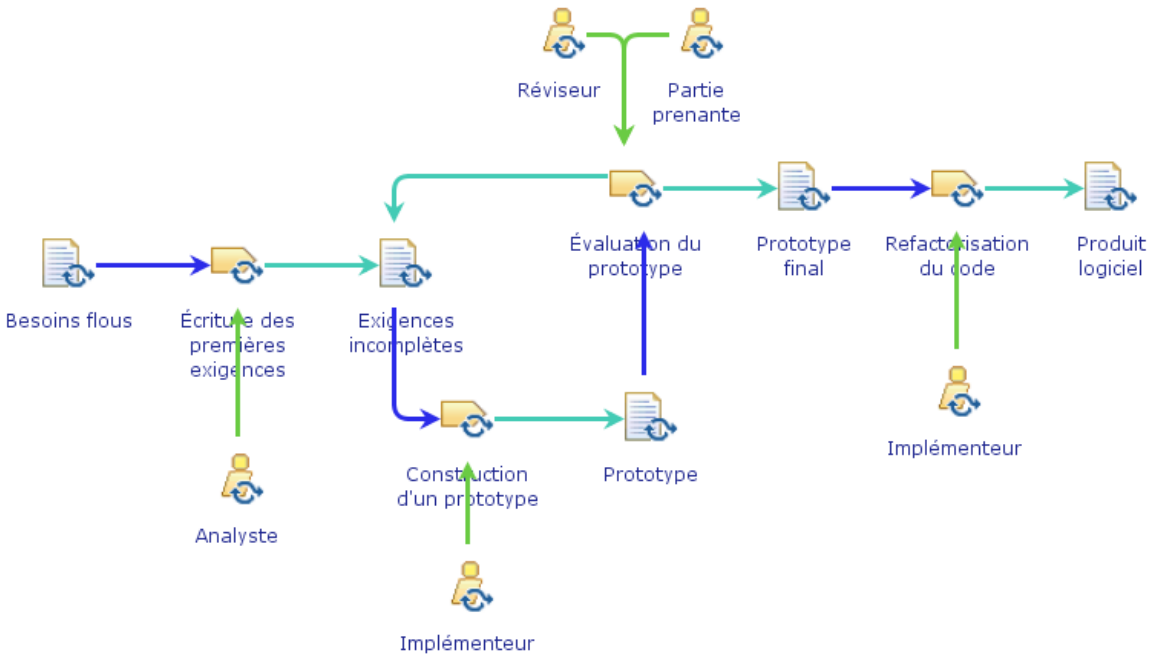
Examen final – LOG3000 – Automne 2014

- Lundi le 22 décembre 2014.
- Durée : 09h30 à 12h00 (total 2h30).
- Local : B-316.1
- Total des points : 20.
- Pondération de l'examen dans la note finale : 40%.
- Sans documentation, sans calculatrice.
- Remettre le questionnaire à la fin de l'examen.

