Overleg refactoring Steffan

Context project: Crisis management

30-4

In dit gesprek heeft Steffan ons geholpen met het opstellen van een goede vragenlijst en heeft ook hier en daar tips gegeven.

1 Vragen

1. Er zijn verschillende stakeholders met verschillende wensen, hou gaan we hiermee om/ houden we hier rekening mee?
2. Hoe zorgen dat we zo snel mogelijk onze speed te weten komen?
3. Men wil een plugin-architecture, maar wat er uiteindelijk mee moet gebeuren is niet heel duidelijk. Waar leg je de basis neer?
4. Repast:
   (a) Als je update naar een nieuwe versie, hoe kun je dan verifieren dat het op dezelfde manier werkt?
   (b) Als we er dingen uithalen, hoe zouden we dat dan vervangen?
   (c) Men wil gebruik maken van *repast* en er zijn ook veel mogelijkheden. Als wij het gaan gebruiken, hoe zorgen we er dan voor dat er makkelijk op voort te bouwen is?
   (d) Met de huidige *stepper* krijg je een andere simulatie afhankelijk van de speed. Is dit met *repast* op te lossen?
5. Hoe zorgen we ervoor dat we zo snel mogelijk de code begrijpen?
6. Hoe debug je dit soort code?
7. Hoe ga je om met exceptions?
8. **Technical dept**: Hoe gaan we om met het ontwikkelen van goede code, tegenover het ontwikkelen van nieuwe features.
9. Hoe gaan we het build process verbeteren? Ookwel het: "don't repeat yourself principle" (bijv. dat niet iedereen eigen dependencies hoeft in te voeren))

10. Aantal cruciale dingen in de code:
    (a) Alles wijst naar de environment (kan een god class zijn (singleton)): hoe gaan we zoiets refactoren.
    (b) Wanneer ga je refactoren en wanneer niet? Je kan alles gelijk gaan refactoren, maar in elk stukje zit weer een tijd van debugging. (hoe ouder de code hoe steviger het is.
11. Waar gaan we beginnen met testen, wat is de strategie?
12. Hoe zorg je ervoor dat je de feedbackloop analyseert?
13. Hoe schrijf je tests die goed de karakteristieken van de huidige code *mappen*, als je dingen vervangt, dat je controleert dat dingen het zelfde blijven?

2 tips

1. Moeten ook unit test/intigrate test toevoegen. Legacy code is vaak gekoppeld, en dan moet je veel gaan mokken, dus daar moeten we opletten.
2. Bij een daily scrum ook apart met het kleinere groepje gaan zitten.
3. Nooit direcht committen naar master file (altijd met pull request)
4. Aanvullen van de DOD(Defenition of done)
    (a) aanvullen na de eerste retro
    (b) Hoe zet je bijv. zoiets als gedocumenteer op? : Een bugfixt heeft namelijk geen documentie.
    (c) Checkstyle prove
5. Bij elke issue hoort eigenlijk een pull-request
6. Niet te open in het gesprek gaan, kom met best wel directe voorstellen
7. Voor de scrum masters wordt sterk aangeraden om zich in te lezen in het boek "de kracht van scrum" (agile manifest)
8. Vragen altijd mailen!
9. Probeer de features zo klein mogelijk te krijgen en ze dan 1 voor 1 te tacklen.
10. in principe estimate je al voor de sprint planning per team(omdat iedereen wel ongeveer op een gegeven moment welke taken bij hun horen) De eerste keer is het waarschijnlijk handig om dit met iedereen te doen.

3 Actiepunten

- Iedereen: Verklaren hoe we de prioriteit van de wensen bepalen.
- Joop: Mail naar stakeholders betreffende aanwezigheid sprint review