Fast track

# Topic 1: Coding rules, MISRA C

**Cosing rules**

(Coding Standards/Guidelines) là tập hợp các quy tắc hoặc hướng dẫn để lập trình viên tuân theo khi viết mã nguồn. Những quy tắc này nhằm đảm bảo rằng mã nguồn:

* **Dễ đọc**: Tăng tính rõ ràng và nhất quán.
* **Dễ bảo trì**: Dễ dàng cập nhật, sửa lỗi hoặc thêm tính năng mới.
* **An toàn**: Giảm thiểu lỗi tiềm ẩn và tăng tính tin cậy.
* **Chuyển giao dễ dàng**: Các thành viên trong nhóm có thể hiểu được mã do người khác viết.

**MISRA C**

MISRA C (**Motor Industry Software Reliability Association for C**) là một tập hợp tiêu chuẩn lập trình C được thiết kế đặc biệt để đảm bảo **độ an toàn**, **độ tin cậy** và **khả năng bảo trì** của phần mềm, đặc biệt trong ngành **ô tô**. MISRA C được phát triển lần đầu vào năm 1998 và hiện nay đã trở thành tiêu chuẩn được sử dụng rộng rãi trong nhiều ngành công nghiệp khác, như y tế, hàng không, và năng lượng.

MISRA C đưa ra các quy tắc:

* **Bắt buộc** (Mandatory): Phải tuân thủ 100%.
* **Khuyến nghị** (Advisory): Khuyến khích nhưng không bắt buộc.

**Tại sao cần tuân thủ?**

* **Cosing rules**: Giúp nhóm làm việc hiệu quả hơn, giảm thời gian sửa lỗi.
* **MISRA C**: Đảm bảo phần mềm đạt tiêu chuẩn an toàn, tránh các lỗi có thể dẫn đến tai nạn hoặc thất bại hệ thống nghiêm trọng.

# Test Automation Framework là gì?

Test Automation Framework là một tập hợp các **công cụ**, **thư viện**, và **hướng dẫn** được xây dựng để hỗ trợ quá trình tự động hóa kiểm thử phần mềm. Framework cung cấp một cấu trúc chuẩn để viết, thực thi, và quản lý các test case, giúp tăng tính hiệu quả, giảm sự lặp lại và đảm bảo tính nhất quán trong quá trình kiểm thử.

## Lợi ích khi sử dụng Test Automation Framework

1. **Tiết kiệm thời gian và chi phí**: Tự động hóa kiểm thử giúp giảm đáng kể thời gian thực thi và chi phí so với kiểm thử thủ công.
2. **Tái sử dụng mã nguồn**: Các thành phần trong framework có thể tái sử dụng cho nhiều dự án hoặc kịch bản kiểm thử khác nhau.
3. **Tính nhất quán**: Quy trình kiểm thử chuẩn hóa giúp đảm bảo các kết quả kiểm thử đồng nhất.
4. **Dễ bảo trì**: Khi có sự thay đổi trong ứng dụng, chỉ cần cập nhật framework hoặc test script mà không phải chỉnh sửa thủ công toàn bộ.
5. **Báo cáo chi tiết**: Framework cung cấp các báo cáo rõ ràng về kết quả kiểm thử.
6. **Dễ dàng tích hợp**: Có thể tích hợp với các công cụ CI/CD (như Jenkins, GitHub Actions) để kiểm thử liên tục.

## Automation testing hoạt động như thế nào?

Quy trình tự động hóa kiểm thử cơ bản bao gồm các bước:

1. **Chuẩn bị môi trường kiểm thử**:
   * Chọn framework phù hợp (e.g., Selenium, Appium, TestNG).
   * Cài đặt các công cụ và cấu hình môi trường.
2. **Viết test script**:
   * Sử dụng ngôn ngữ lập trình (Java, Python, C#) để tạo kịch bản kiểm thử dựa trên yêu cầu phần mềm.
3. **Chạy test script**:
   * Test script được thực thi trên các môi trường hoặc trình duyệt cụ thể.
   * Kết nối với dữ liệu kiểm thử (nếu dùng Data-driven).
4. **Kiểm tra kết quả**:
   * Framework tự động so sánh kết quả thực tế với kết quả mong đợi.
   * Tạo báo cáo kiểm thử (Test Report).
5. **Bảo trì test script**:
   * Cập nhật hoặc sửa đổi các test script khi có thay đổi trong ứng dụng.

