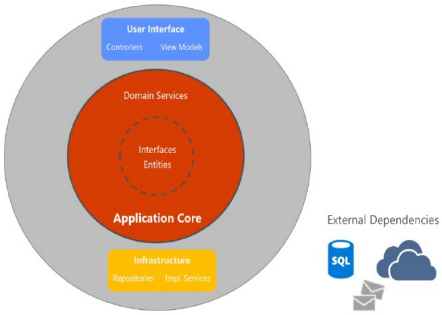
Softwareontwerp & -architectuur 3 Eindopdracht

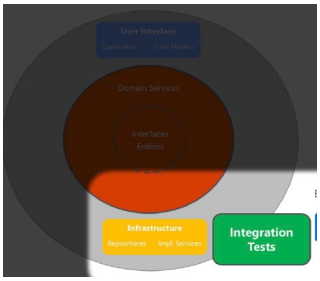
Inleiding

In periode IVT2.1 heb je kennisgemaakt met een aantal applicatiearchitecturen, zoals de clean (onion) architecture (Smith, 2017). In deze architectuurstijl staat de *application core*, het domeinmodel met bijbehorende services, centraal, zoals figuur 1 aangeeft.

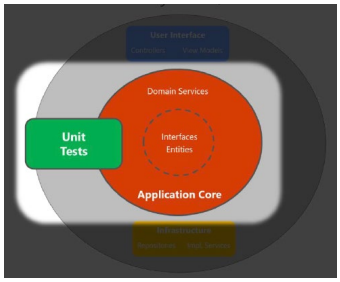


Figuur 1 - clean architecture (Smith, 2017)

In de lessen Softwareontwerp & -architectuur 3 heb je meer geleerd over aspecten van softwarekwaliteit en non-functional requirements. Deze zijn in alle ringen/lagen toe te passen, maar de nadruk ligt voor deze opdracht tot de application core. Stub implementaties voor bijvoorbeeld opslag in een data store of het sturen van een bericht naar een gebruiker via een medium als e-mail (in de infrastructure ring) voldoen en hoeven dus niet echt uitgewerkt te worden. Hierdoor hoeven we geen integratie tests (figuur 2) te schrijven, maar kunnen we ons beperken tot unit testing voor de klassen in de application core (figuur 3).



Figuur 2 - integration tests en de clean architecture (Smith, 2017)



Figuur 3 - unit tests en de clean architecture (Smith, 2017)

Doel

Het doel van deze opdracht is om een domein te ontwikkelen voor een Scrum/DevOps projectmanagement systeem vergelijkbaar met Azure DevOps of Atlassian Jira, waarin je minimaal zes design patterns gebruikt, die je kunt onderbouwen aan de hand van de OO ontwerpprincipes. Je doet dit op basis van vooraf opgestelde functional en non-functional requirements, en je laat zien dat je hieraan voldoet door middel van een ontwikkelstraat die na een push automatisch tests uitvoert en kwaliteitsmetrieken genereert. Je kunt hiervoor de SonarCloud omgeving gebruiken. De eis aan de kwaliteit die we hoe dan ook op leggen is dat je code met een Quality Gate label A wordt beoordeeld. Aan de hand van het ontwerp dat je maakt, implementeer je de application core en passende unit tests. Tijdens het assessment verdedig je, en reflecteer je op je (ontwerp)keuzes. Je ontwikkelt geen user interface, en ook de opslag van de data mag je buiten beschouwing laten.

Opdracht

De beschrijving van de casus is niet volledig, en dat kan ook niet. Het zou heel veel tekst beslaan met erg veel details en dat is niet nodig voor de opdracht. Zie de tekst als de kaders waarbinnen je zelf keuzes maakt in functionaliteiten. Bepaal en beschrijf vooraf wat de functional en non-functional requirements zijn die je in de tekst vindt. Vul open ruimten in de opdrachtbeschrijving op met eigen aannames, en documenteer deze ook.

Als je meerdere varianten hebt van een bepaald concept (bijvoorbeeld GitHub en GitLab) hoef je ze niet allemaal helemaal uit te werken. Eén of twee is voldoende. Het gaat erom dat je de werking van het geheel aan kunt tonen, en dat je d.m.v. het ontwerp hebt aangetoond dat je inziet dat meer variatie mogelijk is en dat die ondersteund wordt met de juiste principes.

Kijk voor inspiratie ook op internet bij materiaal/blogs over Azure DevOps of andere tooling van dit type. Verlies jezelf vooral niet in details, maar houdt altijd in de gaten dat het doel van deze opdracht het toetsen van de leerdoelen van het vak is!

