# CODING STANDARD

## TIÊU CHUẨN KỸ THUẬT — TIÊU CHUẨN VIẾT MÃ

*Automate the Style Debate • Readable Over Clever • Consistent, Not Rigid*

| | |
|---|---|
| **Tổ chức / Organization** | [Tên tổ chức] |
| **Chủ sở hữu / Document Owner** | Head of Engineering hoặc Staff/Principal Engineer được ủy quyền |
| **Phê duyệt / Approved by** | Engineering Governance Council (hoặc ủy quyền cho Kiến trúc sư trưởng/Head of Engineering) |
| **Ngày hiệu lực / Effective Date** | [DD/MM/YYYY] |
| **Phiên bản / Version** | 1.0 |
| **Chu kỳ rà soát / Review Cycle** | Hằng năm, hoặc ngay khi công cụ phân tích tĩnh/formatter/linter tham chiếu có thay đổi đáng kể về ngưỡng mặc định hoặc khuyến nghị |
| **Tài liệu cấp trên / Parent Documents** | Engineering Constitution v1.0 (Tầng 1); Software Quality Policy v1.0, Software Development Governance Policy v1.0 (Tầng 2) |
| **Phân loại / Classification** | INTERNAL |

**SAMPLE DOCUMENT**

# A. Kiểm soát tài liệu

DOCUMENT CONTROL

| Phiên bản | Ngày | Người soạn | Mô tả thay đổi |
|---|---|---|---|
| 1.0 | [DD/MM/YYYY] | [Tên / Vai trò] | Ban hành lần đầu |

# B. Mục lục nội dung

CONTENTS


- 1. Mục đích
- 2. Phạm vi áp dụng
- 3. Liên kết với Policy liên quan
- 4. Định nghĩa
- 5. Yêu cầu kỹ thuật bắt buộc
- 6. Công cụ khuyến nghị
- 7. Đo lường tuân thủ
- 8. Cơ chế ngoại lệ
- 9. Chủ sở hữu tài liệu và chu kỳ rà soát
- 10. Tài liệu liên quan
- 11. Phê duyệt

# 1. Mục đích

**PURPOSE**

1. Cụ thể hóa Software Quality Policy (SQP) Điều 5.6 (Code review chất lượng) và Software Development Governance Policy (SDGP) Điều 5.3 (Chất lượng trong từng giai đoạn) thành các quy tắc viết mã ĐO LƯỜNG ĐƯỢC, KIỂM TRA ĐƯỢC bằng công cụ hoặc checklist — thay vì để "chất lượng code" là một khái niệm cảm tính, phụ thuộc gu cá nhân của người review.
2. Loại bỏ tranh cãi thủ công, tốn thời gian về style/định dạng trong code review bằng cách chuyển toàn bộ việc đó cho công cụ tự động — con người tập trung review vào logic, khả năng đọc hiểu và cấu trúc.
3. Thiết lập một mặt bằng chung tối thiểu về chất lượng/bảo trì mã nguồn áp dụng nhất quán cho mọi ngôn ngữ lập trình dùng trong tổ chức, trong khi vẫn tôn trọng khác biệt hợp lý giữa các hệ sinh thái ngôn ngữ (ví dụ quy ước đặt tên).

Tiêu chuẩn này không lặp lại lý lẽ của SQP/SDGP — người đọc được giả định đã đồng ý các nguyên tắc đó; tài liệu này chỉ quy định CHÍNH XÁC phải làm gì.

