De onweerstaanbare verleiding van het maken van ge nerieke software

DE RULE OF THREE: LASTIG, MAAR NOODZAKELIJK

Het maken van generieke, herbruikbare software is verleidelijk, maar gevaarlijk. Wie de rule of three niet in acht neemt, krijgt al snel te maken met spectaculair falende software-ontwikkelprojecten. Het probleem is namelijk, zeggen Matthijs Maat en Gert Florijn, dat men denkt te weten wat generiek gemaakt kan worden en wat niet. Een vergissing.

door: MATTHIJS MAAT & GERT FLORIJN **beeld:** MARC KOLLE

IT mag dan een relatief jong vakgebied zijn, dat wil niet zeggen dat er geen waardevolle theorie en ervaring voorhanden zijn. Principes als encapsulatie en low-coupling/high-cohesion stammen al uit de vorige eeuw. Toen werd ook de basis gelegd voor iteratief ontwikkelen en continue kwaliteitsborging. Helaas maken we niet altijd gebruik van de lessen uit het verleden. Het bracht Fred Brooks ertoe zijn klassieker the mythical man-month (met daarin Brooks' law: Adding programmers to a late project makes it later) te vergelijken met de Bijbel: 'Everybody quotes it, some people read it, and a few people go by it.'

In dit artikel aandacht voor de rule of three. Deze regel stelt dat ervaring nodig is om een generieke toepassing of een raamwerk te maken. Meer specifiek, dat je tenminste drie concrete voorbeelden moet hebben gemaakt voordat je aan generalisatie begint. Als je dat niet doet, leidt de verleiding om generieke software te maken al snel tot spectaculair falende software-ontwikkelprojecten.

Hergebruik

Het streven naar hergebruik is zo oud als de softwareontwikkeling zelf. Meerdere keren hetzelfde ontwikkelen, dat kan niet efficiënt zijn, toch? Wat als we het nu één keer goed maken en dan overal hergebruiken? Dat moet leiden tot makkelijker en goedkoper aan te passen systemen.

Daarom is in de loop der jaren gezocht naar technieken voor hergebruik. Van modules en bibliotheken tot objectoriëntatie en daarop gebaseerde raamwerken om hergebruik van code mogelijk te maken. Van componenten en services tot multifunctionele systemen om diensten en zelfs hele systemen opnieuw te kunnen gebruiken.

Het is evident dat hergebruik kan werken. Er zijn talloze bibliotheken, raamwerken, platformen en totaaloplossingen die ontwikkelaars veel werk uit handen nemen. Om in deze tijd een complexe webapplicatie, mobiele toepassing, ERP of EPD van de grond af te ontwikkelen is op zijn zachtst gezegd ambitieus.

Dat betekent echter niet dat het maken van herbruikbare software simpel is. Zeker het ontwikkelen van een generieke totaaloplossing die kan worden gespecialiseerd/aangepast voor een specifieke situatie is zeer ingewikkeld. Er zijn tegenkrachten die de beoogde voordelen geheel teniet kunnen doen (zie kader).

Toch wordt dit vaak geprobeerd. We gaan meerdere toepassingen bouwen volgens hetzelfde stramien, dus beginnen we met het generieke deel – anders valt er alsnog niets her te gebruiken. Bovendien, hoe moeilijk kan het zijn? Een verzekering is toch een verzekering? Dus gaan we een generiek zaaksysteem, een generiek uitkeringssysteem, een generiek hypotheeksysteem of een generiek leningen- of toeslagensysteem bedenken dat geschikt is voor alle gevallen.

Het probleem

Precies die verleidelijke redenering brengt veel projecten in problemen. De onderliggende veronderstelling is namelijk dat je vooraf in staat bent te bepalen wat generiek gemaakt kan worden en wat niet. Zo precies, dat je er zelfs software voor kunt maken. En die aanname is, zo blijkt vaak, onterecht. 'No one is that smart,' zo stelden Don Roberts en Ralph Johnson het in 1996. En dat is te zien.

Een dergelijk traject kenmerkt zich vaak door een ellenlange ontwerpfase, omdat de neiging bestaat alle mogelijke situaties te willen voorzien en op te lossen. De conceptuele oplossing wordt mooier, groter en ambitieuzer, omdat er altijd iets te vinden is dat misschien óók nodig is. En ingewikkelder, omdat met allerlei mogelijke, en vaak onvoorziene, uitzonderingssituaties rekening moet worden gehouden. Omdat nooit te bewijzen is dat iets níét nodig is, ontbreekt een natuurlijke rem. En als er dan uiteindelijk daadwerkelijk systemen worden ontwikkeld blijkt altijd dat er alsnog dingen zijn waar niet aan gedacht was – maar al te vaak ook rond fundamentele aannames. Wat ogenschijnlijk gelijk leek, blijkt in de werkelijkheid fundamenteel verschillend. Ook het daadwerkelijk realiseren van dat generieke raamwerk en/of systeem heeft zo zijn problemen. Hoe ga je toetsen of het gerealiseerde voldoet, zonder specifieke toepassing? Wie accepteert wat is gebouwd en bepaalt of dat goed genoeg is? Van wie ís dat generieke stuk eigenlijk en hoe wordt het beheerd?

