Deel III Modelleren en testen

Activity diagrammen

- Hoe modelleer ik een activity diagram bij een use case?
- Welke modelelementen kent een activity diagram voor een use case?
- Wat is control flow? Wat is object flow?
- Hoe plaats ik het stappenplan van een use case in een activity diagram?
- Wat is een activity? Wat zijn control nodes?
- Hoe modelleer ik beslismomenten in een activity diagram?
- Hoe modelleer ik samengestelde beslismomenten in een activity diagram?
- Hoe voer ik een andere use case uit in een activity diagram?
- Hoe valideer ik de postcondities van use cases in een activity diagram?
- Hoe modelleer ik herhalingen in het activity diagram?
- Wanneer gebruik ik initial nodes en activity final nodes?
- Wat zijn fork nodes en join nodes? Wanneer zijn ze nodig?
- Wanneer partitioneer ik een activity diagram?

Testscenario's en testgevallen

- Hoe distilleer ik testscenario's voor een use case?
- Wat zijn deelpaden en hoe identificeer ik deelpaden in een activity diagram?
- Hoe stel ik testscenario's samen uit deelpaden?
- Welke en hoeveel testscenario's heb ik minimaal nodig om een use case te testen?
- Hoe definieer ik testacties?
- Hoe vind ik testattributen bij testacties?
- Hoe definieer ik testgevallen?
- Wanneer zijn er afdoende testgevallen?
- Wat is pair testing?

Activity diagrammen

Do or do not. There is no try.

Yoda (The empire strikes back)

Een van de lastigste rollen in een systeemontwikkelproject is die van tester. Vaak is de tester de boeman. Ontwikkelaars stoppen bloed, zweet en tranen in de applicatie. Vervolgens kijkt een tester er zeer kritisch naar en constateert de ene na de andere fout, pardon verbetering. Testers brengen de tekortkomingen van een project aan het licht. Dit maakt ze niet geliefd bij ontwerpers en ontwikkelaars. Ook niet bij de projectleider, trouwens. Steeds als een tester een fout constateert, loopt het project immers verder uit.

In de kreukelzone

De huidige generatie auto's heeft een kreukelzone, bedoeld om de eerste klappen op te vangen bij een aanrijding. Zolang een auto niet botst, is er niets aan de hand. Maar bij iedere willekeurige aanrijding deukt de kreukelzone het eerst. Testen bevindt zich in de kreukelzone van projecten. Zolang het project als een zonnetje loopt is er voldoende ruimte om te testen. Totdat de deadline in zicht komt. Botsing! Tijdnood! Er moeten impopulaire maatregelen genomen worden. Plots zijn er nog maar twee weken beschikbaar om de applicatie te testen, in plaats van de geplande twee maanden. Of erger nog, de applicatie wordt opgeleverd terwijl deze alleen is getest door de ontwikkelaars.
Opvallend genoeg wordt vooral de opgeleverde applicatie getest. En deze applicatie is traditioneel pas aan het eind van een project gereed. Barry Boehm stelde al in 1976 dat de kosten voor het oplossen van fouten exponentieel toenemen naarmate deze later worden ontdekt. Het is dus zaak fouten in een zo vroeg mogelijke fase van een project te detecteren. Wanneer een project in korte iteraties werkt en steeds een klein deel van de functionaliteit ontwerpt, realiseert en test, zoals in een agile project, wordt al een flinke verbetering bereikt. Testen vindt nu niet meer alleen aan het eind van een project plaats, maar tijdens iedere iteratie. Kent Beck meent zelfs dat met het gebruik van de juiste technieken de curve van Boehm flink wordt afgeplat [Beck-01].

Requirements testen

Stel dat in een project niet alleen de opgeleverde applicatie, maar ook alle requirements en het ontwerp kunnen worden getest. Fouten zijn dan nog eerder uit te bannen. Nog voordat er code wordt geschreven. Tijd om twee vliegen in één klap te slaan. De requirements zijn gemodelleerd in procesdiagrammen en use cases.
Use cases gelden als basis voor het modelleren van de functionaliteit. Om use cases te kunnen testen moeten testgevallen beschikbaar zijn voor alle mogelijke scenario's van de use case. Niet alleen het gewenste, maar ook de faal- en herstelscenario's (zie hoofdstuk *Use cases beschrijven*). Het activity diagram biedt uitkomst. Het leent zich uitstekend voor het identificeren van alle verschillende scenario's van de use case, en dus ook voor de testscenario's en testgevallen. En juist het identificeren van *alle* scenario's is voor een goede test van belang.
Een goed voorbeeld vormt use case **Valideren Creditcard**. Onmisbaar bij het aanmelden als abonnee bij Dare2Date. In het gewenste scenario keurt de creditcardmaatschappij de creditcard van de bezoeker goed. In een van de faalscenario's komt er echter geen goedkeuring. Niet kredietwaardig. De bezoeker wordt geen abonnee van Dare2Date. Zaak gesloten. Maar is de zaak wel echt gesloten? Misschien willen de initiatiefnemers van Dare2Date de bezoeker later nogmaals een abonnement aanbieden. Wellicht is de bezoeker later wel kredietwaardig. Dit herstelscenario wordt maar al te gemakkelijk over het hoofd gezien.
Het testen van de requirements voorkomt zelfs technische fouten. Wat gebeurt er als de creditcardmaatschappij niet reageert? Misschien is de te raadplegen web service wel uit de lucht. Wordt er dan wel of geen abonnement verstrekt? Het is het veiligst om de bezoeker geen abonnement te verstrekken, maar ook hier zijn alternatieven. Wellicht kiest men ervoor de gok te nemen en toch een abonnement te verstrekken. Verstandiger is misschien om te melden dat de creditcard niet kan worden gevalideerd en dat dit op een later tijdstip nog eens wordt geprobeerd. "U hoort nog van ons."
De diverse scenario's bij een use case worden vaak als tekst uitgeschreven. Hierbij is er echter geen garantie dat alle mogelijke scenario's boven water komen. Het modelleren van een activity diagram bij een use case geeft wel een volledig beeld.

Het activity diagram

Net als de traditionele concurrenten, zoals het data flow diagram en de flowchart, wordt het activity diagram voor diverse doeleinden gebruikt:

- *Bedrijfsproces.* In hoofdstuk *Starten met bedrijfsprocessen* is al aangegeven dat een activity diagram bij uitstek geschikt is voor het modelleren van een chronologisch bedrijfsproces, zoals het aanvragen van een levensverzekering.

- *Use case*. Het activity diagram biedt een uitstekend overzicht van het stappenplan van een use case. In zo'n activity diagram zijn de diverse scenario's gemakkelijk te onderscheiden.
- *Methode*. Een methode van een klasse met veel beslismomenten en conditionele handelingen is goed te modelleren in een activity diagram.

Een activity diagram bestaat uit een aaneenschakeling van *activity nodes* (activiteiten). Iedere activity node modelleert een deel van het gedrag van het diagram. Er zijn diverse soorten activity nodes. Het modelleren van gedrag wordt wel *control flow* genoemd. De control flow wordt bewerkstelligd doordat er *transities* bestaan tussen de verschillende activity nodes. Voor het modelleren van use cases zijn de volgende nodes van belang:

- *Activity*. Een activity is de eenheid van gedrag in een activity diagram.
- *Initial node*. De control flow start in een initial node. Anders dan in eerdere versies van UML kan een activity diagram meerdere initial nodes tellen.
- *Activity final node*. De control flow eindigt in een activity final node.
- *Decision node*. Een decision node markeert een beslismoment. In een decision node wordt een keuze gemaakt voor één van de uitgaande transities.
- *Merge node*. In een merge node komen verschillende transities samen. Een merge node kent slechts één uitgaande transitie. Eventueel kunnen desicion nodes en merge nodes worden gecombineerd.
- *Loop node*. Een loop node modelleert het herhaald uitvoeren van een aantal activity nodes.

