному форматі, її рекомендується роздрукувати й зберігати з іншими документами на продукт.

При придбанні корпоративних ліцензій підтвердженням ліцензійних прав є іменний сертифікат.

Збереження зазначених документів здійснюється не тільки для підтвердження ліцензійності продуктів перед правоохоронними органами, а є необхідним для подальшого продажу продукту іншій особі, оскільки в такому разі всі компоненти продукту слід передати цій останній.

Лекція 6. Парадигма якості в програмній інженерії

1.1 Основні поняття в області якості

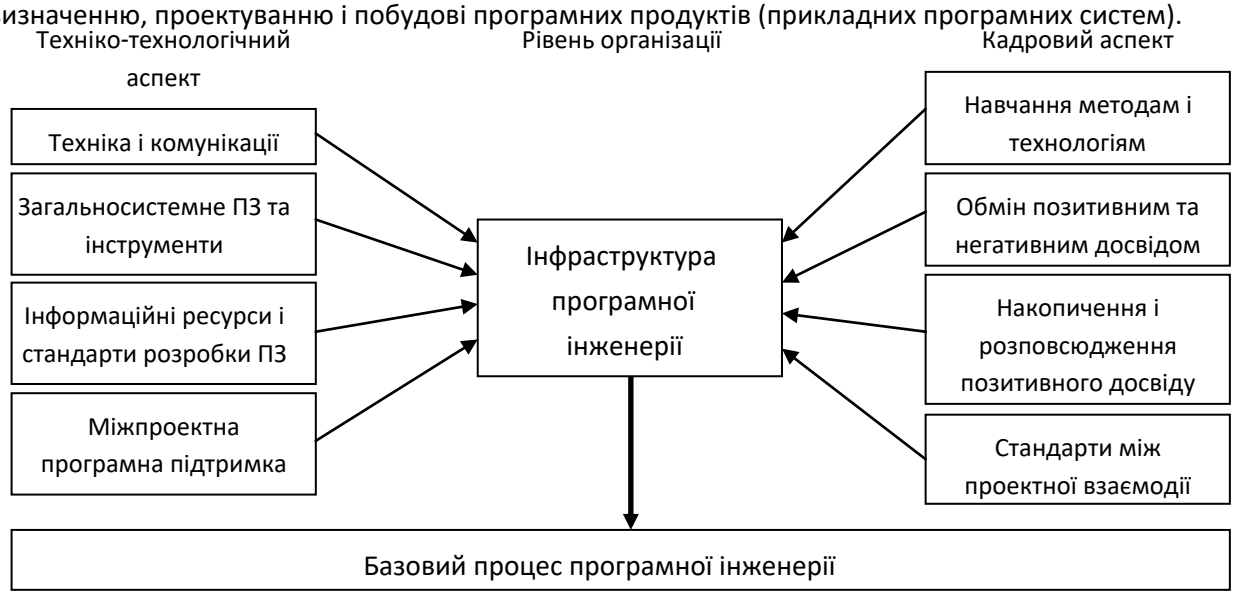
Програмна система (ПС) – група інтегрованих програмних засобів, які підтримують певний діловий процес споживача (або його частину) і використовують загальне сховище даних.

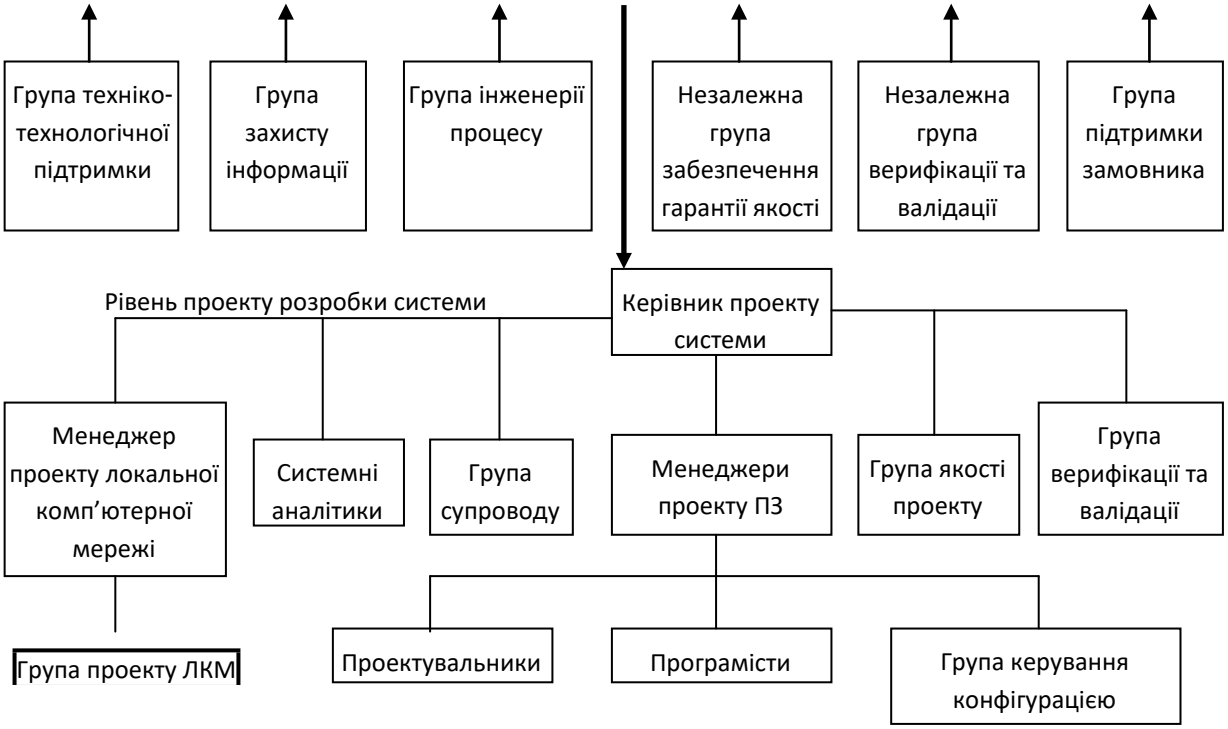
Термін *програмне забезпечення* (ПЗ) використовується застосовано до сукупності програмних засобів, які розробляються з метою експлуатації у складі системи.

Основа якісної розробки програмних систем – раціональна інфраструктура програмної інженерії як виду бізнесу. Спеціалісти-практики в області програмних систем і користувачі сходяться в поглядах на поняття поганого програмного продукту як такого, що:

- Не забезпечує підтримку стратегії бізнесу або потреб користувача
- Недостатньо надійний, гнучкий, ефективний і погано супроводжується
- Є коштовним і занадто довго розробляється

Інфраструктура програмної інженерії – інтегрований набір загальнодоступних технічних, технологічних і методологічних ресурсів організації розробника, які роблять можливим виконання процесу програмної інженерії колективами проектів, які відкриваються по договорах із замовниками. Тут *проект* – це обмежена часовими рамками діяльність, мета якої полягає в створенні унікального програмного продукту. *Процес програмної інженерії* – множина логічно пов'язаних видів діяльності по визначенню, проектуванню і побудові програмних продуктів (прикладних програмних систем).





Компоненти інфраструктури розробки ПС.

Техніко-технологічний аспект.

1. Техніка і комунікації:
   - Комп'ютери користувачів, файлові сервери
   - Локальні комп'ютерні мережі (ЛКМ)
   - Глобальна комп'ютерна мережа (ГКМ)
   - Електронна пошта
   - Техніка для тестування
   - Офісна техніка
   - Інші складові комплексу технічних засобів
2. Загальносистемне ПЗ та інструменти:
   - Клієнт\серверні технології
   - Операційні системи
   - Офісні системи
   - Системи документообігу
   - Утиліти
   - Засоби захисту інформації (антивіруси)
   - CASE-інструменти, системи програмування
   - СУБД
   - Графічні інструменти
3. Інформаційні ресурси і стандарти розробки:
   - Методології розробки
   - Інструменти керування проектами, конфігураціями
   - Системи підтримки використання ресурсів Інтернет
   - Нормативні документи, які стосуються технічних, програмних, комунікаційних засобів, даних і захисту інформації
   - Нормативні документи оформлення матеріалів
   - Методичні матеріали, шаблони і заготовки документів
4. Міжпроектна програмна підтримка
   - Розроблені програми (модулі), визнані здатними до загального користування, документовані та

поміщені під контроль конфігурації.

Кадровий аспект.

1. Навчання методам і технологіям: 3. Накопичення і закріплення позитивного Можливості організації по навчанню досвіду:

спеціалістів методам та прийомам розробки Визначення форматів і засобів накопичення і ПЗ зберігання здобутого досвіду (опитування,

•Можливості вивчення спеціалістами техніко семінари тощо) технологічних компонент інфраструктури Створення бібліотек активів організації за 2. Обмін позитивним та негативним досвідом: принципом «кращий об’єкт». Включення їх у Культура «відкритого» сприйняття/передачі сферу керування конфігурацією.

набутого досвіду, знань, характерних Забезпечення доступності. помилок. Сприяння розповсюдженню 4. Стандарти міжпроектної взаємодії:

позитивного досвіду. Не приховування Визначення стандартів (меж компетенції, власних помилок і не перекладання знань) по процесам ЖЦ створюваної ПС. відповідальності за них. Бажання Уніфікація та стандартизація прийомів навчатись/навчати роботи з метою побудови і підтримки базового процесу програмної інженерії

•Профілювання знань для забезпечення замінюваності спеціалістів в проекті. Дотримання принципу «глибокі знання у вузькій сфері»

Ролі спеціалістів в організаційній структурі розробки

Ролі на рівні організації

1. Група техніко-технологічної підтримки:
   - Вивчення ринку послуг і попиту в організації відносно техніки та загальносистемного ПЗ
   - Придбання/встановлення/підтримка техніки
   - Придбання/встановлення/підтримка загальносистемного ПЗ
   - Навчання/консультаційні послуги співробітникам
   - Рекомендації по застосуванню техніки і технологій в проектах
2. Група захисту інформації:
   - Вивчення стану справ в області захисту інформації і накопичення досвіду
   - Забезпечення захисту інформації в організації
   - Перевірка захисту інформації в організації
   - Підтримка проектів в питаннях захисту інформації
3. Група інженерії процесу

- Визначення, супровід та вдосконалення базового процесу програмної інженерії. Забезпечення нормативно-методичної підтримки виконання процесів ЖЦ. Організація та поповнення сховища (бібліотеки) активів організації
- Допомога менеджерам проектів в адаптації базового процесу до потреб проектів. Підбір або виготовлення форм (шаблонів) документів для інженерії проектів
- Підтримка процесу документування в проектах, зокрема виконання важких графічних робіт, оформлення документів згідно стандартів оформлення. Нормоконтроль та друк документів.
- Міжпроектна координація в частині накопичення досвіду і організації навчання
- Підтримка керування конфігурацією в проектах

