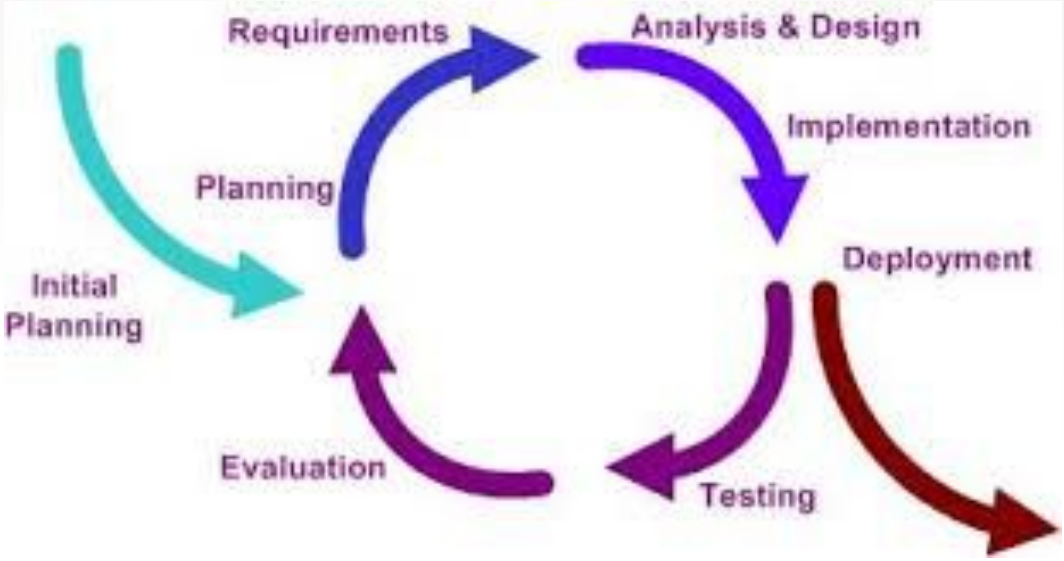


РОЗДІЛ 6. Аналіз вимог: інспекція, атестація, завершеність, виявлення конфліктів і невідповідностей. Аналіз взаємодій елементів функціональності і вирішення протиріч

Аналіз вимог

Аналіз вимог **– це процес виявлення протиріч, неповноти, конфліктів та прийняття дозвілу на конфлікти в процесі розробки програмного забезпечення.**

А також, це процес класифікації вимог, визначення пріоритетів, границь системи і принципів взаємодії зі середовищем функціонування.

Аналіз вимог включає:

- Виявлення та вирішення конфліктів між вимогами;
- Визначення меж задачі, розв'язуваної створюваним програмним забезпеченням;
- В загальному випадку - визначення кордонів і змісту програмного проекту;
- Деталізацію системних вимог для встановлення програмних вимог;
- Практично завжди, хоча це явно і не зазначено в описі аналізу вимог як секції SWEBOK, на практиці виділяються і деталізуються бізнес-вимоги для встановлення програмних вимог.

Погляд на аналіз вимог

Традиційний погляд на аналіз вимог часто сфокусований або зменшений до питань ***концептуального моделювання*** з використанням відповідних аналітичних методів, одним з яких є SADT - Structured Analysis and Design Technique (методологія структурного аналізу і техніки проектування), знайомий багатьом по нотаціям IDEF0 (функціональне моделювання - стандарт IEEE 1320.1), IDEF1X (інформаційне моделювання - стандарт IEEE 1320.2, відомий також як IDEFObject), часто вживаним як для моделювання бізнес-процесів, так і структур даних, зокрема - реляційних баз даних.

Так чи інакше, незалежно від виразних засобів, які є лише інструментом аналізу і фіксування результатів, ***результатом аналізу вимог*** повинні бути однозначно інтерпретовані вимоги, реалізація яких перевіряєма, а вартість і ресурси - передбачувані.

Класифікація вимог (Requirements Classification)

Вимоги можуть класифікуватися по цілому ряду параметрів, наприклад:

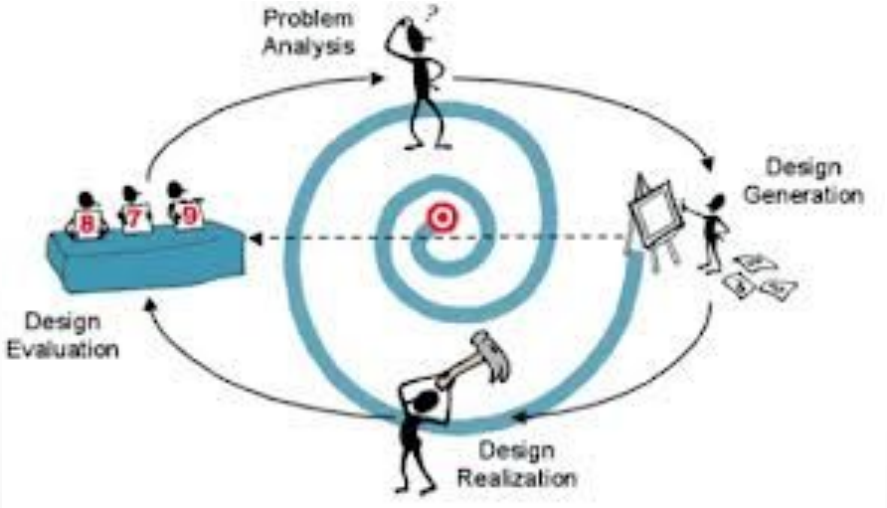
- Функціональні та нефункціональні вимоги;
- Внутрішні (з іншими вимогами) або зовнішні залежності;
- Вимоги до процесу або продукту;
- Пріоритет вимог;
- Зміст вимог щодо конкретних підсистем створюваного програмного забезпечення;
- Змінність / стабільність вимог;
- Інші варіанти класифікації можуть, і часто базуються, на прийнятих в організації підходах, застосовуваних методологіях, методах і практиках, а, найчастіше, і специфіці проектів і навіть вимогах замовників до процесу розробки і, зокрема, визначення вимог та форми подання результатів їх аналізу.