Deze activity nodes zijn voldoende om het gedrag in het stappenplan van een use case te modelleren. Behalve gedrag modelleert een activity diagram ook het bewerken van objecten en gegevens. Dit heet *object flow*. Ook hiervoor zijn diverse nodes beschikbaar. Het modelleren van object flow valt echter buiten het bereik van dit boek.

Van use case naar activity diagram

Dit hoofdstuk laat het gebruik van het activity diagram zien voor het modelleren van het stappenplan van een use case. Het stappenplan van een use case is rechtstreeks om te zetten in één enkel activity diagram. Dit diagram bevat de stappen uit het stappenplan en weerspiegelt zo de volgordelijkheid van deze stappen. Bovendien geeft het de zo belangrijke beslismomenten grafisch weer. Onvolkomenheden in het stappenplan komen zo snel aan het licht.
Het stappenplan kent een beperkt aantal elementen. Het bestaat uit afzonderlijke stappen, die ombeurten worden uitgevoerd door een actor en de applicatie. Tussen de stappen door zijn er beslismomenten. Deze zijn bepalend voor de uitkomst van de use case. Soms wordt de beoogde uitkomst behaald; soms wordt een faal- of een herstelscenario uitgevoerd.

Het omzetten van het stappenplan naar een activity diagram is rechttoe-rechtaan. De start van het stappenplan is ook de start van het activity diagram. Iedere stap in het stappenplan is een activity. Het uitvoeren van een volgende stap representeert een transitie naar een volgende activity node. Een beslismoment in het stappenplan correspondeert met een decision node in het activity diagram. Vanuit een beslismoment vindt één uit meerdere transities plaats, afhankelijk van de uitkomst van het beslismoment. Zelfs de herhalingen in een stappenplan zijn weer te geven als loop nodes. Ook de uitkomsten van de use case zijn vertegenwoordigd. Het zijn de final nodes in het activity diagram. In tabel 5 is het omzetten van een stappenplan van een use case naar een activity diagram vormgegeven.

Stappenplan	**Activity diagram**	**Opmerkingen**
Start	Initial node	
Stap	Activity	
Beslismoment	Decision node	Een decision node kent één ingaande en meerdere uitgaande transities.
Herhaling	Loop node	Herhaling.
Uitkomst	Activity final node	Iedere uitkomst mondt uit in een final node.
Postconditie		Postcondities worden waar bij het bereiken van een van de final nodes.
Scenario	Control flow	Het doorlopen van het activity diagram van de initial node naar een van de final nodes.

tabel 5 – Van stappenplan naar activity diagram

Initial nodes

In eerdere versies van UML startte ieder activity diagram met één startpunt. Dit werd wel de *start state* genoemd. Maar altijd precies één startpunt. Niet meer en niet minder. Een activity diagram kan in UML 2.0 nul, één of meer initial nodes hebben. Wanneer een activity diagram een use case modelleert, neem dan altijd precies één initial node op. Iedere keer dat het activity diagram wordt doorlopen, begint dit vanaf deze node. Een initial node is weergegeven als een dichte bol. Plaats deze boven in het diagram. Het activity diagram leest het prettigst van boven naar beneden.

Tip 63. Plaats één enkele initial node in het activity diagram.

De uitvoering van het stappenplan start hier. Vanaf de initial node vertrekt een eerste transitie naar de eerstvolgende activity node. Dit is de eerste stap in het stappenplan.

Stappen en activities

Iedere stap in het stappenplan correspondeert met een activity node in het activity diagram. Zo'n activity node is ofwel een *control node*, zoals de decision node of de loop node, ofwel een *activity*. Een activity diagram modelleert primair een opeenvolging van activities, waarin mogelijk condities en herhalingen voorkomen. Modelleer daarom alle stappen in het stappenplan die geen conditie valideren als een activity. Een activity wordt in UML weergegeven als een rechthoek met afgeronde hoeken. De rechthoek bevat de beschrijving van de activity.

Tip 64. Modelleer een activity voor iedere stap in het stappenplan die geen conditie valideert.

Modelleer vervolgens een transitie van een activity naar de eerstvolgende activity, als de corresponderende stappen elkaar direct opvolgen in het stappenplan. Geef een transitie weer als een pijl die wijst in de richting van de volgende activity.

Tip 65. Modelleer transities tussen activities waar de corresponderende stappen elkaar direct opvolgen in het stappenplan.

Het is gebruikelijk om vanaf de initial node een transitie te laten vertrekken naar de eerste activity node in het activity diagram. Meestal zal dit een activity zijn, maar soms ook een decision node. Modelleer nooit meerdere transities vanaf de initial node.

Tip 66. Modelleer één enkele transitie vanaf de initial node naar de eerste activity node.

Tijd voor een voorbeeld. Hieronder is use case **Inzien Profiel** weergegeven.

Use case
Inzien profiel

Doel
Abonnee zoekt een passend profiel.
Bezoeker onderzoekt een mogelijk abonnement..

Stappenplan
1. Voer uit **Zoeken Profiel**.
2. Als actor heeft geannuleerd.
2.1 Stop.
3. Ophalen profiel.

4. Ophalen huidige abonnee.
5. Toon webpagina.
6. Actor initieert actie.
7. Als Abonnee inzien voorkeuren initieert.
7.1 Voer uit **Inzien Voorkeuren**.
8. Als Abonnee inzien foto initieert.
8.1 Voer uit **Inzien Foto**.
9. Als Abonnee versturen bericht initieert.
9.1 Voer uit **Versturen Bericht**.

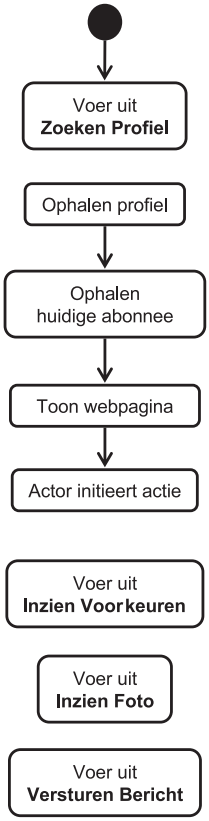
Het stappenplan van **Inzien Profiel** kent genoeg stappen die geen conditie valideren. Dit zijn de stappen **1**, **2.1**, **3**, **4**, **5**, **6**, **7.1**, **8.1** en **9.1**. Met uitzondering van stap **2.1** correspondeert ieder van deze stappen met een activity in het activity diagram. Stap **2.1** geeft één van de final nodes van het activity diagram weer. Final nodes komen later in dit hoofdstuk aan de orde in paragraaf *Final nodes*. Het eerste rudimentaire activity diagram voor **Inzien Profiel** is weergegeven in afbeelding 1.
Modelleer een activity diagram liefst van boven naar beneden. De initial node is boven in het diagram geplaatst. Er is een transitie gemodelleerd vanuit de initial node naar stap **1** uit het stappenplan. Stappen **3**, **4**, **5** en **6** volgen elkaar direct op. Ook hier zijn transities gemodelleerd. Iedere activity start pas als de vorige geheel is afgerond. Als de activity **Ophalen profiel** is afgerond, start **Ophalen huidige abonnee**. Nu de stappen die geen conditie valideren zijn overgenomen, krijgen ook de beslismomenten uit het stappenplan een plaats in het ontluikende activity diagram.

Beslismomenten en decision nodes

Zelfs in de rudimentaire opeenvolging van activities uit afbeelding 31 is het gewenste scenario van use case **Inzien Profiel** snel te herkennen. Dit is geen toeval. Zowel in het stappenplan als in het activity diagram geldt dat het gewenste scenario makkelijk te herkennen moet zijn. Geef het gewenste scenario op de hoofdas van het activity diagram weer. Recht van boven naar beneden.

Tip 67. Modelleer het gewenste scenario in het activity diagram recht van boven naar beneden.

