**Лабораторна робота №8 Визначення вимог до програмних систем. Оформлення вимог. Тематичне опитування**

**На період** карантину в дистанційній формі навчання на надані запитання потрібно надати письмові відповіді, надіславши їх на електронну адресу викладача. Файл надавати з іменем у форматі

**ОPI<Номер групи><Номер лекції / практичної / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, О**PI3104L**buts.doc. Відповіді на запитання повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей -"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи ІПЗ-31 05.04.2021, ІПЗ-32 04.04.2021**

**Мета:** **Навчиться оформлювати вимоги до програмного забезпечення.**

**Завдання:**

1. Опрацювати теоретичні відомості.
2. Використати нові відомості для доповнення створюваних на **Практичні роботі №10** та **Лабораторні роботі №7** документів „Вимоги користувача”, „Функціональні вимоги до програмного забезпечення”, надавши інформацію про **Перелік зацікавлених сторін,** а також окремим переліком визначити

* Вимоги дизайну ;
* Вимоги продуктивності ;
* Поведінкові вимоги ;
* Структурні вимоги ;
* Архітектурні вимоги ;
* Вимоги споживача

1. До роботи повинен надаватися звіт з титульним листом із визначенням «Лабораторна робота № », після цього написати назву системи / застосунку для якого відносяться вимоги. До звіту включаються доопрацьовані на цій лабораторній роботі документи „Вимоги користувача” та „Функціональні вимоги до програмного забезпечення” з відміченими в них доповненнями (заливка кольором або виділення шрифтом іншого читабельного кольору).
2. По закінченню лабораторну роботу потрібно здати на перевірку викладачеві, надіславши електронною поштою. Якщо викладач знаходить помилки чи неточності, він може повернути роботу на доопрацювання.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Аналіз вимог** полягає в визначенні потреб та умов які висуваються щодо нового, чи зміненого продукту, враховуючи можливо конфліктні вимоги різних замовників, таких як користувачі чи бенефіціари.

Бенефіціар — одержувач визначених [вигод](https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%B3%D0%BE%D0%B4%D0%B0), що виникають у результаті реалізації проекту.

Аналіз вимог є критичним для успішної розробки проекту. Вимоги мають бути задокументованими, вимірними, тестовними, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи. Вимоги можуть бути архітектурними, структурними, поведінковими, функціональними, та не функціональними.

Аналіз вимог включає три види діяльності:

* **Виявлення вимог**: задача комунікації з користувачами для визначення їх вимог. Також це називають збором вимог.
* **Аналіз вимог**: виявлення недоліків вимог (неточностей, неповноти, неоднозначностей чи суперечностей) і їх виправлення.
* **Запис вимог**: Вимоги можуть документуватись в різних формах, таких як опис звичайною мовою, прецедентами, користувацькими історіями, чи специфікаціями процесу.

Аналіз вимог може бути довгим та важким процесом що вимагає використання тонких психологічних навиків. Нові системи змінюють середовище і відношення між людьми, тому важливо розпізнати всі зацікавлені сторони, взяти до уваги всі їхні потреби, і переконатись що вони розуміють наслідки які приносить нова система. Аналітики можуть використати кілька методів щоб отримати від споживача вимоги. Історично це включає проведення інтерв'ю, чи фокус-груп (яку в цьому контексті частіше називають як майстерня вимог) і створення списків вимог. До сучасніших підходів відносять прототипування та прецеденти. За потреби аналітик використає комбінацію цих методів щоб встановити точні вимоги зацікавлених сторін, так щоб система відповідала бізнес-потребам.

Систематичний аналіз вимог також відомий як **інженерія вимог**. Часом більш неформально її називають *збором вимог*, чи *специфікацією вимог*. Термін *аналіз вимог* також може застосовуватись до відповідного аналізу, в протилежність до, наприклад, збору чи документування вимог. Інженерія вимог може бути поділена на дискретні хронологічні кроки :

* Збір вимог
* Аналіз вимог і переговори
* Специфікація вимог
* Моделювання системи
* Перевірка вимог
* Управління вимогами

Цей перелік не протирічить схемі на рис.1 з Лекції 07, оскільки надає послідовність виконання, яка містить в собі заходи, визначені в схемі.