4. Незалежна група якості (SQA-група):
   - Планування та виконання дій по контролю і гарантії дотримання дисципліни створення програмної продукції в проектах (організація перевірок робіт в контрольних точках проектів, визначених календарними планами)
   - Контроль документів і продуктів ПЗ в контрольних точках проектів на предмет дотримання діючих стандартів та інших нормативних документів, встановлених у вимогах замовника Звітність безпосередньо перед керівником організації
5. Незалежна група верифікації та валідації (V&V-група):
   - Виконання функції верифікації (по домовленості з групою SQA)
   - Планування і проведення незалежного кваліфікаційного тестування інтегрованих компонент ПЗ або програмних продуктів з метою визначення їх відповідності потребам замовника
   - Координація планів робіт з менеджерами проектів відносно вимог до тестового середовища, строків і порядку передачі ПЗ на тестування
   - Представлення звітів (результатів) тестування менеджерам проектів для прийняття мір по виправленню ПЗ
   - Незалежність від менеджерів проектів в частині визначення об’ємів і методів тестування
   - Звітність перед керівником організації за дотримання порядку тестування і стан розроблених програмних продуктів
6. Група підтримки замовника:
   - Зв’язок із замовником з питань автоматизації ділових процесів
   - Підтримка процесів керування вимогами, навчання користувачів, супроводу (або допомога в їх виконанні на рівні окремих проектів)

Ролі на рівні проекту

1. Керівник проекту системи:
   - Повна фінансова відповідальність за виконання проектних домовленостей перед замовником
   - Керування розробкою складових створюваної продукції – проектів ПЗ, комплексу технічних засобів, засобів захисту інформації
   - Відповідальність за дії виконавців проекту
2. Системні аналітики:
   - Дослідження умов та потреб автоматизації діяльності організації-споживача
   - Системний аналіз вимог споживача і формування концепції системи
   - Контроль обґрунтованості проектних рішень, що приймаються
3. Група якості проекту:
   - Контроль якості робочих продуктів, створених процесами ЖЦ (на

відповідність стандартам, методикам тощо)

- Звітність тільки керівнику проекту
- Може бути відсутньою, якщо на рівні організації діє незалежна група якості

4. Група V&V проекту:
   - Перевірка відповідності робочих продуктів, вироблених на певному етапі ЖЦ, вимог до них, встановлених на попередньому етапі
   - Може виконувати тестування окремих компонент ПЗ, а також системне (інтеграційне) тестування ПЗ, виробленого в проекті
   Звітність тільки керівнику проекту
5. Менеджер проекту ПЗ:
   - Повна відповідальність за усі проектні рішення та дії, пов'язані з розробкою ПЗ в проекті
   - Підбір і контроль ресурсів проекту, а також графіка робіт
   - У великих або розподілених програмних проектах може бути

Архітектура процесів життєвого циклу

декілька менеджерів (по підсистемам або рівням проекту ПЗ)

6. Проектувальники:
   - Прийняття і документування проектних рішень по архітектурі і функціям ПЗ. Узгодження рішень з менеджером проекту ПЗ.
   - Дотримання стандартів якості (забезпечення досягнення характеристик якості)
7. Програмісти:
   - Програмування або моделювання компонентів ПЗ по проектним специфікаціям, підготованих проектувальниками
   - Дотримання стандартів якості при програмуванні (по зручності супроводу коду, зручності застосування програм)
   - Відладка та автономне тестування розроблених компонент
8. Група керування конфігурацією:
   - Виконання процесу конфігураційного керування версій ПЗ і робочих продуктів проекту ПЗ
9. Група супроводу:
   - Виконання процесу супроводу версій ПЗ і робочих продуктів проекту ПЗ під час дослідної експлуатації і під час встановленого періоду супроводу
   - Навчання користувачів
   - Виконання процесу розв'язання проблем
   - Можуть бути членами групи підтримки замовника
10. Група проекту ЛКМ:
    - При розробці системи «під ключ» проектування і монтаж ЛКМ для встановлення в організації споживача
    - Закупівля і встановлення КТЗ і загальносистемного ПЗ, пуско-налагоджувальні дії.

Процес програмної інженерії має ієрархічну структуру і включає множину процесів ЖЦ програмної системи. Вимоги до процесів ЖЦ ПС визначає міжнародний стандарт ISO/IEC 12207, а в Україні йому відповідає ДСТУ 3918-99. Процеси ЖЦ розподілені по трьом групам, які відображають функціональну направленість видів діяльності, які ці процеси регламентують:

- Основні процеси
- Процеси підтримки
- Організаційні процеси Основні процеси ЖЦ:

1. Придбання
   1.1. Підготовка придбання
   1.2. Вибір постачальника
   1.3. Моніторинг діяльності постачальника
   1.4. Прийом споживачем
2. Поставка
   2.1. Участь в тендері
   2.2. Укладення договору
   2.3. Випуск продукту (релізу)
   2.4. Підтримка приймання продукту
3. Розробка
   3.1. Виявлення вимог
   3.2. Аналіз вимог до системи
   3.3. Проектування архітектури системи
   3.4. Аналіз вимог до ПЗ системи
   3.5. Проектування ПЗ
   3.6. Програмування ПЗ
   3.7. Інтеграція ПЗ
   3.8. Тестування ПЗ
   3.9. Системна інтеграція
   3.10. Системне тестування
   3.11. Інсталяція ПЗ
   4. Експлуатація
   4.1. Функціональне використання
   4.2. Підтримка споживача
   5. Супровід

Процеси підтримки інтегруються з будь-якими іншими процесами, розв’язуючи задачі, допоміжні по відношенню до задач цих процесів, і забезпечують якість їх розв’язання в конкретних проектах.

1. Документування
2. Керування конфігурацією
3. Забезпечення гарантії якості
4. Верифікація
5. Валідація
6. Сумісний перегляд
7. Аудит
8. Керування розв’язанням проблем
9. Керування запитами на зміну
10. Забезпечення застосовуваності продукту (підтримка користувача)
11. Оцінювання проекту

Організаційні процеси ЖЦ

1. Керування
   1.1. Організаційне будівництво (формування стандартів, етики)
   1.2. Управління організацією
   1.3. Управління проектом
   1.4. Управління якістю
   1.5. Управління ризиком
   1.6. Вимірювання
2. Підтримка інфраструктури
   3.2. Оцінювання процесів
   3.3. Покращення процесів корпоративної політики, цілей, процесів,
   4. Забезпечення трудовими ресурсами
   4.1. Управління кадрами
   4.2. Навчання
   4.3. Управління знаннями (розповсюдження знань)
   5. Управління активами організації
   6. Управління програмою повторного використання

3. Вдосконалення 7. Доменна інженерія

3.1. Установлення процесів

Базовий процес організації

Сучасною концепцією процесу програмної інженерії є побудова базового процесу організації (БПО), який розробляється, супроводжується, оцінюється і покращується подібно до того як розробляються програмні продукти.

БПО є основою для визначення процесів усіх програмних проектів. Процеси на рівні проектів розробляються шляхом адаптації БПО до характеристик конкретного проекту.
Визначення БПО – це формалізований опис складу процесів ЖЦ, з яких мають побудуватись процеси розробки в програмних проектах, а також взаємозв'язків між елементами цих процесів.

З кожним процесом розробки пов'язуються:

- *Вимоги* до процесу, які вказують, «що» собою являє процес (що він буде робити)
- *Архітектура* і проект процесу, які описують, «як» процес буде визначений (які будуть елементи процесу і як будуть пов'язані)
- *Реалізація* опису процесу в рамках організації програмного проекту (створення елементі процесу і встановл. інтерфейсу)
- *Перевірка* і затвердження визначення процесу
- *Впровадження* процесу в середовище конкретного проекту.

При побудові БПО керівники організації-розробника ПС спочатку визначають склад основних процесів із числа рекомендованих, а потім підтримуючих і організаційних процесів. В своїх рішеннях керуються доцільністю включення певного процесу БПО з позицій необхідності виконання регламентованих процесом дій та їх достатності для досягнення цілей розробки якісного програмного продукту.

В мінімальній конфігурації процесів ЖЦ, крім процесу управління проектом, в БПО обов'язково має знайтись місце для процесу перевірки, а також процесу управління конфігурацією.

Побудований БПО має підтримувати використання моделей ЖЦ, застосування яких допускається в проектах організації. Широко розповсюджені такі основні класи моделей ЖЦ:

- Каскадні o Стандартна o Із зворотнім зв'язком o Пилоподібна
- Ітераційні o З прирощуваннями o Еволюційні
  - Спіраль
  - на
  - Швидк
  - ої розробки програм (RAD)

Вибір моделі суттєво залежить від двох факторів:

А) чи можна спочатку визначити практично повний набір функцій, які необхідно реалізувати в програмному продукті

Б) чи мають усі жадані функції поставлятись замовнику одночасно

Якщо А і Б, то вибираємо каскадні моделі

А і не Б – вибирається ітераційна модель з прирощуваннями

Не А і Б, а також бажана розробка прототипів для моделювання вимог – спіральна модель

Не А і не Б – модель швидкої розробки програм, при умові, що строки розробки не будуть чітко встановлені.

Керування проектами

Застосовано до програмної інженерію види діяльності по управлінню утворюють дворівневу структуру: загальне організаційне управління, керування виконанням програмних проектів.

Керування проектом (в будь-якій галузі) – це область знань, навиків, інструментарію та прийомів для досягнення цілей проектів в рамках узгоджених параметрів якості, бюджету, строків та інших обмежень. Сучасна концепція керування проектом основана на принципі інформативного вимірювання процесів ЖЦ проекту, його ресурсів і створюваних робочих продуктів.

Керування проектом включає наступні кроки:

1. Ініціація проекту і визначення його меж. Визначення вимог до проекту за допомогою застосування методів оцінювання вимог з різних точок зору: технічної, технологічної, фінансової, соціально-політичної)
2. Планування. Передбачає:
   - Вибір моделі ЖЦ проекту і оцінювання процесів ЖЦ в контексті придатності для задовільнення вимог до проекту, адаптацію БПО.
   - Ієрархічну декомпозицію задач проекту, їх специфікацію поряд з встановленими вимогами, асоційованими робочими продуктами.
   - Оцінки об’ємів робіт, трудомісткості і вартості реалізації проекту
   - Розподіл ресурсів по задачам з врахуванням графіку проекту і з позицій раціонального використання персоналу проекту, обладнання та матеріалів.
   - Керування ризиком проекту по критеріям вартості розробки, тривалості проекту і якості програмних продуктів
   - Керування якістю - застосування процедур планування та контролю якості виконання процесів і будь-яких робочих продуктів цих процесів, а також верифікація та валідація продуктів.
   - Керування планом проекту. Необхідність такого керування обумовлена часто змінюваними вимогами замовника і умовами виконання проекту. Це робить процес планування проекту ітеративним
3. Введення в дію. Передбачає виконання обраних процесів ЖЦ у відповідності з планом поряд із змінами, моніторингом та регулюванням процесів і складанням звітів для зацікавлених сторін (керівництва організації, замовників, співвиконавців)
4. Огляд і оцінка виконання проекту. Передбачає виконання всеосяжної перевірки діяльності по проекту і оцінки його руху до встановлених цілей. Проводиться у встановлених критичних точках проекту. Оцінюються не тільки результати виконання процесів, але й ефективність застосованих методів та інструментів, а також продуктивність роботи колективу проекту і можливі труднощі. При необхідності здійснюється регулювання

5. Закриття проекту. Передбачає припинення проекту після завершення процесів та досягнення цілей. Оцінюється успішність проекту по відношенню до встановлених критеріїв. ПС передається в експлуатацію.

Процес вимірювання при керуванні проектами

Сучасний рівень розвитку програмної інженерії дозволяє вивести проблему якості ПС за рамки простого питання «працює система чи ні?» формулюючи її так: «наскільки точно створена система задовольняє встановленим вимогам до її якості». Відкладувати пошук відповіді на це питання до моменту завершення розробки – занадто великий ризик як для замовника, так і для розробника. Вимірювання мають стати невід'ємною частиною ЖЦ проекту.