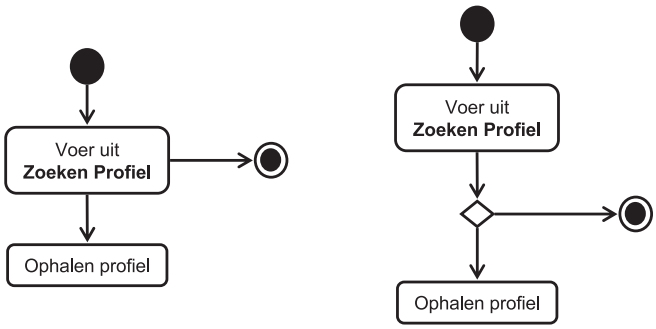
Het activity diagram in afbeelding 31 is echter nog verre van volledig. Het stappenplan kent diverse uitzonderingen op het gewenste scenario. Iedere uitzondering komt voort uit een conditionele stap in het stappenplan. Beschouw in het stappenplan van **Inzien Profiel** stappen **2** en **2.1** maar. Als de actor geen passend profiel selecteert, wordt er ook geen profiel getoond. Iedere conditionele stap leidt tot een decision node in het activity diagram.



afbeelding 31 – Rudimentair activity diagram

Tip 68. Modelleer iedere conditionele stap in het stappenplan als een decision node in het activity diagram.

Vanuit deze decision node vertrekken tenminste twee, maar mogelijk meerdere transities naar volgende activity nodes. Aan de hand van de condities uit het stappenplan wordt bepaald met welke transitie het pad wordt vervolgd. Dit is er altijd precies één. UML voorziet in twee notaties voor het noteren van beslismomenten. Beide zijn weergegeven in afbeelding 32.
In de linkernotatie vertrekken de transities direct vanuit de activity. In de rechternotatie is een decision node toegevoegd aan het diagram. De desicion node is gerepresenteerd als een ruit. De transities vertrekken nu vanuit de decision node. Qua betekenis zijn beide notaties gelijk. De decision node biedt echter een duidelijk voordeel ten opzichte van de intrinsieke notatie. Alhoewel een decision node iets meer ruimte inneemt in het activity diagram, is wel in één oogopslag te zien waar zich splitsingen voordoen.



afbeelding 32 – Twee notaties voor een beslismoment

Tip 69. Geef ieder beslismoment in het activity diagram weer als een decision node.

Zeker in complexe activity diagrammen geven decision nodes meer inzicht. Ook bij het vaststellen van de testscenario's bij een use case op basis van dit activity diagram spelen de decision nodes een rol (zie het hoofdstuk *Testscenario's en testgevallen*).

Enkelvoudige beslismomenten

Bij het modelleren van een beslismoment blijft de decision node leeg. Om aan te geven welke transitie wordt gekozen, worden er in het activity diagram condities bij de transities genoteerd. Deze condities heten *guards*. Modelleer guards tussen rechte haken.
Vanuit een beslismoment wordt altijd precies één transitie gekozen. Het is onmogelijk twee transities tegelijk te bewandelen. Evenzo kan het niet zo zijn dat geen van de transities wordt gekozen. De transities vanuit een decision node dekken samen alle mogelijke overgangen af en sluiten elkaar uit.

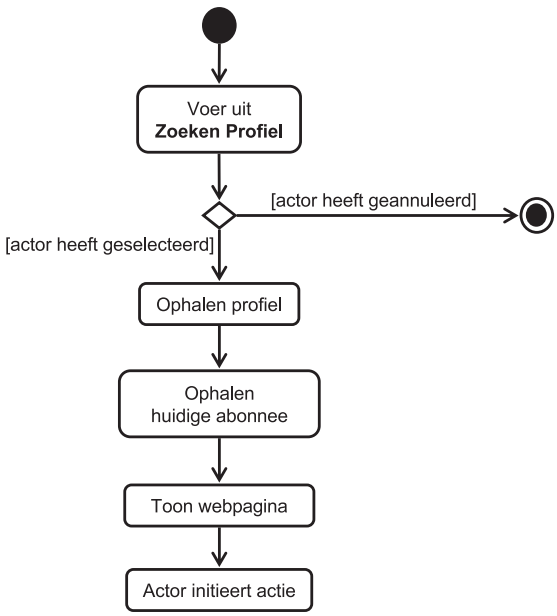
Tip 70. Noteer guards bij de transities vanuit een decision node. Deze guards zijn volledig en sluiten elkaar uit.

Bij het omzetten van het stappenplan naar een activity diagram is zo snel te zien of het stappenplan volledig is. Stappen **2** en **2.1** uit het stappenplan van **Inzien Profiel** beschrijven samen met stap **3** zo'n beslismoment.

Use case
Inzien profiel

Stappenplan
1. Voer uit **Zoeken Profiel**.
2. Als actor heeft geannuleerd.
2.1 Stop.
3. Ophalen profiel.
…

Na het uitvoeren van **Zoeken Profiel** in stap **1** heeft de actor ofwel een profiel gevonden, ofwel geannuleerd. In het eerste geval gaat het activity diagram met stap **3** verder, in het tweede geval stopt het bij stap **2**. Deze situatie is gemodelleerd in afbeelding 33.



afbeelding 33 – Enkelvoudig beslismoment

De transities in deze afbeelding sluiten elkaar duidelijk uit. Wees hier nauwkeurig in! Vanuit **Voer uit Zoeken Profiel** vindt dus altijd precies één transitie plaats. Formuleer guards altijd eenduidig en kwantificeerbaar. Dit is vergelijkbaar met het formuleren van pre- en postcondities bij een use case. Formuleer ze als een eenvoudige stelling. De guard is waar of niet niet waar. Iedere guard straalt zo duidelijk uit of er aan wordt voldaan of niet.

Tip 71. Formuleer een guard eenduidig en kwantificeerbaar, als eenvoudige stelling.

Samengestelde beslismomenten

In een data flow diagram wordt de conditie in de ruit genoteerd. Transities kennen er geen guards. In een data flow diagram kent een beslismoment *altijd* twee transities. Aan de conditie wordt wel of niet voldaan. Ja of nee. Dit markeert een voornaam verschil tussen een data flow diagram en een activity diagram. Een activity diagram kan meerdere transities bij een decision node bevatten. Dit kan omdat de conditie niet in de ruit staat, maar als guard bij de transities is opgenomen. Een activity diagram is dan ook compacter dan een data flow diagram. Ook hiervoor biedt **Inzien Profiel** een passend voorbeeld.

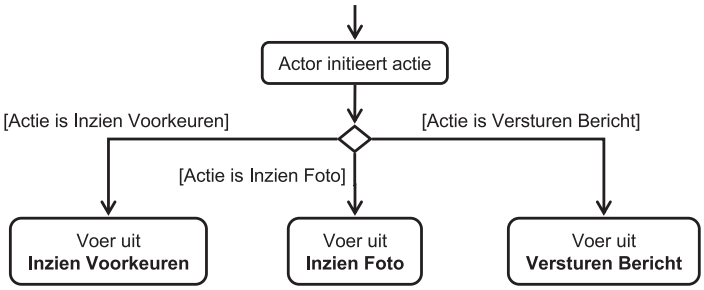
Use case
Inzien profiel

Stappenplan
…
6. Actor initieert actie.
7. Als Abonnee inzien voorkeuren initieert.
7.1 Voer uit **Inzien Voorkeuren**.
8. Als Abonnee inzien foto initieert.
8.1 Voer uit **Inzien Foto**.
9. Als Abonnee versturen bericht initieert.
9.1 Voer uit **Versturen Bericht**.

Dit deel van het stappenplan kent drie beslismoment na elkaar. Aan deze beslismomenten valt iets op. Ze valideren dezelfde conditie. Combineer beslismomenten die na elkaar dezelfde conditie valideren tot één enkel beslismoment. Modelleer dit beslismoment ook weer als een decision node met één inkomende transitie en meerdere uitgaande.

Tip 72. Breng beslismomenten in het stappenplan die na elkaar dezelfde conditie testen samen in één enkele decision node in het activity diagram.

In het geval van **Inzien Profiel** betekent dit een decision node met drie uitgaande transities, zoals is gemodelleerd in afbeelding 34.
In afbeelding 34 staan de drie activities naast elkaar. Bij een gering aantal is dit het meest overzichtelijk. Geef de activities eventueel onder elkaar weer, als er een groot aantal is. Het activity diagram kent immers geen notie van tijd. De locatie van een activity node in een diagram geeft niet aan wanneer deze wordt uitgevoerd. De tijdsduur van activity nodes kan bovendien per keer verschillen.



