**Лабораторна робота №5**

**JS фреймворки та бібліотеки як спрощення реалізації фронтенду програмних систем**

*Мета*: дослідити можливості реалізації фреймворків та бібліотек у розробці фронтенду програмних систем, реалізувати програмно основні аспекти та проаналізувати отримані результати.

*Завдання*

1. Розглянути, проаналізувати основні JS фреймворки та бібліотеки (див. Лекції 11 та 12) та, базуючись на предметній області, вимогах та готових реалізаціях у роботах 1, 2 і 4, обрати фреймворк або бібліотеку для реалізації проєкту, який відповідав би предметній області та завданням робіт 1,2 та 4. Надати у звіті аргументи на користь обраного JS фреймворку/ бібліотеки;

2. При обранні фрейворку обов’язково надати огляд архітектури, яка ним передбачена.

3. За допомогою обраного фреймворку/бібліотеки (див. Додаток 7) програмно реалізувати односторінковий аналог\*, який відповідає тематиці робіт 1,2 та 4 і містить не менше 75% їх контенту.

4. Проаналізувати отриману у п.3 реалізацію за допомогою метрик (LOC, HIT, тощо) та\або швидкодією, швидкістю завантаження сторінки, тощо, у порівнянні з реалізаціями, що отримані у роботах 1,2 та 4 (за можливості). Перелік можливих інструментів надано нижче у теоретичних відомостях.

**Зауваження:** у п.3 крім односторінкового аналогу можливі інші програмні реалізації – форма авторизації, форма оплати, оформлення замовлення, подання статті до друку на сайті новин, тощо.

*Теоретичні відомості*

**Завдання 1.** Фреймворк (framework — конструкція, структура) — програмне середовище спеціального призначення, своєрідний каркас, який використовується для того, щоб істотно полегшити процес об’єднання певних компонентів під час створення програм.

*Основні функції фреймворків:*

Керування станом. Це як інструменти для зручного управління станом додатка, що дає змогу програмістам легко відслідковувати й оновлювати дані в додатку без необхідності вручну керувати станом.

Компонентний підхід. Фреймворки часто використовують цей підхід, який дає змогу розбити інтерфейс на окремі компоненти, кожен з яких відповідає за свою функцію. Це спрощує розробку, повторне використання коду та забезпечує більш чисту й організовану структуру.

Маршрутизація. Це механізми для управління маршрутизацією всередині додатка, що дає змогу створювати різні сторінки та маршрути, обробляти переходи між ними й оновлювати URL відповідно до поточного стану додатка.

Бібліотека — це набір функцій, які розв’язують конкретне завдання в програмі. Вона може використовувати шаблони повідомлень, раніше скомпільований код, класи або підпрограми — і використовуватися багато разів.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**jQuery** є популярною бібліотекою JavaScript, яка спрощує взаємодію з DOM (Document Object Model) та робить роботу з подіями та AJAX запитами більш простою та кросбраузерною.

***Основні функції та можливості jQuery:***

⮚ Вибір елементів: завдяки потужній системі селекторів, можна легко вибрати один чи багато елементів з DOM. Наприклад: $('.myClass') вибере всі елементи з класом "myClass".

⮚ Маніпуляція DOM: методи для зміни структури та вмісту сторінки. Наприклад: .html(), .text(), .append(), .prepend().

⮚ Обробка подій: простий і зручний механізм обробки подій, таких як клік, зміна, наведення тощо. Наприклад: .click(), .on(), .hover().

⮚ Анімація: забезпечує анімаційні ефекти, такі як плавне згортання, розгортання, зміна розміру, переміщення та інші. Наприклад: .fadeIn(), .slideUp(), .animate().

⮚ AJAX: легка і зручна робота з AJAX-запитами для асинхронного обміну даними з сервером. Наприклад: .ajax(), .get(), .post().

⮚ Взаємодія з атрибутами та стилями: можливість отримання та встановлення значень атрибутів та стилів елементів. Наприклад: .attr(), .css().

⮚ Вбудовані ефекти та утиліти: бібліотека має різноманітні утиліти для роботи з масивами, об'єктами, подіями, а також вбудовані ефекти для створення анімації та зміни стану елементів.

⮚ Ланцюговий синтаксис: можливість використовувати ланцюговий синтаксис, що робить код більш компактним та читабельним.

**Деякі фрагменти коду:**

**$(document).ready(function() { ... });**: jQuery-функція, яка викликається тільки після того, як весь HTML-документ буде повністю завантажений. Весь код, який знаходиться всередині цієї функції, виконується після завершення завантаження сторінки;

**let resursElement = $("selector");**: визначає змінну **resursElement** і присвоює їй об'єкт, який представляє елемент що відповідають заданому селектору "**selector**".