Вимірювання – отримання об'єктивних даних про стан продуктів, процесів та ресурсів розробки ПС з метою побудови прогнозуючих та оціночних моделей, які застосовуються для керування проектом і вдосконалення процесів організації.

Виконання вимірювань в проекті це багатокроковий процес, який включає наступні кроки:

1. *Визначення цілей програми вимірювання*
2. *Побудова процесу вимірювань*. Модель розроблюється таким чином, щоб забезпечити побудову точних і аргументованих відповідей на питання, підкріплених кількісними оцінками, основаних на вимірюваних величинах.
3. *Вибір базових мір*. В їх число як мінімум мають входити:
   - Розмір і складність програмного продукту, на основі яких може бути оцінена трудомісткість та вартість розробки
   - Продуктивність праці окремих спеціалістів і колективу проекту в цілому Міри вимірювань атрибутів якості
4. *Організація збору даних для виконання вимірювань*. Дані, що збираються мають забезпечувати не тільки високі передбачувальні можливості вимірювань. Але й бути достатньо «дешевими» з точки зору їх отримання. Збір і обробка даних для вимірювань є трудомістким процесом, який вимагає підтримку керівників організації і додаткових інвестицій в розробку ПС.
5. *Застосування моделей*.

Підходи до підвищення якості програмних систем.

Перший крок полягає у впровадженні спеціально призначених для цього підтримуючих процесів, а саме процесів гарантування якості, верифікації, валідації, сумісного перегляду та аудиту.

Процес гарантування якості (процес SQA) забезпечує гарантії, що програмні продукти і процеси в життєвому циклі проекту відповідають вимогам. Ці гарантії основані на тому, що SQA планує і здійснює контроль і здійснення допомоги в тій діяльності по проекту, що безпосередньо пов'язана із «вбудовою» в програмні продукти тих властивостей, що забезпечують якість. SQA встановлює стандарти та контролює їх дотримання в ході ЖЦ. Мета SQA – встановити чому допускаються помилки і як вони можуть бути виправлені. Об'єктом досліджень SQA є процеси ЖЦ ПС, а не програмні продукти.

Процеси верифікації та валідації визначають, чи дійсно продукти певного етапу процесу розробки і супроводу ПС відповідають вимогам, які до них висуваються. Задача цих процесів

– перевірити та підтвердити, що кінцевий програмний продукт буде відповідати своєму призначенню і задовольняти користувачів.

Методи які використовують як в цілях SQA так і в цілях V&V ділять на статичні та динамічні.

Статичні методи - дослідження документації, інспекції, ревізії, аналіз потоків даних, аналіз дерев подій і дерев відмов, аналіз складності алгоритмів і т.д.

До *динамічних методів* відносять тестування, імітаційне моделювання та символічне виконання.

Керування ризиком в проекті.

Ризик виникає там, де є невизначеність, пов'язана з настанням якої-небудь небажаної події і є можливість потерпіти збитки внаслідок цієї події.

Ризик проекту ПС можна визначити як можливість зниження якості кінцевого продукту, перевищення вартості його розробки, затримки завершення розробки або зриву проекту із-за неефективності і недосконалості процесів ЖЦ ПС.

Величина ризику - це добуток серйозності наслідків небажаної події в проекті на ймовірність появи цієї події.

Ефективне керування ризиком полягає в прийнятті компромісних рішень по оцінюванню трудомісткості позбавлення від визначеного ризику з однієї сторони, і величини негативного впливу цього ризику на якість робочих продуктів і процесів – з іншої.

Підвищення зрілості організації.

Зрілість організації можна охарактеризувати як ступінь чіткості (ясності) визначення, управління, вимірювання, контролю і виконання процесу розробки ПС в організації.

Модель зрілості являє собою шаблон для оформлення маленьких еволюційних кроків для покращення процесу у вигляді 5-и рівнів зрілості.

- Рівень 1 – початковий. Включений в модель тільки з метою утворення точки підрахунку для оцінювання наступних покращень процесу . Характеризується тим, що процес розробки ПС неструктурований та хаотичний, а бюджет, графік і якість розробки непередбачувані.
- Рівень 2 – повторюваний. На цьому рівні керування проектом націлено на контроль дотримання планів по вартості, тривалості і функціональності розробки. Дисципліна розробки дозволяє застосовувати відпрацьовані засоби управління неодноразово у схожих проектах. Розробка нових проектів ведеться на основі накопиченого досвіду і у відповідності з основними стандартами в області програмування
- Рівень 3 – фіксований. Процес програмної інженерії в організації затверджений, стандартизований і документований. Впроваджена програма навчання штату розробників ПС і менеджерів. Колективи окремих проектів слідують базовому процесу розробки в організації і налаштовують його для досягнення цілей конкретного проекту.
- Рівень 4 – керований. Досягається мета кількісної оцінки якості програмних продуктів і процесу розробки в рамках єдиної програми вимірювань. Здійснюється збір і наліз даних по проектам, що дає можливість управляти ризиком проекту і при необхідності повертати процес у встановлені рамки.
- Рівень 5 – оптимізований. Забезпечується неперервне покращення процесу завдяки наявності засобів кількісної оцінки його слабких і сильних сторін. Дані про ефективність

процесу розробки використовуються для проведення аналізу в цілях переходу на нові технології і вдосконалення процесу розробки в організації. Дані про нові засоби інженерії вивчаються і розповсюджуються по організації.

Лекція 5. Технічне завдання

1. Формування технічного завдання
2. Тестування білої скрині

ДСТУ 19.201-78 «Технічне завдання. Вимоги до змісту та оформленню»

Стандарт встановлює порядок побудови і оформлення технічного завдання на розробку програми або програмного виробу для обчислювальних машин, комплексів та систем незалежно від їх призначення та області застосування.

ТЗ має містити наступні розділи:

- Вступ
- Підстави для розробки
- Призначення розробки
- Вимоги до програми
- Вимоги до програмної документації
- Техніко-економічні показники
- Стадії та етапи розробки
- Порядок контролю і прийому ТЗ може містити додатки.

В залежності від особливостей програми допускається уточнювати зміст розділів, вводити нові розділи або об’єднувати окремі з них.

ЗМІСТ РОЗДІЛІВ

1. В розділі «Вступ» вказують найменування, коротку характеристику області застосування програми і об’єкту, в якому будуть застосовувати програму
2. В розділі «Підстави для розробки» мають бути вказані:
   - Документи, на підставі яких ведеться розробка
   - Організація, яка затвердила цей документ і дата затвердження
   - Найменування і умовне позначення теми розробки
3. «Призначення розробки» - функціональне і експлуатаційне призначення програми
4. Розділ «Вимоги до програми» має містити наступні підрозділи:

- Вимоги до функціональних характеристик – вимоги до складу виконуваних функцій, організації вхідних та вихідних даних, часовим характеристикам тощо…
- Вимоги до надійності – вимоги до забезпечення надійного функціонування (контроль вхідної та вихідної інформації, час відновлення після збоїв)
- Умови експлуатації – температура повітря, вологість та інші для вибраного типу носіїв даних, при яких мають забезпечуватись задані характеристики, а також вид обслуговування, кількість та кваліфікацію персоналу;
- Вимоги до складу і параметрам технічних засобів – вказують необхідний склад технічних засобів із вказуванням їх основних технічних характеристик
- Вимоги до інформаційної та програмної сумісності – вимоги до інформаційних структур на вході та виході, методам розв'язку, кодам програми, мовам програмування і програмним засобам, які використовує програма. При необхідності має забезпечуватись захист інформації та програм
- Вимоги до маркування та пакування
- Вимоги до транспортування та зберігання
- Спеціальні вимоги

5. В розділі «Вимоги до програмної документації» має бути вказаний попередній склад програмної документації і спеціальні вимоги до неї
6. В розділі «Техніко-економічні показники» мають бути вказані:
- Орієнтовна економічна ефективність
- Приблизна річна потреба в програмі
- Економічні переваги розробки у зрівнянні з аналогами

7. В розділі «Стадії та етапи розробки» встановлюють необхідні стадії розробки, етапи і зміст робіт (перелік програмних документів, які мають бути розроблені, узгоджені і затверджені), строки розробки і виконавців

В додатках до ТЗ за необхідністю приводять:
- Перелік науково-дослідницьких та інших робіт, які пояснюють необхідність розробки

- Схеми алгоритмів, таблиці, розрахунки та інші документи, які можуть бути використані при розробці
- Інші джерела розробки

Тестування «білої скрині»

Програміст пише програми і сам їх тестує. Ця технологія називається тестуванням «білої скрині» (white box) або «скляної скрині» (glass box).
Переваги тестування «білої скрині»:

1. Направленість тестування. Програміст може тестувати програму по частинах, розробляти спеціальні тестові підпрограмки, які викликають модель, що тестується, і передають йому необхідні дані. Окремий модуль завжди легше протестувати саме як «білу скриню»
2. Повне охоплення коду. Програміст завжди може визначити, які саме фрагменти коду працюють в кожному тесті. Він бачить які гілки залишились не протестованими і може підібрати умови при яких вони будуть виконані
3. Керування потоком. Програміст завжди знає, яка функція має виконуватись в програмі наступною і яким має бути її поточний стан. Тут програміст може користуватись таким зручним засобом як відладчик
4. Відстежування цілісності даних. Програмісту відомо, яка частина програми має змінювати кожен елемент даних. Відстежуючи стан даних, він може виявити такі помилки як зміна даних не тими модулями, їх неправильна інтерпретація або невдала організація. Програміст також може самостійно автоматизувати тестування.
5. Внутрішні граничні точки. В коді видно ті граничні точки програми, які скриті від погляду тестерів

«чорної скрині». Наприклад, для виконання певної дії можна використати декілька алгоритмів і невідомо який з них вибрав програміст. Переповнення буфера – програміст одразу може сказати при якій кількості даних виникне ця помилка.

6. Тестування, яке визначається вибраним алгоритмом. Для тестування обробки даних, яка використовує складні обчислювальні алгоритми, можуть

знадобитись спеціальні технології. Тестування «білої скрині» розглядають як частину програмування. Програміст виконує цю роботу постійно.

Структурне тестування є одним з видів тестування «білої скрині». Головна ідея – вибір правильного програмного шляху. Структурне тестування має потужну математичну теоретичну основу, але більшість тестерів використовують функціональне тестування.

Тестування програмних шляхів: критерії охоплення.

Протестувати усі програмні шляхи неможливо. Тому серед програмних шляхів виділяють ті, що необхідно протестувати обов'язково. Для відбору використовуються критерії охоплення (coverage criteria). Їх також називають логічними критеріями охоплення або критеріями повноти. Найчастіше використовуються критерії:

- Охоплення рядків
- Охоплення розгалужень
- Охоплення умов

Коли тестування узгоджується з цими критеріями кажуть про тестування шляхів.

Критерій охоплення рядків – найслабший. Він вимагає, щоб кожен рядок програми був виконаний хоча б один раз. Він не підходить вже коли зустрічається хоча б один оператор умови.

Наприклад :

If (a < b and c = 5) then

    Do something;

Для перевірки коду необхідно проаналізувати 4 варіанти:

1) a < b and c = 5 : (do something виконується)
2) a < b and c ≠ 5 : (do something не виконується)
3) a ≥ b and c = 5 : (do something не виконується)
4) a ≥ b and c ≠ 5 : (do something не виконується)

Для виконання коду необхідно перевірити тільки 1-й варіант

При тестуванні за критерієм охоплення розгалужень програміст перевіряє варіант 1 і ще один з трьох інших варіантів.

Найбільш суворим є критерій охоплення умов. При такому необхідно перевірити усі 4 варіанти.

