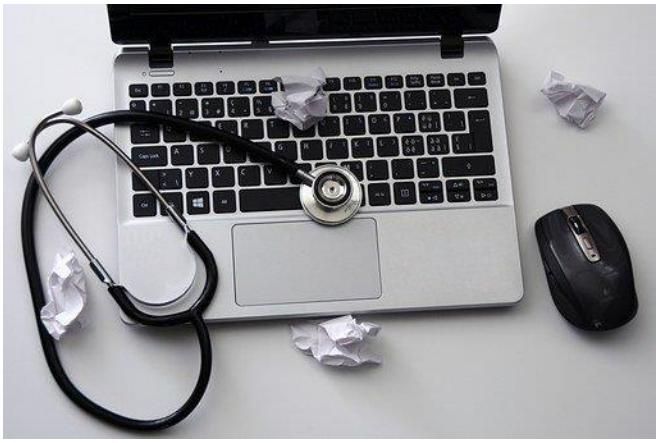
Als de software ziek is, ga je naar de softwaredokter

Deel 1: Inzicht in codekwaliteit



Als je je niet goed voelt, ga je naar de dokter. Het eerste dat de dokter doet is een onderzoek uitvoeren; symptomen waarnemen. Daarna trekt hij op basis van zijn expertise conclusies, bijvoorbeeld dat je ziek bent. Zo is het ook met software. Er zijn symptomen, maar de ziekte is nog onbekend. De symptomen kunnen van uiteenlopende aard zijn. Het is duidelijk dat er een probleem is, maar wat is de oorzaak?

Dit artikel gaat over codekwaliteit. In het eerste hoofdstuk leggen we uit wat wij onder codekwaliteit verstaan. Het onderwerp van het tweede hoofdstuk is het belang van codekwaliteit. Vervolgens gaan we in op het meten van codekwaliteit en hoe je verder kunt gaan na de meting.

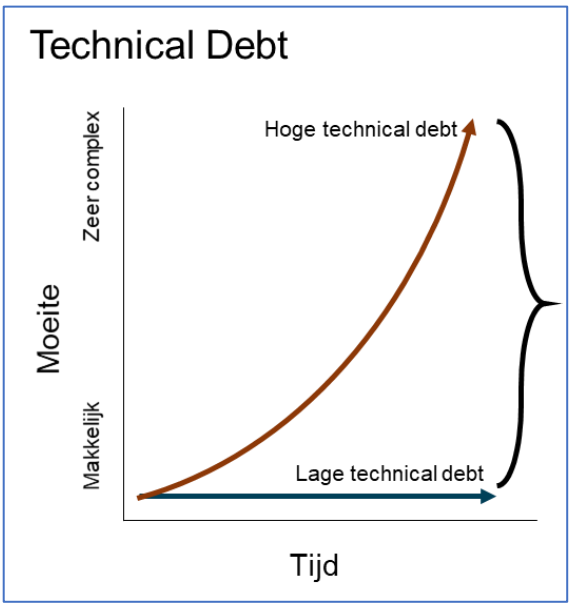
Wat is codekwaliteit?

Iedereen die wel eens softwarecode gezien heeft of zelf geprogrammeerd heeft, heeft gezien dat sommige code beter leesbaar is dan andere. Dit kan te maken hebben met de wijze waarop de code opgemaakt is, zoals inspringen van code en gebruik van lege regels. Het kan ook te maken hebben met het gebruiken van logische namen voor variabelen en functies. En uiteraard is de logica in het ene stukje code lastiger dan in het andere stukje.

Goede code voldoet volgens de experts aan de volgende kenmerken (zie [1]):

- Elegant: goede code is prettig te lezen
- Leesbaar: goede code leest als een goed geschreven roman
- Simpel: goede code heeft één doel
- Testbaar: alle testen draaien foutloos

Waarom is codekwaliteit belangrijk?



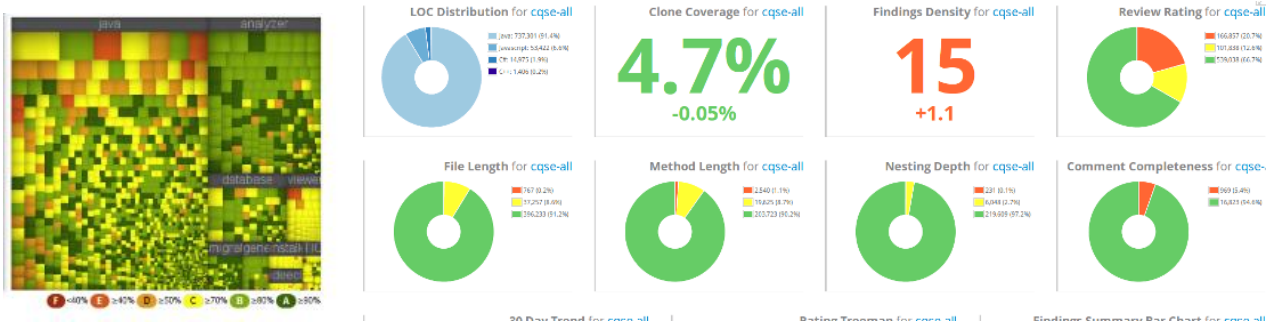
Software moet als eerste doen wat de gebruikers ervan verwachten, qua functionaliteit, snelheid etc. Dit is wat we de buitenkant van het systeem noemen, namelijk datgene wat je ziet wanneer het systeem in gebruik is. Als echter de binnenkant niet goed is, treden er problemen op.