1. *Question sur les cycles de vie (2 points)*

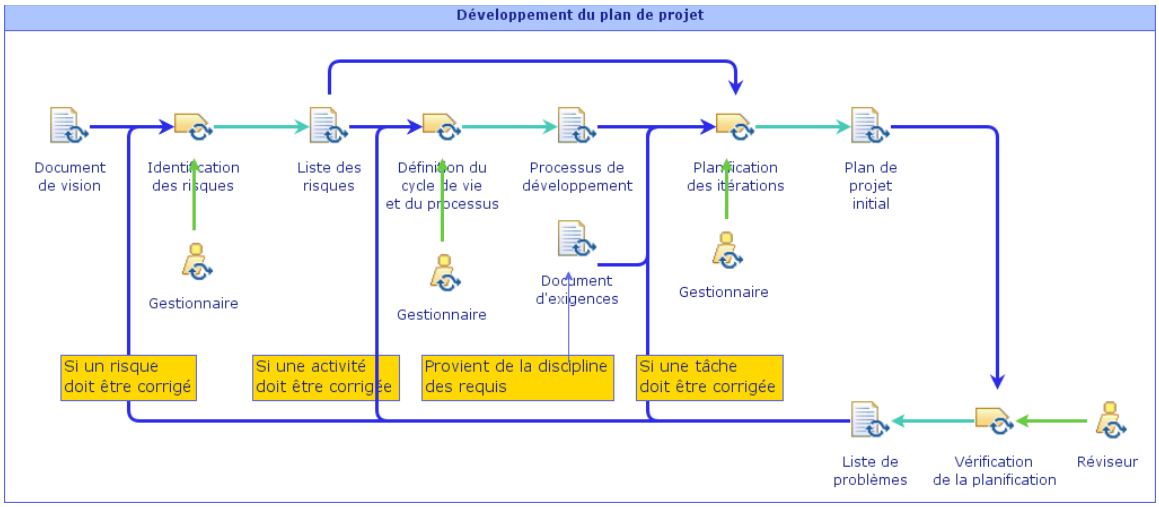
Modélisez un processus transformationnel simple en utilisant le langage SPEM 2.0 (le langage de ProcessEdit). Vous devez utiliser un maximum de cinq activités.

Solution possible :



2. *Question sur la discipline de gestion de projet (3 points)*

Vos collègues vont faire le projet intégrateur de 4e année la session prochaine. Le projet qu'elles ont choisi est plutôt ambitieux, alors elles ont déjà commencé à planifier le travail à faire. En bonnes étudiantes du cours LOG3000, elles ont monté un processus décrivant comment le logiciel sera développé. Le modèle suivant présente les activités pour la discipline de gestion de projet.



Décrivez un problème important que vous pouvez voir dans le modèle (1 point).

Justifier l'importance du problème en décrivant l'impact que celui-ci pourrait avoir sur le développement logiciel (1 point).

Proposez une modification au processus qui permettrait de corriger ce problème (1 point).

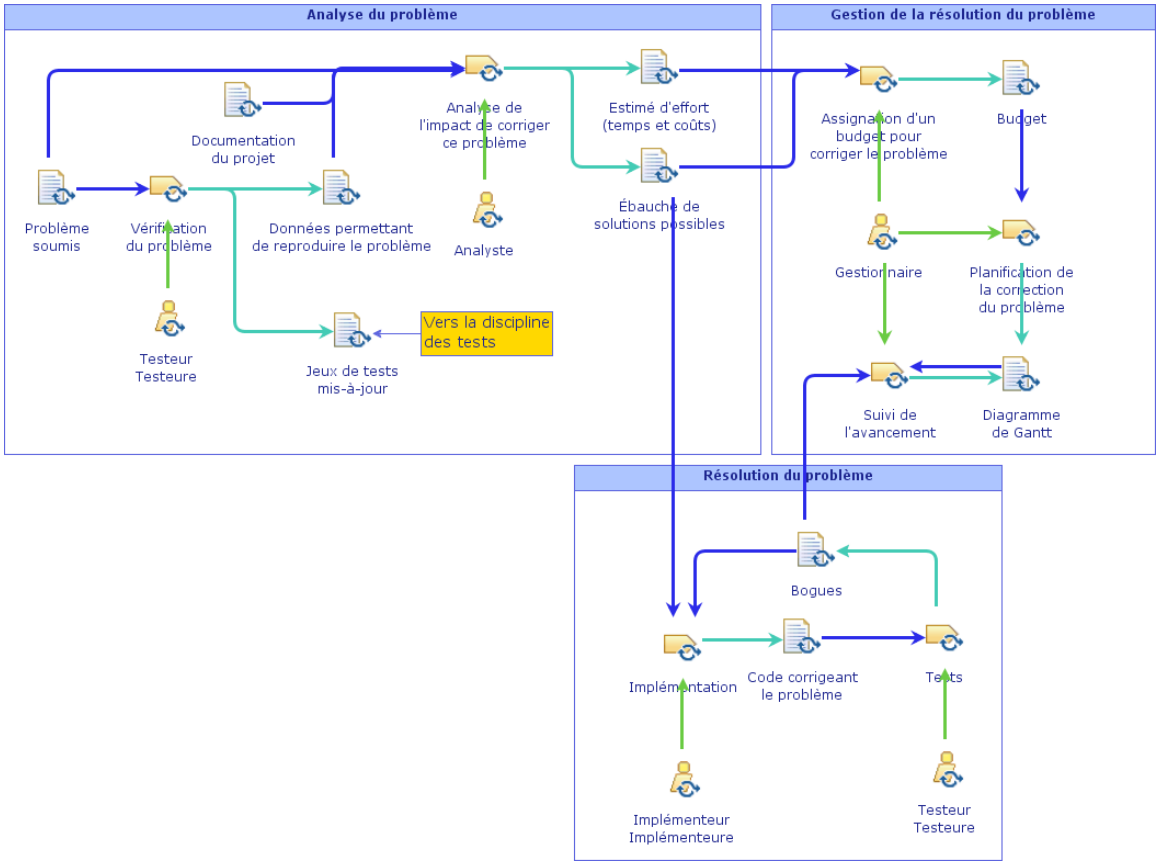
Solution :