Критерії охоплення корисні але їх недостатньо. Приклад

A = B/C

Якщо C ≠ 0, той цей код успішно виконується і критерії охоплення виконуються. Але при C = 0 програма припинить виконання. Різниця в цих двох варіантах не в шляху, а в даних.

Програмний шлях називається чутливим до помилок, якщо при його проходженні помилки можуть проявитись. Якщо ж помилки обов'язково проявляться в проходженні даного шляху, то такий шлях називається таким, що знаходить помилки.

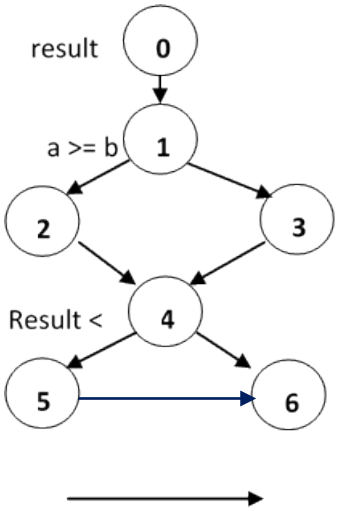
Приклад тестування потоків керування

```
Int TestFunction(int a, int b)
{
        int result = 0;
if(a >= b) result = a-b; else
result = b-a; if(result < 5) result = 0; return result;
}
```

Керуючий граф програми:

result

result return



Тестовий набір, сформований згідно з критерієм покриття команд, має вигляд (X, Yет) = {(9,5,0), (5,9,0)}. В тестованій по даному набору програмі виконуються усі команди і виконуються наступні шляхи:

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 5 \to 6. \end{cases}$$

Тестовий набір, сформований згідно з критерієм покриття розгалужень має вигляд (X, Yет) = {(9, 5, 0), (0, 255, 255)}. В тестованій по даному набору програмі виконуються усі розгалуження і виконується наступна множина шляхів:

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 6. \end{cases}$$

Тестовий набір, сформований згідно з критерієм покриття маршрутів, має вигляд (X, Yет) = {(9, 5, 0), (0, 255, 255), (255, 0, 255), (5, 9, 0)}.

$$M = \begin{cases} 0 \to 1 \to 2 \to 4 \to 5 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 6 \\ 0 \to 1 \to 2 \to 4 \to 6 \\ 0 \to 1 \to 3 \to 4 \to 5 \to 6. \end{cases}$$

Лекція 4. Моделі життєвого циклу програмного забезпечення (ПЗ)

1. Життєвий цикл програмного забезпечення
2. Моделі послідовного виконання стадій
3. Ітераційні моделі
4. Спіральна модель
5. Модель еволюційного прототипування

1. Життєвий цикл ПЗ - стадії, що проходить програмний продукт від появи ідеї до її реалізації в коді, імплементації у бізнес і подальшої підтримки. Моделі життєвого циклу багато в чому зумовлюють і методології розробки ПЗ. Життєвий цикл ПС визначається як «весь період існування системи від початку розробки до завершення її використання» (ДСТУ 2941-94. Розробка систем. Терміни і визначення).

ЖЦ поділяється на впорядковані стадії, основні з яких:

- Визначення потреб
- Аналіз вимог і оформлення концепції
- Розробка
- Виробництво
- Впровадження/продаж
- Експлуатація
- Супровід і підтримка
- Вилучення з експлуатації

Всередині кожної з цих стадій відбувається подальша деталізація по більш дрібним стадіям. Моделі ЖЦ описують взаємозв’язки стадій.

Далі будемо розглядати стадії ЖЦ, які пов’язані з процесом розробки ПС, основні з яких:

- Аналіз вимог
- Проектування (попереднє і детальне)
- Реалізація
- Тестування

Найбільш відомі типи моделей ЖЦ: послідовні та ітераційні. Ці моделі на практиці можуть змішуватись, утворюючи змішані моделі ЖЦ.

Моделі ЖЦ можуть використовуватись для:

- Організації, планування. Розподілення ресурсів (затрат праці і часу) і керування проектом розробки
- Організації взаємодії з замовниками і визначення складу документів (робочих продуктів), які розроблюються на кожній стадії
- Аналізу і оцінювання розподілу ресурсів і затрат на протязі ЖЦ
- Наглядного опису або в якості основи для проведення фінансових розрахунків з замовниками
- Проведення емпіричних досліджень з метою визначення впливу моделей на ефективність розробки і загальну якість програмного продукту

Рекомендації по можливому відображенню процесів ЖЦ на основні моделі розробки приведені в Керівництві по застосуванню стандарту ISO/IEC 12207 (Guide for ISO/IEC 12207 – Software life cycle processes).

2. Моделі послідовного виконання стадій

Каскадна модель

Каскадна (Waterfall) або стандартна модель – найбільш відома модель розробки, яка пропонується стандартами як базова. Ця модель характеризується стадіями, які виконуються послідовно. Кожна стадія має бути завершена до переходу до наступної. Створені в ній робочі продукти після їх верифікації та валідації мають бути «заморожені» і передані на наступну стадію в якості еталону. Користувач бачить працюючий програмний продукт в самому кінці розробки. Найбільш жорстке обмеження цієї моделі – необхідність «заморозки» вимог. При цьому, для того, щоб мінімізувати ризик, допускаються лише невеликі зміни.

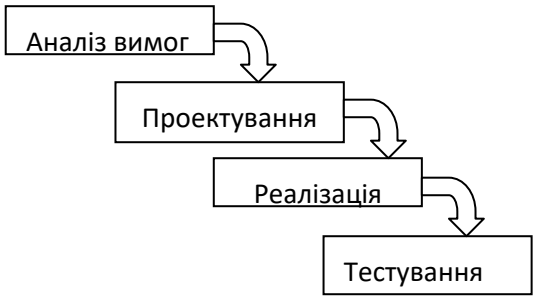


Рис 1. Каскадна модель

З точки зору якості ПС в цій моделі вартість виправлення дефектів на стадії тестування найбільша (у зрівнянні з іншими моделями), оскільки тестування відбувається в самому кінці розробки. Через нестачу часу на переробки і тестування існує значний ризик випуску ПС з серйозними дефектами.

Каскадна модель із зворотнім зв'язком

Ця модель розширює стандартну модель включенням в неї циклів зворотного зв'язку для повернення на попередню стадію при зміні вимог, проекту і по результатам інспекцій або дій по V&V

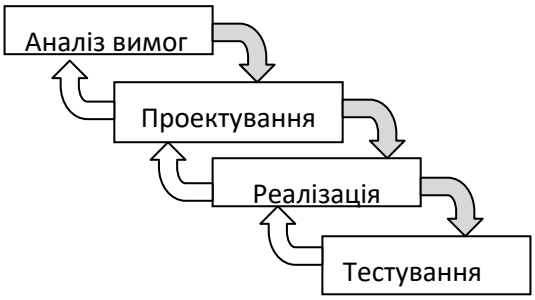


Рис 2. Каскадна із зворотнім зв'язком

Процеси V&V, які виконуються після завершення кожної стадії розробки, грають в цій моделі важливу роль.

Характеристики каскадної моделі:

- Послідовне впорядкування стадій
    - Формальні перевірки по завершенні кожної стадій (інспекції, технічні огляди)
    - Наявність документованих вимог і проекту

Переваги каскадної моделі:

- Застосування формальних перевірок дозволяє вчасно виявляти дефекти
- Чіткі критерії початку і завершення стадій
- Чіткі вимоги і цілі проекту

V-подібна модель

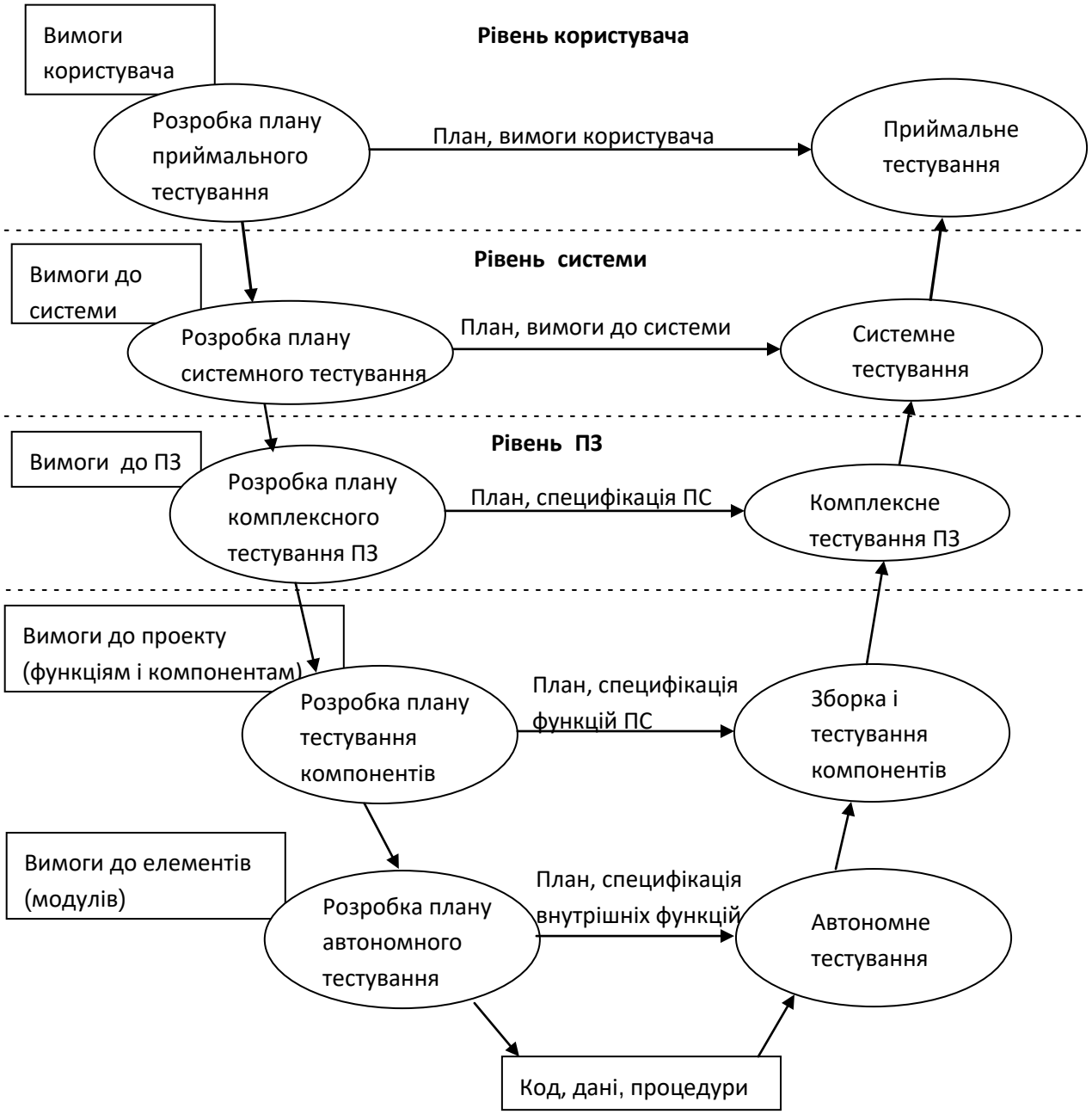


Рис.3. V-подібна модель

V-подібна (V-shape) модель розширює каскадну модель і включає до неї дії по ранньому плануванню тестування.