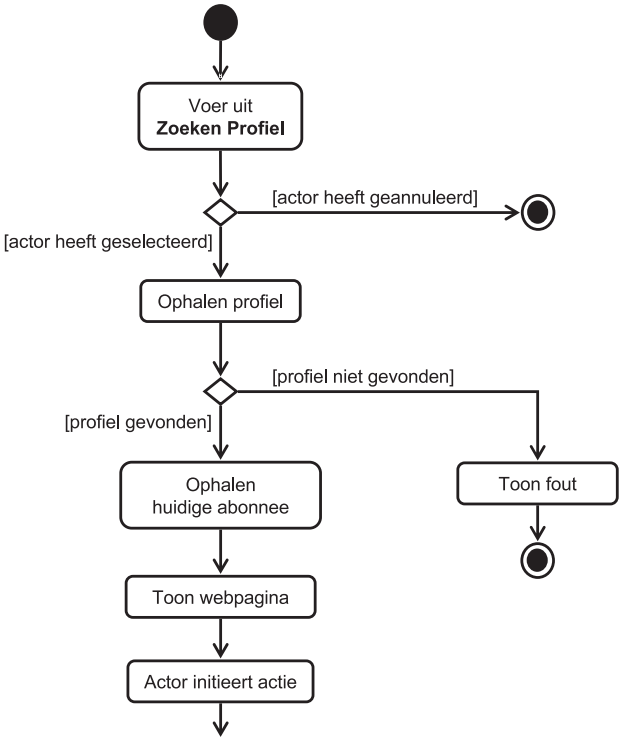
afbeelding 34 – Samengesteld beslismoment

Activities valideren

Bij het opstellen van het activity diagram blijken er dikwijls meer scenario's te zijn dan er uit het stappenplan van de use case bleek. In theorie kan iedere activity in het activity diagram een nieuw beslismoment inleiden. Vrijwel iedere activity kan immers goed of fout gaan. Ga dus bij iedere activity in het diagram de mogelijke uitkomsten na. Kan de activity fout gaan? Zijn er meerdere mogelijke uitkomsten?
Een simpel voorbeeld? Ga activity **Ophalen profiel** uit use case **Inzien Profiel** maar na. Wie garandeert dat deze slaagt? Misschien is de database wel uit de lucht. Wellicht reageert de achterliggende web service niet. Het al dan niet slagen van een activity heeft direct invloed op het verloop in het activity diagram. Moet er in dit voorbeeld een fout worden getoond aan de actor? Neem een nieuwe decision node op als de uitkomst van een activity van belang is voor het verdere verloop van het diagram. Plaats deze nieuwe decision node direct onder de te valideren activity. Valideer hiermee de mogelijke uitkomsten van de activity.

> *Tip 73.* Neem een nieuwe decision node op als een activity meerdere mogelijke uitkomsten heeft, die van belang zijn voor het uitvoeren van de use case.

Als het in use case **Inzien Profiel** niet lukt het geselecteeerde profiel op te halen in activity **Ophalen Profiel**, ontstaat er een faalscenario. Er wordt een fout getoond en **Inzien Profiel** wordt gestaakt. Deze situatie is afgebeeld in afbeelding 35. Hier zijn de mogelijke uitkomsten van activity **Ophalen Profiel** ondervangen in een decision node.
Deze werkwijze leidt tot een volledig en eenduidig activity diagram. Er resteert één uitdaging. Sec gezien heeft vrijwel iedere activity mogelijk meerdere uitkomsten. Zelfs een op het oog triviale activity als **Toon webpagina** kan misgaan. Bereikt de webpagina de browser wel? Als voor iedere activity een decision node wordt toegevoegd aan het activity diagram, volgt een explosie van scenario's. Allemaal heel correct, maar ieder scenario levert uiteindelijk ook een testscenario en testgevallen. Welk detailniveau is wenselijk?



afbeelding 35 – Activity Ophalen Profiel gevalideerd

Helaas heb ik op deze vraag geen pasklaar antwoord. Het detailniveau verschilt van situatie tot situatie, en van project tot project. Onthoud dat functionele beslismomenten altijd van belang zijn, en technische beslismomenten niet altijd. Ook de complexiteit en de fouttolerantie van een applicatie zijn van belang. Voor de ledenadministratie van een lokale voetbalvereniging geldt een heel ander detailniveau dan voor het online verwerken van transacties door een internationale bank. Alistair Cockburn geeft aan dat naarmate projecten groter worden en requirements complexer er meer *ceremonie* wordt toegevoegd aan de werkwijze van het project en aan de precisie van de diagrammen [Cockburn-02]. Modelleren is steeds een afweging tussen voortgang en ceremonie. Tussen functionaliteit en doorlooptijd.

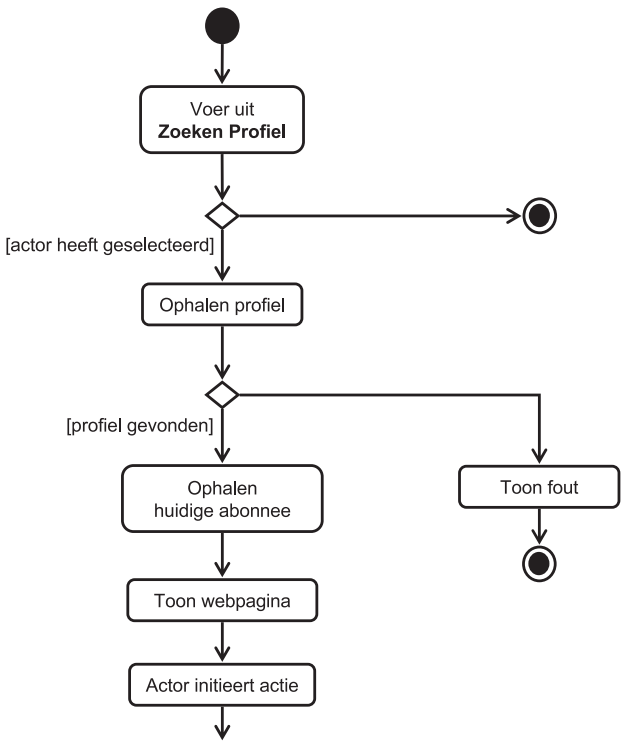
Uitzonderingen

Als er diverse transities zijn vanuit een decision node is niet altijd goed te zien of deze volledig zijn en elkaar uitsluiten. In veel gevallen is het niet opportuun alle mogelijke uitkomsten als individuele transitie te modelleren. Dit vervuilt het activity diagram. UML voorziet bij dit soort gevallen in een eenvoudige notatie. Voorzie één van de transities van een guard die wordt aangeduid met **else**. Als de uitkomst van het beslismoment aan geen van de overige, concrete

guards voldoet, wordt automatisch de transitie **else** genomen. De transitie **else** voorkomt een wirwar aan transities die nogal eens puur voor de volledigheid.worden toegevoegd De guard **else** voorkomt zo een hoop overbodige scenario's.

Tip 74. Neem een transitie met de guard **else** op als de transities vanuit een decision node niet volledig zijn, maar er geen andere betekenisvolle transities zijn te definiëren.

Zelf modelleer ik meestal wel de transitie, maar niet de guard **else**. Deze veronderstel ik impliciet. Het activity diagram is al vol genoeg. Als ik dit doe zorg ik er wel voor dat de guards van het gewenste scenario in elk geval zijn gemodelleerd. De **else**-transities modelleren zo altijd de uitzonderingen. Dit garandeert dat het gewenste scenario altijd correct wordt uitgevoerd.



afbeelding 36 – Impliciete [else] transities

Tip 75. Modelleer de guards bij de transities van het gewenste scenario altijd. Gebruik transities met **else** altijd voor uitzonderingen.