# 2. Phạm vi áp dụng

**SCOPE**

- Áp dụng cho toàn bộ mã nguồn ứng dụng, script build/tiện ích nội bộ và cấu hình dạng code (Infrastructure as Code khi chứa logic lập trình) do tổ chức phát triển hoặc phát triển thuê ngoài.
- Áp dụng cho mọi ngôn ngữ lập trình tổ chức đang sử dụng — quy tắc trong tài liệu này được diễn đạt ở mức nguyên tắc để áp dụng được cho nhiều ngôn ngữ; công cụ/cấu hình cụ thể theo từng ngôn ngữ do đội phụ trách nền tảng/CI công bố kèm theo (ví dụ trong repo template hoặc CI/CD Standard).
- **KHÔNG bao gồm quy tắc bảo mật khi viết mã** (kiểm tra input, xử lý secret, phòng chống lỗ hổng...) — nội dung này thuộc **Secure Coding Standard** (tài liệu Tầng 3 riêng biệt, chưa ban hành tại thời điểm soạn Coding Standard này). Khi Secure Coding Standard được ban hành, hai tài liệu áp dụng song song; nếu có xung đột về phạm vi, Secure Coding Standard là nguồn tham chiếu cho mọi yêu cầu bảo mật.
- **KHÔNG bao gồm quy tắc kiểm thử** (chiến lược test pyramid, ngưỡng coverage...) — nội dung này thuộc Software Quality Policy Điều 5.2 và Testing Standard (Tầng 3, khi ban hành).
- Tiêu chuẩn này tập trung vào: định dạng code, linter tĩnh, đặt tên, độ phức tạp, comment, xử lý lỗi, dependency hygiene và trùng lặp mã — tức khía cạnh CHẤT LƯỢNG và KHẢ NĂNG BẢO TRÌ của mã nguồn.

# 3. Liên kết với Policy liên quan

**ALIGNMENT WITH RELATED POLICIES**

| Yêu cầu trong Standard | Điều khoản Policy cụ thể hóa |
|---|---|
| 5.1 Formatting tự động | SQP Điều 5.6.1 — khả năng đọc hiểu mã nguồn |
| 5.2 Linter tĩnh trong CI | SDGP Điều 5.3.2; SQP Điều 5.3 (quality gate tự động) |
| 5.3 Quy tắc đặt tên | SQP Điều 5.6.1 — khả năng đọc hiểu mã nguồn |

| | |
|---|---|
| 5.4 Giới hạn độ phức tạp | SQP Điều 5.6.1 — khả năng bảo trì, mức độ phức tạp hợp lý |
| 5.5 Comment và code chết | SQP Điều 5.6.1 |
| 5.6 Xử lý lỗi có chủ đích | SDGP Điều 5.3.1; SQP Điều 5.6 |
| 5.7 Dependency hygiene | SDGP Điều 5.10 (chuỗi cung ứng phần mềm) — góc độ bảo trì, không lặp lại góc độ bảo mật/lỗ hổng |
| 5.8 Trùng lặp mã | SQP Điều 5.6.1 — tránh trùng lặp không cần thiết |

> **Không lặp lại Policy/Standard khác:** Bảng trên chỉ nêu điều khoản gốc để truy vết lý do; nội dung chi tiết, đo lường được của mỗi yêu cầu nằm trọn trong Điều 5 của tài liệu này. Yêu cầu bảo mật code review và test coverage đã có tại Secure Software Development Policy / Software Quality Policy — Standard này không quy định lại.

## 4. Định nghĩa

DEFINITIONS

- **Clean Code:** tập nguyên tắc viết mã dễ đọc, dễ bảo trì phổ biến trong ngành, gắn liền với cuốn "Clean Code: A Handbook of Agile Software Craftsmanship" (Robert C. Martin, 2008) — nhấn mạnh đặt tên có ý nghĩa, hàm nhỏ chỉ làm một việc, giảm thiểu side-effect và chỉ dùng comment khi thật sự cần thiết.
- **Cyclomatic Complexity (Độ phức tạp Cyclomatic):** chỉ số đo số lượng đường đi độc lập tuyến tính qua mã nguồn của một hàm/method, do Thomas J. McCabe đề xuất năm 1976 — càng nhiều nhánh rẽ (if, vòng lặp, case...) thì chỉ số càng cao, đồng nghĩa hàm càng khó kiểm thử đầy đủ và khó suy luận logic.
- **Cognitive Complexity (Độ phức tạp nhận thức):** chỉ số do SonarSource (tác giả G. Ann Campbell) giới thiệu qua technical report công bố năm 2017, đo mức độ khó hiểu của mã nguồn ĐỐI VỚI CON NGƯỜI — khác Cyclomatic Complexity ở chỗ cộng thêm điểm phạt khi cấu trúc rẽ nhánh lồng nhau càng sâu, thay vì cộng dồn như nhau cho mọi loại nhánh rẽ. Nhiều công cụ phân tích tĩnh hiện dùng chỉ số này làm căn cứ chính để đánh giá khả năng bảo trì, thay vì chỉ dùng Cyclomatic Complexity.
- **Code Smell:** dấu hiệu bề mặt trong mã nguồn cho thấy khả năng tồn tại vấn đề sâu hơn về thiết kế hoặc khả năng bảo trì (ví dụ hàm quá dài, lớp quá nhiều trách nhiệm, trùng lặp) — không phải lỗi (bug) mà là tín hiệu cần xem xét tái cấu trúc (refactor).
- **Linter:** công cụ phân tích tĩnh mã nguồn phát hiện lỗi tiềm ẩn, vi phạm coding standard hoặc pattern rủi ro mà không cần chạy chương trình.
- **Formatter:** công cụ tự động viết lại định dạng mã nguồn (thụt lề, khoảng trắng, xuống dòng...) theo một bộ quy tắc cố định, loại bỏ hoàn toàn nhu cầu tranh luận thủ công về style.
- **Dependency Hygiene:** thực hành quản lý thư viện/gói phần mềm bên thứ ba theo hướng tối giản — chỉ thêm khi thực sự cần thiết và gỡ bỏ khi không còn dùng — nhằm giảm bề mặt bảo trì và rủi ro (khác với việc quét lỗ hổng bảo mật dependency đã quy định tại SDGP Điều 5.10, vốn tập trung vào rủi ro bảo mật).