# Makefile là gì?

Makefile là một tệp cấu hình chứa **các quy tắc (rules)** để tự động hóa quá trình biên dịch và xây dựng chương trình, thường được sử dụng trong các dự án viết bằng ngôn ngữ C/C++ hoặc các ngôn ngữ khác. Tệp này thường được sử dụng với công cụ make để thực hiện các lệnh xây dựng tự động dựa trên sự phụ thuộc giữa các tệp nguồn và tệp đối tượng.

Ví dụ, Makefile sẽ giúp:

* **Tự động hóa biên dịch**: Chỉ cần gõ lệnh make, thay vì nhập từng lệnh biên dịch.
* **Xây dựng dự án theo điều kiện**: Chỉ biên dịch lại các tệp đã thay đổi, tiết kiệm thời gian.
* **Quản lý phức tạp**: Điều phối nhiều tệp nguồn, thư viện, và tệp đầu ra.

## Makefile hoạt động như thế nào?

Makefile hoạt động dựa trên các **quy tắc (rules)**, bao gồm:

1. **Mục tiêu (Target)**: Là tệp bạn muốn tạo ra, ví dụ: file .o, file thực thi .exe.
2. **Phụ thuộc (Dependencies)**: Là những tệp cần thiết để tạo ra Target.
3. **Hành động (Commands)**: Là các lệnh cần thực hiện để xây dựng Target từ Dependencies.

## Ví dụ và các thành phần của Makefile

**1.3.1 Rule (Quy tắc)**

* Một rule gồm target, dependencies, và commands.
* Các lệnh trong commands phải được thụt vào bằng một **tab**.

**1.3.2 Variable (Biến)**

* Biến giúp tái sử dụng giá trị trong Makefile.
* Định nghĩa biến:

makefile

Sao chép mã

CC = gcc

CFLAGS = -Wall

* Sử dụng biến:

makefile

Sao chép mã

$(CC) $(CFLAGS) -o program main.c

**1.3.3 Condition (Điều kiện)**

* Dùng để kiểm tra điều kiện và thực hiện hành động tương ứng.
* Cú pháp:

makefile

Sao chép mã

ifeq ($(OS), Windows\_NT)

RM = del

else

RM = rm -f

endif

**1.3.4 Function (Hàm)**

Makefile hỗ trợ một số hàm tích hợp:

* **$(wildcard pattern)**: Lấy danh sách các tệp phù hợp với mẫu.

makefile

Sao chép mã

SRC = $(wildcard \*.c)

* **$(patsubst find,replace,text)**: Thay thế chuỗi.

makefile

Sao chép mã

OBJS = $(patsubst %.c, %.o, $(SRC))

* **$(shell command)**: Chạy lệnh shell và lấy kết quả.

makefile

Sao chép mã

DATE = $(shell date)

**1.3.5 Clean Rule (Quy tắc dọn dẹp)**

* Dùng để xóa các tệp tạm thời.
* Ví dụ:

makefile

Sao chép mã

clean:

rm -f \*.o program

# Overview Debug technology

**Phương pháp Debugging (Gỡ lỗi)**

Debugging là quá trình xác định, phân tích và khắc phục các lỗi hoặc vấn đề trong phần mềm hoặc hệ thống. Quá trình này giúp đảm bảo phần mềm hoạt động đúng theo mong đợi.