- Il n'y a pas d'activité de suivi d'avancement en cours de projet.
- « Le plan n'est pas le territoire. » Il y a toujours des imprévus en cours de projet qui demandent de modifier le plan initial. En l'absence de ces activités, le déroulement réel du projet peut être très différent du déroulement planifié, et le projet peut ne pas être terminé aux dates limites imposées.
- Il faudrait une activité qui assigne le travail aux développeurEs, et une autre activité qui obtient un feedback des mêmes développeurEs afin de corriger le plan de travail selon les dernières données disponibles.

3. *Question sur la discipline de gestion du changement (3 points)*

Vous travaillez pour une entreprise qui offre des librairies de code spécialisées. Ces librairies sont très complexes, et chaque changement peut coûter très cher. De plus, chaque erreur dans les librairies peut avoir un impact critique au niveau de la satisfaction de la clientèle. L'entreprise a donc un processus élaboré pour la gestion des changements. Le modèle ci-dessous présente son processus de gestion du changement, où une demande de changement s'appelle « problème ».

Par contre, la qualité des correctifs de code est faible : Les corrections introduisent souvent de nouveaux problèmes, ne corrigent pas le problème trouvé, ou bien ils sont mal écrits et difficiles à comprendre. L'entreprise n'y comprend rien, elle fait pourtant un suivi très serré de son processus !



Décrivez un problème important que vous pouvez voir dans le modèle suivant (1 point).

Justifier l'importance du problème en décrivant l'impact que celui-ci pourrait avoir sur le développement logiciel (1 point).

Proposez une modification au processus qui permettrait de corriger ce problème (1 point).

Solution :

- Il y a deux problèmes : (1) L'implémentation devrait avoir la description initiale du problème et des données pour le reproduire. (2) Le processus est élaboré pour le côté gestion, mais il n'y a rien du côté de la résolution de problème. Il faudrait des activités d'analyse architecturales, des tests unitaires, des tests d'intégrations, des tests de vérification, des tests de validation, tests de régression, etc.
- Pour les deux problèmes : (1) En l'absence des détails sur la définition du problème, le développeur peut partir sur une mauvaise tangente et résoudre le mauvais problème. (2) En l'absence de détails, il est possible que les implémenteurEs écrive juste assez de

code pour résoudre le problème (ce qui explique le code difficile à comprendre), et que les testeurEs testent juste assez pour ne pas trouver de bogue (ce qui explique l'ajout de nouveaux bogues).

- Pour les deux problèmes : (1) Il faudrait juste s'assurer que les implémenteurEs ont toutes les données pertinentes pour faire leur travail. (2) Il faudrait monter un processus plus détaillé pour la résolution de problème, notamment avec un travail sur l'architecture et plus de détails sur les types de tests importants.