Wanneer het lastiger wordt om wijzigingen of aanvullingen in de software door te voeren, zeggen we dat de technical debt groeit. Het is belangrijk om de technical debt te beperken. Zie ook de grafiek (gebaseerd op [2]). Als de technical debt niet op tijd opgelost wordt, kost het steeds meer moeite (tijd) om wijzigingen of aanvullingen te maken. Als je wacht met het oplossen van de technical debt, zal deze blijven groeien. Dit maakt dat het steeds meer moeite het kost (bruine lijn) om wijzigingen te maken. Bovendien bestaat er een groter risico dat een nieuwe fout wordt geïntroduceerd bij een wijziging. Door de technical debt naar beneden te brengen – naar de blauwe lijn toe – wordt wijzigen makkelijker. De kans van het introduceren van fouten wordt ook minder.

Een aspect van technical debt is codekwaliteit, naast bijvoorbeeld andere aspecten, zoals architectuur en security. Door inzicht in de kwaliteit van je code, kun je afgewogen beslissingen nemen over te nemen acties voor verbeteringen.

Hoe kan ik inzicht krijgen in codekwaliteit?

Om inzicht in codekwaliteit te krijgen, wordt gebruik gemaakt van statische analysetools. Die tools voeren metingen uit op aspecten in de software en geven daar een waarde aan, zogenaamde metrieken. Die metrieken geven indicaties voor de kwaliteit van je code op aspecten, zoals betrouwbaarheid en onderhoudbaarheid die gebaseerd zijn op ISO 25010. Sommige metrieken gaan uit van de grootte van de module en meten bijvoorbeeld het aantal regels code (Lines of Code; LOCs). Andere metrieken geven een maat aan de complexiteit van de code, bijvoorbeeld Cyclomatische complexiteit (McCabe index), die aangeeft hoeveel beslispaden er door de code heen mogelijk zijn. Weer andere metrieken kijken meer naar het proces dat de ontwikkelaar gebruikt om de code te schrijven door het aantal commits te meten, voordat de code in het systeem wordt geïntegreerd.
Voorbeelden van statische analysetools zijn TICS en SonarQube. Zij tonen metrieken op een dashboard. Zie hieronder enkele voorbeelden.



Nu zou je kunnen zeggen: “Als je de metrieken van de tools hebt, weet je toch precies waar het aan schort?” Helaas is het niet zo eenvoudig. Vaak zijn er aanvullende redenen waarom een deel van de software een slechte score krijgt. Volgens de theorie betekent een hoge cyclomatische complexiteit slechte code. In de praktijk kan het zijn dat er een goede reden is voor die hoge score. Het wil dus niet per definitie zeggen dat de code slecht is. Een ander voorbeeld is gegenereerde code die vaak ook slecht scoort. Gegenereerde code is vrijwel altijd minder leesbaar dan code die direct door een ontwikkelaar is geschreven. Er dient in dit geval niet gekeken te worden naar de gegenereerde code, maar naar de bron waarvan de gegenereerde code is afgeleid.

We weten de toestand van de code. En nu?

Er kunnen verschillende oorzaken zijn voor het hebben van slechte code. Het kan bijvoorbeeld te maken hebben met de opleiding van de ontwikkelaars of de aansturing van het project. Om verbeteringen door te voeren gebruiken experts de metrieken als vertrekpunt in de analyse van de codekwaliteit. Daarnaast worden aanvullende analyses uitgevoerd, zoals code reviews en een analyse van de projectaansturing. Andere gegevens kunnen ook gebruikt worden, zoals bijvoorbeeld testrapporten en incidentrapporten. Op die manier kan een goede inschatting gemaakt worden van de gezondheid van de code. In deel 2 wordt verder besproken hoe en waar verbeteringen doorgevoerd kunnen worden.

Waarom Improve Quality Services?

Improve heeft zijn oorsprong in testen en kwaliteit. Meer dan 20 jaar zijn de experts van Improve al bezig om verbeteringen door te voeren in de kwaliteit van software. Veel experts van Improve komen uit de techniek en veel hebben ontwikkeld of ontwikkelen tot op de dag van vandaag nog steeds. Ze werken in velerlei branches, zoals banken en verzekeringen, maar ook civiele en industrieprojecten, zoals tunnels en productieplants.

Een belangrijke dienst van Improve is het verzorgen van trainingen. Zo geven de trainers van Improve workshops aan ontwikkelaars op het vlak van testen en kwaliteit. Tevens zijn de trainers van Improve in staat om ontwikkelaars te begeleiden op de weg naar het schrijven van code met een hogere kwaliteit.

Referenties

[1] Clean Code, Robert Martin, 2009 Pearson Education, Inc.
[2] The UX of Technical Debt, Glen Lipka, http://commadot.com/the-ux-of-technical-debt/