CRÉDIT AGRICOLE
CORPORATE & INVESTMENT BANK

GitHub Copilot impact on the Software Development Lifecycle

Table of the Laws

MARCH 2025

Reminder of GitHub Copilot main use cases

	Business Demand	Design	Coding	Build	Test	Release	Deploy	Operate	Monitor
New	Business analysis User Stories Documentation	UI Design Software architecture	Code suggestions Code review & QA Debugging Unit tests generation Code documentation	Packages build	Test case generation Test synthetic data Test result analysis Root cause analysis Security vulnerabilities	Execution platform provisioning Release notes Infrastructure as Code		Monitoring and anomaly detection Predictive maintenance & optimization Log analysis User feedback analysis Assistance to incident analysis / resolution	
Legacy		Low-Code powered by GenAI Application configuration (SAP, SalesForce…) Software refactoring	Code migration Code retro documentation						
Agile	Roadmap and sprint planning, effort estimations		Retrospective		Team communication and collaboration				
LLM	Azure Open AI Services								

GitHub Copilot *Use case covered by GitHub Copilot*

→ Copilot is able to perform different use cases, especially on the coding part of the software development lifecycle
In the following slides, we focus on the applicative build processes.

Table of the Laws

SDLC transversal rules linked to the introduction of GitHub Copilot

- **Automation:** **automate** as much as possible **the CI/CD pipelines**, in priority code quality monitoring and test automation (unit test & functional tests)
- **Ways of working:** **deploy agility incrementally** in the view of implementing Agile@Scale (capitalize on existing Agile Center of Excellence)
- **Monitoring tools:**
  - If not already done, **set-up SonarQube & DevOps Sentinel** on the perimeter
  - If not done, **configure the following indicators in SonarQube/Sentinel**: LTTP, LTTP implementation, numbers of resolved JIRA tickets, deployment frequency, unit test coverage, technical debt, code smells, code duplication, maintainability rating, reliability rating, security rating → *If not already done, please liaise with the Software Factory team to implement required configuration on your applicative perimeter*
  - If not already in place, **set-up Checkmarx vulnerability detections**
- **Quality gates:**
  - Set-up **quality gates**, based on monitoring tools *(see next slide for configuration)*
    - Mandatory on critical application and/or application where the lifecycle is fully automated.
    - Not mandatory for all other applications if code reviews are effective
- **Quality engineering:**
  - **Empower the operational managers** on the definition of quality requirements (ex: definition of the test, quality thresholds…)
  - Perform a Product Risk Analysis, and define a clear test strategy
- **Linters:** Install **linters** in code editors (checking coding good practices/frameworks)
  - Identify the appropriate linter to use based on the programming language (e.g., Checkstyle / PMD / FindBugs for Java, ESLint, JSHint, Prettier for Typescript, Cppcheck for C/C++) → *Please liaise with the Software Factory team for the tooling strategy*
  - Use standard configuration to keep homogenous the configuration between the perimeters
- **Playbooks:** **Formalize the project knowledge base,** including example of application
- **GenAI strategy:**
  - Centrally define the **GenAI strategy**, to address the whole SDLC (ex: requirements gathering, user stories, test case generation, test script automation, code migration (transcoding), and version upgrades)
  - Declinate this strategy at BLI level

Recommendations directly linked to the introduction of GitHub Copilot

Table of the Laws

Mandatory processes linked to the introduction of GitHub Copilot, per phase of the SDLC

Documentation: Perform functional and technical retro-documentation (eased with GitHub Copilot, especially for legacy applications)

Software refactoring:

- **Prioritize the technical debt,** starting by the most critical applications and resources skills shortage
- Keep **part of the projects' effort** to treat the technical debt

Code review & QA:

- Implement **peer reviews** before a commit, when a specific business or technical expertise is required
- Set-up **mandatory code review** by senior profiles before a merge for all perimeters, **in a continuous mode** throughout the process (when a code push is performed, not a periodic review)

Unit tests generation

- **Create and perform unit tests**, whatever the level of maturity of the perimeter, to facilitate long-term maintenance:
  - Differentiate coverage levels for backend and frontend (unit tests for backend, E2E tests for frontend)
  - Code coverage should reach ideally 60%-80% for back-end code and 40%-60% for front-end code
  - Improve coverage step by step to gradually reduce technical debt
  - Reinforce unit test automation as part of the CI/CD
- **Set-up quality gates on unit tests**
  - On back-end code: blocking if code coverage ≤ 60%, review required by the tech lead if code coverage is between 60% and 80%, passing if code coverage ≥ 80%
  - On front-end code: blocking if the code coverage ≤ 40%, review required by the tech lead if code coverage is between 40% and 60%, passing if code coverage ≥ 60%
- **Create a regression unit test** to each bug identified in a run mode

Code documentation: Write clear comments and document the code (it is not always self-standing), especially classes/functions/methods with functional or business complexity:

- Classes/function headers should be mandatory and include input & output parameters
- Besides, GitHub Copilot will rely on these code comments to be more efficient.