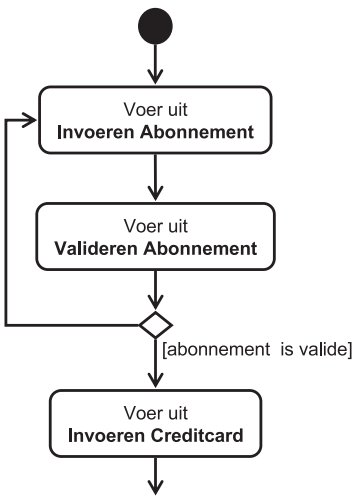
In afbeelding 36 is het geoptimaliseerde activity diagram voor **Inzien Profiel** getoond.

Beide beslismomenten in dit activity diagram zijn vereenvoudigd door het gebruik van impliciete **else**-transities. Het gewenste scenario leest nog steeds van boven naar beneden. De hierbijbehorende guards zijn expliciet gedefinieerd.

Herhaling

Soms is het nodig een of meer activity nodes te herhalen in een activity diagram. Een actor krijgt bijvoorbeeld drie pogingen om in te loggen. Lukt het de eerste keer niet, dan kan nog een poging worden ondernomen totdat de actor correct inlogt of drie keer foutief heeft ingelogd. Het activity diagram is bij uitstek geschikt voor het visualiseren van herhalingen. Een herhaling is te modelleren als een loop node die twee activity nodes met elkaar verbindt middels een transitie. Deze transitie heeft vaak de tegenovergestelde richting van de andere transities in het diagram. Loop nodes vallen zo direct op in het activity diagram.

Tip 76. Modelleer een herhaling in een stappenplan als een loop node in een activity diagram.



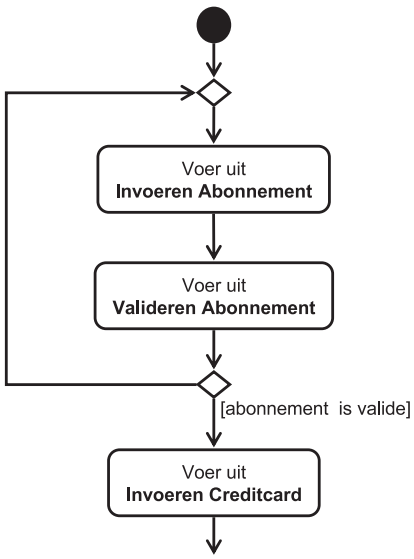
afbeelding 37 – Herhaling

Ook bij het aanvragen van een abonnement kan het een en ander misgaan. Veronderstel dat use case **Valideren Abonnement** een negatieve uitkomst heeft, bijvoorbeeld omdat het opgegeven e-mailadres bij een reeds bestaand abonnement hoort. De use case **Invoeren Abonnement** wordt nu opnieuw uitgevoerd, waarbij de reden van afwijzen wordt getoond. Dit stelt de actor in staat zijn aanvraag te verbeteren. Het activity diagram van **Aanvragen Abonnement** in afbeelding 37 spreekt boekdelen.

Een loop node impliceert een transitie. Deze wijst net als een reguliere transitie naar de eerstvolgende uit te voeren activity node. Hier is dat activity **Voer uit Invoeren Abonnement**.

Herhaling met merge node

Er is een tweede notatie voor het modelleren van herhalingen. De herhaling wijst nu niet direct naar de eerstvolgende activity node, zoals in afbeelding 37, maar naar een merge node. Een merge node heeft net als een decision node de vorm van een ruit. Vanuit de merge node vertrekt één transitie naar de eerstvolgende activity node. In afbeelding 38 is opnieuw het activity diagram voor **Aanvragen Abonnement** getoond, nu met merge node.



afbeelding 38 – Herhaling met verzamelpunt

In dit diagram komt de herhaling uit in de merge node boven de eerstvolgende activity **Invoeren Abonnement**. De notaties uit afbeelding 37 en afbeelding 38 verschillen niet van betekenis. Het gebruik van een merge node in een herhaling maakt het activity diagram wel duidelijker. Onmiddelijk is zo te zien welke activity nodes binnen de herhaling vallen. Dit is vooral praktisch als het diagram meerdere herhalingen kent die bijvoorbeeld bij dezelfde activity node uitkomen.

Tip 77. Sluit iedere herhaling aan op een merge node.

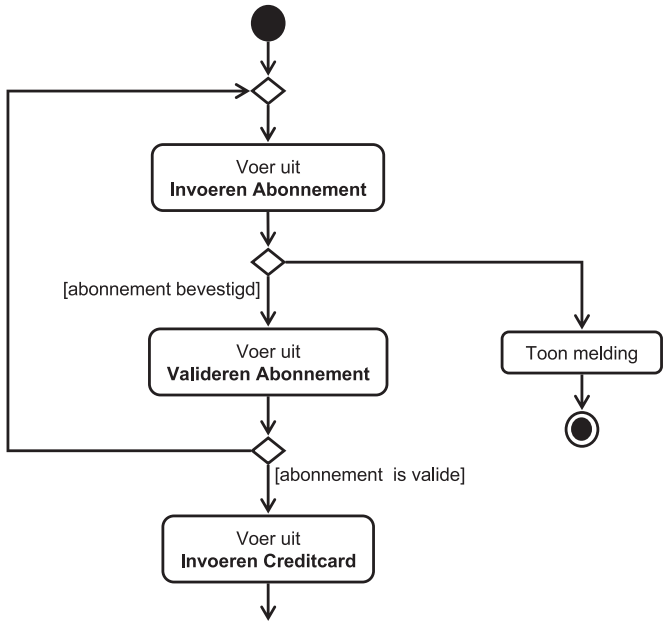
Naast de verbeterde inzichtelijkheid van het diagram biedt de merge node nog een voordeel. Wanneer er een guard geldt bij het uitvoeren van de eerste activity node binnen de herhaling, dan krijgt deze een plaats bij de uitgaande transitie van de merge node. Zo is gegarandeerd dat de guard onder alle omstandigheden is getest. In de notatie uit afbeelding 37 is dit niet het geval.
Er is nog een reden voor het hanteren van merges nodes. Samen met de initial node en de final nodes delen de decision nodes en merge nodes het activity diagram op in deelpaden. Dit is een van de stappen die gebruikt wordt om alle scenario's bij een use case te vinden. Dit onderwerp wordt behandeld in het hoofdstuk *Testscenario's en testgevallen*.
Er is overigens een verschil tussen een beslismoment en een verzamelpunt. Een verzamelpunt kent meerdere inkomende transities en maar één uitgaande. Een beslismoment kent eveneens mogelijk meerdere inkomende transities, maar telt minimaal twee uitgaande transities.

Herhaling onderbreken

Net als in code is het ook in een activity diagram van belang dat iedere herhaling wordt onderbroken. Het activity diagram bij een use case mag geen oneindige herhaling bevatten. Zorg ervoor dat iedere herhaling tenminste één decision node bevat die de herhaling onderbreekt.

Tip 78. Doorbreek iedere herhaling met minimaal één decision node.

Het activity diagram voor use case **Aanvragen Abonnement** in afbeelding 38 bevat zo'n decision node. De herhaling wordt onderbroken als het abonnement valide is. Toch geeft deze oplossing geen bevredigend gevoel. Wat als de actor er niet in slaagt om een valide abonnement in te voeren? Er ontbreekt iets. Ook activity **Voer uit Invoeren Abonnement** heeft mogelijk meerdere uitkomsten. De actor bevestigt het abonnement of annuleert. Aha! Het activity diagram in afbeelding 38 valideert de uitkomst van activity **Invoeren Abonnement** niet. Net wat er ontbreekt. Het nieuwe activity diagram is gemodelleerd in afbeelding 39.
Nu is de herhaling in elk geval onderbroken. Modelleer uitgaande transities en guards bij onderbrekende beslismomenten zodanig dat het gewenste scenario nog steeds van boven naar beneden is te volgen, zoals in afbeelding 39.



afbeelding 39 – Herhaling onderbroken

Interactie onderbreken

Het enige onverwachte dat ons nu nog kan gebeuren is dat de actor de use case lukraak beëindigt. In webapplicaties komt dit zeker voor. Zo nu en dan boek ik een vliegticket via internet. Door schade en schande heb ik geleerd om daarbij niet op **Back** te klikken in de browser. Wat er ook gebeurt. Het gebeurt maar zelden dat webapplicaties **Back** fatsoenlijk afvangen. De meeste webapplicaties raken hier danig van in de war. Het afvangen van **Back** is een boeiende niet-functionele requirement.
In een webapplicatie kan sowieso elke interactie tussen actor en applicatie op ieder moment worden onderbroken. Deze situatie is vrijwel onmogelijk te modelleren. Dit zou betekenen dat bij iedere activity een decision node moet zijn opgenomen die dit onderbreken afvangt.