**$("body").css("background-color", "otherColor");**: змінює колір фону сторінки з "*background-color*" на "*otherColor*".

**$("# nameId").click(function() { ... });**: додає обробник події click для елемента з ідентифікатором **#nameId**. Коли цей елемент клікають, викликається певна функція. **setTimeout(function() { loginButton.removeClass("nameClass"); }, 500);** - Використовуємо функцію **setTimeout** для встановлення затримки перед видаленням класу "nameClas**s**". Після 500 мілісекунд (0.5 секунди) затримки викликається функція, яка видаляє клас.

Більше за посиланням: https://www.w3schools.com/jquery/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Svelte** - це модерний фреймворк для створення користувацьких інтерфейсів. Його основна концепція полягає в тому, що код написаний на Svelte компілюється в чистий, оптимізований JS під час збірки, що дозволяє уникнути використання великої кількості коду під час виконання програми. ***Основні функції та можливості фреймворку Svelte***:

⮚ Одноразовий компілятор (Single-file Component Compilation): у svelte використовується підхід компіляції в процесі розробки, під час якого його код перетворюється на оптимізований нативний JavaScript в момент збірки. Це означає, що власне на етапі виконання немає потреби включати бібліотеку Svelte, що полегшує кінцевий розмір коду для клієнтів.

⮚ Реактивність "вбудована у фреймворк" (Built-in Reactivity): svelte автоматично відслідковує зміни даних і оновлює DOM, коли дані змінюються. Це відбувається без додаткового коду, що дозволяє розробникам уникнути деяких типових задач, пов'язаних з управлінням станом у інших фреймворках.

⮚ Менша кількість коду: svelte пропонує спрощені шаблони та менше кількість додаткового коду порівняно з іншими фреймворками. Це дозволяє швидше створювати функціональність і зменшує шанси на помилки в коді.

⮚ Менше навантаження на виконання: оскільки Svelte використовує підхід компіляції, результатом є оптимізований, ефективний код, що прискорює завантаження сторінки та покращує продуктивність додатків.

⮚ Компонентний підхід: як і в інших фреймворках, Svelte підтримує створення додатків через компоненти. Це дозволяє створювати відокремлені, перевиконані частини коду.

⮚ Можливості анімації: svelte надає різноманітні можливості для роботи з анімаціями, що полегшує створення привабливих та динамічних інтерфейсів.

**Деякі фрагменти коду: за посиланнями https://svelte.dev/examples/hello-world та https://developer.mozilla.org/en-US/docs/Learn/Tools\_and\_testing/Client side\_JavaScript\_frameworks#svelte\_tutorials**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**React** – це бібліотека для веб-інтерфейсу та рідного інтерфейсу користувача. Створюйте інтерфейс користувача з окремих частин, які називаються компонентами, написаними на JS.

***Основні функції та можливості бібліотеки React:***

⮚ Компоненти: React базується на концепції компонентів. Компонент - це відокремлений, самодостатній блок, який може бути використаний для побудови інтерфейсу. Ви можете створювати власні компоненти, що дозволяє легко організовувати код та покращувати його читабельність.

⮚ Віртуальний DOM: React використовує віртуальний DOM для оптимізації оновлення елементів на сторінці. Замість того, щоб маніпулювати кожним елементом безпосередньо, React працює з віртуальним представленням сторінки та визначає оптимальні зміни перед оновленням реального DOM.

⮚ JSX: JSX - це синтаксис, який дозволяє вписувати HTML-подібний код безпосередньо в JavaScript. Це полегшує читання та написання коду, а також дозволяє використовувати всю потужність JavaScript для роботи з компонентами.

⮚ Стан і Властивості: React використовує концепцію стану та властивостей для управління даними у додатку. Стан - це внутрішній стан компонента, а властивості - це дані, які передаються в компонент ззовні.

⮚ Життєвий цикл компонентів: Кожен React компонент має життєвий цикл, що включає в себе методи, які викликаються на різних етапах життя компонента. Наприклад, метод componentDidMount викликається після того, як компонент був відображений на сторінці.

⮚ Роутінг: Для управління переходами між сторінками React використовує роутінг. Популярні бібліотеки, такі як React Router, надають зручний інтерфейс для роботи з роутінгом у React додатках.

⮚ Керування станом: Для управління станом додатку React використовує підйом стану та контекст. Це дозволяє передавати стан вниз по дереву компонентів.