Ontwerp

Je onderbouwt de keuze van design patterns met een ondersteunend verslag, waarin je ook de resultaten van de code-analyse opneemt en toelicht. Het ontwerp bevat minimaal 6 design patterns, waarvan één creational pattern meetelt (meer toepassen mag, maar die tellen niet mee), en van de overige 5 minimaal 4 verschillende structural/behavioral patterns. Schematisch:

<table>
<tr><td rowspan="3">Minimaal 6 patterns</td><td colspan="2">1 creational telt hierin mee;
meer mag in het ontwerp</td></tr>
<tr><td rowspan="2">5 behavioral/structural patterns</td><td>1 of meer vrije keuze
(eventueel 1 van de 4 nogmaals toegepast)</td></tr>
<tr><td>4 verschillende</td></tr>
</table>

Er zijn meer OO design patterns dan we in de les behandeld hebben, zie bijvoorbeeld het boek Head First Design Patterns. Die mag je natuurlijk ook toepassen! (m.u.v. singleton, dat is eigenlijk een anti-pattern). Je maakt het ontwerp in geschikte UML-diagrammen. Zorg ervoor dat het geheel overzichtelijk blijft – splits dus diagrammen waar nodig.

Code

Je werkt je ontwerp uit in een OO-taal naar keuze. Geef in de code aan waar je welk design pattern toe hebt gepast. Repositories kun je opnemen in je ontwerp, maar daar hoef je geen implementaties met echte databases voor te realiseren. Minimale implementaties met objecten in geheugen / uit tekstbestanden die het testen ondersteunen zijn voldoende (fake repositories). Hetzelfde geldt voor de implementatie voor het sturen van berichten via media als e-mail, Slack, sms etc. Een stub-implementatie voldoet hier. Mocken is een verplicht onderdeel van je testen.

Testen

Voor logica in het domein implementeer je unit testen. Redeneer hier vanuit de use cases / user stories en denk hierbij aan bijvoorbeeld het veranderen van de status van backlog items (en de spelregels daarbij!). Je test dus geen standaard getters en setters van klassen waarbij je het idee hebt dat de tests weinig toevoegen, maar wel die code waar meer logica is die aan bepaalde regels moet voldoen. Voor onderdelen met een complexere structuur (hogere cyclomatische of cognitieve complexiteit) stel je testen op met de stappen voor path coverage. In het kader van onderhoudbaarheid heeft de quality officer bij Avans bepaald dat de kwaliteit van de code door SonarCloud beoordeeld dient te worden met een Quality Gate label A (Sonar way). Dit is een aanknopingspunt om de non-functionals specifiek en meetbaar te kunnen definiëren.

Oplevering

Je werkt met een DevOps-omgeving naar keuze waarin je SCM (source-code-management) toepast, en gebruik maakt van development pipelines die minimaal je code bouwen, testen en een statische code analyse uit (laten) voeren. Verder lever je op, in één zip bestand, via de inleverlink op Brightspace (i.v.m. archivering binnen Avans):

- pdf met:
    - requirements (functioneel en niet-functioneel, definieer dus ook je codemetrieken).
    - acceptatiecriteria voor alle requirements, zodat je hierop je testcases kunt baseren
    - relevante UML-diagrammen die jouw systeemontwerp inzichtelijk maken, met een toelichting en een onderbouwing van je keuzes.
    - een testaanpak, waarin je beschrijft welke use cases, userstories of business rules je test,
    - een overzicht van testcases, die herleidbaar (traceable) zijn naar de requirement die zij testen,
    - een rapportage van de code-analyse, met een evaluatie daarop.
- zip-bestand met daarin de code (zonder binaries).

Casus

Avans DevOps is een suite van DevOps tools die Scrumprojecten van studenten ondersteunt, inclusief integratie van development pipelines (ontwikkelstraten) en een discussieforum over het te verrichten werk. De scope voor deze opdracht is om tot een uitgewerkte application core te komen, die zowel geprogrammeerd is als getest is op cruciale logica en non-functionals.

In de tool kunnen projecten aangemaakt worden en voor elk project bestaat de ondersteuning uit in ieder geval de volgende onderdelen:

- Projectmanagement - het werk dat gedaan moet worden volgens de Scrum-methode, met mogelijkheid tot discussie via berichten.
- Het code-archief: software configuration management (SCM) voor de ontwikkelde code van het project.
- DevOps d.m.v. een development pipeline (ontwikkelstraat) met ondersteuning voor bouwen van code, testen, analyseren en deployen.