4. Question sur le CMMI (2 points)

Vous travaillez dans une jeune entreprise qui développe le logiciel très informellement, sans processus défini. Votre patron arrive un matin et dépose le document décrivant le CMMI sur votre table de travail. Il vous dit qu'un client de l'entreprise demande maintenant une certification de CMMI niveau 3 pour les projets logiciels. Il demande donc à vous et à votre équipe qu'il vous faut travailler au CMMI niveau 3 à partir de maintenant.

Que va-t-il se passer, d'après vous, au sein de votre équipe de travail (1 point) ?

Que feriez-vous à la place de votre patron (1 point) ?

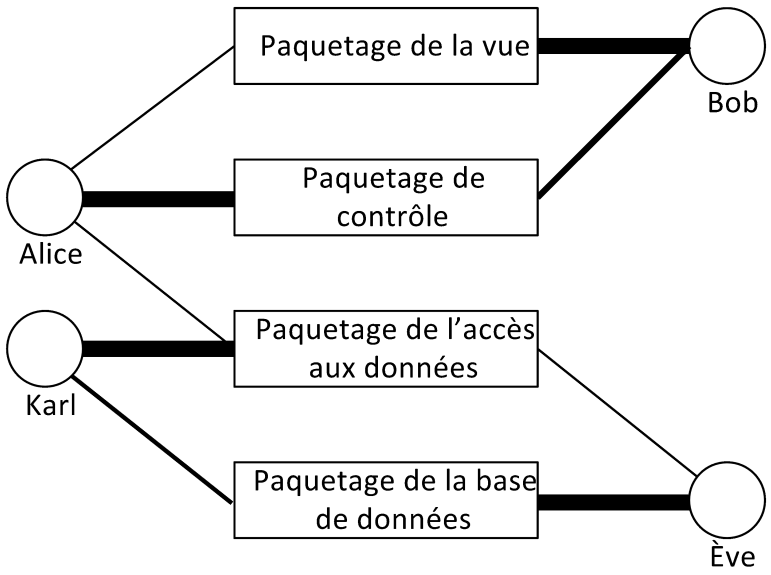
Solution : Il ne s'agit pas vraiment d'une question sur le CMMI, mais d'une question sur la résistance au changement. Il faut juste être capable d'identifier que l'entreprise est près du CMMI-1 et que de passer au niveau 3 représente un effort important.

- L'équipe va résister à ce qu'on lui demande de faire tout ce travail supplémentaire du jour au lendemain.
- Il faudrait introduire progressivement des pratiques du CMMI afin d'atteindre éventuellement le niveau 3. Le passage du niveau 1 au niveau 3 ne se fait pas de manière instantanée.

5. Question sur le travail en équipe (3 points)

5.a. Sociogramme

Le sociogramme suivant présente la participation de chaque personne dans l'équipe en termes de lignes de code ajoutées, lignes de code effacées et lignes de code modifiées (*code churn*). Le code est séparé en quatre paquetages correspondant aux quatre niveaux de l'architecture. Plus le trait entre une personne et un paquetage est épais, et plus cette personne a effectué des soumissions de code pour ce paquetage. Quel problème peut-on voir au sein de cette équipe (1 point) ?