**Деякі фрагменти коду: за посиланням https://react.dev/learn/tutorial-tic-tac-toe та https://developer.mozilla.org/en-US/docs/Learn/Tools\_and\_testing/Client side\_JavaScript\_frameworks#react\_tutorials та https://www.w3schools.com/react/default.asp**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Vue.js –** фреймворк, що генерує відповідно до свого класу одразу HTML, CSS та JS код, який можна використовувати для зручної та швидкої побудови веб-сайтів та API. Він є відкритим і легко розширюваним, що дозволяє розробникам поетапно впроваджувати його у свої проекти. Vue.js фокусується на забезпеченні простоти

інтеграції та використання, а також на ефективності у створенні односторінкових застосунків (SPA) та інших сучасних веб-застосунків.

***Основні функції та можливості Vue.js:***

⮚ Двостороннє зв'язування даних (Two-way Data Binding): Зміни в інтерфейсі автоматично оновлюють дані, і навпаки. Це робить управління станом застосунку більш зручним.

⮚ Компонентна архітектура: Застосунки на Vue.js будуються за допомогою компонентів, які можуть бути багаторазово використані в різних частинах застосунку.

⮚ Директиви (Directives): Vue.js надає безліч вбудованих директив для управління DOM-елементами. Наприклад, v-if, v-for, v-bindта інші. ⮚ Обробка подій: Легкість у обробці подій, таких як кліки, зміни введення та інші, за допомогою директив типу v-on.

⮚ Декларативний шаблонізатор: Vue.js використовує декларативний шаблонізатор, який дозволяє вбудовувати вирази та вказівки в HTML-подібному коді. Це полегшує розробку та розуміння структури сторінки.

⮚ Компоненти: Vue.js підтримує концепцію компонентів, що дозволяє розділити інтерфейс на невеликі та перевикористовувані частини. Кожен компонент має свій власний стан, методи та рендеринг.

**Деякі фрагменти коду: за посиланнями https://ua.vuejs.org/guide/essentials/component-basics.html та https://developer.mozilla.org/en-US/docs/Learn/Tools\_and\_testing/Client side\_JavaScript\_frameworks#vue\_tutorials**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Next.js** - це реактивний фреймворк для розробки веб-додатків на базі React. Він дозволяє легко створювати універсальні (із можливістю серверного рендерингу), статичні та односторінкові додатки.

***Основні функції та можливості Next.js:***

⮚ Універсальні додатки (Server-Side Rendering): одна з ключових особливостей - це підтримка серверного рендерингу, що поліпшує продуктивність та оптимізує індексацію пошуковими системами; кожна сторінка може мати свій серверний рендер, що дозволяє отримувати динамічний вміст з сервера перед тим, як сторінка буде відправлена клієнту.

⮚ Статичне покоління (Static Site Generation): підтримує статичне покоління для генерації статичних HTML-файлів для сторінок додатку; полегшує кешування сторінок та поліпшує швидкість завантаження, особливо для статичних вмістових сторінок.

⮚ Швидкість та оптимізація: вбудована оптимізація для завантаження зображень, використання CDN та автоматична оптимізація для шляхів веб-додатків; забезпечує можливість вибору між об’єктом get Static Props для статичного покоління та getServerSideProps для серверного рендеринга в залежності від потреб.

⮚ Routing на основі файлової структури: використовує конвенцію на основі файлової структури для роутингу. Файли в папці pages автоматично стають шляхами до відповідних сторінок.

⮚ Маршрутизація: підтримка динамічної маршрутизації, де шляхи можуть містити параметри, що робить простим обробку динамічного контенту. ⮚ API Routes: зручна можливість визначення API-маршрутів безпосередньо в проекті Next.js.

⮚ Декларативне визначення мета-тегів: забезпечує легке визначення мета-тегів для кращої оптимізації для пошукових систем та соціальних мереж. ⮚ Розширення за допомогою плагінів: екосистема плагінів, яка дозволяє розширювати та налаштовувати функціональність Next.js.

Next.js найкраще підходить для:

• Розробки односторінкових додатків (SPA) та багатосторінкових додатків (MPA).

• Створення статичних сайтів.

• Реалізації серверних рендерингових додатків.

• Розробки онлайн-навчальних платформ.

**Деякі фрагменти коду:**

Команда для створення шаблонного проєкту: npx create-next-app@latest . Після чого необхідно відповісти yes\no на ряд питань для додаткових налаштувань. Далі переходимо у директорію проєкту. Більше за посиланням: https://nextjs.org/docs/pages/api-reference/components/head

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Backbone.js** — це фреймворк веб-додатків із розширеним клієнтом на основі JS, заснований на парадигмі проектування модель–подання–контролер, призначений для підключення до API через інтерфейс RESTful JSON.