# 5. Yêu cầu kỹ thuật bắt buộc

**MANDATORY TECHNICAL REQUIREMENTS**

Từ khóa PHẢI (MUST) / KHÔNG ĐƯỢC PHÉP (MUST NOT) / NÊN (SHOULD) / CÓ THỂ (MAY) dùng theo đúng quy ước tại Engineering Constitution (RFC 2119).

## 5.1 Định dạng code (Formatting) — PHẢI tự động hóa hoàn toàn

1. Mọi codebase PHẢI (MUST) dùng một công cụ formatter tự động thống nhất cho mỗi ngôn ngữ (ví dụ `Prettier` cho JavaScript/TypeScript/CSS, `Black` hoặc `Ruff format` cho Python — chỉ là ví dụ minh họa, không bắt buộc công cụ cụ thể).
2. Formatter PHẢI (MUST) được tích hợp vào pre-commit hook và/hoặc bước kiểm tra định dạng trong CI; CI PHẢI (MUST) chặn merge nếu mã nguồn chưa được định dạng đúng.
3. Người review KHÔNG ĐƯỢC PHÉP (MUST NOT) đưa ra góp ý về style thuần túy (khoảng trắng, thụt lề, vị trí dấu ngoặc, độ dài dòng...) trong code review — đây là việc máy làm được. Mọi bất đồng về style PHẢI (MUST) được giải quyết bằng cách sửa cấu hình formatter dùng chung, không phải bằng tranh luận trên từng pull request.

## 5.2 Linter tĩnh trong CI

1. Mỗi codebase PHẢI (MUST) có linter tĩnh chạy tự động trong CI trên mọi pull request.
2. CI PHẢI (MUST) chặn merge khi linter báo lỗi ở mức `error`.
3. CI không bắt buộc chặn merge ở mức `warning`, trừ khi đội phát triển tự quyết định nâng một số quy tắc cụ thể từ warning thành error — quyết định này PHẢI (MUST) được ghi nhận trong cấu hình linter dùng chung của repo, không để tùy tiện theo từng lần review.
4. Việc tắt một quy tắc linter cho một dòng/khối mã cụ thể (ví dụ dùng comment disable) PHẢI (MUST) đi kèm lý do ngắn gọn; KHÔNG ĐƯỢC PHÉP (MUST NOT) tắt toàn bộ linter ở cấp file hoặc cấp dự án để né tránh sửa lỗi.

## 5.3 Quy tắc đặt tên

1. Tổ chức KHÔNG áp đặt một naming convention cứng duy nhất cho mọi ngôn ngữ, vì mỗi ngôn ngữ/hệ sinh thái có quy ước khác nhau (ví dụ `camelCase` phổ biến trong JavaScript/Java, `snake_case` trong Python/Ruby, `PascalCase` cho tên class/type trong nhiều ngôn ngữ).
2. Trong cùng một codebase/ngôn ngữ, quy ước đặt tên PHẢI (MUST) nhất quán — không trộn lẫn nhiều convention cho cùng một loại đối tượng (biến, hàm, hằng số, class) trong cùng một dự án.
3. Tên biến, hàm, class PHẢI (MUST) mô tả đúng ý nghĩa/mục đích sử dụng; tránh viết tắt tối nghĩa (ví dụ `tmp1`, `dataX`, `procInfo2`) trừ các viết tắt đã là quy ước phổ biến, dễ hiểu trong ngành hoặc trong miền nghiệp vụ của tổ chức (ví dụ `id`, `URL`, `HTTP`).
4. Linter tĩnh NÊN (SHOULD) được cấu hình quy tắc kiểm tra naming convention tự động khi ngôn ngữ và công cụ hỗ trợ, để giảm phụ thuộc vào việc con người tự nhớ quy ước.