Project management volgens Scrum

De requirements voor elk project worden bijgehouden in de vorm van een product backlog en dat is zoals je weet een geordende lijst van product backlog items. Omdat een backlog item soms te groot is als taak voor één developer, ondersteunt de tool het aanmaken van activiteiten binnen een backlog item. Backlog items kunnen gekoppeld worden aan een sprintbacklog en daarin ook aan een developer. Er kan maximaal één developer aan een backlog item gekoppeld worden. Indien er meer developers aan gaan werken, dienen activiteiten onder het backlog item bijgemaakt te worden. Een backlog item kan pas done zijn, indien alle onderliggende taken dat zijn.

Een sprint kan als doel hebben een deelproduct af te hebben voor een sprintreview voor feedback van de product owner en andere stakeholders, of een release van de software (deployment). In beide gevallen doorloopt de sprint verschillende stadia. Het eerste stadium is dat de sprint alleen nog is gecreëerd en alleen eigenschappen als naam, begin- en einddatum, gewijzigd kunnen worden. Backlog items kunnen aan de sprint(backlog) worden toegevoegd. Wanneer de sprint uitgevoerd wordt, kunnen voorgaande activiteiten niet meer aangepast worden. Backlog items van een sprint kennen de fasen todo, doing, ready for testing, testing, tested en done, en additionele fasen zijn ooit denkbaar. Backlog items beginnen zoals je weet in de todo fase bij aanvang van de sprint en gaan liefst in de hierboven beschreven volgorde uiteindelijk naar done. Wanneer een backlog item in de fase ready for testing komt, krijgen testers een notificatie. Mocht een tester erachter komen dat er toch iets ontbreekt of fout gaat met een item waarvan de developer aangeeft dat hij klaar was (ready for testing done), dan gaat het item terug naar todo. Terug naar doing kan niet: als het goed is, werkt iedereen aan één item en volgens scrum moet het wisselen van taken voorkomen worden. Overigens krijgt de scrum master een notificatie wanneer dit gebeurt en kan hij de developer die het item zogenaamd af had gemaakt hierop aanspreken. Wanneer een backlog item getest is, controleert een (lead) developer via de definition of done of het echt naar de done toestand mag. Mocht dat niet zo zijn, dan gaat het item eerst terug naar ready for testing. De tester gaat eerst opnieuw testen. Gaat dat goed, dan gaat het item opnieuw naar tested. Zo niet, dan betekent dit werk voor de developer: het item komt opnieuw in to do. Alle notificaties kunnen via e-mail, maar ook via andere media als Slack etc. (combinaties zijn mogelijk).

Wanneer een sprint af is gelopen (‘de tijd is op’) krijgt deze de status ‘finished’. Afhankelijk van het type sprint kan vanaf daar een review plaatsvinden of de release in gang gezet worden wanneer de resultaten goed genoeg zijn. Mocht dat niet zo zijn, dan wordt de release geannuleerd – een vervelende situatie die een eindstatus is van de sprint met automatisch bericht naar de product owner én scrum master via mail, Slack, …. Mocht de release wel in gang worden gezet, dan wordt de

voor die sprint ingestelde development pipeline gestart (installeren packages, build, test etc. tot en met deployment). Zolang de activiteiten van de development pipeline worden uitgevoerd, kan de sprint niet gewijzigd worden. Indien alle activiteiten van de pipeline succesvol zijn uitgevoerd, is de sprint gereleased en daarmee gesloten; notificatie naar scrum master en product owner. Indien er onverhoopt toch een fout optrad, krijgt de scrum master een melding. Deze kan ervoor kiezen opnieuw het release proces proberen uit te voeren (misschien was er in de context iets mis als een server die niet bereikbaar was), of om de release te annuleren.

Bij sprints die afgesloten worden met een sprint review, volgt een fase die de sprint echt afsluit. De scrum master initieert de bijbehorende actie en kan deze alleen uitvoeren wanneer hij een samenvatting van de review als document voor de sprint heeft geüpload.