***Основні функції та можливості фреймворку Backbone***:

⮚ Моделі: представляють собою дані додатка і можуть включати логіку для їх обробки; забезпечують спрощений спосіб взаємодії з сервером;

⮚ Види: відповідають за представлення даних користувачу і взаємодіють з DOM; пов'язані з конкретним елементом в DOM і можуть реагувати на події.

⮚ Колекції: представляють собою групи моделей і дозволяють виконувати операції з групою моделей одночасно; забезпечують ряд методів для фільтрації та сортування даних.

⮚ Маршрутизатори: дозволяють визначати, які частини інтерфейсу відображаються при зміні URL; забезпечують взаємодію з історією браузера та можливість обробки подій маршрутування.

⮚ Події: використовує механізм подій для взаємодії між різними частинами додатку; моделі, види, колекції і інші об'єкти можуть висилати та слухати події. ⮚ Синхронізація з сервером: дозволяє взаємодіяти з сервером за допомогою AJAX-запитів, що спрощує отримання та відсилання даних.

⮚ Інтеграція з іншими бібліотеками: Backbone легко інтегрується з іншими бібліотеками та фреймворками, наприклад, з jQuery або Underscore.js.

**Деякі фрагменти коду: за посиланням https://docs.org.ua/**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ember.js -** це відкритий фреймворк для розробки веб-додатків, написаний на JavaScript. Він базується на парадигмі архітектури Model-View-Controller (MVC) та надає інструменти для ефективної розробки великих та складних клієнтських додатків.

***Основні функції та можливості Ember.js:***

⮚ Шаблонізатор Handlebars: Ember.js використовує шаблонізатор Handlebars для відображення вмісту веб-сторінок. Handlebars дозволяє використовувати динамічні дані в HTML-подібних шаблонах.

⮚ Автоматичне зв'язування даних (Data Binding): Ember.js вбудовує механізми автоматичного оновлення інтерфейсу користувача при зміні даних моделі. Це спрощує управління станом додатка та покращує взаємодію з користувачем.

⮚ Роутінг та навігація: фреймворк надає потужну систему роутінгу, яка дозволяє легко управляти станом додатка та переходами між різними сторінками. ⮚ Компоненти: Ember.js підтримує концепцію компонентів, яка дозволяє розділити інтерфейс на невеликі та перевикористовувані блоки коду. Компоненти можуть мати власний внутрішній стан та логіку.

⮚ Моделі та Адаптери: Ember.js надає засоби для визначення моделей даних та їх взаємодії з сервером через адаптери. Це полегшує роботу з серверними даними та їх синхронізацію з клієнтською частиною додатка.

⮚ Інструменти для тестування: Ember.js ставить на тестування, надаючи інтеграцію зі зручними інструментами для автоматизованого тестування коду.

⮚ CLI (Command Line Interface): Ember CLI дозволяє легко створювати, розгортати та управляти проектами Ember.js. Він надає швидкий спосіб генерувати код, тести, моделі та інші елементи додатка.

⮚ Розширюваність: Ember.js дозволяє розширювати його функціональність через різноманітні додаткові бібліотеки та модулі.

**Деякі фрагменти коду: за посиланням https://developer.mozilla.org/en US/docs/Learn/Tools\_and\_testing/Client**

**side\_JavaScript\_frameworks#ember\_tutorials**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Nuxt.JS** — фреймворк із серверним рендерингом Vue.js. Він вміє генерувати cтатичні HTML-сайти на сервері, а також охоплює багато готових рішень для складання вебдодатків із хорошим SEO і UX. З ним просто й легко робити інтерактивні вебдодатки, які будуть відмінно індексуватися пошуковими мережами.

***Основні функції та можливості Nuxt.JS***:

⮚ Серверний рендеринг: технологія, при якій весь HTML-код сторінки генерується на сервері, дає змогу вебдодаткам швидше завантажуватися, незалежно від потужності пристрою. Це особливо вигідно для сторінок зі статичною інформацією.

⮚ Модульна архітектура: наявність готових рішень для підключення Google Аnalytics, генерації sitemap, модулі для pwa, axios, firebase і багато чого іншого. Водночас все це оптимізовано для SEO і продуктивно, а головне можна тонко налаштовувати не втрачаючи у швидкості роботи.

**Деякі фрагменти коду: за посиланням https://nuxt.com/docs/examples/hello-world**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Angular** — це веб-фреймворк, який дозволяє розробникам створювати швидкі та надійні програми. Angular, який підтримується спеціальною командою в Google, надає широкий набір інструментів, API та бібліотек для спрощення та оптимізації робочого процесу розробки. Angular дає надійну платформу, на якій можна створювати швидкі, надійні програми, які масштабуються як з розміром вашої команди, так і з розміром вашої кодової бази.