## 5.4 Giới hạn độ phức tạp

1. Hàm/method PHẢI (MUST) được giữ độ phức tạp dưới ngưỡng do công cụ phân tích tĩnh của tổ chức quy định trong cấu hình dùng chung, đo bằng Cognitive Complexity (Điều 4) làm chỉ số chính — nhiều

công cụ phân tích tĩnh hiện nay (ví dụ SonarQube/SonarLint) ưu tiên chỉ số này hơn Cyclomatic Complexity vì phản ánh sát hơn mức độ khó hiểu thực tế đối với người đọc.

2. Điểm khởi đầu tham khảo (đã xác minh là ngưỡng mặc định phổ biến của SonarQube/SonarLint, tổ chức CÓ THỂ điều chỉnh trong cấu hình dùng chung tùy đặc thù hệ thống):

| # | Chỉ số | Ngưỡng mặc định phổ biến |
|---|---|---|
| **1** | Cognitive Complexity / hàm (đa số ngôn ngữ: Java, JS/TS, Python, C#...) | 15 |
| **2** | Cognitive Complexity / hàm (họ ngôn ngữ C/C++) | 25 |
| **3** | Cyclomatic Complexity / hàm (chỉ số tham khảo phụ trợ theo McCabe) | 10 |

3. Hàm/method vượt ngưỡng đã cấu hình PHẢI (MUST) được gắn cờ (flag) bởi CI; việc merge thay đổi vượt ngưỡng KHÔNG ĐƯỢC PHÉP (MUST NOT) trừ khi có lý do kỹ thuật rõ ràng được ghi nhận (ví dụ hàm sinh tự động, bảng switch-case tra cứu đơn giản không thể tách nhỏ có ý nghĩa) và được reviewer chấp thuận rõ ràng.
4. Hàm/method quá dài hoặc đảm nhận nhiều trách nhiệm khác nhau (vi phạm nguyên tắc "một hàm làm một việc" của Clean Code, Điều 4) NÊN (SHOULD) được tách nhỏ thành các hàm có tên rõ ràng, dễ kiểm thử độc lập — không chờ đến khi vượt ngưỡng công cụ mới xử lý.

## 5.5 Comment

1. Comment PHẢI (MUST) chỉ được dùng để giải thích LÝ DO (why) của một quyết định khi lý do đó không hiển nhiên từ bản thân đoạn mã (ví dụ lý do chọn một giải pháp không trực quan để né một giới hạn của hệ thống bên ngoài, hoặc một đánh đổi hiệu năng có chủ đích).
2. Comment KHÔNG ĐƯỢC PHÉP (MUST NOT) chỉ mô tả lại ĐIỀU GÌ (what) đã hiển nhiên qua tên biến/hàm rõ ràng (ví dụ `// tăng i lên 1` phía trên `i++`) — nếu cần comment để giải thích "đang làm gì", đó thường là dấu hiệu tên biến/hàm chưa đủ rõ ràng và nên đổi tên thay vì thêm comment.
3. Code đã comment-out (code chết còn để lại dưới dạng bị vô hiệu hóa bằng comment) KHÔNG ĐƯỢC PHÉP (MUST NOT) tồn tại lâu dài trong nhánh chính (main/trunk) — lịch sử mã nguồn đã có version control lưu giữ, không cần giữ lại bằng comment. Code chết phát hiện trong review PHẢI (MUST) được yêu cầu xóa trước khi merge, trừ trường hợp có ghi chú rõ ràng kèm thời hạn xử lý cụ thể.