**Phân loại lỗi (Error Classification)**

Các lỗi gặp phải khi gỡ lỗi thường được phân loại như sau:

1. **Lỗi cú pháp (Syntax Errors)**
   * Xảy ra khi mã nguồn vi phạm quy tắc của ngôn ngữ lập trình.
   * Ví dụ: thiếu dấu chấm phẩy, ngoặc không khớp, khai báo biến sai.
   * Phát hiện: Do trình biên dịch hoặc thông dịch báo lỗi trong quá trình biên dịch.
2. **Lỗi thời gian chạy (Runtime Errors)**
   * Xảy ra khi chương trình đang chạy.
   * Ví dụ: chia cho số 0, truy cập vùng nhớ không hợp lệ, hoặc trỏ tới con trỏ NULL.
   * Phát hiện: Chương trình gặp sự cố hoặc hành vi không mong muốn khi thực thi.
3. **Lỗi logic (Logical Errors)**
   * Chương trình chạy bình thường nhưng đưa ra kết quả sai do lỗi logic trong mã.
   * Ví dụ: điều kiện lặp sai, thuật toán sai, hoặc sử dụng sai toán tử logic.
   * Phát hiện: Yêu cầu kiểm thử kỹ lưỡng và phân tích kết quả đầu ra.
4. **Lỗi ngữ nghĩa (Semantic Errors)**
   * Xảy ra khi mã nguồn thực thi thành công nhưng không đáp ứng đúng chức năng theo yêu cầu.
   * Ví dụ: triển khai sai tính năng do hiểu sai yêu cầu.
   * Phát hiện: Thông qua việc kiểm tra chức năng và xác nhận yêu cầu.

# SW Requirement Analyze

**1. Requirement Concepts (Khái niệm về yêu cầu)**

**Yêu cầu** là những điều kiện hoặc năng lực cần thiết mà một hệ thống phần mềm phải đáp ứng để giải quyết một vấn đề cụ thể hoặc đạt được một mục tiêu.

**Phân loại yêu cầu**

1. **Yêu cầu chức năng (Functional Requirements)**
   * Mô tả các hành vi, tính năng, hoặc chức năng mà hệ thống phải cung cấp.
   * Ví dụ:
     + "Người dùng có thể đăng nhập vào hệ thống bằng tài khoản và mật khẩu."
     + "Hệ thống gửi thông báo qua email khi có sự kiện mới."
2. **Yêu cầu phi chức năng (Non-functional Requirements)**
   * Xác định các tiêu chí về hiệu suất, bảo mật, khả năng mở rộng, hoặc tính khả dụng của hệ thống.
   * Ví dụ:
     + "Hệ thống phải xử lý 1000 yêu cầu/giây."
     + "Dữ liệu người dùng phải được mã hóa theo chuẩn AES-256."
3. **Yêu cầu miền (Domain Requirements)**
   * Các yêu cầu xuất phát từ lĩnh vực hoặc ngành cụ thể mà hệ thống được phát triển.
   * Ví dụ: Trong ngành tài chính, hệ thống phải tuân thủ quy định về bảo mật dữ liệu PCI-DSS.

**2. Software Requirement Process (Quy trình yêu cầu phần mềm)**

Quy trình phát triển yêu cầu phần mềm bao gồm các bước sau:

**2.1 Elicitation (Khám phá yêu cầu)**

* Thu thập thông tin từ các bên liên quan để hiểu nhu cầu của họ.
* Kỹ thuật sử dụng:
  + Phỏng vấn
  + Hội thảo
  + Quan sát quy trình làm việc hiện tại

**2.2 Analysis (Phân tích yêu cầu)**

* Làm rõ và ưu tiên các yêu cầu thu thập được.
* Xác định xung đột giữa các yêu cầu và giải quyết chúng.
* Kết quả: Một tập hợp yêu cầu rõ ràng, nhất quán và khả thi.

**2.3 Specification (Đặc tả yêu cầu)**

* Ghi lại các yêu cầu đã được phân tích trong tài liệu SRS.
* Tài liệu này nên dễ hiểu và đầy đủ để cả nhóm phát triển lẫn khách hàng đều thống nhất.

**2.4 Validation (Xác minh yêu cầu)**

* Đảm bảo các yêu cầu được xác định đúng và phù hợp với mục tiêu.
* Cách thức:
  + Xem xét tài liệu SRS với các bên liên quan.
  + Kiểm tra các yêu cầu có thể kiểm thử được không.

**2.5 Management (Quản lý yêu cầu)**

* Theo dõi các thay đổi trong yêu cầu và đảm bảo tất cả thay đổi đều được cập nhật trong tài liệu.

REQ