***Основні функції та можливості***

⮚ Розвинуті інструменти командного рядка - Angular CLI. З його допомогою можна керувати проектом виключно за допомогою командного рядка, наприклад: створювати проект, модулі, компоненти.

⮚ В основі Angular ключовими є концепції інкапсуляції та чіткої ієрархії. Компоненти є за замовчуванням ізольованими, і втручання можливе лише за певних налаштувань. Отримати доступ до компонента можливо лише, якщо для цього вибудована чітка ієрархія, з окресленими шляхами доступу до, наприклад, станів.

⮚ В основі Angular лежить мова TypeScript. Головною відмінністю її від JS - типізація, яка має спросити операції над даними, роботу з ООП і т.д. ⮚ Як і більшість великих проектів, якими зазвичай є сайти-сервіси, вони завжди застосовують фреймворки для спрощення роботи, швидкості створення проекту. Angular є визнаним корпоративним рішенням, який широко застосовується.

**Деякі фрагменти коду: за посиланням https://developer.mozilla.org/en US/docs/Learn/Tools\_and\_testing/Client**

**side\_JavaScript\_frameworks#angular\_tutorials**

**Завдання 3.** Допустимим є аналіз за допомогою:

1. Lighthouse, обираючи робочий процес, який вам найкраще підходить: Chrome DevTools (перевірити сторінки і наявність звіту в зручному форматі); з командного рядка (автоматизовані запуски за допомогою сценаріїв оболонки); модуль Node (інтеграція у свої системи).

2. Web Vitals - вимірює основні веб-показники, надаючи миттєвий відгук про показники завантаження, інтерактивності та зміни макета.

3. Інтуїтивний (або використання імплементованих рішень в IDE) - оцінка метрик LOC, HIT, тощо. Більше метрик у Додатку 8.

**Додаток 7.**

| **№** | **Назва фреймворку/бібліотеки** |
| --- | --- |
| 1 | React (ланка: https://react.dev/ ) |
| 2 | Vue.js (ланка: https://vuejs.org/) |
| 3 | jQuery (ланка https://jquery.com/) |
| 4 | Ember (ланка https://emberjs.com/) |
| 5 | Backbone.js (ланка https://backbonejs.org/) |
| 6 | Svelte (ланка: https://svelte.dev/) |
| 7 | Next.js (ланка: https://nextjs.org/) |
| 8 | Nuxt.js (ланка: https://nuxt.com/) |
| 9 | Angular (ланка: https://angular.dev/)\*\* (вимагає знань TypeScript!) |
| \* | Інший фреймворк/бібліотека з аргументацією вибору (Preact, Polymer, т.п.) |

Додаток 8.

| **Назва**  **Короткий опис Детальний опис Клас або тип метрики**  **метрики** |
| --- |
| Інтервальна міра Складність програм визначається у вигляді відрізка [Z (G), Z (G) + h], де  **Метрика**  Z (G) - метрика МакКейба, h = n - 1, для n-місного предиката і h = 0, для простого  Топологічна  **Майерса**  предиката |
| Логічна складність програми визначається як насиченість програми виразами типа IF-THEN-ELSE. При  **Метрика**  цьому вводяться дві характеристики: CL - абсолютна складність програми, що характеризується кількістю  **Джилба**  операторів умови; cl - відносна складність програми, що характеризується насиченістю програми операторами  умови, т. е. cl визначається як відношення CL до загальної кількості операторів:  Топологічна  • Кількість операторів циклу;  • Кількість операторів умови;  • Число модулів або підсистем;  • Ставлення числа зв'язків між модулями до числа модулів;  • Ставлення числа ненормальних виходів з безлічі операторів до загальної кількості операторів. |
| Пара (цикломатичне  (A, B), де A - метрика МакКейба, І - число виконуваних операторів. Топологічна  **Метрика**  число, число операторів)  **Хансена** |
| Топологічна метрика  M(G) = (n (G), N, Q0)  **Метрика**  Чена  Топологічна  **Чена** |
| Вузлова міра (число  На полях лістингу малюються лінії передачі управління від одного  **Метрика**  вузлів передач управління)  оператора до іншого (передбачається, що один оператор займає рівно один  Топологічна  **Вудворда**  рядок). Число перетинів цих ліній дає значний метрики. |
| Функціональне число  Дана міра враховує рівень вкладеності і протяжності програми.  **Метрики**  (сума наведених складнощів  Кожній вершині присвоюємо свою складність відповідно до  **Харрисона,**  всіх вершин керуючого графа);  операторами-ром, який вона зображає. Ця початкова складність вершини може  **Мейджела**  Функціональне  значиться будь-яким способом, включаючи використання заходів Холстеда.  відношення (відношення числа  Функціональна міра (SCOPE) програми - це сума наведених складнощів  Топологічна  вершин графа до  всіх вершин керуючого графа.  функціонального числа);  Функціональним відношенням (SCORT) називається відношення числа  Регулярні вирази (число  вершин в керуючому графові до його функціональної складності, причому з  операндів, операторів і дужок  числа вершин виключаються термінальні.  в регулярному виразі  SCORT приймає різні значення для графів з однаковим цикломатичним  керуючого графа програми)  числом. |
| Модифікована  Міра Пивоварського ставить за мету врахувати в оцінці складності ПС  **Метрика**  цикломатична міра складності  відмінності не тільки між послідовними і вкладеними керуючими  **Пивоварского**  конструкціями, але і між структурованими і неструктурованими програмами. |