Інженерія вимог є "піддисципліною системної інженерії та програмної інженерії яка причетна до визначення цілей, функцій та обмежень апаратних та програмних систем. В деяких моделях життєвих циклах, процес інженерії вимог починається з діяльності по вивченню здійсненності, результатом якої є звіт про здійсненність. Якщо звіт припускає що продукт може бути створеним, то починається аналіз вимог. Якщо аналіз вимог передує дослідженням здійсненності, що може сприяти нестандартному мисленню, тоді здійсненність має визначитись перед завершенням аналізу вимог.

**Розділи аналізу вимог**

**Визначення зацікавлених сторін** ми детально розглядали в Лекції 07, можна провести таке узагальнення, розширивши коло осіб з погляду на розроблення масштабних систем.

Зацікавлені сторони (ЗС) це особи чи організації, які мають дійсний інтерес до системи. Вона може впливати на них прямо чи опосередковано.

Визнається, що зацікавлені сторони не обмежуються організацією що найняла аналітиків. До зацікавлених сторін також відносять:

* кожного хто керуватиме системою (звичайні та обслуговуючі оператори);
* будь-кого хто отримає вигоди від системи (функціональні, політичні, фінансові та соціальні бенефіціари);
* кожен хто бере участь в придбанні чи закупці системи. В розробці продуктів для масового ринку, відділ менежменту продукту, маркетингу (іноді і відділу продаж), які діють як сурогатні споживачі, щоб направляти розробку продукту;
* організації, що регулюють аспекти системи (фінансові, безпеки, та інші регулятори);
* люди та організації, що протистоять системі (негативні зацікавлені сторони);
* організації відповідальні за системи, які будуть взаємодіяти з системою, що розробляється;
* ті організації, які горизонтально інтегруються з організацією, для якої аналітики конструюють систему.

**Інтерв'ю з зацікавленими сторонами**

Інтерв'ю з ЗС є рядовим підходом що використовується в аналізі вимог. Ця техніка може служити як шлях отримання висококонцентрованого знання, яке часто не виявляється в спільних сесіях розробки вимог, де увага зацікавленої сторони підпорядкована забезпеченню більш крос-функціонального контексту. Більш того, особистий характер інтерв'ю надає більш розслаблююче середовище, де хід думок може бути детальніше пояснений.

**Спільні сесії Розробки Вимог (СРР)**

Вимоги часто мають крос-функціональні наслідки, які невідомі окремим зацікавленим сторонам і часто пропускаються чи неправильно описуються протягом інтерв'ю з ЗС. Ці крос-функціональні наслідки можуть бути виявленими проведенням сесій СРР в контрольованому середовищі, стимульованому кваліфікованим посередником, де ЗС беруть участь в дискусії з метою виявлення вимог, аналізують їх деталі і розкривають крос-функціональні наслідки. Мають бути присутні спеціально призначені секретар, та бізнес-аналітик для документування дискусії. Використання навиків навченого посередника для управління дискусією звільняє бізнес-аналітика, дозволяючи йому сфокусуватись на процесі визначення вимог.

Сесії СРР подібні до сесій спільного проектування ПЗ. Спершу сесії виявляють вимоги, які направляють дизайн, а потім виявляють властивості, які мають бути реалізовані, щоб задовольнити отримані вимоги.

**Списки вимог в стилі контракту**

Також традиційним способом документування вимог є список вимог в стилі контракту. Такі списки не дуже цінуються в сучасному аналізі, оскільки вони показали малу успішність в досягненні своїх цілей. Але, їх іноді можна побачити і сьогодні.

**Переваги**

* Надає чіткий попунктовий список вимог.
* Надає контракт між спонсорами проекту та розробниками.
* Для великої системи може надати опис високого рівня.