Концептуальне моделювання

(Conceptual Modeling)

Розробка моделі проблеми реального світу - ключовий елемент ***аналізу вимог.***

Мета моделювання - розуміння проблеми, завдання та методів їх вирішення до того, як почнеться вирішення проблеми.



Часто доводиться чути, що прагматичність підходу щодо програмних проектів полягає на "пропуск" етапі (або стадії, фазі) моделювання.

У свою чергу, часто ставлять знак рівності між моделюванням і "*цими красивими квадратиками із стрілками*".

Ні те, ні інше твердження невірні. Наприклад, в XP і в інших гнучких (Agile) практиках існують і історії користувачів, і картки завдань, та процедури аналізу (зокрема, пов'язаних з ними "мозкових штурмів", як запланованих, так і, на жаль, не дуже), в результаті якого ми сформулювали завдання, високорівневі можливості - "фічі" продукту (feature - особливість), а також необхідні моделі.

Обсяг моделей, їх деталізація та засоби представлення можуть бути різні. Їх вибір базується і/або диктується конкретним культурним контекстом організацій, залучених до проекту, і практик, які застосовуються проектною групою. Саме не форма, але сама ідея моделювання як спроба спростити і однозначно інтерпретувати на концептуальному рівні проблематику діяльності в реальному світі - обов'язкова складова як керування вимогами, так і програмної інженерії, в цілому.

Серед факторів, які впливають на вибір моделі, методу і деталізації її подання, ступенем пов'язаності з програмним кодом та іншими питаннями є:

- Природа проблеми (проблемної області);
- Експертиза та досвід інженерів;
- Вимоги замовника до процесу;
- Доступність методів та інструментів;
- Внутрішньокорпоративні стандарти та регламенти;
- Культура розробки.

Питання моделювання тісно пов'язані з ***застосовуваними методами*** і ***підходами***.

Однак, **приватні методи або нотації**, так чи інакше слідують **поширеним в індустрії практикам** і тяжіють до тих форм, з якими пов'язані накопичений досвід і підтверджені загальноприйнятою практикою знання.

Можуть бути розроблені різні види моделей, що включають потоки робіт і даних, моделі станів, трасування подій, взаємодії користувачів, об'єктні моделі, моделі структур даних, і т.п.

Концептуальне моделювання базується на візуальному моделюванні

В комплексі сукупність включених до методу діаграм відображає найважливіші випадки функціонування системи. Перелічимо їх:

а) діаграми класів;
б) діаграми сценаріїв використання;
в) діаграми поведінки об'єктів, а саме:
    1) діаграми послідовності;
    2) діаграми співробітництва;
    3) діаграми активності;
    4) діаграми станів;
г) діаграми реалізації, а саме:
    1) діаграми компонент;
    2) діаграми розміщення.

Перевірка вимог

(Requirements Validation)

Визначення вимог безпосередньо пов'язано з процедурами *перевірки* (verification) і *затвердження* (атестації - validation, як це сформульовано в ГОСТ Р і ISO / IEC 12207).

V & V

Прийнято вважати, що вимоги описані неповністю, якщо для них не задані правила V & V (verification & validation - перевірка і атестація), тобто не визначені способи перевірки і затвердження.

Процедури перевірки є відправною точкою для інженерів-тестувальників і спеціалістів з якості, що безпосередньо відповідають за відповідність одержуваного програмного продукту пропонованим до нього вимогам.

Створення правильного продукту

Якщо стандарти життєвого циклу описують як правильно створювати продукт, то стандарти (у тому числі, внутрішньокорпоративні) відповідають за створення правильного продукту, тобто того продукту, який відповідає очікуванням користувачів та інших зацікавлених осіб.

Для досягнення цієї мети застосовується ряд практик, в тому числі, *представлених нижче*.

- *Огляд вимог (Requirements Review)*
- *Прототипування (Prototyping)*
- *Затвердження моделі (Model Validation)*
- *Приймальні тести (Acceptance Tests)*

Огляд вимог (Requirements Review)

Один з поширених методів перевірки вимог - *інспекція або огляд вимог*.

Суть його полягає в тому, що ряд осіб, залучених до проекту (для великих проектів - спеціально виділені фахівці), "вичитують" вимоги в пошуках необгрунтованих припущень, описів вимог, що допускають множинні інтерпретації, протиріч, неузгодженості, недостатній мірі деталізації, відхилень від прийнятих стандартів і т.п.

Прототипування (Prototyping)

У загальному випадку, ***Прототипування*** представляє собою перевірку інженерної інтерпретації програмних вимог і витяг нових вимог, невизначених або неясних на ранніх ітераціях збору вимог.

Існує безліч підходів до прототипування, як з точки зору деталізації, так і того, чому приділяється увага при прототипіюванні.