| Вона виражається відношенням N (G) = v \* (G) + Σ Pt, де v \* (G) - модифікована  цикломатична складність, обчислена так само, як і V (G), але з однією  Топологічна  відмінністю: оператор CASE.  Для підрахунку глибини укладення предикатних вершин  використовується число "сфер впливу". Під глибиною вкладеності розуміється  число всіх "сфер впливу" предикатів, які або повністю утримуються в сфері  даної вершини, або перетинаються з нею. Глибина вкладеності збільшується за  рахунок вкладеності не самих предикатів, а "сфер  впливу". |
| --- |
| Міра складності,  Виділяються три етапи обчислення даної метрики:  **Метрика**  заснована на числі можливих  1) Для кожної керуючої змінної i обчислюється значення її складної  **Мак-Клура**  шляхів виконання програми,  функції C (i) за формулою:  зокрема керуючих конструкцій  C (i) = (D (i) \* J (i)) / n  і змінних;  Де D (i) - величина, що вимірює сферу дії змінної i. J (i) - міра складності  взаємодії модулів через змінну i, n - число окремих модулів в схемі розбиття.  2) Для всіх модулів, що входять в сферу розбиття, визначається значення  їх складних функцій M (P) за формулою  M (P) = fp \* X (P) + gp \* Y (P)  де fp і gp - відповідно число модулів, які безпосередньо передують і  безпосередньо наступних за модулем P, X (P) - складність звернення до модуля  Метрика, спрямована  P, Y (P) - складність управління викликом з модуля P інших модулів.  на зміну архітектури  3) Загальна складність MP ієрархічної схеми розбиття програми на модулі  системи  задається формулою  MP = {Сума по всіх можливих значеннях P - модулів програми} (M (P))  Дана метрика орієнтована на програми, добре структуровані, складені з  ієрархічних модулів, які задають функціональну специфікацію і структуру  управління. Також мається на увазі, що в кожному модулі одна точка входу і  одна точка виходу, модуль виконує рівно одну функцію, а виклик модулів  здійснюється відповідно до ієрархічної системи управління, яка задає  відношення виклику на безлічі модулів програми. |
| Вводяться поняття локального і глобального потоку:  **Метрика**  Локальний потік інформації з A в B існує, якщо:  **Кафура**  Модуль А викликає модуль В (прямий локальний потік);  Модуль В викликає модуль А і А повертає В значення, яке використовується в В (непрямий локальний  потік);  Модуль С викликає модулі А, В і передає результат виконання модуля А в В;  Глобальний потік інформації з А в В через глобальну структуру даних D існує, якщо  модуль А поміщає інформацію в D, а модуль В використовує інформацію з D.  На основі цих понять вводиться величина I - інформаційна складність процедури:  I = length \* (fan\_in \* fan\_out) ^ 2 |