> *Tip 79.* Modelleer het op ieder willekeurig moment kunnen onderbreken van de interactie niet in het activity diagram.

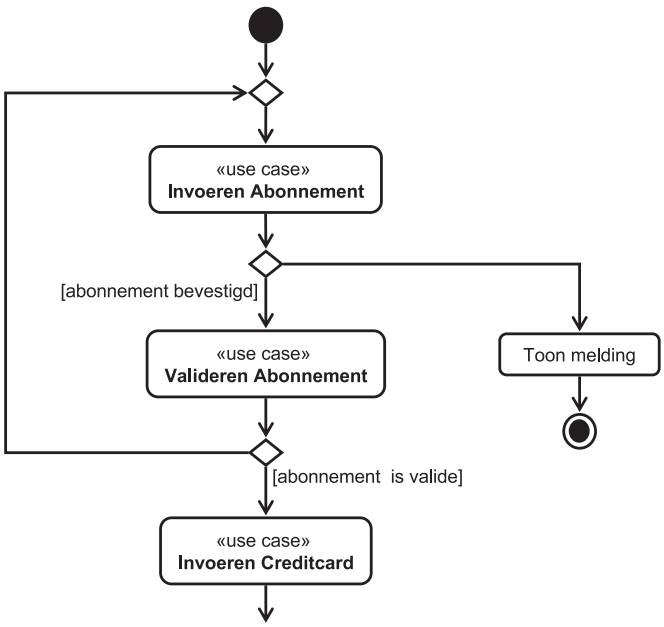
Use cases en stereotypen

Er zijn stappen in het stappenplan van een use case die het uitvoeren van een andere use case beschrijven. Dit geldt vooral primaire use cases, die gebruik maken van een of meer secundaire use cases. Het geldt ook voor secundaire use cases die op hun beurt andere secundaire use cases uitvoeren. Elke stap die een

andere use case aanroept, is al als activity opgenomen in het activity diagram. Deze activity representeert in feite het uitvoeren van de andere use case. Dit moet herkenbaar zijn in het activity diagram. Geef zo'n activity altijd de naam van de use case die wordt uitgevoerd. Nog duidelijker is het als de activity ook is voorzien van een kenmerkend stereotype. Gebruik hiervoor het stereotype **«use case»**.

Tip 80. Geef een activity die een use case uitvoert de naam van deze use case. Voorzie de activity van het stereotype **«use case»**.

Het activity diagram voor **Aanvragen Abonnement** is opnieuw weergegeven in afbeelding 40.



afbeelding 40 – Use cases als activity, inclusief stereotype

Nog zo'n stereotype dat ik regelmatig gebruik in activity diagrammen is **«form»**. Hiermee geef ik aan dat een activiteit een scherm of een webpagina toont. Ik geef de activity dan de naam van het te tonen scherm of webpagina.

Tip 81. Geef een activity die een scherm toont de naam van dit scherm, en voorzie de activity van stereotype **«form»**.

Bij het gebruik van een modelleeromgeving kan een diagram niet altijd naar wens worden vormgegeven. Elke modelleeromgeving kent zo zijn eigen wijze van implementeren van UML. Het kan maar zo zijn dat uw modelleeromge-

ving geen stereotypen kent bij activities. Wees dan creatief. Oormerk een activity-als-use-case of een activity-als-form bijvoorbeeld door een sterretje achter de naam van de activity te plaatsen of door haken om de de naam van de use case of het scherm te plaatsen.

Postcondities valideren

Zoals het voorbeeld in afbeelding 40 laat zien, is het voor sommige activities zinvol de mogelijke uitkomsten te toetsen. Ingeval een use case wordt uitgevoerd als activity is dit zeker belangrijk. Voor de meeste use cases zijn immers meerdere uitkomsten mogelijk. En deze zijn zeker van belang. De mogelijke uitkomsten van een use case zijn beschreven in de postcondities. Valideer na het uitvoeren van een use case altijd de bijbehorende postcondities.

Tip 82. Valideer in het activity diagram altijd de postcondities van use cases die als activity worden uitgevoerd.

Vanuit een use case die slechts één enkele postconditie waarmaakt, vertrekt ook maar één transitie. Vanuit een use case die meerdere postcondities kent, volgen waarschijnlijk meerdere transities. Neem een nieuwe decision node op in het activity diagram. Plaats deze direct na de uitvoering van de activity-als-use-case. Neem evenveel transities op vanuit deze decision node als er postcondities voor de uitgevoerde use case gelden. Plaats ieder van de postcondities als guard bij een van deze transities.

Tip 83. Neem direct na het uitvoeren van een use case in een activity een decision node op. Modelleer vanuit deze decision node een transitie voor ieder van de postcondities. Gebruik deze postcondities als guards.

De use case **Aanvragen Abonnement** gebruikt verschillende andere use cases, waaronder **Invoeren Abonnement** en **Valideren Abonnement**. De postcondities van **Invoeren Abonnement** zijn hieronder beschreven.

Use case
Invoeren Abonnement

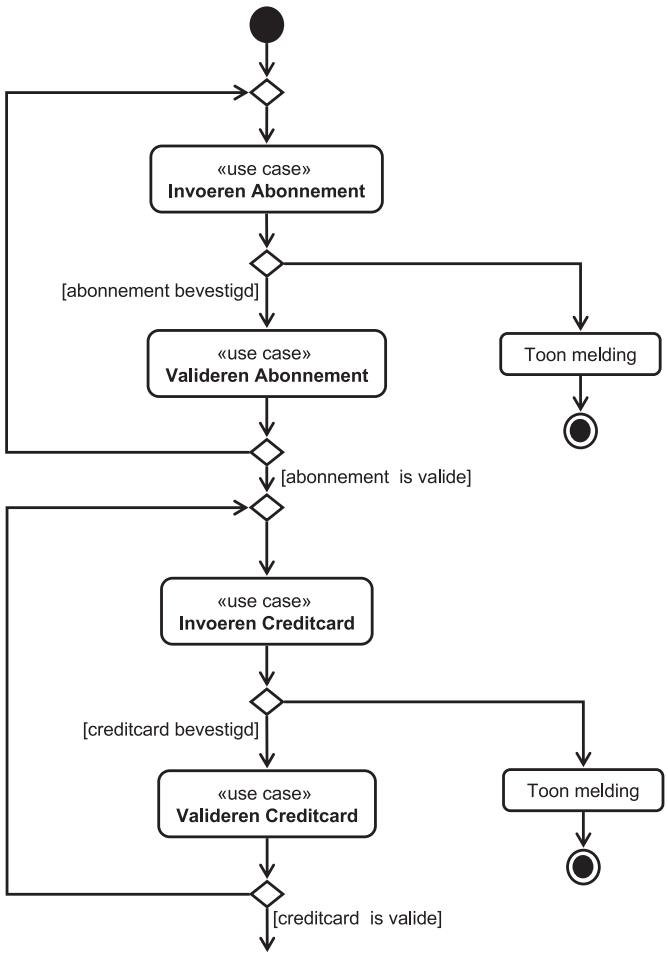
Postcondities
Abonnement is bevestigd en opgeslagen of
Abonnement is geannuleerd.

Als de actor tijdens **Invoeren Abonnement** *annuleert, stopt ook* **Aanvragen Abonnement**. *Anders vervolgt het activity diagram met use case* **Valideren Abonnement**. *De postcondities van deze use case zijn hieronder beschreven.*

Use case
Valideren Abonnement

Postcondities
Abonnement is valide of
Abonnement bevat fouten.

