Algemene opmerkingen m.b.t. UML

Versie: 1.2

Datum: 7-2-2023

Inhoud

1. UML diagrammen

In boeken over UML zie je dat de verschillende diagrammen vaak als indeling rondom de uitleg worden gebruikt.

Het is echter goed om je te realiseren dat zelfs de auteurs van UML de diagrammen niet als het centrale deel van UML zien (Fowler, 2004, p.11).

Vaak kun je legaal elementen van het ene diagramtype op een ander diagram gebruiken. De UML-standaard geeft aan dat bepaalde elementen typisch op bepaalde diagramtypes worden getekend, maar dit is geen voorschrift (Fowler, 2004, p.11).

Een duidelijk voorbeeld is het package element. Op zich is er een package diagram, waarbij je door middel van afhankelijkheden tussen packages bijvoorbeeld een indeling van software kunt weergeven. Denk aan namespaces in C# en packages in Java. Maar het package element kun je in allerlei diagrammen gebruiken. Bijvoorbeeld in het use case diagram om use cases te groeperen. Of in een klassendiagram om klassen te groeperen. Dat kan binnen een complex diagram overzicht geven. Of wat te denken van de pijl die voor afhankelijkheid staat. Deze pijl kun je in allerlei diagrammen terug vinden bijvoorbeeld in het use case diagram, klassendiagram en package diagram.

2. Correct UML

UML is zo complex dat de standaard vaak op meerdere manieren geïnterpreteerd kan worden (Fowler, 2004, p.13).

Niet alle UML-tools houden zich volledig aan de UML-standaard of ondersteunen niet alle UML-onderdelen. Het is daarom belangrijk dat we met elkaar richtlijnen afspreken in de gevallen waar dat nodig is.

Je moet in gedachten houden dat een algemeen principe binnen UML is, dat alle informatie voor een bepaald diagram in principe achterwege gelaten kan worden. Dat kan bijvoorbeeld zijn, alle attributen verbergen, of heel specifiek, een drietal klassen niet tonen. Van een diagram kun je nooit iets afleiden uit het ontbreken van iets. (Fowler, 2004, p14).

Je hoort vaak beweren dat als je geen multipliciteit weergeeft bij een associatie dat het een 1 op 1 relatie is. Dat is niet correct. Het betekent gewoon dat de degene die het diagram getekend heeft het niet weet, of voor de context niet belangrijk vindt.

3. Ook met UML kun je niet alles

UML kent vele diagrammen. Veel diagrammen zijn ook heel bruikbaar en worden in de praktijk veel gebruikt. Het is ook heel handig dat we als ontwikkelaars dezelfde diagrammen gebruiken. We hoeven dan immers niet meer uit te leggen hoe de diagrammen gelezen moeten worden.

Maar UML ondersteunt ons niet in alle situaties. Zo is er geen diagram voor het weergeven van een schermflow. Je zou daarvoor een UML-diagram oneigenlijk kunnen gebruiken, bijvoorbeeld het toestandsdiagram. Je kunt natuurlijk ook een eigen diagramnotatie daarvoor bedenken.

In de praktijk zie je dat ontwikkelaars vaak terug vallen op eigen tekentechnieken wanneer ze op hoog niveau een architectuur weergeven. Het componenten diagram en het deployment diagram zie je daarvoor niet veel gebruikt worden. Blijkbaar bevalt ons dit als ontwerpers niet zo. Misschien komt dat ook omdat in verschillende versies van UML er andere toepassingen zijn geweest voor componenten. Martin Fowler schrijft daarover:

“In earlier versions of the UML, components were used to represent physical structures, such as DLLs. That’s no longer true; for this task, you now use artifacts” (Fowler, 2004, p. 141).

In het werkveld zie je dat voor diagrammen zoals het component en het deployment diagram een alternatief komt opzetten, namelijk het C4-model. Het C4-model is makkelijk te leren, en leent zich uitstekend om een software architectuur mee weer te geven. In ons UML onderwijs behandelen we daarom in plaats van het componenten en deployment diagram het C4-model (zie https://c4model.com/).

4. UML en Scrum

Vaak zie je studenten UML alleen gebruiken om achteraf de code te documenteren. Dan doe je UML te kort. UML is heel geschikt tijdens het analyseren en ontwerpen. Bijvoorbeeld tijdens een scrum refinement of sprint planning. Dit komt omdat de vele diagrammen je op verschillende manieren naar je domein en architectuur laten kijken. Succesvolle ontwerpers zie je hun ideeën communiceren door het schetsen van UML diagrammen op een whiteboard.