Release notes: Automate and generate release notes

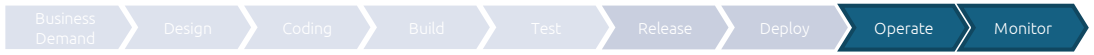
Test synthetic data: Generate test synthetic data, using GitHub Copilot

Root cause analysis: Perform **in-depth root cause analysis** for failures

Set-up quality gates on end-to-end tests: follow the **test strategy** defined in the Product Risk Analysis:

- In case of a critical anomaly, the quality gate should be blocking, and the code directly sent back to the developer
- In case of 3 major anomalies or 10 minor anomalies, the quality gate should be blocking and generate the opening of a corrective JIRA ticket (after analysis from the tech lead or quality manager)

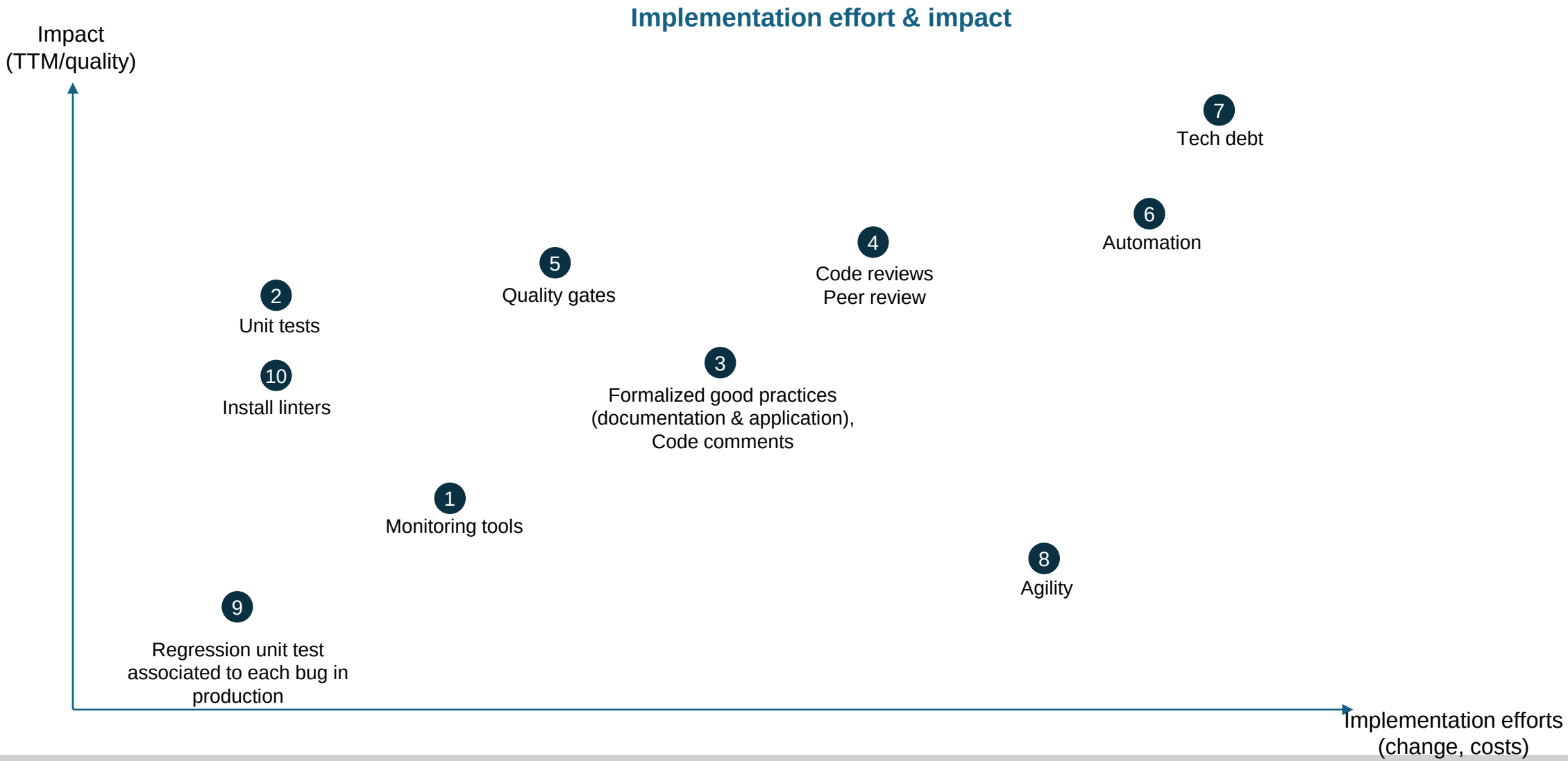
Infra as code: Accelerate on infra as code, using GitHub Use Copilot to **generate infra as code scripts**
Release notes: *cf. Build phase*



- *N/A*

Recommendations directly linked to the introduction of GitHub Copilot

Recommended prioritization based on the implementation effort & associated impact



CRÉDIT AGRICOLE
CORPORATE & INVESTMENT BANK