Alleen als de postconditie (en guard) **abonnement is valide** waar is, vervolgt het activity diagram met het invoeren van de creditcard. Het bijbehorende activity diagram is weergegeven in afbeelding 41. De negatieve postcondities van de use cases zijn hier gemodelleerd als een **else**-transitie.

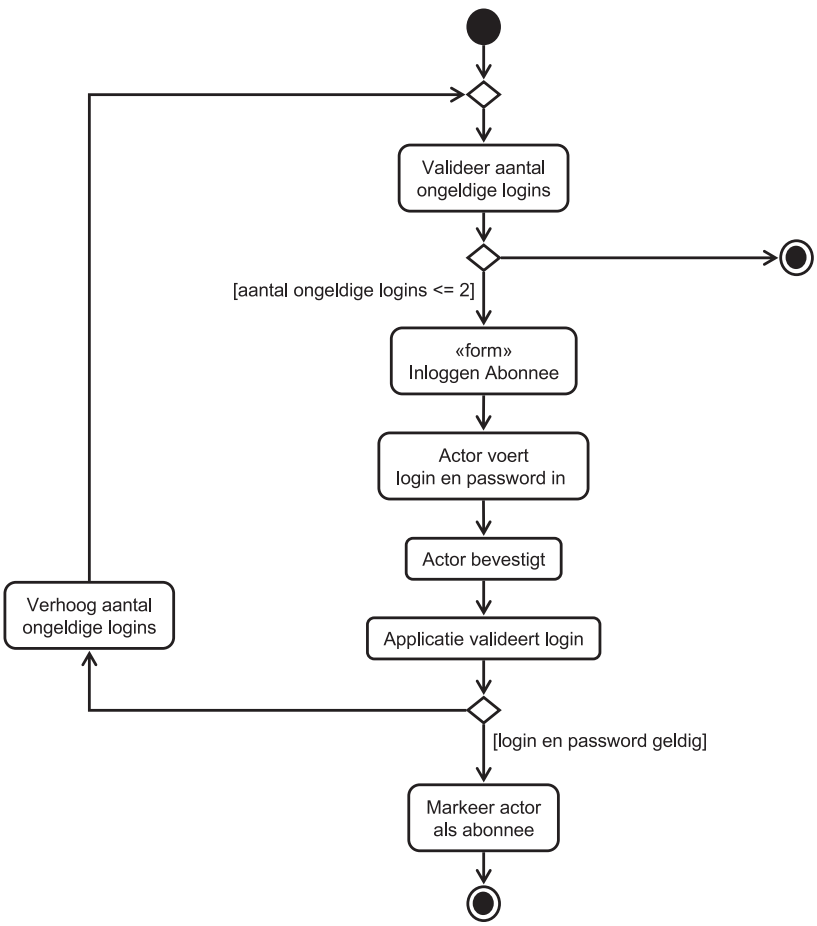


afbeelding 41 – Valideren van postcondities van use cases

Final nodes

Afhankelijk van de beslismomenten in een stappenplan zijn er verschillende scenario's mogelijk bij een use case. Ieder van deze scenario's leidt tot het beëindigen van de use case, of het nu het gewenste scenario, een faalscenario of een herstelscenario betreft. Kijk maar. Wanneer een bezoeker **Invoeren Abonnement** annuleert, stokt ook **Aanvragen Abonnement**. Dit leidt onherroepelijk tot een faalscenario.
Modelleer het beëindigen van een scenario door een activity final node op te nemen en een transitie te modelleren van de laatste gepasseerde activity node naar deze final node. Modelleer een activity final node als een bol met een open ring.

Tip 84. Beëindig ieder scenario in een activity final node. Modelleer een transitie van de laatste activity node naar deze activity final node.



afbeelding 42 – Activity final nodes

De use case **Inloggen Abonnee** van Dare2Date is een goed voorbeeld. **Inloggen Abonnee** scheidt abonnees van bezoekers. Het bijbehorende activity diagram is weergegeven in afbeelding 42.
Het activity diagram bij **Inloggen Abonnee** kent meerdere scenario's. In het gewenste scenario logt de actor correct in. Het faalscenario treedt in werking als de actor drie keer foutief inlogt. Er is een aantal herstelscenario's, namelijk als de actor er in twee of drie pogingen in slaagt correct in te loggen. Het activity diagram telt twee final nodes. De eerste wordt bereikt door alle faalscenario's. De tweede door het gewenste scenario en de herstelscenario's.

Postcondities en activity final nodes

De postcondities van een use case reflecteren direct het doel van de use case. Als een use case is uitgevoerd is altijd één van de postcondities waargemaakt. Ook het bijbehorende activity diagram maakt deze postcondities waar. Het activity diagram eindigt altijd in één van de activity final nodes. Hier aangekomen is dus altijd één van de postcondities waargemaakt. Koppel de postcondities aan de final nodes. Voeg de postconditie toe als tekst bij de bijpassende final node. De koppeling is niet een-op-een. Dezelfde postconditie kan worden bereikt in meerdere final nodes.

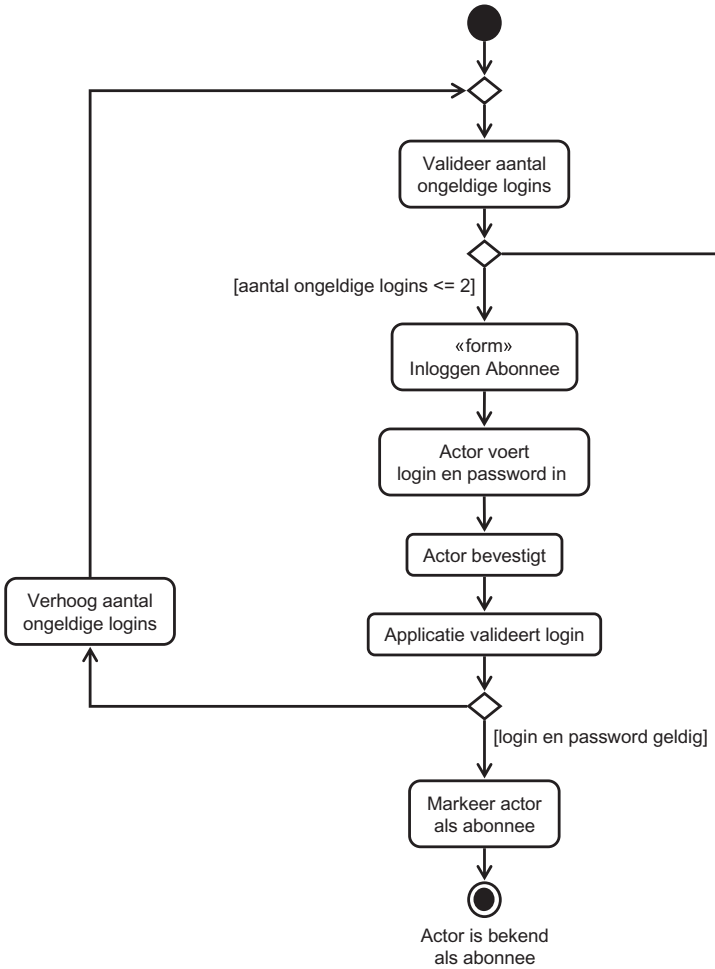
Tip 85. Noteer bij elke activity final node in het activity diagram de postconditie die er is waargemaakt.

De postcondities van **Inloggen Abonnee** zijn eenvoudig. De actor is bekend als abonnee of niet. Ze zijn hier weergegeven.

Use case
Inloggen Abonnee

Postcondities
Actor is bekend als abonnee of
Actor is niet bekend als abonnee

Het bijbehorende activity diagram is gemodelleerd in afbeelding 43, met de postcondities bij de final nodes geplaatst.
De postcondities van een use case zijn zo gemakkelijk te valideren. Ieder van de gedefinieerde postcondities moet bij minimaal één final node voorkomen. Het kan niet zo zijn dat een van de postcondities nooit wordt bereikt in het diagram. Er zijn tenminste net zo veel postcondities als er final nodes zijn.



afbeelding 43 – Postcondities bij de activity final nodes

Tip 86. Controleer of elke postconditie is waargemaakt in minimaal één activity final node.