Найбільш часто прототипи створюються ***для оцінки способу реалізації інтерфейсу користувача і перевірки архітектурних підходів і рішень.***

Серед можливих негативних наслідків прототипування варто виділити наступні:

- зміщення уваги з цільових функцій прототипу і, як наслідок, незадоволеність користувачів огріхами прототипу;
- відсутністю реальної функціональності (для прототипів користувальницького інтерфейсу);
- помилками в прототипі і т.п.;

- перетворення прототипу в реальну систему за рахунок постійного додавання нових властивостей і функціональності "для перевірки" - часто буває порушена архітектурна цілісність, незабезпечена необхідна масштабованість і якість одержуваного програмного продукту;

- переключення уваги зацікавлених осіб на ергономіку і деталі дизайну графічного інтерфейсу користувача, при початковій меті побудови прототипу для виявлення функціональних і інших вимог і навпаки.

Проблема не в увазі користувальницькому інтерфейсу, проблема в підміні, якщо так можна висловитися, функціональної складової користувача інтерфейсом (тобто, "я не про 'кнопочки' і 'віконця', я про завдання ...").

Звичайно, зрозуміло, що ці фактори можна перетворити і *в позитивні сторони* прототипу. Крім того, не варто вважати що прототип це завжди щось, втілене в код. Прототипом для користувача інтерфейсу може бути, наприклад, просто "промальований" на папері або в електронній формі набір переходів між екранами / діалоговими вікнами системи (до речі, це підхід, що часто використовується в Agile-практиках, але аж ніяк не замінює вимог до системи).

Так чи інакше, вибір того чи іншого методу прототипування, та й самого факту такого способу перевірки вимог або технологічних ідей, повинен грунтуватися на тимчасових і інших наявних ресурсах, досвіді в прототипіюванні і, звичайно, ступені складності створюваної програмної системи.

Затвердження моделі

(Model Validation)

Затвердження або атестація моделі пов'язана з питаннями забезпечення прийнятної якості продукту. Впевненість у відповідності моделі заданим вимогам може бути закріплена формально з боку користувачів / замовника.

У той же час, перевірка і атестація моделі, наприклад, об'єктно-орієнтованого представлення бізнес-сутностей і зв'язків між ними може бути перевірена з тим або іншим ступенем використання формальних методів, наприклад, статичного аналізу (пошук зв'язків і шляхів взаємодії між описаними об'єктами і виділення різного роду невідповідностей).

Це - інша сторона затвердження моделі.

Приймальні тести

(Acceptance Tests)

Вимоги повинні бути верифіковані. Вимоги, які не можуть бути перевірені і атестовані (затверджені) - це всього лише "побажання".

Саме так вони буду сприйматися розробниками, навіть у разі їх високої значимості для користувачів.

Якщо описана вимога не супроводжується процедурами перевірки - в більшості випадків говорять про недостатню деталізації або неповному опису вимоги і, відповідно, специфікація вимог повинна бути відправлена на доопрацювання і якщо необхідно, повинні бути зроблені додаткові зусилля, спрямовані на збір вимог.

Процедура аналізу вимог вважається виконаною тільки тоді, коли всі вимоги, включені в специфікацію, володіють методами оцінки відповідності їм створюваного програмного продукту. Найчастіше таке суворе обмеження на ступінь завершеності специфікації накладається на функціональні вимоги і атрибути якості (наприклад, час відгуку системи).

Ідентифікація та розробка приймальних тестів для нефункціональних вимог часто виявляється найбільш трудомістким завданням.

Для її рішення зазвичай "шукають точку опори", тобто можливість погляду на описувані вимоги з кількісної точки зору, навіть до переформулювання і більшою мірою деталізації опису таких вимог.

Використання моделей для проведення аналізу вимог

Щоб полегшити процес формулювання і розуміння вимог для Замовника, існує ряд прийомів.

По-перше, ***вимоги можна формулювати на різних рівнях абстракції.***

Так, рівень опису вимог, підтримуваний в документі "Бачення", є достатньо збалансованим. Теж можна сказати і про короткі (в один абзац) описи ключової функціональності системи. Діючи таким чином, ми, очевидно, вирішимо проблему залучення Замовника в аналіз задач, однак зазначені вище ризики будуть знижені недостатньо: концептуальні описи функціональності залишають багато свободи для тлумачення в ту або іншу сторону.

Процес аналізу вимог тісно пов'язаний, з одного боку, з аналізом проблемної області, з іншого - з архітектурним аналізом і проектуванням. Часто на практиці буває важко відокремити кордони компетенцій цих потоків робіт. Так, модель аналізу потоків даних, широко використовувана в аналізі проблемної області, згадується багатьма авторами, як модель, корисна в аналізі вимог. Ряд дослідників вважає доцільним ілюструвати опис вимог *діаграмами класів*, хоча, строго кажучи, виділення класів належить не до аналізу вимог, а до архітектурного аналізу.

Як визначити доцільність використання тих чи інших прийомів, методик, мов моделювання при аналізі вимог?

Тут можна запропонувати **три практичні рекомендації**:

По-перше, **аналіз вимог покликаний вивчати взаємодії автоматизованої інформаційної системи та її середовища,** тобто користувачів, мережевих і системних компонентів, що знаходяться поза системою.

Отже, бізнес-моделі, що описують взаємодії між компонентами організаційної системи, строго кажучи, можна розглядати лише як "сировину" для витягання вимог, але не як моделі, що описують вимоги.

По-друге, **аналіз вимог повинен знаходити відповідь на те, ЩО робить система, абстрагуючись від деталей реалізації, тобто того, ЯК вона це робить.**

Тому, припустимо, діаграма взаємодії об'єктів, що реалізують той чи інший варіант використання, можна розглядати швидше як ілюстрацію внутрішнього устрою системи, корисну для програміста, ніж модель для замовника.

Однак, найбільш важливим є *третє міркування*, в чомусь "опозиційне" по відношенню до перших двох.

Для моделювання аналізу вимог слід застосовувати моделі, найбільш адекватно проясняють функціональність системи та особливості її використання.