**Недоліки**

* Такі списки розтягуються на сотні сторінок. Такі документи майже неможливо прочитати цілком і отримати повне розуміння системи.
* Такі списки абстрагують всі вимоги, тому містять мало контексту.
* Їх абстракція робить неможливим побачити як вимоги сполучаються чи працюють разом.
* Така абстракція заважає правильно розставляти пріоритети між вимогами.
* Така абстракція збільшує подібність та ймовірність неправильної інтерпретації вимог, чим більше людей її прочитають, тим більша кількість різних інтерпретацій з'явиться.
* Така абстракція означає, що надзвичайно важко впевнитись, що вимагає більшість вимог.
* Такі списки створюють фальшиве відчуття взаємного розуміння між зацікавленими сторонами та розробниками.
* Списки в стилі контракту дають ЗС фальшиве відчуття безпеки, що розробники мусять досягти певних речей. Через природу таких списків вони упускають критичні вимоги, які виявляються пізніше в процесі. Розробники можуть використати ці відкриті вимоги щоб переглянути умови договору на свою користь.
* Такі списки вимог не домагають в проектуванні системи

**Альтернативи до списків вимог**

Як альтернатива до великих, попередньо-описаних списків вимог гнучка розробка програмного забезпечення використовує історії користувачів щоб описати вимогу в повсякденних термінах. Історія користувача (*user story*) - це одне чи більше речень, звичайною мовою предметної області, які описують чого користувач хоче досягти.

**Вимірювані цілі**

Найкращі практики беруть складений список вимог як підказку, і постійно запитують "Чому?", поки не стане зрозумілою справжня бізнес ціль. Зацікавлені сторони і розробники можуть скласти тести, що вимірюють рівень досягнення цілей. Такі цілі змінюються повільніше ніж довгий список конкретних але невимірних вимог. Як тільки невеликий набір критичних, вимірних цілей буде встановлений, швидке прототипування та короткі ітеративні фази розробки можуть приносити справжні цінності для зацікавленої сторони, задовго до середини проекту.

**Прототипи**

В середині 1980-тих прототипування розглядалось як найкраще рішення до проблеми аналізу вимог. Прототипи - це макети застосунку. Макети дозволяють користувачам візуалізувати застосунок, який ще не створено. Прототипи допомагають користувачам уявити як буде виглядати система, і зробити простішим для користувачів прийняття конструкторських рішень. Після введення прототипів спостерігаються значні покращення в комунікації між користувачами й розробниками. Раннє бачення застосунку приводить до меншої кількості змін у майбутньому і тому зменшує загальну вартість проекту.

Тим не менш, протягом останнього десятиліття, прототипування хоча й зарекомендувало себе як корисна техніка, але не розв'язало проблему вимог:

* Як тільки менеджери бачать прототип, вони перестають розуміти, що проект ще не буде завершений протягом деякого часу.
* Дизайнери часто думають, що змушені використовувати склеєний докупи код прототипу в реальній системі, бо вони бояться "втратити час" починаючи заново.
* Прототипи загалом допомагають в конструкторських рішеннях та проектуванні інтерфейсу користувача, але вони не можуть вказати, які вимоги були спочатку.
* Дизайнери і кінцеві користувачі можуть занадто сфокусуватись на конструюванні інтерфейсу користувача і занадто мало на створенні системи, що виконує бізнес процес.
* Прототипи добре працюють для інтерфейсів користувача, та подій на екрані, але не дуже корисні для пакетних чи асинхронних процесів які можуть включати складні розрахунки чи запити до даних.

Прототипи можуть бути плоскими діаграмами (які часто називаються каркасними моделями), чи робочими застосунками, що використовують синтезовану функціональність. Каркасні моделі створюються в різноманітних графічних дизайн-документах, і часто видаляють увесь колір з дизайну (використовують чорно-білу палітру) у випадках, коли кінцеве ПЗ буде мати відповідний графічний дизайн. Це допомагає уникнути плутанини між концептом програми в дизайн-документі та її кінцевим виглядом.

**Прецеденти**

Прецеденти - це технологія документування потенційних вимог до нової, чи зміненої програмної системи. Кожен прецедент надає один чи більше *сценаріїв,* які виражають те, як система взаємодіє з користувачем чи іншою системою, щоб досягти конкретної бізнес цілі. Прецеденти зазвичай уникають технічного жаргону, віддаючи перевагу мові кінцевого користувача чи експерта в предметній області. Інженери вимог та ЗС часто виступають співавторами прецедентів.