5. Requirements analyse

In een scrum project is het team samen met de product owner continue bezig om de requirements boven water te krijgen. Verschillende UML diagrammen kunnen hiervoor ingezet worden:

- Een use case diagram om de verschillende actoren te onderkennen en de functionaliteit die zij nodig hebben om hun werk te kunnen doen.
- Een klassendiagram om het domein te leren kennen, en de taal van de klant te leren spreken.
- Een activiteitendiagram om de werkprocessen in kaart te brengen en na te denken over de implementatie van een use case.
- Een toestandsdiagram om je te laten nadenken over de levenscyclus van een object en de verschillende events die daar invloed op hebben. Je zult verbaasd staan over het aantal requirements dat je boven water kunt halen, alleen al door een toestandsdiagram van een bepaald object te gaan uitwerken.

Het is belangrijk dat het ontwikkelteam beseft dat de UML diagrammen door iedereen begrepen moeten worden, zeker ook door de gebruikers en klanten van het toekomstige systeem waarmee je in gesprek bent. Dus maak de UML diagrammen in deze fase niet te complex als je met een gebruiker of klant communiceert.

6. Ontwerpen

In de ontwerpfase worden de diagrammen meer technisch van aard en is het ook belangrijker om correcte UML diagrammen te maken, zodat de ontwikkelaars elkaar goed begrijpen. Verschillende diagrammen kunnen in deze fase heel nuttig zijn:

- Een klassendiagram om de softwareklassen en de relaties daartussen overzichtelijk te tonen. Let op, we hebben het hier niet over één klassendiagram voor de gehele software. Dat maakt juist dat je door de bomen het bos niet meer ziet en daarvoor kun je beter een package diagram gebruiken. Je kunt een klassendiagram wel goed gebruiken als je een deel of een bepaald aspect van je ontwerp wilt uitleggen.
- Een packagediagram om een globaal overzicht van de software te geven.
- Een sequentiediagram om de interactie tussen klassen te laten zien. Je kunt bijvoorbeeld laten zien hoe een scenario van een use case in de software geïmplementeerd is.
- Een toestandsdiagram voor objecten die in verschillende toestanden kunnen verkeren, en waarbij je overzichtelijk wilt maken hoe ze van de ene toestand in een andere toestand kunnen overgaan. Het geeft daarmee een goed overzicht van de overeenkomstige software code.

7. Documentatie

UML is heel geschikt als ondersteuning bij het documenteren van wat er uiteindelijk gerealiseerd is. Het is dan wel van belang dat de documentatie overeenstemt met hetgeen er gemaakt is. Is dat niet het geval, dan verlies de documentatie zijn waarde, en zullen ontwikkelaars de documentatie gaan negeren en direct in de code kijken. Scrum ondersteunt het op peil houden van de documentatie door in de definition of done op te nemen dat een item pas af is als de documentatie is bijgewerkt.

Het is de kunst om niet te weinig, maar ook niet te veel te documenteren. Ergens ligt een optimum en dat is erg afhankelijk van de context. Het is in ieder geval belangrijk om steeds het doel van een diagram goed in het oog te houden. Neem bijvoorbeeld niet alle operaties van een klasse op in een klassendiagram, maar alleen die operaties die je helpen om de essentie van een ontwerp te begrijpen. Doe je dat niet, dan zie je al snel door de bomen het bos niet meer.

Hetzelfde geldt voor sequentiediagrammen. Het is echt niet nodig om voor ieder scenario van alle use cases een sequentiediagram te tekenen.

Ook ga je niet voor iedere klasse een toestandsdiagram tekenen. Vaak is er in een klein schoolproject slechts één klasse in het systeem waarvoor dit nuttig is.

Tenslotte kan het nog nuttig zijn om een complex stuk software met een activiteitendiagram te ondersteunen.

8. Meerdere versies van een diagram

Zoals beschreven kun je UML in meerdere fases van je project inzetten. Het is dan ook niet ongebruikelijk om bijvoorbeeld van een klassendiagram of use case diagram meerdere versies te hebben. Bijvoorbeeld een use case diagram van hetgeen er tot dan daadwerkelijk gerealiseerd is, en een versie waarmee je als ontwikkelteam al aan het vooruitdenken bent. Hetzelfde zal vaak voor een klassendiagram het geval zijn.