## 5.6 Xử lý lỗi (Error Handling)

1. Lỗi PHẢI (MUST) được xử lý hoặc truyền (propagate) một cách có chủ đích — mọi khối bắt lỗi (`catch` / `except` / `rescue` ...) PHẢI (MUST) có hành động rõ ràng: xử lý, ghi log, chuyển đổi thành lỗi ở tầng cao hơn, hoặc truyền tiếp có kiểm soát.
2. KHÔNG ĐƯỢC PHÉP (MUST NOT) để khối bắt lỗi trống (empty catch block) nuốt lỗi âm thầm mà không ghi chú lý do — nếu một lỗi thực sự cần được bỏ qua có chủ đích, PHẢI (MUST) có comment giải thích rõ lý do bỏ qua (đây là một trường hợp comment giải thích "why" hợp lệ theo Điều 5.5).
3. Thông tin lỗi bị nuốt hoặc xử lý sai cách KHÔNG ĐƯỢC PHÉP (MUST NOT) làm mất dấu vết phục vụ gỡ lỗi/audit sau này — tối thiểu phải được ghi log ở mức phù hợp trước khi bị bỏ qua. Việc ghi log lỗi PHẢI

(MUST) tuân thủ Secure Coding Standard khi tài liệu đó được ban hành — không được log nguyên trạng dữ liệu nhạy cảm (mật khẩu, token, dữ liệu cá nhân) kèm theo exception.

## 5.7 Dependency Hygiene

1. Đội phát triển KHÔNG ĐƯỢC PHÉP (MUST NOT) thêm một dependency/thư viện bên thứ ba mới chỉ để dùng một hàm tiện ích nhỏ mà đội có thể tự viết ngắn gọn, dễ bảo trì trong nội bộ — mỗi dependency mới làm tăng bề mặt bảo trì, rủi ro chuỗi cung ứng (SDGP Điều 5.10) và thời gian build.
2. Việc thêm dependency mới có quy mô đáng kể (không phải tiện ích nhỏ) NÊN (SHOULD) được cân nhắc theo nguyên tắc tái sử dụng trước khi xây mới tại SDGP Điều 5.9.
3. Dependency không còn được sử dụng trong mã nguồn (phát hiện qua review, refactor, hoặc công cụ rà soát dependency không dùng) PHẢI (MUST) được gỡ bỏ khỏi danh sách phụ thuộc của dự án. Việc rà soát này NÊN (SHOULD) tận dụng chung công cụ Software Composition Analysis (SCA) đã dùng cho mục đích quét lỗ hổng bảo mật, tránh phải vận hành hai công cụ quét dependency riêng biệt.
4. Yêu cầu quét lỗ hổng bảo mật của dependency và xử lý theo SLA đã được quy định tại SDGP Điều 5.10 — Coding Standard này không lặp lại, chỉ bổ sung góc độ vệ sinh/bảo trì (hygiene).

## 5.8 Trùng lặp mã (Duplication)

1. Trùng lặp mã PHẢI (MUST) được rà soát định kỳ bằng công cụ phân tích tĩnh có khả năng phát hiện duplication (cùng công cụ dùng cho linter/độ phức tạp tại Điều 5.2, 5.4 nếu công cụ hỗ trợ).
2. Điểm khởi đầu tham khảo (đã xác minh là ngưỡng mặc định của quality gate "Sonar way" trên SonarQube/SonarQube Cloud): tỷ lệ dòng trùng lặp (*duplicated lines density*) trên mã nguồn mới (new code) không vượt quá `3%`. Tổ chức CÓ THỂ (MAY) quy định ngưỡng khác trong cấu hình công cụ dùng chung, tương xứng với đặc thù từng hệ thống — con số trên chỉ là điểm khởi đầu tham khảo, không phải yêu cầu tuyệt đối áp cho mọi trường hợp.
3. Khi trùng lặp vượt ngưỡng đã cấu hình, đội phát triển PHẢI (MUST) đưa hạng mục tái cấu trúc (refactor) vào backlog theo cơ chế quản lý nợ kỹ thuật tại SQP Điều 5.4 — không bắt buộc xử lý ngay lập tức nếu rủi ro thấp, nhưng phải được ghi nhận, không bị bỏ qua âm thầm.

