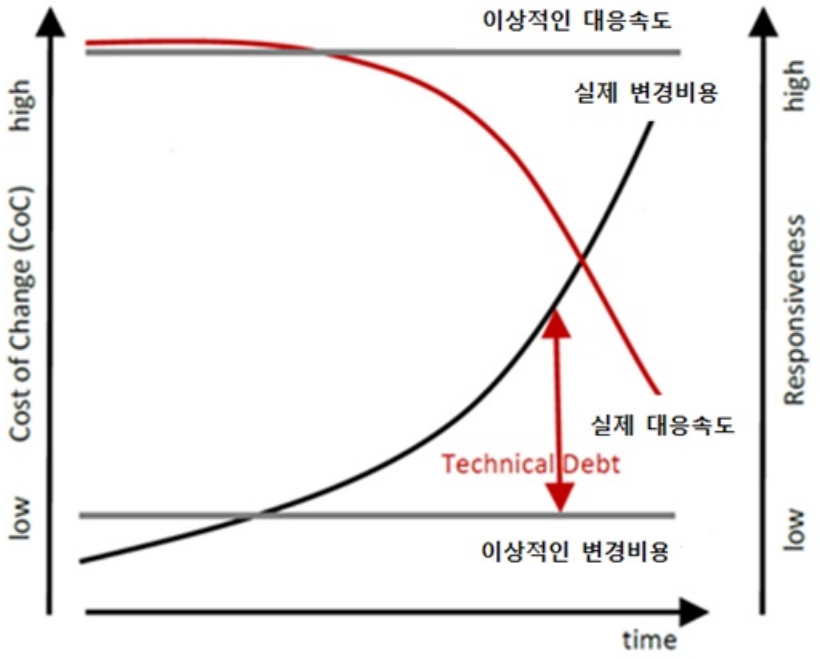
클린코드 & 리펙토링 & 시큐어코딩

클린코드

클린코드는 의도를 명확하게 드러내며, 가독성이 뛰어나며, 유지보수하기 쉬우며, 재사용성이 높은 코드를 작성하는 것을 지향하는 개발 방법론입니다. 클린코드의 가장 중요한 요소 중 하나는 가독성이라고 볼 수 있습니다. 즉, 모든 팀원이 이해(understandability)하기 쉽도록 작성된 코드 입니다.



만약 클린 코드를 실천하지 않고 복사&붙여넣기와 같은 방법을 택한다면 **Technical dept**가 생길것입니다.
현 시점에서 더 나은 접근방식보다 더 쉬운 솔루션을 채택함으로써 발생되는 추가적인 재작업의 비용입니다.

⭐ Technical dept이란 기술 부채를 의미합니다.

🌈클린 코드의 주요원칙

1️⃣ *Follow Standard Conventions*

코딩 표준, 아키텍쳐 표준 및 설계 가이드를 준수해야 한다.

2️⃣ *Keep it simple, Stupid*

단순한 것이 효율적이며, 복잡함을 최소화해야 한다.

3️⃣ *Boy Scout Rule*

참조되거나 수정되는 코드는 원래보다 클린해야 한다.

자신이 담당한 코드는 담당하기 이전의 코드보다 더 클린하게 만들어야한다.

4️⃣ *Root Cause Analysis*

항상 근본적인 원인을 찾아야한다. 그렇지 않으면 반복될 것이다.

5️⃣ ***Do Not Multiple Languages in One Source File***

하나의 파일은 하나의 언어로 작성한다.

리팩토링

리팩토링은 기능을 변경하지 않으면서도 코드의 구조를 개선하여 버그의 발생을 줄이고, 유지보수 및 확장성을 개선하는 것입니다. 또한, 리팩토링은 소프트웨어 개발 과정에서 지속적으로 이루어져야 하며, 코드를 작성하는 동안에도 항상 리팩토링 가능성을 고려해야 합니다.

```
function sumNumbersAboveThreshold(numbers, threshold) {
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] >= threshold) {
      sum += numbers[i];
    }
  }
  return sum;
}
```

이 함수는 입력된 배열에서 특정 값 이상인 숫자들의 합을 반환합니다.

하지만, 이 함수는 몇 가지 문제가 있습니다.

1. 함수 이름이 너무 길고, 함수의 목적을 명확하게 드러내지 않습니다.
2. `for` 루프를 사용하는 대신 배열 메서드 `reduce()`를 사용하면 코드를 더 간결하게 만들 수 있습니다.
3. 함수의 매개변수로 배열을 받는 대신, 가변 인수를 사용하여 여러 개의 숫자를 입력받을 수 있도록 만들면 사용하기 더 편리할 것입니다.

위의 문제점을 개선한 리팩토링된 함수는 다음과 같습니다.

```
function sumAbove(threshold, ...numbers) {
  return numbers.reduce((sum, number) => number >= threshold ? sum + number : sum, 0);
}
```

이렇게 개선된 코드는 함수 이름이 짧아져 더 명확해졌고, 배열 메서드 `reduce()`를 사용하여 코드가 더 간결해졌습니다. 또한, 가변 인수를 사용하므로 함수를 더 쉽게 사용할 수 있게 되었습니다.

시큐어코딩

시큐어 코딩(Secure Coding)은 소프트웨어 개발 과정에서 보안 위협에 대한 이해와 대비를 고려하여 안전한 소프트웨어를 개발하는 방법론입니다.

소프트웨어 보안 위협으로는 버퍼 오버플로우, 인젝션, 인증 및 권한 부여 오류, XSS(Cross-Site Scripting) 등이 있습니다. 이러한 보안 위협으로부터 안전한 소프트웨어를 개발하기 위해서는 보안 취약점을 식별하고, 이를 해결하기 위한 코딩 기술과 안전한 코딩 습관을 반드시 준수해야 합니다.

시큐어 코딩은 프로그래밍 언어나 프레임워크에 따라 다양한 기술과 방법이 존재합니다. 예를 들어, 자바에서는 정적 코드 분석 도구를 사용하거나, SQL 인젝션 공격을 막기 위해 PreparedStatement나 Stored Procedure를 사용하는 것 등이 시큐어 코딩의 방법 중 일부입니다.

예를 들어, 입력 값 검증이 없는 경우, 악의적인 사용자가 악성 스크립트나 SQL 삽입을 통해 시스템에 해킹을 할 수 있습니다. 따라서 입력 값 검증 기능을 추가하여 이를 방지할 수 있습니다. 아래는 JavaScript에서 입력 값 검증 기능을 추가하는 간단한 예시 코드입니다.

```
function getUserInput() {
  var input = document.getElementById("inputField").value;
  return input;
}

function validateInput(input) {
  if (input.includes("<script>") || input.includes("<img>") || input.includes("SELECT"))
{
    alert("Invalid input detected!");
    return false;
  } else {
    return true;
  }
}

function processInput() {
  var input = getUserInput();
  if (validateInput(input)) {
    // process input
  }
}
```

위 코드에서 `getUserInput()` 함수는 HTML 입력 필드에서 값을 가져오는 함수입니다. `validateInput()` 함수는 입력된 값을 검증하여 악성 스크립트나 SQL 삽입을 검사합니다. 이를 통해 보안 취약점을 방지할 수 있습니다. `processInput()` 함수에서는 `validateInput()` 함수를 호출하여 검증한 후, 안전한 입력 값만 처리합니다. 이와 같이 시큐어코딩은 취약점을 찾아내고 이를 방지하는 방법을 제시하여 보안을 강화합니다.