В цій моделі тестування розглядається як неперервний процес, інтегрований в процес розробки ПС. Він включає два взаємопов’язаних під процесів – планування тестування в рамках процесів розробки системи (ліва гілка) і проведення тестування відповідних об’єктів (права гілка). На практиці задачі тестування ПЗ і системного тестування часто об’єднуються в один процес,

однак для складних ПС тестування технічних характеристик має виконуватись окремо.

Характеристики V-подібної моделі:

- Перевірка і оцінка тестопридатності вимог на ранніх стадіях розробки (з допомогою аналізу, який виконується під час тестування)
- Наявність документованих тестових вимог Переваги V-подібної моделі:
- Забезпечує зворотний зв'язок з користувачем на ранніх стадіях ЖЦ
- Покращує планування і розподіл затрат на тестування
- Чіткі документовані цілі тестування

Каскадна модель з прототипуванням (пилкоподібна модель)

Модель є модифікацією V-подібної моделі з включенням в неї прототипів для моделювання вимог і проекту

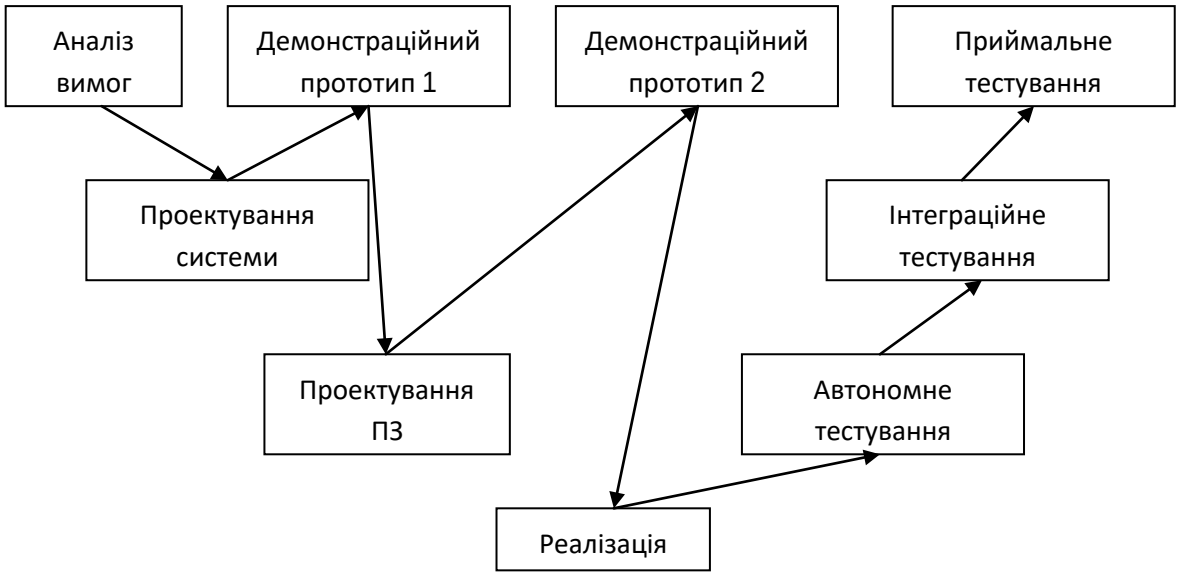


Рис.4. Каскадна модель з прототипуванням

Прототипи слугують для демонстрації і після розробки проекту їх викидають, а реалізація проекту може виконуватись в іншому середовищі.

- Характеристика: для аналізу і моделювання проектних рішень застосовуються прототипи
    - Переваги: усуває проблеми, пов'язані з неповнотою і нечіткістю вимог

Ризики застосування послідовних моделей:

- Вимоги не повністю зрозумілі
- Система занадто велика, щоб бути реалізованою одразу
- Швидкі зміні в технологіях
- Часта зміна вимог
    - Користувач не може використовувати проміжні результати

Коли краще застосувати послідовну модель:

- Вимоги зрозумілі і не будуть суттєво мінятись
- Система має невеликий розмір і складність
- Усі можливості мають бути реалізовані одразу
- Нова система розробляється на заміну старої і необхідно повністю замінити стару систему

3. Ітераційні моделі

Ітераційні моделі загалом можна розділити на два класи: моделі з приростом (Incremental) і еволюційні (Evolutionary). У відповідності з цими моделями програмний продукт розроблюється ітераціями, і кожна ітерація закінчується випуском працездатної версії програмного продукту. Основна відмінність між моделями – підхід до визначення вимог.

Ітераційні моделі з приростом

Програмний продукт розробляється ітераціями – з додаванням на кожній функціональних можливостей. При цьому спочатку визначаються усі вимоги до ПС, і можливо розроблюється попередній проект. Подальша розробка ПС розбивається на ітерації. В першій ітерації реалізується набір основних вимог, які забезпечують базову функціональність. Інші ітерації реалізуються в порядку критичності вимог для кінцевого користувача. При появі в середині ітерації нового набору вимог. вони відкладаються до реалізації наступної версії. В реальному житті це допущення може порушуватись і допускається перегляд вимог.

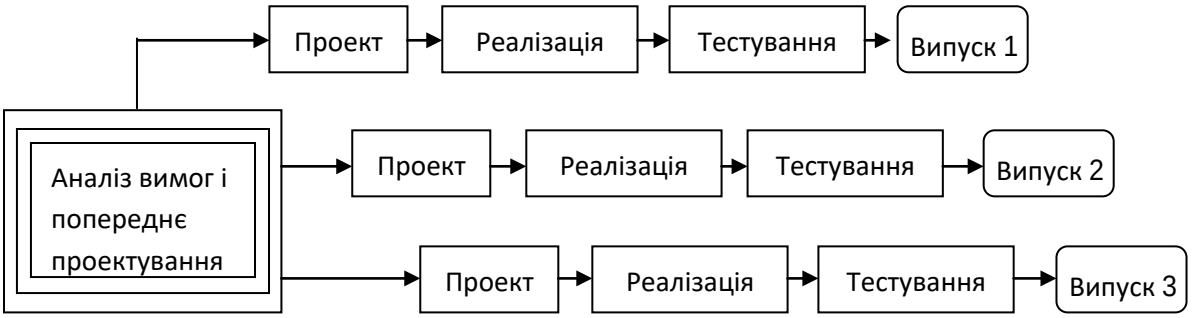


Рис 5. Ітераційна модель з приростом

В різних моделях цієї групи ітерації можуть виконуватись послідовно або з перекриттям (нова ітерація починається до завершення попередньої). Ітераційні моделі з приростом широко використовуються для розробки комерційних програмних продуктів, які розвиваються на протязі довгого періоду часу або для яких зовнішні вимоги змінюються слабо.

Характеристики ітераційних моделей з приростом:

- Аналіз і проектування виконуються для усієї системи
- Базові функціональні вимоги реалізуються першими
- Інші вимоги реалізуються в наступних версіях
- Проміжні версії придатні для використання

Переваги ітераційних моделей з приростом:

- Критичні функції реалізуються в першу чергу
- Критичні функції тестуються більш ретельно
- Найменш критичні задачі реалізуються останніми, що мінімізує наслідки відмови із-за дефектів
- Завершення першої версії остаточно затверджує вимоги і проект
- Раннє планування виконання тестування
- Раннє виявлення дефектів користувачами

Ризики, пов’язані з вибором моделі:

- Вимоги не повністю зрозумілі
- Вимоги не стабільні
- Усі можливості мають бути реалізовані одразу
- Швидкі зміни в технології

Коли краще застосовувати модель:

- Вимагається швидка реалізація основних можливостей
- Якщо проект системи можна природним чином поділити на незалежні частини

Еволюційні моделі

На відміну від моделей з приростом, еволюційні моделі застосовуються в тих випадках, коли усі вимоги не можуть бути визначені одразу або відомо, що вони можуть змінитись. Розробка проекту по цим моделям також виконується ітераціями. Але кожна ітерація охоплює усі стадії розробки, від аналізу вибраного набору вимог до випуску версії. На кожній ітерації виконується прототипування вимог і проекту. До найбільш відомих еволюційних моделей відносяться спіральна модель і модель еволюційного прототипування.

3. Спіральна модель

Розроблена Боемом. Відображає керований ризиком процес еволюції проекту від аналізу до готовності продукту.

На кожному витку спіралі (стадії) виконуються наступні дії:

1. Визначаються цілі стадії. Розглядаються альтернативні рішення для досягнення цих цілей
2. Проводиться оцінювання цих рішень. Ідентифікуються ризики завершення стадії і виконується їх аналіз. Приймаються рішення про продовження або завершення стадії 3. Розробляються робочі продукти стадії та план для наступної стадії

3. Останній виток спіралі може мати структуру каскадної моделі.

Види розглядуваних ризиків – ризики, які стосуються технічних аспектів розробки, фінансові ризики, ризики експлуатації. Спіральна модель застосовується для складних проектів або в тих випадках, коли проблеми проекту недостатньо зрозумілі.

Характеристики спіральної моделі:

- Перший прототип моделює концепцію. Результатом є план вимог. Перед переходом до розробки наступного прототипу виконується аналіз ризику

- Другий прототип моделює вимоги до ПЗ. Результатом є план розробки. Виконується аналіз ризику
- Третій прототип моделює проект. В результаті створюється інтегрований і протестований прототип. Виконується аналіз ризику.
- Останній прототип (робочій) використовується як основа для детального проектування, кодування та тестування.

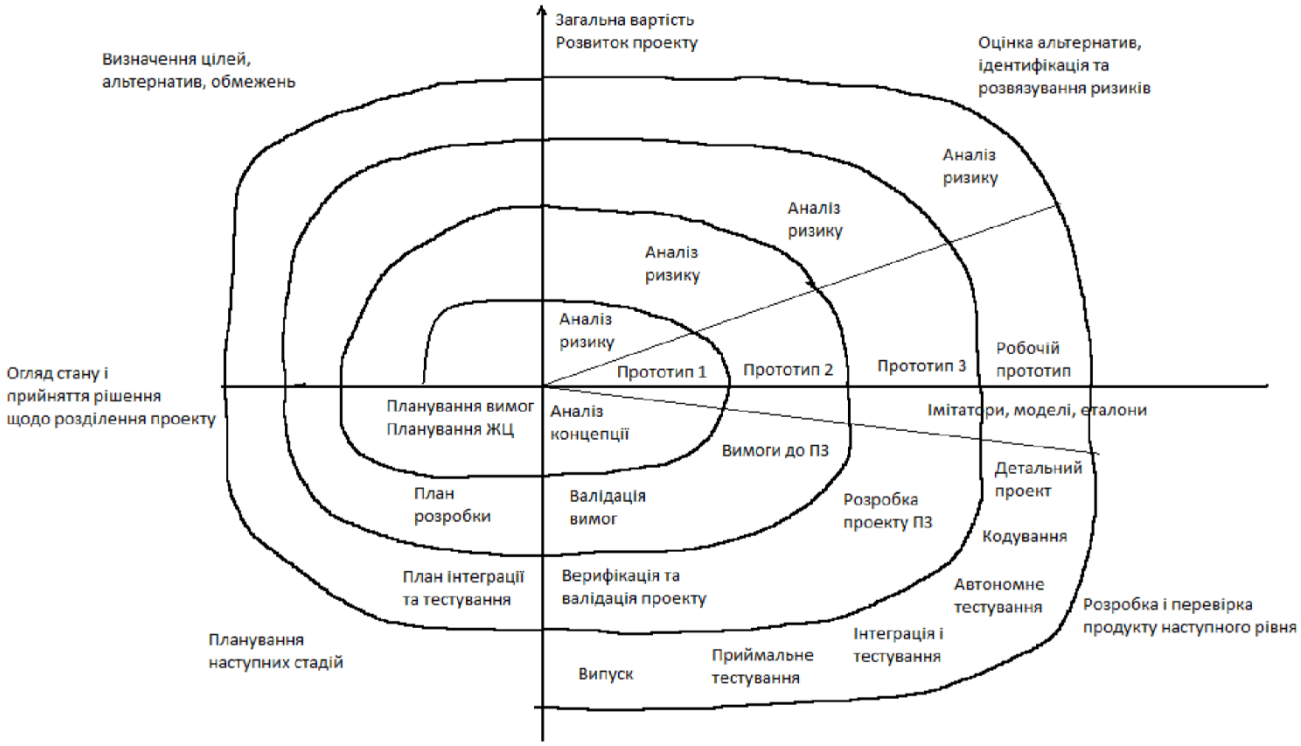


Рис. 6. Спіральна модель

Ризики пов'язані з вибором моделі:

- Усі можливості мають бути реалізовані одразу
- Проект неможна природним чином розділити на незалежні частини

Коли краще застосовувати модель:

- Проект крупний, складний і вимоги не можуть бути визначені одразу
- Нова технологія і вимагається її вивчення
- Користувачі не можуть чітко сформулювати вимоги
- Вимагається рання демонстрація можливостей

Подальшим розвитком цієї моделі є Win-Win Spiral Model, яка основана на залученні до розробки різних категорій учасників проекту і визначенні умови успіху системи, які обговорюються на кожній ітерації.

