Développement d'Applications Python – Projet 13
« Projet final : prêt pour le feu d'artifices ? »
Analyse vis-à-vis de l'eXtreme programming

Voici une analyse de ma pratique lors de ce projet final vis-à-vis des 13 bonnes pratiques de l'eXtreme programming. Les pratiques citées ci-dessous (titres et explications) sont tirées du site www.techno-science.net.

1. Client sur site

Explication :
Un représentant du client doit, si possible, être présent pendant toute la durée du projet. Il doit avoir les connaissances de l'utilisateur final et avoir une vision globale du résultat à obtenir. Il réalise son travail habituel tout en étant disponible pour répondre aux questions de l'équipe.

Ma pratique sur ce projet :
Durant ce projet, le client était représenté par le président de l'association concernée. Je travaillais seul, en-dehors de sa présence, mais il était joignable par téléphone et par mail. Nous nous sommes réunis deux fois : au démarrage pour fixer le cadre, puis un peu avant la fin pour valider certains points ensemble.

2. Jeu du Planning

Explication :
Le client crée des scénarios pour les fonctionnalités qu'il souhaite obtenir. L'équipe évalue le temps nécessaire pour les implémenter. Le client sélectionne ensuite les scénarios en fonction des priorités et du temps disponible.

Ma pratique sur ce projet :
J'ai créé moi-même les scénarios et priorisé en fonction des besoins du client. J'ai essayé de quantifier le temps de travail pour chaque fonctionnalité, mais je l'ai trop souvent sous-estimé.

3. Intégration continue

Explication :
Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.

Ma pratique sur ce projet :
J'ai en effet procédé de cette manière lors de ce projet, avec l'intégration continue supportée par le jeu de tests.

4. Petites livraisons

Explication :
Les livraisons doivent être les plus fréquentes possible. L'intégration continue et les tests réduisent considérablement le coût de livraison.

Ma pratique sur ce projet :
J'ai en effet procédé de cette manière lors de ce projet, avec de petites livraisons (à chaque *User Story*).

5. Rythme soutenable

Explication :
L'équipe ne fait pas d'heures supplémentaires deux semaines de suite. Si le cas se présente, il faut revoir le planning. Un développeur fatigué travaille mal.

Ma pratique sur ce projet :
J'ai essentiellement travaillé pendant la journée et en semaine sur ce projet, rarement le soir et le week-end. De toute façon, je me suis aperçu que j'étais moins efficace en fin de journée, et que je ne disposais pas d'assez de temps le week-end.

6. Tests de recette (ou tests fonctionnels)

Explication :
À partir des scénarios définis par le client, l'équipe créé des procédures de test qui permettent de vérifier l'avancement du développement. Lorsque tous les tests fonctionnels passent, l'itération est terminée. Ces tests sont souvent automatisés, mais ce n'est pas toujours possible.

Ma pratique sur ce projet :
J'ai en effet codé des tests fonctionnels avec *Selenium* pour valider chaque *User Story*. Je ne passais pas à l'étape suivante sans validation de ces tests (1 à 4 par *User Story* en fonction des scénarios).

7. Tests unitaires

Explication :
Avant d'implémenter une fonctionnalité, le développeur écrit un test qui vérifiera que son programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, on lance tous les tests écrits par tous les développeurs, et on sait immédiatement si quelque chose ne fonctionne plus.

Ma pratique sur ce projet :
C'est en effet de cette manière que j'ai procédé pour les tests unitaires, et ceci pour la première fois en *Test Driven Development* (TDD). J'ai pu y parvenir grâce à l'expérience accumulée sur les projets précédents du parcours.

8. Conception simple

Explication :
L'objectif d'une itération est d'implémenter les scénarios sélectionnés par le client, et uniquement cela. Envisager les prochaines évolutions nous ferait perdre du temps sans avoir la garantie qu'on en gagnera plus tard. Les tests nous permettront de changer l'architecture plus tard si nécessaire. Plus l'application est simple, plus il sera facile de la faire évoluer lors des prochaines itérations. De même, la documentation doit être minimale : on préfèrera un programme simple qui nécessite peu d'explications à un système complexe.

Ma pratique sur ce projet :
J'ai essayé de tendre vers cette bonne pratique, tant pour le développement des fonctionnalités que pour la documentation. J'ai notamment appliqué cela en divisant les responsabilités en un nombre assez conséquents d'*apps* Django, afin de limiter la complexité.

9. Utilisation de métaphores

Explication :
On utilise des métaphores et des analogies pour décrire le système et son fonctionnement. Le fonctionnel et la technique se comprennent beaucoup mieux lorsqu'ils sont d'accord sur les termes qu'ils emploient.

Ma pratique sur ce projet :
Bien que je pratique régulièrement cette méthode pour communiquer, je ne l'ai pas, ou très peu, utilisée dans le cadre de ce projet.

10. Refactoring (remaniement du code)

Explication :
Amélioration régulière de la qualité du code sans en modifier le comportement. On retravaille le code pour repartir sur de meilleures bases tout en gardant les mêmes fonctionnalités. Les phases de refactoring n'apportent rien au client mais permettent aux développeurs d'avancer dans de meilleures conditions, et donc plus vite.

Ma pratique sur ce projet :
J'ai procédé au remaniement de code dans ce projet, en particulier au niveau des tests, car des répétitions apparaissaient. Le code était donc davantage maintenable après le remaniement.

11. Appropriation collective du code

Explication :
L'équipe est collectivement responsable de l'application. Chaque développeur peut faire des modifications dans toutes les portions du code, même celles qu'il n'a pas écrites. Les tests diront si quelque chose ne fonctionne plus.

Ma pratique sur ce projet :
Ayant travaillé seul sur le code du projet, cette pratique n'est pas applicable ici.

12. Convention de nommage

Explication :
Puisque tous les développeurs interviennent sur tout le code, il est indispensable d'établir et de respecter des normes de nommage pour les variables, méthodes, objets, classes, fichiers, etc.

Ma pratique sur ce projet :
J'ai essayé de respecter un maximum des conventions de nommage, afin d'obtenir du code le plus propre possible, dans lequel il sera facile de se « replonger » plus tard. En particulier, en plus de le PEP8, j'ai essayé de respecter le PEP257 concernant les *docstrings*.

13. Programmation en binôme

Explication :
La programmation se fait par deux. Le premier, appelé *driver* (ou pilote), a le clavier. C'est lui qui va travailler sur la portion de code à écrire. Le second, appelé *partner* (ou co-pilote), est là pour l'aider, en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes. Les développeurs changent fréquemment de partenaires, ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

Ma pratique sur ce projet :
Ayant travaillé seul sur le code du projet, cette pratique n'est pas applicable ici.

Synthèse

Hormis le fait de travailler seul sur ce projet, j'ai globalement respecté les bonnes pratiques de l'eXtreme programming. Cette analyse me permet de comprendre les points où je pourrais améliorer ma pratique, c'est donc une très bonne chose que de la mener sur ce projet.