Прецеденти є оманливо простими інструментами для опису поведінки системи. Прецедент містить текстовий опис всіх способів, якими користувачі можуть працювати з програмним забезпеченням, але прецеденти не описують жодних внутрішніх процесів у системі і не пояснюють, як система має реалізовуватись. Вони просто показують кроки, які користувач має здійснити, щоб виконати свою задачу. Так можна описати всі способи взаємодії з системою.

**Специфікація вимог до програмного забезпечення**

Специфікація вимог до програмного забезпечення (SRS) - це повний опис поведінки системи, що розробляється. Він включає набір прецедентів, що описують всі взаємодії користувача з системою. Прецеденти інколи визначають як функціональні вимоги, але останній термін є більш поширеним, а перший часто асоціюється з UML-діаграмами, які є доречною ілюстрацією, але не завжди повністю висвітлюють проблему . На додачу до прецедентів, SRS також містить нефункціональні (чи додаткові вимоги). Нефункціональні вимоги є вимогами які накладають обмеження на проект, чи реалізацію (такі як вимоги інженерії продуктивності, стандарти якості, чи обмеження проектування).

Рекомендовані підходи до специфікації вимог до ПЗ описані в стандарті IEEE 830-1998. Цей стандарт описує можливі структури, бажаний вміст і якості специфікації вимог.

**Типи вимог**

Вимоги категоризуються кількома способами. Нижче подана звичайна категоризація вимог, яка стосується технічного менеджменту:

* Вимоги дизайну
* Вимоги продуктивності
* Нефункціональні вимоги
* Функціональні вимоги
* Поведінкові вимоги
* Структурні вимоги
* Архітектурні вимоги
* Вимоги споживача

# Функціональні та нефункціональні вимоги ми розглядали в Лекції 07 та Практичній роботі №10. Коротко зупинимося на інших.

Вимоги дизайну - вимоги «будувати до», «кодувати до», і «купувати до» для продуктів, і «як виконати» для процесів виражених в технічних пакетах даних та інструкціях.

Вимоги продуктивності – вимоги, що визначають до якої міри функції системи повинні бути виконані і зазвичай вимірюються в термінах кількості, якості, охопленні, своєчасності чи готовності. Протягом аналізу вимог, вимоги продуктивності зазвичай доповнюються та уточнюються для всіх виявлених функції з урахуванням їх відношення до інших вимог.

Поведінкові вимоги пояснюють, що має бути зроблено за умов реалізації визначеної необхідної поведінки системи.

Структурні вимоги пояснюють, що має бути зроблено за умов реалізації визначеної необхідної структури системи.

Архітектурні вимоги пояснюють, що має бути зроблено за умов реалізації визначеної необхідної системної архітектури.

# Вимоги споживача описують очікування до системи в термінах цілей, середовища, обмежень та міри ефективності й придатності. Ці вимоги повинні дати відповідь на такі запитання:

* *Операційне поширення і розгортання*: Де використають систему?
* *Профіль чи сценарій місії*: Як система буде виконувати свої завдання?
* *Продуктивність та пов'язані параметри*: Які параметри критичні для виконання місії системи?
* *Використання середовища*: Як будуть використовуватись різноманітні компоненти системи?
* *Вимоги ефективності*: Якою ефективною має бути система для виконання своєї місії?
* *Операційний життєвий цикл*: Як довго система буде використовуватись споживачем?
* *Середовище*: Яких середовищ система очікує щоб працювати ефективно?

Потрібно зауважити, що споживач і користувач не є тотожними поняттями в даному контексті. Користувач безпосередньо використовує програмний продукт в своїй діяльності, а споживач може використовувати лише результати роботи програмного продукту.

Окрім того, можуть виникати успадковані вимоги. Це – вимоги, які зазвичай висуваються на основі попереднього досвіду роботи з аналогічними програмними продуктами або попередньою систему, на заміну якої створюється. Вони часто можуть конфліктувати з вимогами до нової системи.