| тут:  length - складність тексту процедури (міряється через деяку із метрик обсягу, типу метрик Холстеда,  МакКейба, LOC і т.п.)  fan\_in - число локальних потоків всередині процедури плюс число структур даних, з яких процедура бере  інформацію  fan\_out - число локальних потоків з процедури плюс число структур даних, які оновлюються процедурою  Можна визначити інформаційну складність модуля як суму інформаційних складнощів вхфдних до нього  процедур.  Наступний крок - розглянути інформаційну складність модуля щодо деякої структури даних. Інформаційна  міра складності модуля щодо структури даних:  J = W \* R + W \* WrRd + WrRd x R + WrRd \* (WrRd - 1)  тут:  W - число процедур, які тільки оновлюють структуру даних;  R - Тільки читають інформацію зі структури даних;  WrRd - і читають, і оновлюють інформацію в структурі даних.  Метрика, заснована на обліку потоку даних |
| --- |
| Зважена сума різних  (Структура, взаємодія, обсяг, дані) (складність інтерфейсу,  **Метрика**  індикаторів  обчислювальна складність, складність введення / виведення, читабельність) Гібридна  **Зольновського,**  **Сіммонса, Тейера** |
| Складність  Логічна складність з урахуванням історії обчислень  **Метрика**  проектування  **Янгера**  насиченість  Топологічна  коментарями  Число зовнішніх  звернень  число операторів |
| Потік управління  У графі програми, де кожному оператору відповідає вершина, т. Е. Не  **Метрика**  програми вважається більш  виключені лінійні ділянки, при передачі управління від вершини a до b номер  **'Підрахування**  складним, якщо виникають  оператора a дорівнює min (a, b), а номер оператора b - max (a, b). Точка перетину  **точок перехрестя'**  точки перетину дуг передачі  дуг з'являється, якщо  управління.  Метрика складності  min (a, b) <min (p, q) <max (a, b) & max (p, q)> max (a, b) |  потоку керування програм  min (a, b) <max (p, q) <max (a, b) & min (p, q) <min (a, b).  Іншими словами, точка перетину дуг виникає в разі виходу управління за  межі пари вершин (a, b).  Кількість точок перетину дуг графа програми дає характеристики не  структурованості програми. |

| Вважається число  Нехай G = (V, E) - орієнтований граф програми. Число вхідних вершин у  **Метрика**  входів і виходів з вузлових  дуг називається негативною ступенем вершини, а число вихідних з вершини дуг  **граничних значень**  точок графа програми.  - позитивної ступенем вершини. Вершини у яких позитивна ступінь <= 1 назвемо  приймають вершинами, а вершини у яких позитивна ступінь> = 2 вершинами  відбору.  Для отримання оцінки за методом граничних значень необхідно розбити  граф G на максимальне число підграфів G ', що задовольняють наступним  умовам: вхід в підграф здійснюється тільки через вершину відбору; кожен  Метрика складності  підграф включає вершину (звану надалі нижньою межею подграфа), в яку можна  потоку керування програм  потрапити з будь-якої іншої вершини подграфа.  Число вершин, що утворюють такий підграф, так само скоригованої  складності вершини відбору. Кожна приймаюча вершина має скориговану  складність, рівну 1, крім кінцевої вершини, скоригована складність якої  дорівнює 0. Скориговані складності всіх вершин графа G підсумовуються,  утворюючи абсолютну граничну складність програми. Після цього визначається  відносна гранична складність програми:  S0 = 1 (v-1) / Sa,  де S0 - відносна гранична складність програми; Sa - абсолютна гранична  складність програми; v - загальне число вершин графа програми. |
| --- |
| Пара (модуль,  Пара "модуль-глобальна змінна" позначається як (p, r), де p - модуль, який  **Метрика**  глобальна змінна) (фактична /  має доступ до глобальної змінної r. Залежно від наявності в програмі реального  **звернення до**  можлива)  звернення до змінної r формуються два типи пар "модуль-глобальна змінна":  **глобальних**  Aup, Pup, Rup  фактичні і можливі. Можливе звернення до r за допомогою p показує, що область  **змінних**  існування r включає в себе p.  Метрика складності  Характеристика Aup говорить про те, скільки разів модулі Up дійсно  потоку даних  отримували доступ до глобальних змінних, а число Pup - скільки разів вони  могли б отримати доступ.  Ставлення числа фактичних звернень до можливих визначається  Rup = Aup / Pup.  Ця формула показує наближену ймовірність посилання довільного  модуля на довільну глобальну змінну. Очевидно, чим вище ця ймовірність, тим  вище ймовірність "несанкціонованої" зміни будь-якої змінної, що може істотно  ускладнити роботи, пов'язані з модифікацією програми. |
| Визначення спина  Спен - це число тверджень, які містять даний ідентифікатор, між його  **Метрика**  ґрунтується на локалізації  першим і останнім появою в тексті програми. Отже, ідентифікатор, що з'явився  Метрика складності  **Спена**  звернень до даних всередині  n раз, має Спен, рівний n-1. При великому Спен ускладнюється тестування і  потоку даних  кожної програмної секції.  налагодження. |
| Вводиться ряд метрик, в  WMC - Weighted methods per class (обчислюється складність методів  **Набір**  сумі дають уявлення про  будь-яким способом і береться сума складнощів)  **метрик Чидамбера**  складність ТОВ-коду.  DIT - Depth of Inheritance Tree (глибина дерева спадкування)  **і Кемерера (WMC,** |