Zo ontstaan projecten die zich kenmerken door grote ambitie, veel vertrouwen en energie, heel veel inspanning, een lange doorlooptijd en weinig tastbaar resultaat. En dat zijn nu eenmaal projecten die de neiging hebben om stopgezet te worden. Inderdaad, voordat ooit van de beoogde voordelen geprofiteerd kan worden. En zelfs als de fase van oogsten wel bereikt wordt, dan valt die oogst vaak tegen. Omdat alsnog veel inspanning nodig blijkt om het generieke stuk 'af' te maken. Omdat eindgebruikers vooral de beperkingen zien van een oplossing die (net) niet dát biedt wat voor hen handig is. Of omdat iemand alle bestede uren bij elkaar optelt en zich afvraagt of het uiteindelijk niet gewoon goedkoper was geweest om verschillende systemen te ontwikkelen én te onderhouden.

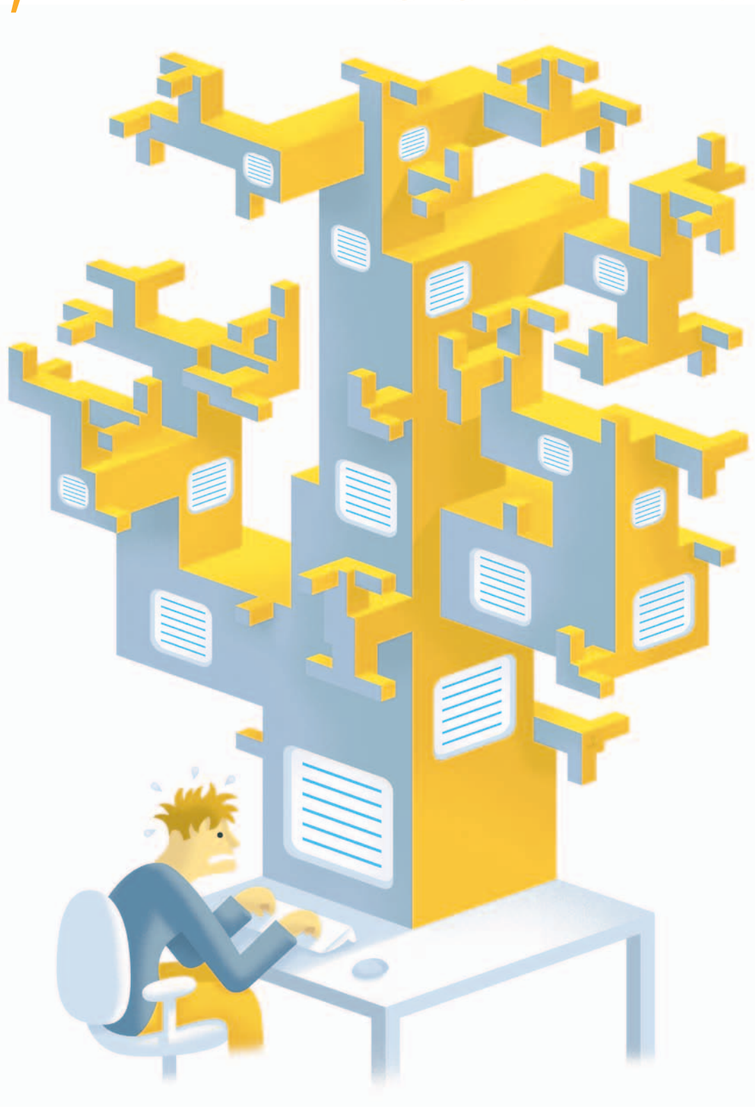
De les

Dus: niet doen, dat is de boodschap van de rule of three. Niet starten met het ontwikkelen van iets generieks. Als je besluit dat het generiek oplossen van een bepaald probleem nut heeft, zorg dan eerst dat je drie concrete toepassingen hebt gerealiseerd waarvan je denkt dat die zouden kunnen profiteren van het generieke deel. Dan wordt duidelijk hoeveel en welke abstracties gedeeld kunnen worden – en welke dus kandidaat zijn om generiek te maken. Dat realiseren kan misschien deels parallel met het ontwikkelen van het generieke deel, maar dat vereist extra veel discipline en aandacht.