5. Модель еволюційного прототипування.

Ця модель основана на застосуванні еволюційного прототипування в рамках усього ЖЦ розробки (а не тільки моделювання вимог). В літературі вона часто називається моделлю швидкої розробки програм (RAD – Rapid application development). Моделювання включає наступні кроки:

Крок	Дія
1. Аналіз застосовуваності моделі	Вивчення можливості застосування моделі для проекту
2. Обстеження замовника	Вивчення потреб користувача та розробка плану створення прототипу
3. Ітерація розробки функціонального прототипу	Створення і узгодження прототипу інтерфейсу користувача. Визначення не функціональних вимог і стратегії реалізації системи
4. Ітерація проектування і побудови	Побудова протестованої системи, яка задовольняє усім функціональним та не функціональним вимогам. На кроках 3 і 4 розробники визначаються прототипи, узгоджують строки розробки, побудову і перевірку прототипів. Ці кроки виконуються
	ітеративно і включають три ітерації: початкове ознайомлення, уточнення. Узгодження
5. Реалізація	Встановлення системи в середовищі замовника, розробка документації і навчання

Ця модель застосовується для розробки не критичних бізнес-програм. Для яких найбільш важливими є функціональні можливості. Її застосування передбачає тісну взаємодію розробника і користувача.

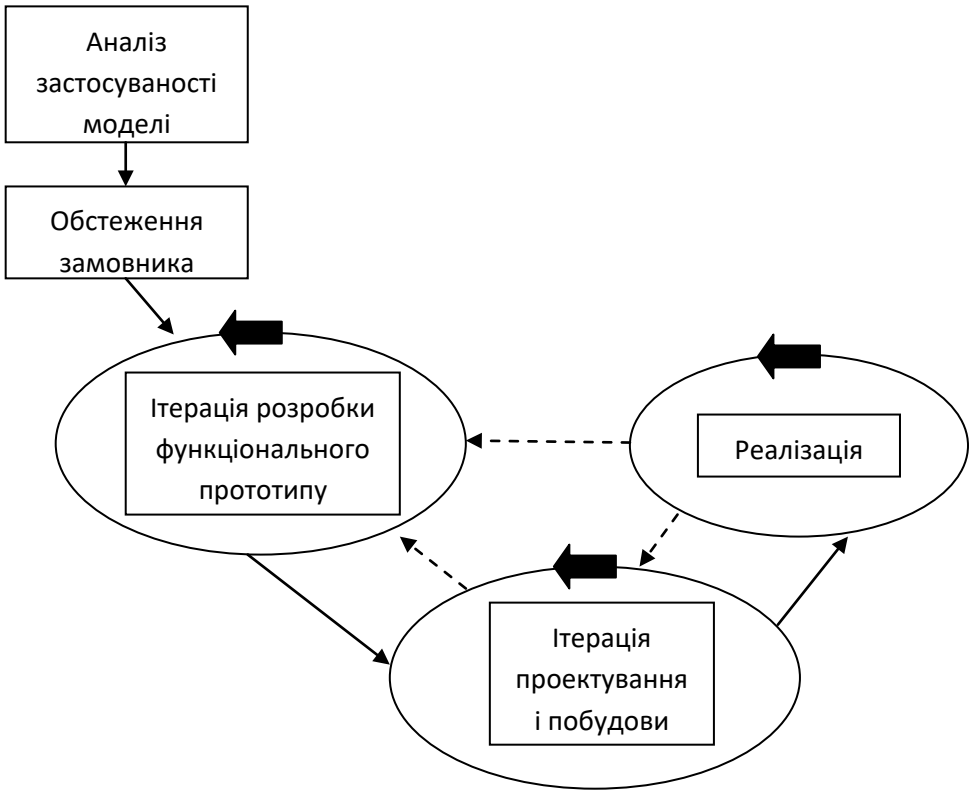


Рис. Модель еволюційного прототипування

Характеристики моделі:

- Гнучкість. Можливість швидко реагувати на зміни і розширення вимог
- Пріоритети функціональних характеристик перед технічними (якості)

Переваги моделі:

- Раннє виявлення дефектів в інтерфейсі
- Швидка демонстрація функціональних можливостей

Ризики пов'язані з вибором моделі:

- Від розробника вимагається хороше володіння CASE-засобами та інструментами
- Програми не моє бути критичною
- Вимагається наявність потужних CASE-засобів

Коли краще застосовувати модель

- Користувачі не можуть чітко сформулювати вимоги
- Вимагається рання демонстрація можливостей

Вибір моделі суттєво залежить від двох факторів:

А) чи можна спочатку визначити практично повний набір функцій, які необхідно реалізувати в програмному продукті

Б) чи мають усі жадані функції поставлятись замовнику одночасно

Якщо А і Б, то вибираємо каскадні моделі

А і не Б – вибирається ітераційна модель з прирощуваннями

Не А і Б, а також бажана розробка прототипів для моделювання вимог – спіральна модель

Не А і не Б – модель швидкої розробки програм, при умові, що строки розробки не будуть чітко встановлені.

Лекція 3. Серія тестів

Перший цикл тестування.

Отримуємо програми і наступний опис її функціонування:

Призначення програми – скласти два введених користувачем числа. В кожному з чисел може бути одна або дві цифри. Програма відображує введені числа і після цього виводить їх суму. Введення кожного числа закінчується натисненням клавіші Ентер. Запускається програма з допомогою команди ADDER (adder.exe).

КРОК 1. Простий і найбільш очевидний тест

В програмах, представлених для першого формального тестування, часто одразу виникає збій. Тому варто виконати найпростіший тест.

Дія	Що вібувається
Вводимо ADDER і натискаємо <Ентер>	Зверху екрану з'являється знак питання. Курсор мигає
Натискаємо 2	За знаком питання з'являється цифра 2
Натискаємо <Ентер>	В наступному рядку з'являється знак питання
Натискаємо 3	За другим знаком питання з'являється цифра 3
Натискаємо <Ентер>	В третьому рядку з'являється цифра 5. На декілька рядків нижче з'являється ще один знак питання

Елементарний тест програма проходить і отже вона робоча. Але проблеми все одно присутні.

Звіт про проблеми першого тесту.

1. Помилка проектування. Нема вказівок на те з якою саме програмою ви працюєте
2. Помилка проектування. На екрані нема жодних інструкцій. Звідки знати що робити? Відобразити інструкцію на екрані нескладно, а друкована документація може загубитись.

3. Помилка проектування. Як зупинити виконання програми? Ця інструкція має бути на екрані.
4. Помилка кодування. Цифра 5 виведена в стороні від доданків.

Обов'язково необхідно представляти детальний звіт про кожну проблему. Можна згрупувати помилки в один звіт, але цього робити не варто.

КРОК 2. Що ще має бути протестовано

Необхідно скласти серії тестів. Їх краще записати. Ці записи в подальшому приймуть вигляд формалізованих описів тестів. Такі документи можуть в подальшому використовуватись для перевірки наступних версій програми.

Приклад серії тестів.

Вхідні дані	***Очікуваний результат***	***Замітки***
99 + 99	198	Пара найбільших чисел які може скласти програма
-99 + -99	-198	В документації не сказано, що неможна скаладати від'ємні числа
99 + -14	85	Перше велике число може вплинути на інтерпретацію системою другого
-38 + 99	61	Перевіримо додавання від'ємного і додатного чисел
56 + 99	155	Друге велике число може вплинути на інтерпретацію системою першого
9 + 9	18	9 – найбільше число з однієї цифри
0 + 0	0	Програми часто злітають на нулях
0 + 23	23	Програма може неправильно обробляти 0 і тому його треба перевірити в якості обох доданків
-78 + 0	-78	

Загальна кількість тестів 39601. Допустимий діапазон (-99,99) - усього 199. Отже, 199^2. Це без врахування будь-яких складних дій користувача (наприклад нанесення бекспейсів і делете).

Ефективніше усього перевіряти граничні умови. Якщо для двох тестів очікується однаковий результат, то тести належать до одного класу. В нашому випадку 81 тест відноситься до класу «пара однозначних додаткових чисел».

Для виконання слід вибирати ті тести з класу, на яких ймовірніше усього може відбутись збій програми. Класом можна назвати групу значень, які програма обробляє однаковим чином. А граничними значеннями класу є ті вхідні дані, на яких програма міняє свою поведінку. Границю завжди слід перевіряти з обох боків. Програмісти часто переконуються, що критичний фрагмент коду працює на одному із значень і забувають це зробити на другому.

КРОК 3. Перевірка недопустимих значень.

Програму тестують тому, що вона може не працювати.

На цьому етапі слід перевірити недопустимі значення.

В нашому випадку:

1. Додаткові числа і нулі обробляються правильно.
2. Не працює жоден тест з від'ємними числами. Після вводу другої цифри комп зависає. Очевидно програма не очікує вводу від'ємних чисел

КРОК 4. Трохи тестування в режимі «вільного польоту»

На цьому етапі включається інтуїція. Це етап в очікуванні виправлення вже виявлених недоліків. Розробляти нові тести немає смислу. Тому не слід втрачати час на планування. Можна провести дослідницьку роботу. Завжди записуйте те, що ви робите і що відбувається під час дослідницьких тестів. Приклад на нашій програмі:

Тест	***Чи цікавий тест***	***Зауваження***
100 + 100	Гранична умова: числа більша за допустимий максимум	Програма прийняла 10. Коли ви ввели другий нуль, програма повела себе так, ніби була натиснутий ентер. Те саме з другим числом. В результаті – сума 20
<Enter> + <Enter>	Відсутність введених даних	Коли ви натиснули Ентер, програма надрукувала 10, останнє введене вами число

123456 + 0	Побільше цифр	Програма прийняла перші дві цифри як і у випадку з цифрою 100. В майбутній версії програма має працювати з більшими числами. Як тоді вона має реагувати на такі ситуації?
1,2 + 5	Десятковий знак	Реакція на десятковий знак така ж як і на Ентер
A + b	Недопустимі символи	Ви натиснули Ентер – після <A> програма зависла
<Ctlr>+<A> + <Ctrl>+<B>, <F1>+<Esc>	Керуючі символи і функціональні клавіші часто є джерелами пролем	Для усіх комбінацій клавіш програма виводить графічні символи, потім після натиснення на Ентер зависає

КРОК 5. Підсумки про недоліки програми

1. У програми дуже обмежений інтерфейс
2. Програма не працює з від'ємними числами. Найбільша обчислювана сума – 198, найменша – 0.
3. Третій ввідний символ програма інтерпретує як натиснутий Ентер
4. Поки не натиснуто Ентер будь-які символи сприймаються як допустимі
5. Програма не перевіряє чи дійсно введена цифра.