## 6. Công cụ khuyến nghị

RECOMMENDED TOOLING

Bảng minh họa dưới đây KHÔNG bắt buộc toàn tổ chức dùng một ngôn ngữ hay một bộ công cụ cụ thể — mỗi đội chọn công cụ tương đương phù hợp với ngôn ngữ/hệ sinh thái đang dùng, miễn đáp ứng yêu cầu tại Điều 5.

| Hạng mục | Ví dụ minh họa (JS/TS) | Ví dụ minh họa (Python) |
|---|---|---|
| Formatter | `Prettier` | `Black` hoặc `ruff format` |
| Linter tĩnh | `ESLint` | `Ruff` hoặc `Pylint` |
| Độ phức tạp & duplication | `SonarQube` / `SonarLint` | `SonarQube` / `SonarLint` |

> **Lưu ý cập nhật thực tiễn ngành:** Từ bản 8.53 (2023), ESLint đã loại bỏ dần các quy tắc thuần định dạng khỏi lõi công cụ và khuyến nghị dùng riêng một formatter (ví dụ Prettier) hoặc plugin `ESLint Stylistic` nếu vẫn muốn dùng ESLint cho mục đích đó — cặp "linter lo logic/chất lượng, formatter lo định dạng" theo Điều 5.1–5.2 vẫn đúng tinh thần, chỉ đổi cách phân công cụ thể trong nội bộ ESLint. Trong hệ sinh thái Python, `Ruff` ngày càng được nhiều dự án dùng để đảm nhiệm đồng thời cả vai trò linter và formatter thay vì phải phối hợp nhiều công cụ riêng lẻ (Flake8/Pylint + Black). Đây là ví dụ minh họa xu hướng công cụ, không phải yêu cầu bắt buộc của Standard này.

## 7. Đo lường tuân thủ

COMPLIANCE MEASUREMENT

- Tỷ lệ pull request bị chặn bởi formatter/linter chưa qua, và thời gian trung bình để khắc phục.
- Số lượng comment code review liên quan đến style thuần túy còn phát sinh (mục tiêu tiệm cận 0 vì đã tự động hóa theo Điều 5.1).
- Số hàm/method vượt ngưỡng độ phức tạp (Cognitive/Cyclomatic Complexity) đang mở, xu hướng theo thời gian.
- Tỷ lệ trùng lặp mã (duplication density) theo từng repo, xu hướng theo thời gian.
- Số dependency được gỡ bỏ vì không còn sử dụng trong mỗi chu kỳ rà soát.
- Số khối bắt lỗi (catch/except) trống bị linter/review phát hiện và tỷ lệ đã khắc phục.

## 8. Cơ chế ngoại lệ

EXCEPTIONS

Ngoại lệ đối với bất kỳ yêu cầu PHẢI (MUST) nào trong Standard này áp dụng theo đúng cơ chế đã quy định tại Engineering Constitution Điều 8 và Software Development Governance Policy Điều 8 — Standard này không thiết lập quy trình ngoại lệ riêng. Vai trò, trách nhiệm chung và hậu quả không tuân thủ áp dụng theo SDGP Điều 6 và Điều 9; Standard này không lặp lại các nội dung đó.

## 9. Chủ sở hữu tài liệu và chu kỳ rà soát

**DOCUMENT OWNERSHIP & REVIEW CYCLE**

Document Owner: Head of Engineering hoặc Staff/Principal Engineer được ủy quyền. Chu kỳ rà soát: hằng năm, hoặc ngay khi công cụ phân tích tĩnh/formatter/linter tham chiếu có thay đổi đáng kể về ngưỡng mặc định hoặc khuyến nghị của ngành.

## 10. Tài liệu liên quan

**RELATED DOCUMENTS**

- Engineering Constitution v1.0 (Tầng 1)
- Software Quality Policy, Software Development Governance Policy (Tầng 2)
- Secure Coding Standard (Tầng 3 — chưa ban hành), Testing Standard, CI/CD Standard (Tầng 3)

*Style không còn là điều để tranh cãi, vì máy đã quyết thay;*
*độ phức tạp có ngưỡng, không có giới hạn của trí nhớ con người;*
*comment chỉ nói lý do, không lặp lại điều code đã tự nói;*
*lỗi không bị nuốt trong im lặng;*
*và mỗi dependency thêm vào đều có lý do đứng vững qua thời gian.*

## 11. Phê duyệt

**APPROVAL**

| Vai trò | Họ tên | Chữ ký | Ngày |
|---|---|---|---|
| Chair, Engineering Governance Council | | | |
| Chief Technology Officer | | | |
| Head of Engineering / Staff Engineer được ủy quyền (Document Owner) | | | |
| Head of Quality / QA Lead | | | |