Er kunnen postcondities ontbreken in het activity diagram. Indien dit het geval is, onderzoek dan of er te weinig postcondities zijn of teveel final nodes. Het is allebei mogelijk. Meestal ontbreken er postcondities bij de use case. Het activity diagram is vrijwel altijd volledi ger.

Tip 87. Neem nieuwe postcondities op als er activity final nodes zijn die geen van de bestaande postcondities van de use case waarmaken.

Niet alle modelleeromgevingen staan het toe tekst op te nemen bij een activity final node. Wees wederom creatief. Neem een *note* op bij elk van de de final nodes. Een note is een betekenisloos modelelement uit UML dat overal kan worden gebruikt. Noteer de postcondities in de notes.

Minder is meer

Naast de hier gebruikte modelelementen kent het activity diagram nog talrijke andere modelelementen. Deze zijn echter bij het modelleren van een activity diagram voor een use case niet van toepassing, maar worden gebruikt wanneer het activity diagram andere doeleinden dient. Zo zijn er nog verschillende andere control nodes, zoals fork nodes en join nodes. Hiermee is de parallelle uitvoering van verschillende flows te modelleren. Ook is het mogelijk de object flow te modelleren in het activity diagram. Als laatste noem ik het partitioneren van het activity diagram waarmee de activity nodes over verschillende verantwoordelijke actoren of bijvoorbeeld geografische locaties te verdelen. Alhoewel ieder van deze modelelementen van tijd tot tijd zijn nut bewijst, geldt hier een eenvoudige richtlijn. Don't use every notation in the book. Gebruik alleen het strikt noodzakelijke. Minder is altijd meer.

Fork nodes en join nodes

Fork nodes en join nodes worden in een activity diagram gebruikt om het parallel uitvoeren van meerdere flow te modelleren. Dit komt vooral voor in gedistribueerde applicaties, waarbij bijvoorbeeld tegelijkertijd op de client en op de server activiteiten plaatsvinden. Hier geldt overigens de wet van Martin Fowler over het ontwikkelen van gedistribueerde applicaties: *do not distribute your objects* [Fowler-02].
Een fork node geldt als de start voor de parallel uit te voeren flows. Een join node geeft aan dat alle parallelle flows gereed zijn. Beide modelelementen zijn gemodelleerd als een brede balk. Er leiden transities naartoe en er komen transities vanaf. Slechts bij hoge uitzondering kom ik fork nodes en join nodes tegen in activity diagrammen voor use cases.
Bij het modelleren van complexe bedrijfsprocessen zijn fork nodes en join nodes reuze handig. Om de doorlooptijd van een bedrijfsproces in te korten, worden vaak verschillende flows parallel uitgevoerd.

Een papieren proces
Een organisatie met vijfduizend verkooppunten optimaliseerde het wijzigen van basisgegevens. Nu vult een medewerker een papieren formulier in. Dit wordt door een logistieke afdeling gekopieerd en verspreid over de diverse afdelingen die hun goedkeuring moeten geven. Nadat alle kopieën weer terug zijn beland bij de logistieke afdeling, wordt de wijziging doorgegeven

aan een afdeling wier enige taak het is de wijzigingen in de diverse applicaties in te voeren. Pure bezigheidstherapie.

Dit parallellisme komt terug in het modelleren van de workflow. Doel? Het terugbrengen van de doorlooptijd van het wijzigen van gegeven van drie weken naar één dag. Fork nodes en join nodes komen hier goed van pas.

Als het activity diagram wordt gebruikt om een enkele use case te modelleren is het gebruik van forks en joins onwaarschijnlijk. De granulariteit van pragmatische use cases is niet van dien aard dat parallelle flows voorkomen.

Partitioneren

Een activity diagram kan worden gepartitioneerd als de verschillende activity nodes in groepen met gelijke karakteristieken zijn te splitsen. In vorige versies van UML kon een activity diagram in één dimensie worden gesplitst. Hiervoor was de term swimlane in gebruik. Swimlane refereert aan de banen in een zwembad. Een zwemmer blijft bij een zwemwedstrijd voortdurend in zijn eigen baan. Zo ook in het activity diagram.
In UML 2.0 kan een activity diagram in willekeurige groepen activity nodes worden gepartitioneerd. Dit maakt partitionering in meerdere dimensies mogelijk. Partities worden voorzien van een identificerende naam en markeren een gebied in het activity diagram, meestal gekaderd in een rechthoek. Partities kunnen bijvoorbeeld een matrix vormen. Een eendimensionale partitionering blijft echter het meest gebruikelijk.

Kriskras
Voor het modelleren van een activity diagram is een modelleeromgeving geen overbodige luxe. Vrijwel alle modelleeromgevingen kunnen partities introduceren in een activity diagram. Het ziet er verleidelijk uit. Ik betrap mezelf er regelmatig op graag eens partities te willen gebruiken. Toch botert het niet tussen mij en de partities.

Ervan overtuigd dat partities verhelderend zijn, introduceer ik ze maar weer eens. Met steeds opnieuw het gevolg dat het hele activity diagram op zijn kop staat in mijn modelleeromgeving. Activity nodes die eerst prettig gegroepeerd waren, verspreiden zich over het diagram. De omvang van het diagram neemt barbaarse vormen aan. Vooral in de breedte. De transities, eerder nog korte pijltjes, wijzen kriskras over het activity diagram en kruisen elkaar voortdurend. Dit komt de duidelijkheid niet ten goede. Zuchtend besluit ik ook dit keer maar weer geen partities te gebruiken.

Gebruik partitionering met mate. Alhoewel ze verhelderend kunnen zijn, dwingen partities het activity diagram ook in een stramien. Voorkom koste wat kost dat het diagram uitmondt in een wirwar van elkaar voortdurend kruisende transities.

Object flow

Vergis u niet. Een activity diagram kan ook worden gebruikt voor het modelleren van object flow. Een object node geeft het bestaan van een object weer in het activity diagram, gevisualiseerd als een rechthoek, met daarin de naam en de status van het object. Object flow wordt weergegeven als een onderbroken pijl van een activity node naar een object node en vice versa. Houd bij het modelleren van een activity diagram goed het gezichtspunt in het oog. Voor wie is het bedoeld? Waarom wordt het activity diagram hier opgesteld? Is mijn doelgroep geïnteresseerd in het modelleren van object flow? Ik waag een gokje? Vrijwel nooit.

Pragmatisch modelleren

Gebruik het opstellen van activity diagrammen bij use cases om de kwaliteit van de requirements te waarborgen. Modelleer deze activity diagrammen wederom als team, het liefst tijdens een workshop. Dit zet de neuzen van alle betrokkenen in dezelfde richting en benut de specifieke deskundigheid van de betrokkenen.

> ***Een andere kijk***
> *In Smart is een activity diagram bij een use case het uitgangspunt voor het opstellen van testscenario's en testgevallen voor deze use case. Zo'n activity diagram wordt bij voorkeur opgesteld tijdens een kleine workshop waaraan bijvoorbeeld gebruikers, ontwerpers, ontwikkelaars en vooral testers deelnemen. Meestal vinden deze workshops direct na de dagelijkse stand-up meeting plaats.*
>
> *Het is heel aardig om te zien dat ontwerpers en testers heel verschillend kijken naar het stappenplan van een use case. Ontwerpers zijn vooral geïnteresseerd in het gewenste scenario. Testers letten vooral op de uitzonderingen. Regelmatig word ik bij deze workshops door de testers terechtgewezen. "En wat gebeurt er nu als de gebruiker annuleert?" O ja, daar hadden we nog niet aan gedacht.*

Testers hebben doorgaans een heel andere kijk op requirements dan ontwerpers en ontwikkelaars. Laat testers daarom al in een vroeg stadium in een project participeren. Dit voorkomt fouten, zelfs nog voordat er één regel code is geschreven.