Développement logiciel moderne avec Git, GitHub et l'intégration / déploiement continus (CI/CD)

En bref

Langue de cours: Anglais

Présentation

Prérequis

- Maîtrise des bases de la programmation en Python (variables, fonctions, classes, modules).
- Familiarité avec l'utilisation d'un terminal en ligne de commande.
- Compréhension générale des concepts fondamentaux du développement logiciel.
- Disposer d'un compte GitHub ainsi que d'un ordinateur portable connecté à Internet.
- Aucune expérience préalable de Git, Docker, du CI/CD ou du développement logiciel collaboratif n'est requise.

Objectifs d'apprentissage

Ce cours introduit les étudiants à l'ensemble du cycle de vie du développement logiciel tel qu'il est pratiqué dans les environnements modernes d'ingénierie logicielle. Au-delà de la programmation elle-même, les étudiants apprendront à collaborer au sein d'équipes de développement, à garantir la qualité logicielle grâce aux tests, à automatiser les processus de validation et de déploiement, ainsi qu'à produire des applications fiables en s'appuyant sur les pratiques contemporaines du DevOps.

À travers une approche pédagogique fondée sur la réalisation d'un projet, les participants expérimenteront l'ensemble du flux de travail d'un projet logiciel collaboratif : gestion des tâches et des tickets (issues), contrôle de version, revue de code, tests automatisés, documentation technique, conteneurisation et intégration/déploiement continus (CI/CD). Une attention particulière sera portée au travail en équipe, à la reproductibilité, à la maintenabilité, à la qualité du code et à l'adoption des standards professionnels du développement logiciel.

À l'issue de cet enseignement, les étudiants seront capables de :

- Utiliser efficacement Git pour la gestion de versions et le suivi de l'historique du code source.
- Collaborer sur des projets logiciels en utilisant les workflows GitHub (issues, branches, pull requests et revues de code).
- Mettre en œuvre les bonnes pratiques modernes de l'ingénierie logicielle dans un contexte de développement collaboratif.

- Concevoir et développer des tests automatisés à l'aide de PyTest.
- Produire une documentation technique claire, structurée et maintenable à l'aide des docstrings et du langage Markdown.
- Concevoir et déployer des pipelines d'intégration continue (CI) à l'aide de GitHub Actions.
- Conteneuriser des applications avec Docker afin de faciliter leur portabilité et leur déploiement.
- Résoudre des conflits de fusion (merge conflicts) et intégrer les contributions de plusieurs développeurs.
- Utiliser de manière responsable et efficace des outils d'assistance au développement basés sur l'intelligence artificielle, tels que GitHub Copilot.
- Contribuer à un projet logiciel partagé en appliquant les méthodes, outils et standards professionnels du développement logiciel.

Description du programme

Le premier jour débute par une matinée consacrée aux fondamentaux de Git et aux workflows collaboratifs proposés par GitHub. À l'issue de cette première session, les étudiants sauront créer des commits, gérer des branches, ouvrir et suivre des issues, soumettre des pull requests et réaliser des revues de code, avant même d'avoir commencé le développement du projet. Les vingt dernières minutes sont consacrées à la présentation du défi fil rouge et à la constitution de deux équipes, chacune responsable du développement d'un module distinct au sein d'un même dépôt partagé.

L'après-midi débute par une introduction à GitHub Copilot et aux outils d'assistance au développement basés sur l'intelligence artificielle. Les étudiants passent ensuite à la mise en pratique. Les deux équipes développent simultanément une couche logicielle commune ainsi que leurs modules respectifs en utilisant des branches de fonctionnalité (*feature branches*) et des *pull requests*.

Le deuxième jour est consacré à la qualité logicielle et à l'automatisation des processus de développement. La matinée couvre les tests automatisés avec PyTest ainsi que la documentation technique à l'aide des docstrings et du langage Markdown. Les étudiants développent les tests associés à leur module et produisent la documentation correspondante en s'appuyant sur les mêmes workflows collaboratifs GitHub découverts la veille.

L'après-midi permet d'aborder les principes du CI/CD (*Continuous Integration / Continuous Deployment*) à travers GitHub Actions. Les étudiants mettent en place une chaîne d'intégration continue exécutant automatiquement les tests à chaque modification du dépôt, publient une documentation générée automatiquement sur GitHub Pages et construisent une image Docker distribuée via GitHub Container Registry.

La formation se conclut par une phase d'intégration collaborative au cours de laquelle les deux équipes soumettent leurs contributions au dépôt principal du projet. Les étudiants sont confrontés à des conflits de fusion (*merge conflicts*) résultant de modifications concurrentes sur les mêmes fichiers et apprennent à les résoudre collectivement afin d'obtenir une base de code cohérente, reproduisant ainsi des situations couramment rencontrées dans les projets logiciels professionnels.