Однак, аналітик вільний вибирати ті мови та методики, які дозволять домогтися цільової функції: зниження ризиків нерозуміння між Виконавцем та Замовником і розмиття кордонів.

Ілюстровані сценарії та прототипи. Цілі прототипування

Розглянемо основні цілі, що вимагають застосування прототипів:

- прояснити неясні вимоги до системи;
- вибрати одне з різних концептуальних рішень;
- проаналізувати здійсненність.

Неясні вимоги

Часто Замовнику буває важко сформулювати вимоги до того, що він очікує від системи.

У цьому випадку прототип інтерфейсу користувача (User Interface, UI), оперативно створений за результатами інтерв'ю, дає йому можливість побачити схематичну реалізацію того, як Виконавець побачив відповідну частину системи.

В даному випадку корисний будь-який результат прототипування: якщо Виконавець зрозумів вимоги добре - користь очевидна;

якщо не дуже - користь полягає в тому, що Замовник може вказати, в чому полягає нерозуміння, тим самим вирішивши основне завдання - **зробити неясне ясним**.

Різні варіанти вирішення

Будь-яку технічну задачу можна вирішити різними способами. Це стосується як завдання формулювання вимог, так і її реалізації в UI.

Аналіз здійсненності

Часто буває так, що комбінація функціональних, нефункціональних вимог і обмежень така, що виникає ризик неможливості їх реалізації.

Як правило, такий ***ризик пов'язаний з вимогами до швидкодії системи при відомих обмеженнях середовища її реалізації***.

У цьому випадку створюються прототипи (не обов'язково, пов'язані з UI), що реалізують відповідну частину системи, які імітують потоки даних, що надходять на її вхід і їх обробку.

Прототипи

Створення прототипу є одним з найбільш важливих етапів реалізації проекту.

Професійно виконаний прототип зможе зберегти багато часу як для клієнта, так і для команди розробників.

В основному, прототип - це ескіз проекту, який буде розроблятись.

Навіщо потрібні прототипи?

Прототипи допомагають у різних випадках :

- **для команди проекту** - для однакового розуміння майбутнього проекту ;
- **розробникам** простіше реалізовувати проект за наявності не тільки технічного завдання , але і прототипу - щоб звірятися с «картинкою » у разі непорозуміння;
- **для замовника** - візуальне представлення його вимог і побажань;
- **для користувачів**, як первинний інструмент тестування зручності;
- **для веб- студій та фрілансерів** прототип може стати хорошим візуальним пропозицією для потенційного клієнта.

Виходячи з первинних вимог клієнта ви можете створити прототип як концепцію проекту, пропонуючи на прикладі як буде виглядати і працювати програмне забезпечення.

Ви також можете акцентувати особисті рішення, які ви придумали.

Прототип допомагає:

- Зменшити ризики проекту

Наприклад, коли при первинних вимогах не продумали можливості користувачів та їх взаємодію.

- Зменшити кількість помилок

Помилки, допущені в кінці створення проекту - найдорожчі; доводиться переробляти систему.

Прототип дозволяє виявити помилки на ранній стадії і мінімізувати появу нових.

- Підвищити якість проекту за рахунок зниження кількості помилок
- Зменшити строки і вартість розробки

Кому потрібен прототип???

Для чого це потрібно замовнику?

Клієнт отримує повне уявлення про всі нюанси проекту.

На цьому етапі можна бачити всі логічні прорахунки, невідповідності в розташуванні елементів або неточності по самій функціональності, які можуть бути скоректовані.

Для чого це потрібно розробникам і дизайнерам?

Економія значної частини роботи та часу розробників і дизайнерів.

Використовуючи узгоджений із замовником прототип, дизайнер одразу отримує основну ідею про те, що саме хоче клієнт.

Розробник також отримує і чітке уявлення про те, яка саме функціональність повинна бути розроблена, як мають взаємодіяти окремі елементи різних компонентів, яка логіка стоїть за кожним запитом клієнта.

Класифікація прототипів

За К. Вігерсом розглянемо наступні класифікації прототипів:

- горизонтальні і вертикальні;
- одноразові і еволюційні;
- паперові та електронні, розкадровка.

Горизонтальний прототип

Горизонтальний або поведінковий прототип (horizontal prototype, behavioral prototype) моделює інтерфейс користувача програми, не зачіпаючи логіку обробки та базу даних.

Горизонтальні прототипи слід використовувати для досягнення мети прояснення неясних або багатоальтернативного вимог.

Вертикальний прототип

Вертикальний або структурний прототип (vertical prototype, structural prototype) не обмежується інтерфейсом користувача. Він реалізує вертикальний "зріз" системи, зачіпаючи всі рівні її реалізації. При створенні такого роду прототипів рекомендується використовувати ті мови та середовища реалізації, що і при виготовленні цільової системи (що, взагалі кажучи, зовсім не обов'язково для горизонтальних прототипів).

Основні *цілі застосування* такого роду прототипів - аналіз застосовності, перевірка архітектурних концепцій.