Якщо програміст не зовсім некомпетентний, то для таких результатів має бути причина. Скоріш за все програміст намагався зробити програму якомога меншою за розміром або швидкою.

Код обробки помилок займає пам'ять. Це ж стосується заголовків, повідомлень про помилки і інструкцій. Якщо програма дійсно має поміститись в маленьку фрагменті пам'яті то на все це місця немає. Це ж стосується часу. Час необхідне для перевірки вводу допустимих символів, для перевірки чи дійсно третя натиснута клавіша – це Ентер, і на друк повідомлень на екрані і на очистку змінних перед виконанням наступного завдання.

Для того, щоб все це вияснити не обхідно переговорити з програмістом.

Підсумки першого циклу тестування

Оскільки програма пройшла простий тест, булла розроблена серія формальних тестів для перевірки роботи з допустимими даними. Ці тести будуть використані і далі. Оскільки частину перевірок програма не пройшла, на планування подальших тестів поки доцільно не витрачати час. Замість цього було проведено ряд неформальних експериментів і виявлено, що програма дуже нестабільна. До написаних зауважень слід повернутись при наступному тестуванні програми.

ДРУГИЙ цикл тестування.

Програміст повідомив, що швидкість роботи є дуже важливою, а об'єм коду не має значення. Програміст поставив свої резолюції.

Помилка	Резолюція
На екрані нема назви програми	Не буде виправлена
На екрані нема інструкцій	Не буде виправлена. Вивід інструкцій уповільнить роботу програми
Як зупинити програму?	Виправлена. На екрані відображається підказка «Для виходу натисніть <Ctrl>+<C>»
Сума 5 виводиться в стороні від доданків	Виправлена
Програма зависає на від'ємних числах	Виправлена. Програма буде складати і від'ємні числа
Програма інтерпретує третій введений символ як Ентер	В стадії розробки (ще не виправлена)
Збій при вводі нечислових даних	Не проблема. Коментар: не вводьте нечислові дані
Збій при вводі керуючих символів	Не проблема
Збій при натисненні функціональних клавіш	Не проблема

КРОК 1. Уважно прочитайте резолюції програміста і визначте, що треба робити, а що ні.

Із резолюцій чітко видно які тести більше проводити не треба, а які будуть замінені новими. Резолюції програміста є ключем до створення нової серії тестів

КРОК 2. Проаналізуйте коментарі до помилок, які не будуть виправлені. Можливо необхідно провести додаткове тестування.

Для того, щоб добитись виправлення помилки, необхідно продемонструвати ситуацію в якій її появлення абсолютно недопустиме. Програма зависає при введені будь-якого недопустимого символу – необхідно продемонструвати програмісту, що це неправильно.

КРОК 3. Передивіться записи першого циклу, додайте нові зауваження і переходьте до тестування. Слід провести серію уже проведених тестів, для того щоб переконатись, що програма їх виконує. Після їх проведення ви помітили, що програма відображає підказку «Для виходу натисніть <Ctrl>+<C>» після кожної операції додавання. Можна скласти наступне зауваження:

Помилка проектування. На вивід на екран підказки тратиться зайвий час. Можна просто написати внизу екрана цю підказку перед початком роботи. Заодно і додати заголовок програми і короткі інструкції. Оскільки програма продовжує зависати при введенні недопустимих символів, то слід описати проблему ще раз. Можливий варіант виправлення – перевіряти кожний введений символ. Недопустимі ігнорувати і виводити повідомлення про помилку.

Головне – указати, що проблема серйозна.

Хороший тестер не той, хто виявить найбільше помилок і не той, хто примусить збентежитись навіть самого першокласного програміста. Кращим є той, хто досягне виправлення найбільшої кількості помилок.

Лекція 2. МІЖНАРОДНИЙ СТАНДАРТ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ ISO 9126

1. Мета, завдання та функції стандартизації
2. Організації зі стандартизації
3. Міжнародний стандарт якості програмних засобів ISO 9126

1. *Мета, завдання та функції стандартизації*

Стандартом називаються нормативно-технічні документи, які встановлюють єдині обов'язкові вимоги щодо типів, розмірів, якості, норм й інших особливостей продукції та послуг з метою упорядкування діяльності в певній галузі економічного використання ресурсів, підтримки техніки безпеки, підвищення якості продукції (процесів, робіт, послуг).

Мета стандартизації – оптимальне впорядкування об’єктів стандартизації.

Головне завдання стандартизації – створювати системи нормативної документації, що визначають прогресивні вимоги до продукції та послуг.

Основні завдання стандартизації:

- реалізація єдиної технічної політики в сфері стандартизації, сертифікації та метрології;
- захист інтересів споживачів та держави в питаннях безпеки продукції, охорони здоров'я;
- забезпечення якості продукції відповідно з досягненнями науки та техніки;
- забезпечення уніфікації, сумісності, взаємозамінності та надійності продукції;
- раціональне використання всіх ресурсі, поліпшення техніко-екологічних показників виробництва;
- безпека народногосподарських об'єктів і попередження аварій та техногенних катастроф;
- створення нормативної бази функціонування систем стандартизації та сертифікації, проведення державної політики в області ресурсозбереження;
- усунення технічних та термінологічних перешкод для створення конкурентоспроможної продукції та її виходу на світовий ринок;
- упровадження та застосування сучасних виробничих та інформаційних технологій;
- співучасть у забезпеченні обороноздатності та мобілізаційної готовності країни.

2. *Організації зі стандартизації*

Міжнародна організаці зі стандартизації (International Standardization Organization, **IS0**), була засновано 23 лютого 1947 року двадцятьма п'ятьма національними організаціями зі стандартизації, як координуючий орган. У складі ISO 160 країн-учасниць, близько 3000 структурних підрозділів технічних

комітетів (ТК); 650 підкомітетів; 2188 робочих груп; понад 15000 опублікованих стандартів ISO.

Міжнародна електротехнічна комісія (*International Electrotechnical Commission*, **IEC**). У відповідності з діючим договором між ISO і IEC, деякі види робіт виконуються спільно ISO. Питання інформаційних технологій, мікропроцесорної техніки тощо — це об'єкти спільних розробок ISO/IEC.

До основних пріоритетів діяльності ISO належать:

- заходи, які сприяють координації та уніфікації національних стандартів;
- розроблення та затвердження міжнародних стандартів;
- обмін інформацією з проблем стандартизації;
- співробітництво з іншими міжнародними організаціями, які зацікавлені у вирішенні суміжних проблем, і на їх прохання, вивчає проблеми стандартизації та ін.

Європейський комітет стандартів **(***Comité Européen de Normalisation*, **CEN)** заснований у 1961 році представниками національних організацій зі стандартизації на нараді в Парижі. Головним завданням цього комітету є розроблення загальних стандартів для країн, що входять до Європейського економічного співтовариства та Європейського товариства вільної торгівлі.

Національний орган стандартизації– *орган стандартизації, визнаний на національному рівні, що має право бути національним членом відповідних міжнародних та регіональних організацій стандартизації,* який розпочав свою діяльність 03.01.2015.

Аналоги в світі:

ANSI (США), DIN (Німеччина), BSI (Великобританія), UNI (Італія), AFNOR (Франція), TSI (Туреччина) тощо.

«Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості» або — Українське Агентство зі Стандартизації.

Категорії та види стандартів

- державні стандарти України – ДСТУ;
- галузеві стандарти України – ГСТУ;
- стандарти науково-технічних та інженерних товариств і спілок України - СТТУ;
- технічні умови України – ТУУ;
- стандарти підприємств – СТП;
- кодекси усталеної практики.

3. *Міжнародний стандарт якості програмних засобів ISO 9126*

ISO 9126 — **«Інформаційна технологія. Оцінка програмного продукту. Характеристики якості та керівництво по їх застосуванню».**

ISO 9126 це міжнародний стандарт, який визначає оцінні якості програмного забезпечення. Стандарт поділяється на 4 частини, які описують наступні питання: модель якості, зовнішні метрики якості, внутрішні метрики якості, метрики якості у використанні.

Модель якості, встановлена у першій частині стандарту 9126-1, класифікує якість ПЗ в 6-ти структурних наборах характеристик, які в свою чергу деталізовані під-характеристиками (субхарактеристиками), такими як:

Функціональність (functionality)– набір атрибутів, який характеризує відповідність функціональних можливостей ПЗ набору потрібної користувачу функціональності. Деталізується субхарактеристиками:

- Придатність до застосування (suitability) – здатність ПС надавати належну множину функцій для розв'язання специфікованих задач і досягнення цілей користувача;
- Коректність (правильність, точність) (accuracy) – здатність ПС забезпечувати правильні або погоджені результати або впливи з необхідним ступенем точності;
- Здатність до взаємодії (в тому числі мережевої) (interoperability) – здатність взаємодіяти з однією чи більше специфікованими системами;
- Захищеність (security) – здатність забезпечувати захист інформації від несанкціонованого доступу осіб або систем і її доступність особам та системам з необхідними правами доступу. **Надійність** (reliability) – набір атрибутів, які відносяться до здатності ПЗ зберігати свій рівень якості функціонування в встановлених умовах за визначений період часу. Субхарактеристики:
- Рівень завершеності (відсутність помилок) (maturity) – здатність уникати відмови із-за внутрішніх дефектів;
- Стійкість до дефектів (fault tolerance) – здатність підтримувати встановлений рівень функціонування в умовах прояви дефектів в ПС, помилок даних або порушень специфікованого інтерфейсу;
- Відновлюваність (recoverability) – здатність відновлювати функціонування на заданому рівні і відновлювати пошкоджені програми та дані; Доступність; Готовність.

Практичність (зручність застосування) (usability)– набір атрибутів, які відносяться до об'єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів. Субхарактеристики:

- Зрозумілість (understandability) – властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв'язання визначених задач і які умови її використання;

- Простота використання (learnability) – властивості ПС, які забезпечують користувачу можливість засвоїти прийоми її застосування;
- Керованість (operability) властивості ПС, які забезпечують користувачу можливість керувати або контролювати її дії;
- Привабливість (attractiveness).

Ефективність (efficiency)– набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах. Суб:

- Часова ефективність (time behavior) - властивості ПС, які спричиняють її здатність забезпечувати належний час відклику (відповіді) і обробки завдань, а також рівень пропускної здатності при виконанні функцій у встановлених умовах застосування;
- Використовуваність ресурсів (resource utilization) властивості ПС, які спричиняють її здатність використовувати ресурси в необхідні періоди часу при виконанні своїх функцій у встановлених умовах застосування.

Супроводжуваність (maintainability) – набір атрибутів, які відносяться до об'єму робіт, які необхідні для проведення конкретних змін (модифікацій). Субхарактеристики:

- Зручність для аналізу (analyzability) властивості ПС, які спричиняють можливість діагностування її недоліків або причин відмов, а також ідентифікації частин, які мають модифікуватись;
- Змінюваність (changeability) властивості ПС, які спричиняють можливість виконання встановлених видів модифікації;
- Стабільність (stability) - властивості ПС, які спричиняють її здатність мінімізувати неочікувані ефекти модифікацій;
- Зручність для тестування (testability) - властивості ПС, які спричиняють її здатність сприяти перевірці модифікованого ПЗ..