Tout au long des deux journées, les étudiants travaillent sur un projet unique et partagé, chaque nouveau concept étant immédiatement appliqué lors de démonstrations et de travaux pratiques. À l'issue de la formation, chaque participant aura contribué à un dépôt GitHub collaboratif comprenant des développements fonctionnels, des tests automatisés, une documentation publiée, une image Docker et des contributions intégrées via des *pull requests*. L'ensemble constitue un premier portfolio technique illustrant l'application des pratiques professionnelles modernes du développement logiciel.

Outils utilisés : Git, GitHub, Visual Studio Code, GitHub Copilot, PyTest, Docker et GitHub Actions.

Compétences et connaissances scientifiques et techniques visées dans la discipline

- Ce cours contribue au développement des compétences suivantes du référentiel de formation d'un ingénieur Centrale :

  Compétences scientifiques et techniques

  - Maîtriser les méthodologies modernes de développement logiciel.
  - Mettre en œuvre les bonnes pratiques d'assurance qualité logicielle.
  - Comprendre les principes de l'automatisation, de l'intégration continue et du DevOps.
  - Concevoir des systèmes logiciels reproductibles, robustes et maintenables.

  Compétences numériques et liées aux données

  - Utiliser des plateformes collaboratives de développement logiciel.
  - Gérer le code source et les différentes étapes du cycle de vie d'une application.
  - Exploiter de manière critique et efficace les outils d'assistance au développement basés sur l'intelligence artificielle.

  Compétences en gestion de projet et innovation

  - Travailler efficacement au sein d'équipes techniques pluridisciplinaires.
  - Mettre en œuvre des processus de développement collaboratif.
  - Réaliser des revues de code et intégrer les retours de ses pairs.
  - Contribuer à la réalisation de livrables techniques partagés.

  Compétences professionnelles et transversales

  - Communiquer efficacement des informations techniques à travers une documentation claire et structurée.
  - Développer des capacités de résolution collaborative de problèmes.
  - Acquérir de l'autonomie dans des environnements professionnels de développement logiciel.
  - Comprendre et appliquer les standards et pratiques utilisés dans l'industrie du logiciel.

Modalité de contrôle des connaissances

L'évaluation repose sur un contrôle continu réalisé tout au long du projet pratique.

Les étudiants sont évalués selon les critères suivants :

Critère d'évaluation	Pondération
Utilisation efficace de Git et des workflows GitHub	20 %
Qualité des contributions au code et des *pull requests*	20 %
Mise en œuvre des tests automatisés	15 %

Qualité de la documentation technique	15 %
Mise en place du pipeline CI/CD	15 %
Participation à l'intégration collaborative et à la résolution des conflits de fusion (*merge conflicts*)	15 %

Les livrables attendus comprennent :

- Un historique de contributions Git (*commits*) et de *pull requests* sur un dépôt collaboratif.
- Une suite de tests automatisés développée avec PyTest.
- Une documentation technique du projet.
- Un workflow d'intégration continue mis en œuvre avec GitHub Actions.
- Une image Docker publiée sur un registre de conteneurs.
- Une version finale intégrée du dépôt logiciel résultant des contributions des différentes équipes.

L'évaluation porte autant sur la qualité technique des réalisations que sur la capacité à appliquer des pratiques professionnelles de développement collaboratif.

Bibliographie

Version Control & Git

- Chacon, S., Straub, B. (2023). *Pro Git* (2nd Edition). Apress. Available online: Pro Git Book

Software Engineering

- Martin, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- Hunt, A., Thomas, D. *The Pragmatic Programmer* (20th Anniversary Edition). Addison-Wesley.

Testing

- Okken, B. *Python Testing with Pytest*. Pragmatic Bookshelf.

Documentation

- Di Pierro, M. *Documentation Best Practices for Software Projects*.

DevOps & CI/CD

- Humble, J., Farley, D. *Continuous Delivery*. Addison-Wesley.
- Kim, G., Humble, J., Debois, P., Willis, J. *The DevOps Handbook*. IT Revolution.

Docker

- Poulton, N. *Docker Deep Dive*. Independently Published.

Official Documentation

Développement logiciel moderne avec Git, GitHub et l'intégration / déploiement continus (CI/CD)

- Git Documentation
- GitHub Documentation
- GitHub Actions Documentation
- PyTest Documentation
- Docker Documentation

Equipe pédagogique

- Julien ZOUBIAN

Total des heures		**10h**
CM	Cours Magistral	7h
TD	Travaux Dirigés	3h