Одноразовий прототип

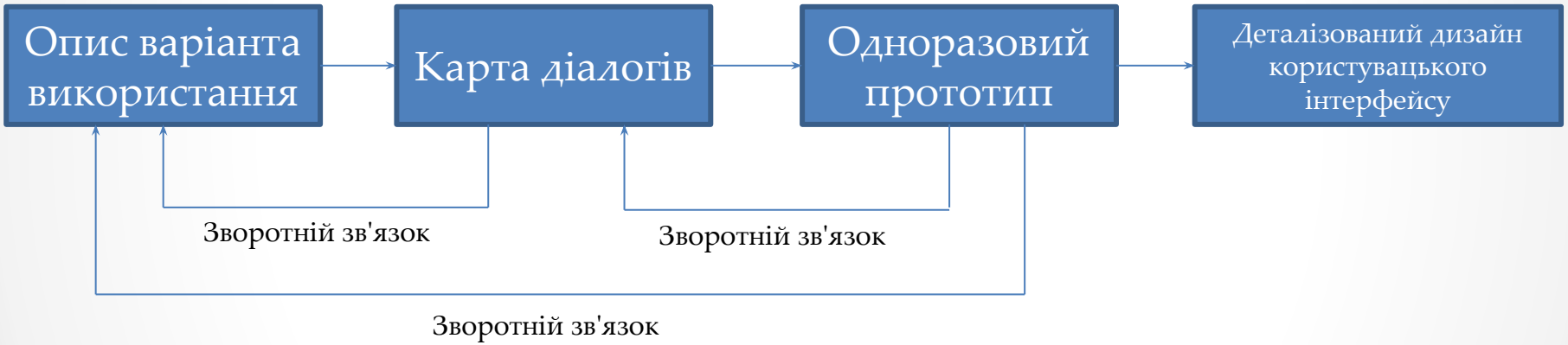
Одноразовий чи дослідницький прототип (throwaway prototype, exploratory prototype) створюється, коли потрібно швидко промакетитувати ті чи інші аспекти і компоненти системи.

Цілям створення дослідних прототипів служить технологія RAD (rapid application development) - швидка розробка застосувань.

Одноразовий прототип повинен створюватися швидко. При його розробці не слід приділяти увагу питанням повторного використання коду, якості, швидкодії, технологічності і т.п.

У результаті виходить "сирий" код, який може містити значну кількість дефектів. Необхідно вжити заходів до того, щоб фрагменти коду, що реалізують такого роду прототипи, не стали частиною цільової системи.

К. Вігерс наводить таку схему переходу від одноразового прототипу до детально відпрацьованого UI:



Еволюційний прототип

Еволюційний прототип (evolutionary prototype) створюється, як перше наближення системи, покликане стати згодом самою системою.

Код еволюційного прототипу повинен послідовно, протягом однієї або більше ітерацій, перерости в код цільового застосування. Тому даний вид прототипів вимагає всього того, від чого слід відмовитися при створенні одноразових прототипів: скрупульозної розробки, застосування технологічних методів і прийомів, тестування результатів і т.п.

У таблиці наведено співвідношення між розглянутими вище 4 видами прототипів:

	Одноразові	Еволюційні
Горизонтальні	• Вияснення та уточнення прикладів використання функціональних вимог • Виявлення пропущених вимог • Дослідження можливих варіантів інтерфейсу користувача	• Реалізація базових варіантів використання • Реалізація додаткових варіантів використання за пріоритетами • Реалізація і доопрацювання web-сайту • Адаптація системи до швидко змінюючим вимогам бізнесу
Вертикальні	• Демонстрація технічної здійсненності	• Реалізація та збільшення ключової клієнт-серверної функціональності та рівнів комунікації • Реалізація та оптимізація основних алгоритмів • Тестування та налаштування виробничості

Паперовий прототип

Паперовий прототип (paper prototype) - відмінна альтернатива розглянутим вище різновидам електронних прототипів у випадку, коли Розробник обмежений у ресурсах. Начерки інтерфейсів на папері, звичайно, не замінять інтерфейс, створений в середовищі розробки. Однак, при всіх недоліках, у таких прототипів є два істотні достоїнства.

Замовник не стане акцентувати увагу на колірному рішенні, формі кнопок і т.п., відволікаючись від аналізу функціональності.

Замовник ніколи не скаже, дивлячись на паперовий інтерфейс: "Та ви, я бачу, вже створили систему на 85%! Давайте закінчимо її в перебігу тижня".

Розкадровка

Рішенням проміжного між електронним і паперовим варіантами прототипів UI класу є презентації, виготовлені за допомогою засобів електронного офісу (наприклад, комбінації Microsoft Visio та Microsoft PowerPoint). У цьому випадку користувач позбавлений свободи вибору, наданої йому поведінковим прототипом. Але ідею покрокової зміни екранів в процесі реалізації сценарію варіанту використання цілком можна реалізувати. Даний вид рішення визначається як *пасивна розкадровка*. *Активна розкадровка* є подальшим розвитком поняття пасивної розкадровки, із застосуванням засобів анімації і т.п. Третій вид розкадровки - *інтерактивна* являє собою електронний одноразовий горизонтальний прототип.

Найбільш розповсюджена класифікація видів прототипів

- **Паперові**
  - швидко створюються,
  - не вимагають великих витрат
  - складно використовувати далі (при розробці , тестуванні)
  - не можливо надати замовнику
  - вимагають деталізації і глибини розрахунку
- **Скетчі , мокап**
  - найбільше використовуються в команді для обговорення пропозицій і нововведень в проекті
  - легкі у розробці
  - існують онлайн-версії програм для розробки таких прототипів
  - використовують при розробці проекту
  - у більшості своїй не дозволяють спробувати продукт , є лише статичні сторінки
- **Інтерактивні прототипи**
  - дозволяють повноцінно залучити користувача ,
  - показати систему цілком і в дії найбільш ефективні
  - необхідно більше часу на розробку прототипу
  - схожі на готовий продукт

Інтерактивні прототипи інтерфейсів

Інтерактивний прототип - це діюча модель користувальницького інтерфейсу. Він імітує роботу системи, так що її можна оцінити в дії ще до того, як розпочато розробку.

Хоча прототип не зберігає дані і не працює з базою даних, в іншому він може бути максимально наближений до майбутнього продукту.