Het kan zijn dat er onduidelijkheid is over een backlog item. Dan kan men daarover discussiëren door naar elkaar toe te stappen of, indien developers niet fysiek bij elkaar zitten, te Skypen/mailen etc. Uiteindelijk worden resultaten van deze discussies in het forum van de applicatie gezet. Daar kunnen verschillende discussies met diverse onderwerpen gestart worden of men kan reacties toevoegen. Er kunnen dus diverse discussiethreads ontstaan, zoals je op diverse fora ziet. Elke thread is gerelateerd aan een backlog item. Zodra het backlog item, waar de discussie betrekking op heeft, afgerond is, kan niets meer worden gewijzigd in de discussie(onderdelen) en kunnen er geen nieuwe berichten aan die threads worden toegevoegd of nieuwe threads worden gemaakt. Let erop dat een backlog item in de kolom ‘done’ kan worden geplaatst, maar daar ook weer terug naar ‘todo’ kan, mogelijk gepaard mét of naar aanleiding van een discussie. Dus waarschijnlijk is die status niet voldoende… Bij reacties in de discussie krijgen teamleden een bericht.

Voor elke sprint is rapportage te genereren, bijvoorbeeld met informatie als teamsamenstelling, burndownchart, aantal effort-punten per developer, etc. Om de rapportage een professioneel tintje te geven, is het mogelijk headers en footers toe te passen. Daarin kunnen bedrijfsnaam/logo, project naam, versie, datum etc. in opgenomen worden. Het rapport is vervolgens in verschillende formaten (bv. pdf, png) op te slaan.

Elke sprint is gekoppeld aan een aantal personen, die verschillende rollen kunnen vervullen: developers, scrum master (1 per sprint). Op projectniveau komt daar de product owner nog bij.

Software Configuration Management

Natuurlijk gaan we voor het SCM-deel zelf geen version control software maken. Er zijn al erg goede systemen daarvoor zoals Git, Subversion etc. Deze worden geïntegreerd in ‘ons’ systeem Avans DevOps. Vermijd daarbij al te veel detail. Kijk in Azure DevOps hoe de integratie met Git is. Veel gebeurt via command-line commando’s en waarschijnlijk heb je daarbij aan een beperkte set version control (Git) concepten zoals commit, branch etc. genoeg om een koppeling te maken van backlog items/activities/sprints/… naar de code base.

DevOps

Hiervoor geldt hetzelfde als voor SCM: dit gaan we niet zelf ontwikkelen, maar we integreren bestaande faciliteiten met Avans DevOps. In Azure DevOps is deze werkwijze vergelijkbaar: er worden niet alleen Visual Studio builds ondersteund, maar er is een hele verzameling acties voor diverse platformen om een development pipeline samen te stellen. Avans DevOps development pipelines zijn opgebouwd uit verschillende typen acties:

- Sources
  Activiteiten om de source code die gebouwd (en mogelijk getest en gedeployed) moet worden op te halen naar een context waarin de gehele pipeline wordt uitgevoerd.

- Package
  Met activiteiten in deze categorie kun je diverse (3rd party) packages/libraries installeren waar je eigen software afhankelijk van is.
- Build
  Dit zijn acties die je software builden en linken. Naast .NET en .NET Core builds worden diverse andere builds ondersteund, zoals Maven, Ant, maar ook een build job via Jenkins.
- Test
  Voert test uit, bv via NUnit, Selenium etc. Deze categorie bevat ook acties om test resultaten te publiceren, of coverage resultaten.
- Analyse
  Voor het analyseren van code via een externe tool als SonarQube. Sommige van deze tools hebben meerdere acties, zoals analysis preparation, analysis execution, analysis reporting.
- Deploy
  Bevat acties om deployment op bv Azure uit te voeren.
- Utility
  Dit kunnen diverse acties zijn, die niet onder bovengenoemde categorieën vallen zoals het runnen van een batch script, files copiëren/deleten/downloaden, command line acties.

In de casus gaat het erom dat het ontwerp definitie van basis development pipelines ondersteunt en de uitvoering van de acties tijdens testen mag je natuurlijk simuleren. Kijk in Azure DevOps documentatie om een idee te krijgen wat voor settings een rol spelen en neem enkele belangrijke op in je ontwerp – meer voor het idee dan dat je je verliest in allerlei details.

Een deployment sprint is gekoppeld aan een development pipeline die eindigt met deployment. Andere sprints kunnen ook worden gekoppeld aan een development pipeline die automatisch/handmatig wordt uitgevoerd. Daarvan kan het einde ook deployment zijn (naar bv een testomgeving), maar het kan ook eindigen na het uitvoeren van de tests en publicatie van testresultaten.

Bibliografie

Smith, S. (2017). *Architecting Modern Web Applications with ASP.NET Core and Azure.* Redmond, Washington: Microsoft Corporation.