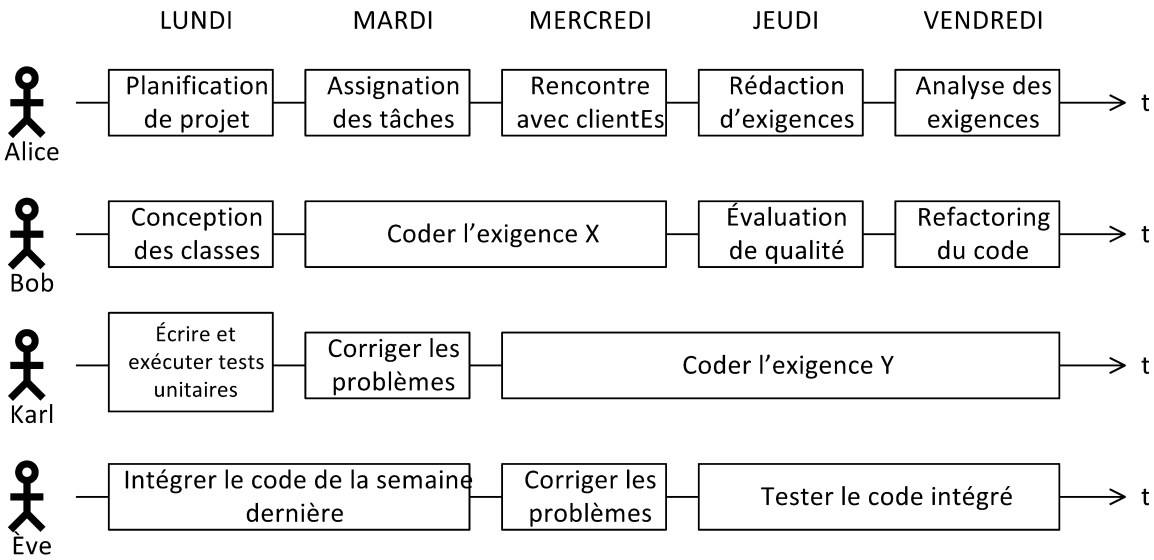


Solution :

- On remarque que les développeurEs ont travaillé essentiellement en silo. Le principal problème est que si certaines développeurs quittent, presque personne ne sais comment un paquetage fonctionne. Par exemple, si Alice quitte l'équipe, il n'y a que Bob qui a un peu de connaissances sur le paquetage de contrôle. On perd en plus toutes les connaissances sur comment le paquetage de contrôle et le paquetage d'accès aux données communiquent ensembles.

5.b. Chronogramme

Le chronogramme suivant présente une semaine de travail typique au sein d'une équipe de développement logiciel. Quel problème peut-on voir au sein de cette équipe (1 point) ?