De les heeft een lastige boodschap in zich. Je wilt iets generieks maken en dan start je met dat juist niet doen? En dan daarbij of later alsnog inspanning moeten leveren om al werkende toepassingen te refactoren en het generieke deel te oogsten? Niet erg aantrekkelijk.

Dat is een van de oorzaken dat deze bestpractice vaak genegeerd wordt. De belofte van een uniforme, generieke en instelbare oplossing is aantrekkelijker dan het doemscenario dat dat punt waarschijnlijk nooit bereikt wordt. Het is enthousiasme en positivisme tegenover voorzichtigheid en realisme. Het is het onwrikbare vertrouwen in eigen kunnen tegenover een genuanceerd verhaal over uitdagingen en risico's.

Wie echter geen trek heeft in dure, vaak mislukkende IT-projecten en bijbehorende publiciteit, doet er goed aan de eigen plannen en ambities te toetsen aan de rule of three. «



TOEPASSING

Het toepassen van de rule of three geeft zo zijn eigen uitdagingen:

- Als er al drie kant-en-klare applicaties draaien, wie gaat dan nog de moeite nemen om daar de generieke delen uit te destilleren? Dat betekent immers geld en tijd steken in iets dat vervolgens nog precies hetzelfde doet als daarvoor. Hergebruik is een algemeen belang – en wie vertegenwoordigt dat met voldoende gewicht als de specifieke belangen al behartigt zijn?
- Hoe voorkom je dat bij het maken van de drie voorbeelden dusdanige keuzes gemaakt worden dat later generiek maken moeilijk, zo niet onmogelijk wordt? Enige vorm van vooraf nadenken of zelfs realiseren is misschien toch niet geheel onverstandig. Maar let op – dit is de eerste stap om dan maar meteen een paar meer of alle mogelijke generieke zaken eerst te maken. Alleen met de allergrootste terughoudendheid doen, dus.

De prijs van genericiteit

De belofte van generieke software is beter onderhoudbare én stabielere software. Onderhoudbaarder, omdat gedeelde elementen één keer uitgeprogrammeerd zijn en dus ook maar één keer aangepast of gecorrigeerd hoeven te worden. Stabieler, omdat dezelfde software telkens opnieuw gebruikt en verbeterd kan worden.

Dingen generiek maken heeft echter ook nadelen, zoals:

- **Generieker betekent vaak ook abstracter, met bovendien soms ingewikkelde constructies van elkaar aanroepende generieke en specifieke delen. En dus ook moeilijker over te dragen en lastiger te begrijpen – juist minder onderhoudbaar dus. De afweging moet zijn: is vijf keer iets simpels vinden en aanpassen echt meer werk dan één keer iets moeilijks?**
- **Aanpassingen in generieke delen hebben impact op alle plaatsen die die delen gebruiken. Dat vraagt grotere omzichtigheid (en is dus minder onderhoudbaar) en geeft kans op onbedoelde zij-effecten (en is dus minder stabiel). Een gecontroleerde evolutie van generieke software is een vraagstuk op zich.**
- **Afhankelijkheden op softwareniveau geven afhankelijkheden – en dus spanning – tussen verder eigenlijk losstaande projecten of organisatieonderdelen. Wie beslist of iets specifiek of generiek gemaakt moet worden? Wie is eigenaar van het generieke deel? Wie bepaalt de prioriteiten in wensen voor dat deel?**

Generieker heeft dus een prijs. Er is een omslagpunt waar de nadelen de beoogde voordelen teniet doen – een punt dat vaak veel dichterbij is dan verwacht.

HERKOMST RULE OF THREE

In 1999 beschreef Martin Fowler in Refactoring: Improving the Design of Existing Code de rule of three als vuistregel voor het bepalen wanneer code generiek gemaakt moet worden: namelijk, als je voor de derde keer hetzelfde aan het doen bent. Fowler schrijft de vuistregel toe aan Don Roberts, die met Ralph Johnson in 1996 in het artikel Evolving Frameworks de rule of three – in de vorm van een patroon – beschreef als antwoord op de vraag: hoe start je met het ontwerp van een raamwerk? Roberts en Johnson verwijzen weer naar de rule of three die voortkwam uit werk van Ted Biggerstaff en Will Tracz uit 1988.

Matthijs Maat (matthijs.maat@mxi.nl) en **Gert Florijn** (gert.florijn@mxi.nl) zijn werkzaam bij M&I/Partners, adviseurs voor management en informatie (www.mxi.nl).