Мобільність (portability)– набір атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше. Суб:

- Здатність до адаптації (adaptability) - властивості ПС, які спричиняють її здатність адаптуватись для застосування в різноманітних специфікованих середовищах без використання дій або засобів відмінних від тих, що спеціально призначені для цих цілей;
- Простота встановлення (installability)
- Співвіснування (відповідність) (co-existence) властивості ПС, які спричиняють її здатність співіснувати з іншими незалежними ПС в спільному середовищі, розділяючи спільні ресурси;
- Здатність до заміщення (replaceability) - властивості ПС, які спричиняють її здатність використовуватись замість інших специфікованих ПС в середовищі їх застосування..

Підхарактеристика «**Відповідність**» не приведена і списку, але вона належить усім характеристикам. Ця характеристика має відображати відсутність протиріч з іншими стандартами або характеристиками.

Наприклад, відповідність надійності і практичності.

Кожна під характеристика якості далі розбивається на атрибути. Атрибут – це сутність, яка може бути перевірена або виміряна в програмному продукті. Атрибути не визначені в стандарті із-за їх різноманітності в різних програмних продуктах.

В стандарті виділена модель характеристик експлуатаційної якості. Основні:

- Результативність – ступінь в якій користувачами досягаються задані цілі по точності і повноті розв'язування задач у встановленому контексті використання ПС
- Продуктивність – продуктивність при розв'язанні основних задач програмної системи, яка досягається при реально обмежених ресурсах в конкретному зовнішньому середовищі застосування.
- Безпека – надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища.
- Задоволення потреб і витрат користувачів у відповідності з цілями застосування ПЗ. Друга частина стандарту ISO 9126-2 присвячена формалізації зовнішніх метрик. Зовнішні метрики відносяться до атрибутів, які визначають поведінку ПЗ у зовнішньому середовищі і взаємодію з іншими програмами.

Третя частина стандарту ISO 9126-3 присвячена формалізації внутрішніх метрик. Такі метрики вимірюють атрибути внутрішніх характеристик ПЗ.

Метрика – це комбінація конкретного методу вимірювання (способу отримання значень) атрибуту сутності і шкали вимірювання (засобу, який використовується для структурування отриманих значень). Викладені загальні рекомендації по використанню відповідних метрик і взаємозв'язку між типами метрик.

Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, супроводжуючих, користувачів і менеджерів якості ПЗ. В ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПЗ.

Лекція 1. Якість програмного забезпечення

Що таке якість?

Основною проблемою в управлінні якістю є той факт, що визначення якості неясне і неоднозначне. Це викликано тим, що зазвичай термін «якість» розуміють неправильно.

Причини:

1. Якість – це не окрема ідея або поняття, а багатомірна і різнопланова концепція.
2. Для будь-якого поняття і визначення існує декілька рівнів абстракції, наприклад, коли люди говорять про якість, одна частина розуміє під цим занадто широкий і розмитий смисл, а інша може вказувати на конкретне визначення.
3. Термін «якість» є невід'ємною частиною нашого повсякденного спілкування, але загальноприйняте і професійне використання може сильно відрізнятись.

Популярний погляд на якість

Загальноприйнята думка про якість – це дещо нематеріальне і «неосяжне» - про нього можуть вестись суперечки та дискусії, його можна критикувати і вихваляти, але зважити і виміряти неможливо. Вирази типу «хороша якість» і «погана якість» служать наглядним прикладом. Люди так кажуть про щось невизначене для них, що вони не можуть чітко охарактеризувати і визначити. Отже, люди сприймають і інтерпретують якість по різному. Якість не може бути контрольованою і керованою. Якість не може бути кількісно виміряна. Такий погляд різко контрастує з професійним підходом до управління якістю – якість чітко визначена величина, яку можна виміряти і проконтролювати, вона піддається управлінню і покращенню.

Інша популярна думка – якість нерозривно пов'язана з розкішшю, першим сортом і тонким смаком. Дорогий, досконально продуманий і технічно більш складний продукт розглядається як гарантія вищої якості ніж більш дешеві аналоги. За такою логікою Каділлак – якісна машина, а Шевроле – ні, незважаючи на надійність і кількість поломок. Якщо слідувати такому підходу,

то якість обмежена визначеним класом дорогих продуктів. Недорогі продукти ж важко віднести до якісних.

Професійний підхід до якості

Існують чіткі та зручні для роботи визначення.

В 1979 році Філ Кросбі визначив якість як «відповідність вимогам» ("conformance to requirements"). В 1970 році Юран і Гріна визначили якість як «придатність до використання» ("fitness for use"). Ці два визначення тісно пов’язані і добре узгоджуються.

Ще декілька визначень якості:

Уотс Хемпфрі – досягнення відмінного рівня придатності до використання;

Компанія IBM - якість, яка управляється ринковими потребами (“market - driven quality”).

Критерій Белдріджа для організаційної якості - “якість, яка задається споживачем”

Визначення системи менеджменту якості ISO 9001 - ступінь відповідності наявних характеристик вимогам.

«Відповідність вимогам» припускає, що вимоги мають бути настільки чітко визначені, що вони не можуть бути зрозумілі і інтерпретовані некоректно. Пізніше, на етапі розробки, проводяться регулярні вимірювання розробленого продукту для визначення відповідності вимогам. Будь-які невідповідності розглядаються як дефекти – невідповідність якості. Наприклад, специфікація на деяку модель радіостанції може вимагати можливість приймати визначену частоту радіохвиль на відстані більше ніж за 30 км від джерела віщання. У випадку, якщо радіостанція не може виконати дану вимогу, вона не відповідає вимогам до якості і має бути визнаною негідною і неякісною. Виходячи з таких принципів, якщо Каділлак відповідає усім вимогам до машин Каділлак, то це якісна машина. Якщо Шевроле відповідає усім вимогам до машин Шевроле, то це теж якісна машина.

«Придатність до використання» приймає до уваги вимоги і очікування кінцевого користувача, який очікує, що продукт або надаваний сервіс буде зручним для нього. Однак різні користувачі можуть використовувати продукт по

різному і це означає, що продукт має володіти максимально різноманітними варіантами використання. Згідно визначенню Юран кожен варіант використання – це характеристика якості і усі вони можуть бути класифіковані по категоріям в якості параметрів придатності до використання.

Ці два визначення якості по суті однакові. Різниця в тому, що варіант «придатність до використання» вказує на важливу роль вимог і очікувань замовника. Роль замовника, пов'язана з якістю, не може бути переоцінена. З точки зору замовника, якість продукту, який він придбав, складається з багатьох факторів: ціна, продуктивність, надійність і т.д.

Тільки замовник може розказати про якість, оскільки це єдине, що він дійсно купляє. Замовник не купляє продукт. Він купляє гарантії того, що усі його очікування до продукту будуть реалізовані. Висновки

Отже, визначення якості 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.

Отже, якість – це відповідність реальним вимогам, явним і неявним. Дуже часто неявні вимоги настільки очевидні для замовника або користувача, що він навіть не припускає, що вони невідомі розробникам. На прикладі автомобілів: замовник може детально описати вимоги до дизайну, параметрам двигуна, оформленню салону, кольору кузова, але ніде не вказати, що шини мають бути круглими, а лобове скло – прозорим.

Замовник буде повністю задоволеним тільки тоді, коли куплений продукт буде повністю відповідати його реальним і життєвим вимогам – специфікованим і ні. Існує дві професії пов'язані з якістю програмного забезпечення: тестер (Quality Control) і QA manager.

В чому різниця між Тестуванням и QA (Quality Assurance)?

Контроль якості (QUALITY CONTROL) це вимірювання якості продукту.

Забезпечення якості (QUALITY ASSURANCE) – це вимірювання і управління якістю процесу, який використовується для створення якості продукту (або якісного продукту).

Іншими словами.

Quality Assurance – попередження дефектів шляхом перевірки і тестування процесу.

Quality Control - виявлення дефектів шляхом перевірки і тестування продукту.

Що таке якість програмного забезпечення?

Якщо спитати про це достатньо широку групу людей, які мають справу з розробкою, продажем або використанням ПЗ, можна отримати наступні відповіді:

- Легко використовувати
- Хороши продуктивність
- Нема помилок
- Не псує даних користувача при збоях
- Можна використовувати на різних платформах
- Може працювати 24 години на сутки і 7 днів на тиждень
- Легко добавляти нові можливості
- Задовольняє потреби користувача
- Добре документовано Якість ПЗ по МакКолу

Першою широко відомою моделлю якості ПЗ стала запропонована в 1977 МакКолом та іншими модель.

В ній характеристики якості розділені на три групи:

Фактори, які описують ПЗ з позицій користувача. Задаються вимогами.

Критерії, які описують ПЗ з позицій розробника. Задаються як цілі.

Метрики, які використовуються для кількісного описання і вимірювання якості.

Фактори якості, яких було виділено 11, групуються в три групи по різним способам роботи людей з ПЗ.

Отримана структура відображується у вигляді трикутника МакКола

Критерії якості – це числові рівні факторів, поставлених як цілей при розробці. Об’єктивно оцінити або виміряти фактори якості безпосередньо важко. Тому МакКол ввів метрики якості, які з його точки зору легше виміряти і оцінити. Оцінки в його шкалі приймають значення від 0 до 10.

Рис. 1. Треугольник МакКола.

Метрики:

- Зручність перевірки на відповідність стандартам (auditability)
- Точність керування і обчислень (accuracy)
- Ступінь стандартності інтерфейсів (communication commonality)
- Функціональна повнота (completeness)
- Однорідність використовуваних правил проектування і документації (consistency)
- Ступінь стандартності форматів даних (data commonality)
- Стійкість до помилок (error tolerance)
- Ефективність роботи (execution efficiency)
- Розширюваність (expandability)
- Широта області потенційного використання (generality)
- Незалежність від апаратної платформи (hardware independence)
- Повнота протоколювання помилок та інших подій (instrumentation)
- Модульність (modularity)
- Зручність роботи (operability)

- Захищеність (security)
- Самодокументованість (selfdocumentation)
- Простота роботи (simplicity)
- Незалежність від програмної платформи (software system independence)
- Можливість співвідношення проекту з вимогами (traceability)
- Зручність навчання (training).

Кожна метрика впливає на оцінку деяких факторів якості. Числовий вираз фактору представляє собою лінійну комбінацію значень впливаючих на нього метрик. Коефіцієнти цього виразу визначаються по різному для різних організацій, команд розробки, видів ПЗ та інш.

Якість ПЗ по Боему

У 1978 Боем запропонував свою модель, яка по суті є розширенням моделі МакКола.

Атрибути якості поділяються за способом використання ПЗ (primary use).

Визначено 19 проміжних атрибутів (intermidiate construct), які включають усі 11 факторів якості по

МакКолу. Проміжні атрибути розділяються на примітивні, які в свою чергу, можуть бути оцінені за допомогою метрик.

До факторів МакКола Боем додав наступні атрибути:

- ясність (clarity)
- зручність внесення змін (modifiability)
- документованість (documentation)
- здатність до відновлення функцій (resilience)
- зрозумілість (understandability)
- адекватність (validity)
- функціональність (functionality)
- універсальність (generality) • економічна ефективність (economy).