Зазвичай це з'єднані між собою *HTML-сторінки* імітації реакцій системи через *JavaScript*. Хоча в деяких випадках він може бути зроблений на *Flash* або інших технологіях.

Наприклад: http://demo.uimodeling.ru/bfm-gazeta/prototype/

Призначення інтерактивних прототипів

- ***Демонстрація інвесторам***. Діючу модель продукту можна показати інвесторам ще до того, як вкладені час і гроші в розробку.
- ***Доопрацювання концепції***. При роботі над складними та інноваційними проектами постійно йдуть експерименти над концепцією і, відповідно, інтерфейсом. У прототип легко вносити правки, а значить і порівнювати альтернативні рішення.
- ***Раннє юзабіліті-тестування***. Прототип дозволяє перевірити зручність і ефективність системи, показавши її потенційним користувачам.
- ***Частина технічного завдання для розробників.*** Команді розробникам простіше зрозуміти як повинна працювати система, попрацювавши з її діючою моделлю.

Етапи створення інтерактивних прототипів інтерфейсів

I. Створення прототипу

II. Об'єднання і наповнення прототипу

III. Доопрацювання прототипу

Фінальний етап. Приймання

Софт для створення прототипів

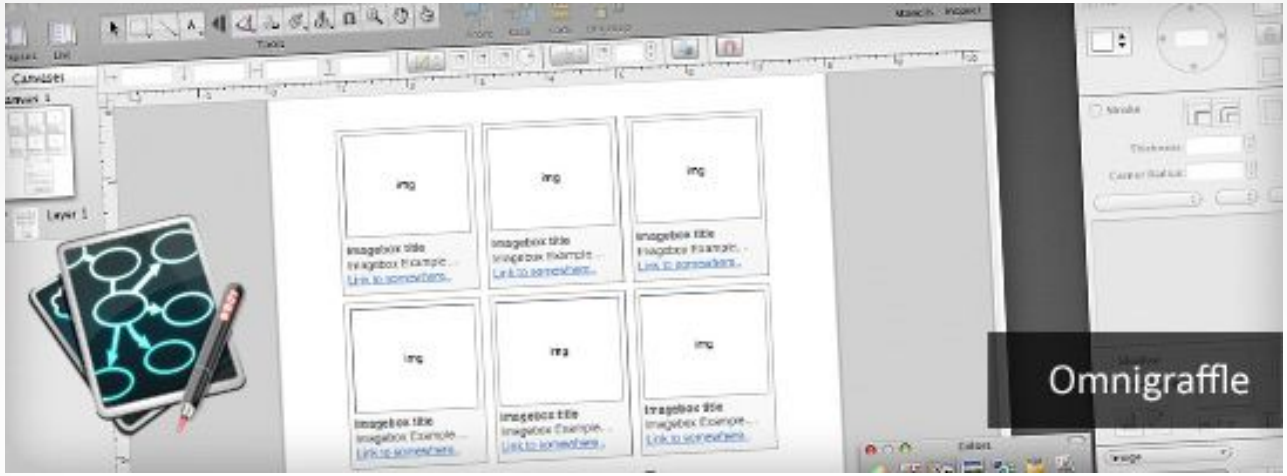
Інтерактивні прототипи:

axure.com
adobe.com
flairbuilder.com
foreui.com
guimachine.ru
proto.io
pidoco.com
protoshare.com

Скетчі, мокапи:

balsamiq.com
mockupbuilder.com
gomockingbird.com
iphonemockup.lkmc.ch

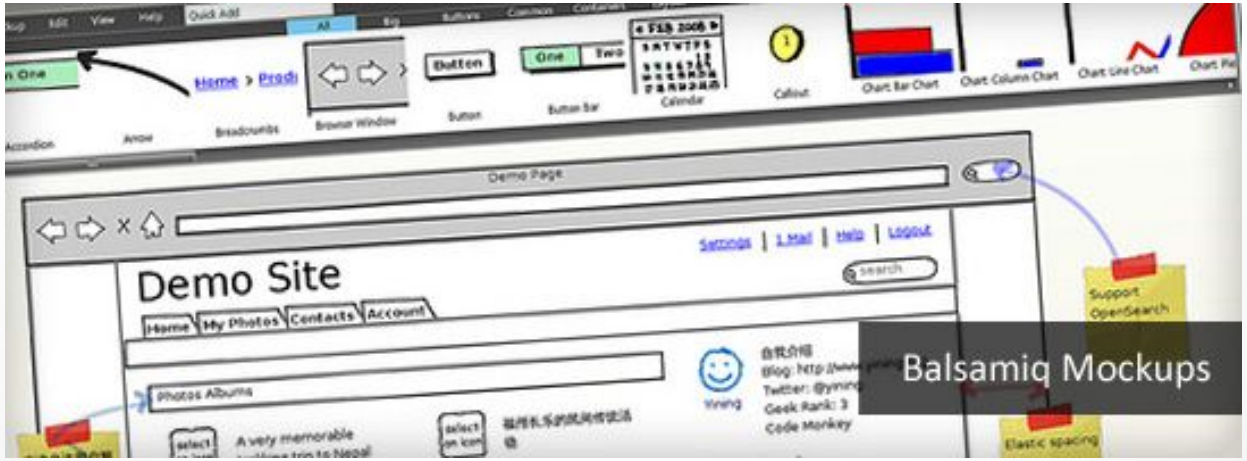
1. Omnigraffle



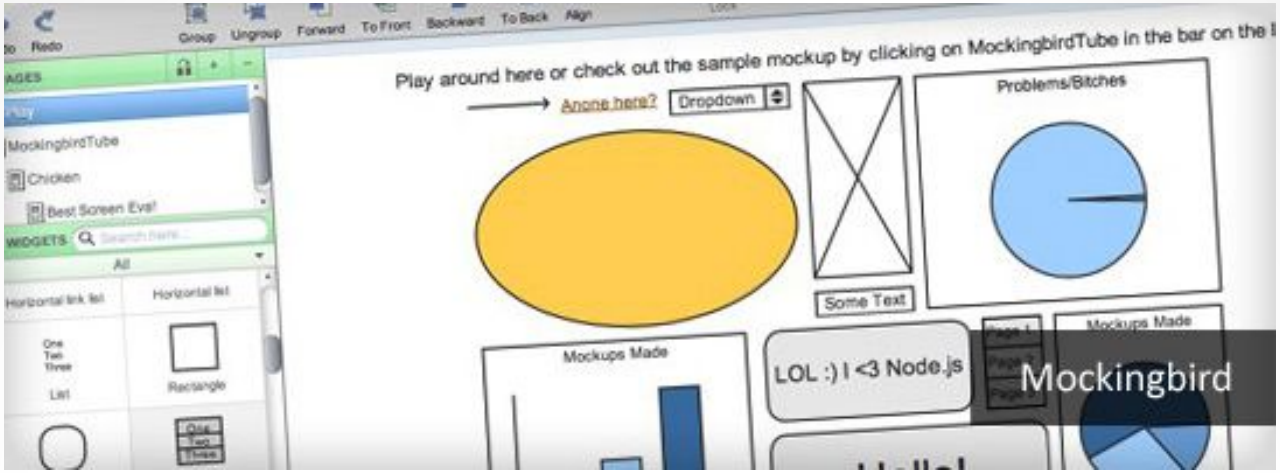
2. ConceptDrawPro

3. Pidoco

4. BalsamiqMockups



5. Mockingbird



6. iPlotz

Ілюстровані сценарії прецедентів

Ілюстровані сценарії прецедентів, ІСП, поряд з прототипами дозволяють досягти кращого розуміння між Замовником та Розробником. Але якщо прототипи адресовані скоріше Замовнику, ніж Розробнику, то з ІСП ситуація обстоїть навпаки: вони містять додаткові відомості, що допомагають Розробнику краще зрозуміти специфіку проблемної області і, тим самим, краще відобразити її в інтерфейсі користувача.

Основна ідея ІСП - "розбавити" текст опису сценарію варіанту використання аспектами застосовності.

Аспект застосовності - інформація, що дозволяє розширити опис прецеденту описами, конкретизують ті чи інші його особливості і, в кінцевому підсумку, підвищити ступінь комфортності користувача.

Розрізняють 3 різновиди аспектів застосовності: орієнтири, середні значення атрибутів і обсяги об'єктів, середня інтенсивність використання.

Орієнтири

Орієнтири - це опис опціональних функціональних можливостей системи. Відсутність таких можливостей не призводить до фатальної невдачі. Присутність - покращує застосовність, постачаючи корисною інформацією. Орієнтири слід розцінювати не як вимоги, а як побажання або рекомендації.

Середні значення атрибутів і обсяги об'єктів

Дана інформація дозволяє оптимальніше побудувати користувацький інтерфейс і оцінити на ранніх стадіях проекту "вузькі місця" в обробці даних, які можуть вплинути на продуктивність системи.

Середня інтенсивність використання

Середня інтенсивність використання дозволяє виділити сценарії "масового" використання, в яких все повинно бути ідеально (швидкодія, зручність користування, мінімум дій на виконання операцій). Наприклад - інтерфейс касира в супермаркеті. Інша крайність - сценарії, що виконуються від випадку до випадку, не кожен день і не потребують особливої оперативності (наприклад, розрахунок заробітної плати за місяць). Ці дані дозволяють структурувати подачу інформації, прибрати з "головних" інтерфейсів рідко використовувані опції і т.п.