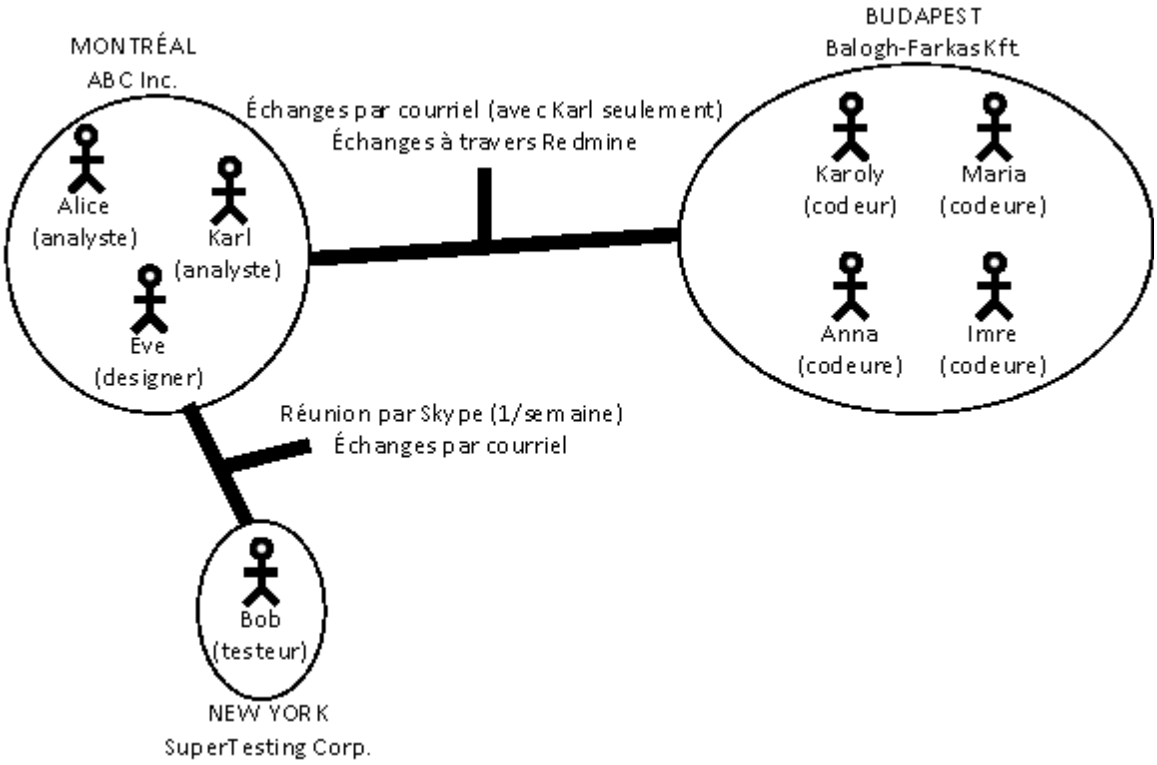


Solution :

- Encore une fois, il s'agit d'un problème de développement en silos. Tout le monde travaille chacun de son côté et ne touche pas au travail de son voisin.

5.c. Canaux de communication

Le diagramme suivant présente les canaux de communication entre les différentes équipes d'un projet de développement logiciel. Alice et Eve parlent français et anglais, tandis que Karl parle français, anglais et hongrois. Bob ne parle qu'anglais et l'équipe de Budapest ne parle que le hongrois. Décrivez deux problèmes dans le modèle suivant (1 point).



Solution :

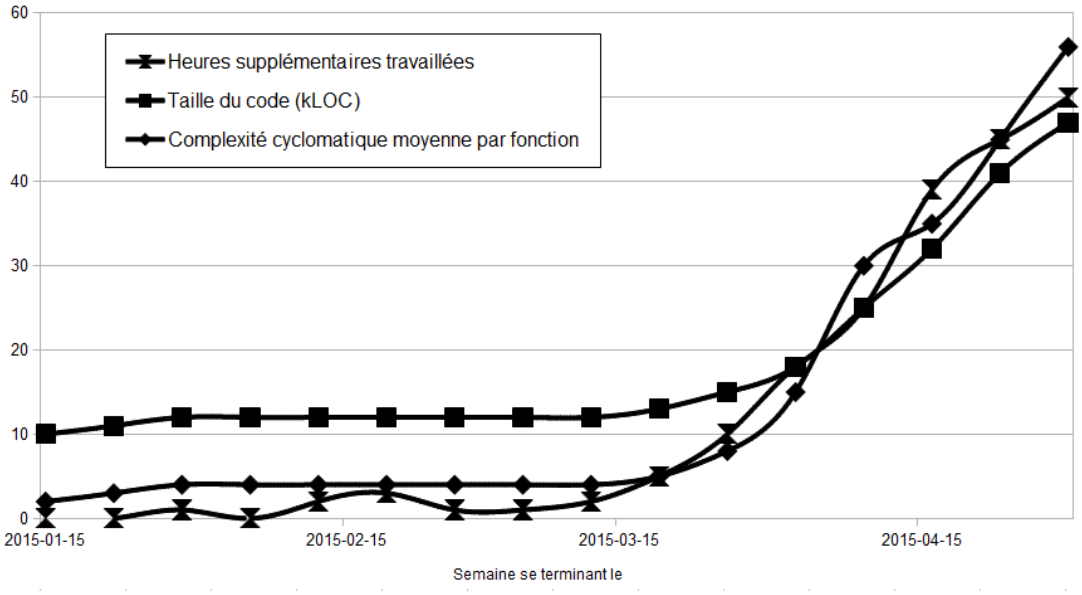
- Problème #1 : Les équipes sont cloisonnées sur leur site. Il faudrait qu'il y ait unE analyste à Budapest, unE codeurE à Montréal, etc.
- Problème #2 : Les canaux de communications pour les échanges entre Montréal et Budapest ne sont pas très riches. Il faudrait au moins un appel Skype entre Karl et l'équipe de Budapest. Il faudrait idéalement que les équipes se rencontrent au moins une fois en personne.
- Problème #3 : Aucune communication possible entre le testeur et l'équipe de codeur à Budapest ? Cela va rendre la correction des bogues particulièrement difficile.