| NOC - Number of Children (число нащадків)  Метрика для ООП  **DIT, NOC, CBO,**  CBO - Coupling between objects (зв'язність між об'єктами)  програм  **RFC, LCOM)**  RFC - Response For a Class(зв'язність класів)  LCOM - Lack of Cohesion in Methods (нестача зчеплення методів) |
| --- |
| **Зв'язність** Ступінь залежності  Content coupling (зв'язність у вмісті)  кожного модуля від кожного з  Common coupling (спільна зв'язність)  інших модулів  External coupling (зовнішня зв'язність)  Control coupling (зв'язність управління)  Stamp coupling (Data-structured coupling)  Топологічна  Data coupling (зв'язність даних)  Message coupling (low)  No coupling (відсутність зв'язності)  Subclass Coupling (зв'язність підкласів)  Temporal coupling (зв'язність за часом) |
| Обчислюється  Для керуючого графа програми будується ієрархія вкладених підграфів з  **Характерист**  складність з урахуванням  одним входом і одним виходом у вигляді діаграми, в якій точки на (i + 1) м рівні  **ичний поліном**  циклічної структури.  позначають окремі вершини і вкладені структури підграфа, розміщеного на i-му  **PCN**  рівні.  Топологічна  При обчисленні характеристичного полінома кожної вершині  зіставляється число CN, а циклічною структурі - змінна c. Задається алгоритм  обчислення характеристичних чисел для структур і діаграм в залежності від  рівня.  Характеристичний поліном PCN - це характеристичне число на самому  верхньому рівні. |
| Потік даних вважається  Програма розбивається на лінійні непересічні ділянки - промені -  **Метрика**  більш складним, якщо він  операторів, які утворюють керуючий граф програми.  **Овієдо**  більш складний у складових  Автор метрики виходить з наступних припущень:  частинах програми, де при  - Програміст може знайти відношення між визначальними і  певному розбитті програми він  використовують входженнями змінної всередині променя легше, ніж між  вважається досить просто.  променями.  - Число різних визначальних входжень в кожному краще більш важливо,  ніж загальне число використовують входжень змінних в кожному промені  Метрика, заснована  Позначимо через R (i) безліч визначають входжень змінних, які  на обліку потоку даних  розташовані в радіусі дії променя i (визначальне входження змінної знаходиться  в радіусі дії променя, якщо змінна або локальна в ньому і має визначальне  входження, або для неї є визначальне входження в деякому попередньому  промені і немає локального визначення шляхом).  Позначимо через V (i) безліч змінних, використовують входження котрі  уже є в промені i. Тоді міра складності i-го променя задається як:  DF (i) = {сума від i до || V (i) ||} (DEF (vj)) |

| де DEF (vj) - число визначають входження змінної vj з безлічі R (i), а || V  (i) || - Потужність безлічі V (i). |
| --- |
| **Метрика Тая** Поліпшення метрики  Складність програми визначається виміром інформаційного потоку  Метрика, заснована  Маккейба.  даних. Метрика досліджує вплив визначень даних і їх використання при  на обліку потоку даних  наявності операторів if і while. |
| Комплексна методика,  H\_M = (M + R1 \* M (M1) + ... + Rn \* M (Mn) / (1 + R1 + ... + Rn)  **Метрика**  заснована на більш простих  Де M - базова метрика, Mi - інші цікаві заходи, Ri - коректно підібрані  **Кокола**  методиках.  коефіцієнти, M (Mi) - функції.  Гібридна  Функції M (Mi) і коефіцієнти Ri обчислюються за допомогою  регресійного аналізу.  В результаті досліджень, автор виділив три моделі для заходів: Маккейб,  Холстеда і LOC, де в якості базової виступає міра Холстеда. Ці моделі отримали  назву "найкраща", "випадкова" і "лінійна". |
| Метрика заснована на  VOD - Violations of the Law of Demeter (порушення закону Деметера)  **Метрика**  припущенні, що чим менше  Закон Деметера намагається мінімізувати взаємодія між класами. Якщо  **Деметера**  взаємодія між класами, тим  клас наслідує цей закон, то його методи можуть викликати методи з обмеженої  Метрика для ООП  простіше, а отже краще,  множини інших класів. Таке безліч складається з атрибутів об'єкта, класів  програм  вихідний код.  параметрів методу або класів об'єктів, створених локально в процесі виконання  методу. |
| Обчислення числа  W = (N / T) ^ (T / 2)  **Метрики**  вершин у завданні засноване  T - загальна кількість завдань в програмі  **оцінки графа**  на підрахунку числа вершин її  N - загальне число вершин у всіх завданнях в програмі.  **паралельності**  керуючого графа. Після цього  N / T - середнє число вершин у завданні Метрика складності  легко знаходиться число  для розподілених систем  вершин графа паралельності. |