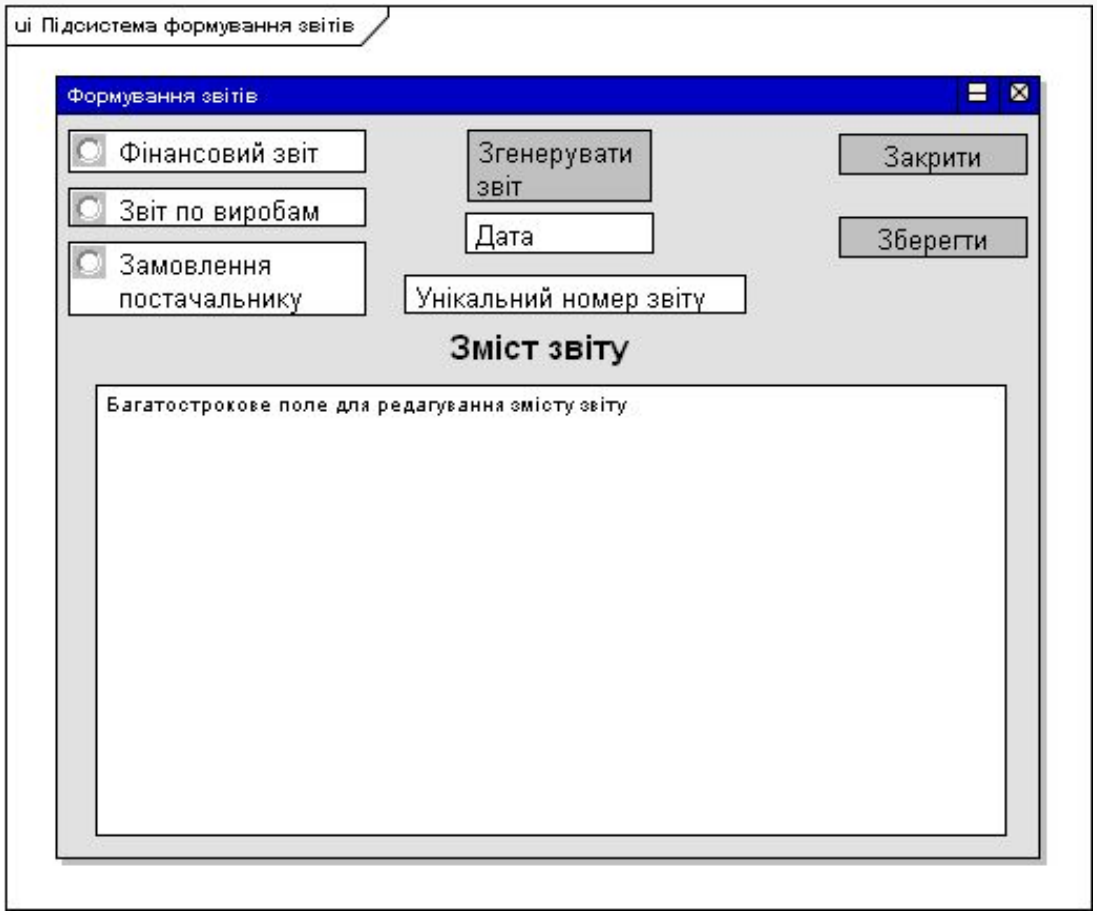


Рисунок 1 – Прототип до екранної форми «Підсистема формування звітності» розроблений засобами Enterprise Architect

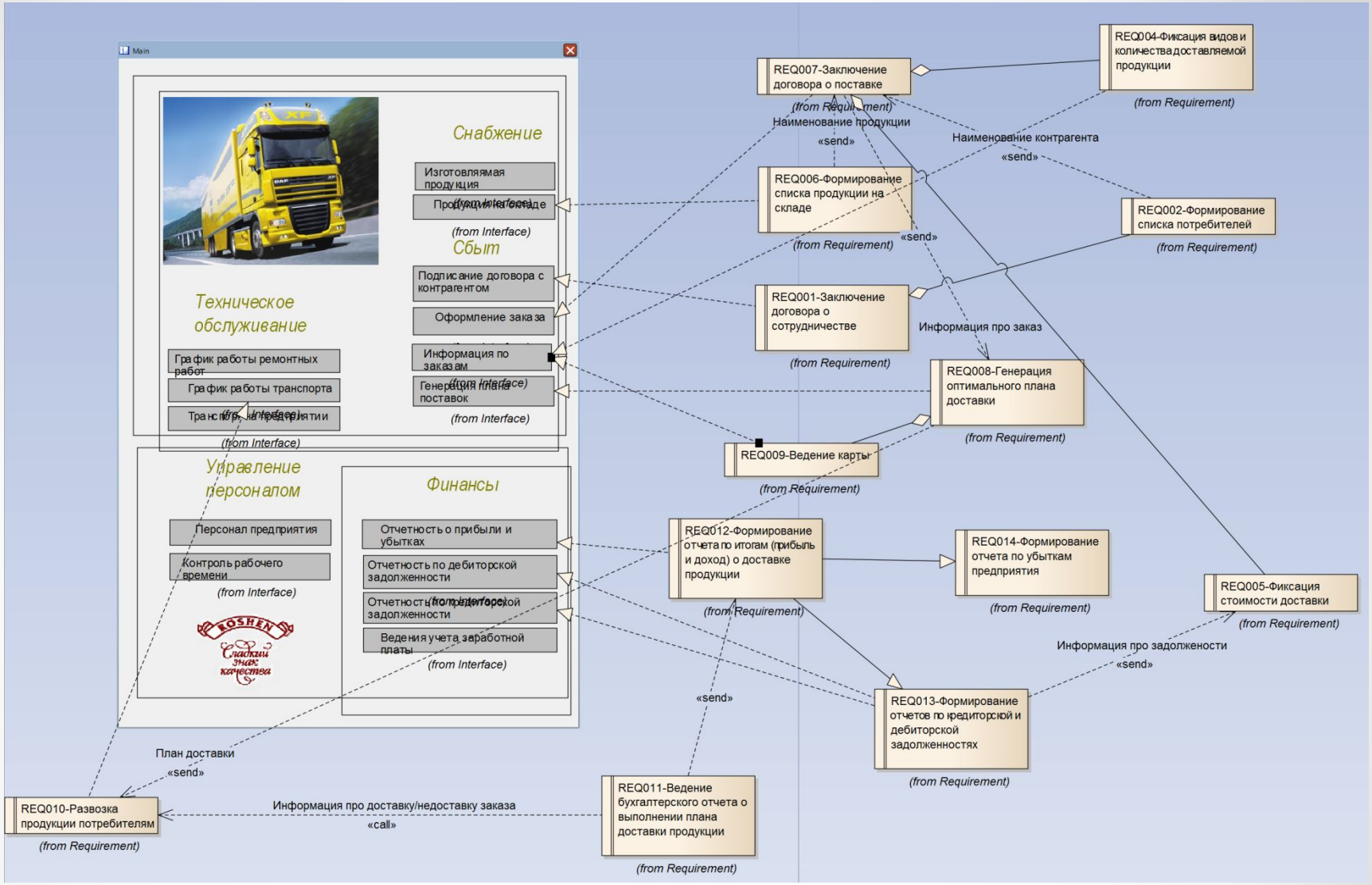


Рисунок 2 – Модель взаємозв'язку вимог і інтерфейсних форм