6. *Question sur l'analyse de processus (2 points)*

Un projet de développement logiciel vient de se terminer chez ABC inc. Par contre, le résultat n'est pas de bonne qualité : Le logiciel fonctionne mais il est plein de bogues et il est très difficile à corriger. En tant qu'expert dans l'analyse des processus, vous êtes engagéE par ABC inc. comme consultantE afin de diagnostiquer la cause de la mauvaise qualité du logiciel.

Vous avez donc compilé des données sur le développement du logiciel et vous avez monté les courbes suivantes, présentant le nombre d'heures supplémentaires travaillées à chaque semaine, la taille du logiciel en milliers de lignes de code (kLOC) et la complexité cyclomatique moyenne de toutes les fonctions du logiciel.

La complexité cyclomatique est la somme des branches se trouvant dans une fonction. Plus une fonction possède de branches, plus sa complexité cyclomatique est élevée, et plus il est difficile de comprendre, tester et modifier la fonction. Le consensus actuel est qu'une complexité au-dessus de 30 est problématique.



La complexité cyclomatique moyenne du logiciel est effectivement très élevée ! Selon les données du graphique, qu'est-ce qui pourrait expliquer une complexité cyclomatique moyenne aussi élevée en fin de projet (1 point) ?

Pourrait-il y avoir une cause autre que celle présentée dans le graphique qui pourrait expliquer une complexité cyclomatique moyenne aussi élevée (1 point) ?

Solution :

- La réponse la plus évidente est que les heures supplémentaires travaillées reflètent une pression importante sur les développeurEs pour finir le travail, ce qui peut se traduire

par du code de moins bonne qualité. Les développeurs sous pression mettent de plus en plus de code dans chaque méthode, au point où celles-ci deviennent trop complexes pour être maintenues.

- Par contre, la corrélation entre heures supplémentaires travaillées et complexité cyclomatique ne veut pas dire que l'un est la cause de l'autre. En fait, ça pourrait même être le contraire, où il faut travailler plus pour corriger les problèmes causés par la complexité cyclomatique élevée. La complexité cyclomatique n'est pas seulement élevée, elle est *extrêmement* élevée : Pour que la moyenne soit au-dessus de 50, il faut qu'il y ait des fonctions réellement problématiques … La cause réelle est peut-être un design initial déficient, qui met trop de code dans un nombre restreint d'éléments.

7. Question sur la modélisation de processus (5 points)

Grâce à vos efforts acharnés et à votre talent inégalé, vous réussissez à vous trouver un travail de développeur logiciel à la NASA. Vous vous retrouvez donc en charge de la maintenance d'un vieux logiciel utilisé dans une sonde spatiale de la NASA. Même après des décennies de vol, on trouve encore des problèmes à régler. La sonde Voyager 2, par exemple, a été lancée en 1977 et a eu une dernière mise à jour de son logiciel en 2010.

Votre travail consiste donc à :

1. Recevoir des rapports de problèmes liés à la sonde spatiale.
2. Valider la présence du problème, et valider que ce problème est bien lié au logiciel.
3. Corriger le problème.
4. Déployer le correctif.

Enfin d'éviter une erreur qui pourrait être fatale pour la sonde spatiale, vous décidez de monter un processus de maintenance complet.

Il y a beaucoup de solutions possibles. Ce que nous voulons voir minimalement :

- Au moins une activité liée à la validation du problème (+1 point),
- Au moins une activité d'analyse et/ou design (+1 point),
- Au moins une activité de programmation et une activité de tests unitaires (+1 point),
- Au moins une activité de tests de vérification et une activité de validation (+1 point),
- Au moins une activité de déploiement (+1 point),
- Respect des règles de modélisation (correction négative, -0,25 point par erreur).