VOP: de richtlijnen

In het *Vakoverschrijdend (eind)project*, kortweg *VOP*, is het de bedoeling dat je in **teams** van (meestal) vier studenten een geïntegreerd softwareproduct ontwikkelt. Hierbij beroep je je op de vaardigheden die je in verschillende opleidingsonderdelen hebt verworven.

We lieten je vooraf kiezen uit enkele onderwerpen. Onder andere op basis van de resultaten van die peiling vormden we teams, die elk een onderwerp kregen toegewezen. Verschillende teams werken dus dezelfde projectopgave uit; in de loop van het project zullen de teams echter ongetwijfeld andere pistes gaan bewandelen.

1 Rollen

Om het project dat ietsje realistischer te maken, nemen voor elk team twee docenten de rol van **klant** aan. De klant vertolkt zijn eisen aan het team. Hij is echter niet altijd even onderlegd in de technische kant van het gebeuren: de ene keer weet hij helemaal niet wat er zoal mogelijk is, de andere keer is hij net veel te ambitieus. Het kan ook voorkomen dat hij bepaalde zaken over het hoofd ziet of dat hij midden in het project radicaal van gedachten verandert. Je team moet dus de dialoog aangaan om de eisen van de klant te rijmen met de haalbaarheid van het project!

De docenten die de rol van klant niet vervullen, zijn beschikbaar als **coach**. Omdat hun tijd echter beperkt is, zal er strikt op worden toegezien dat deze ook nuttig wordt besteed. Zo wordt op geziene leerstof *niet* teruggekomen. Merk op: het VOP is niet enkel een herkauwing van geziene stof. Je zult ook nieuwe kennis (moeten) vergaren en toepassen.

2 Praktisch

Zoals vermeld op de studiefiche, wordt verwacht dat elk teamlid **ongeveer 175 uur** spendeert aan het project. Ongeveer de helft daarvan zijn contacturen (op dinsdag- en vrijdagnamiddag), de rest moet daarbuiten gebeuren. Het is dan ook de bedoeling dat je steeds nauwgezet bijhoudt waar je mee bezig bent en hoeveel tijd je hieraan spendeert. Blijkt dat je hiertegen zondigt of dat je uiteindelijk minder dan 175 uur hebt gespendeerd aan het project, dan zal dit ingrijpende gevolgen hebben voor je eindquotering.

Er wordt verondersteld dat je tijdens de lesuren voorzien voor het project steeds aanwezig bent. Afwezigheden wettig je met een doktersattest. Excessen resulteren in een onvoldoende. Merk op dat het VOP *niet* hernomen kan worden in tweede zittijd.

3 Vereisten

Hoewel de kennis van de *klant* op het vlak van technologie beperkt is, leggen we als *docenten* enkele vereisten op:

- Bepaalde technologieën, zoals *Java Servlets*, *JDBC* en *JavaScript*, **moeten** aan bod komen in het project.

- Bepaalde technologieën mag je dan weer **niet** gebruiken. Wens je gebruik te maken van technologie die niet expliciet in dit document is opgenomen, zoals bijvoorbeeld een externe Java-bibliotheek, dan *moet* dit vooraf besproken worden met de docenten! Het is in de eerste plaats immers de bedoeling dat je gekende leerstof toepast. Daarnaast is het uitbesteden van functionaliteit aan een bibliotheek misschien niet zo fair tegenover de andere teams …
- Ontwerp en code moeten voldoen aan bepaalde **kwaliteitseisen**. Denk hierbij aan leesbaarheid, herbruikbaarheid, flexibiliteit, robuustheid, toepassing van design patterns, meerlagenarchitectuur …
- We hechten veel belang aan grondig **testen** van de software, met behulp van (onder andere) *unit tests* en *integration tests*.
- We verwachten voldoende relevante **analyse- en ontwerpdocumenten**, in correcte UML. Deze kunnen deels ook aan de klant voorgelegd worden.
- Aan het eind van het project verwachten we een **presentatie** en **projectdossier**, waarover later meer.
- We zien toe op de juiste en zinvolle toepassing van de **Scrum**-methodologie.

4 Scrum

Om het project in goede banen te leiden, beroepen we ons (in grote lijnen) op de **Scrum**-ontwikkelingsmethodologie, die deel uitmaakt van de *Agile* filosofie. De twaalf VOP-lesweken worden opgedeeld in drie projectfasen, de zogenaamde **sprints**. In elke sprint werkt je team een deel van het project af; daarbij doorloop je telkens eigenlijk een volledig ontwikkelingsproces (analyse, ontwerp, implementatie, testen, ontplooien).

4.1 Vóór een sprint

Een sprint begint met een **sprint planning meeting**. Hier wordt samen met de klant vastgelegd wat het team zal verwezenlijken in de sprint. Dit komt erop neer dat een aantal *features* (ook wel *stories* genoemd) uit de *master backlog* in de *sprint backlog* opgenomen worden, rekening houdend met prioriteit, risico, haalbaarheid enzovoort. Dit is een soort contract met de klant. Bij de eerste sprint hebben wij (als docenten/klanten) deze keuze reeds gemaakt, zodat je onmiddellijk van start kunt gaan.

Na de sprint planning meeting analyseert het team de verwachte features verder; dit wordt ook *grooming* genoemd. Bepaal de actoren en werk de features uit in use cases. Bij elke use case worden zowel succes- als failure-scenario's uitgewerkt. Maak schetsen van de user interface. Communicatie met de klant is hier heel belangrijk, enerzijds ter verduidelijking van de gewenste functionaliteit, anderzijds ter goedkeuring van de gemaakte analyse en voorgestelde uitwerking.

De features worden nader opgedeeld in taken (*tasks)*, die gemiddeld zo'n vier uur in beslag nemen. De hoeveelheid werk wordt uitgedrukt in *story points*; de initiële schatting wordt weleens verwezenlijkt door een spelletje *planning poker*. De geschatte totale hoeveelheid werk per feature verkrijg je dus door die van de taken op te tellen.

Een realistische schatting is van groot belang. Een van de basisprincipes van Scrum is immers dat de deadline van een sprint absoluut niet uitgesteld mag worden. Desnoods worden bepaalde features (deels) verschoven naar een volgende sprint. Zo weet de klant na elke sprint hoever hij staat.

4.2 Tijdens een sprint

Ook *tijdens* elke sprint vergadert je team op regelmatige basis: *bij elke bijeenkomst* hou je een **daily standup**. Op deze korte vergadering beantwoordt elk teamlid telkens drie vragen:

1. Wat heb je gerealiseerd sinds de vorige bijeenkomst?
2. Wat zal je realiseren tegen de volgende bijeenkomst?
3. Welke problemen ondervond je en verwacht je te ondervinden?

Op deze manier behoudt het voltallige team het overzicht over het volledige project.

De daily standup duurt slechts enkele minuten. Om te vermijden dat de vergadering zou uitlopen, wordt ze al rechtstaand gehouden – vandaar ook de naam *standup*.

4.3 Na een sprint

Na elke sprint volgen ten slotte nog twee vergaderingen. Op de **sprint review meeting** presenteert het team zijn resultaten aan de klant. Daar hoort steeds een demonstratie van de afgewerkte features bij.

Aansluitend bespreekt het team onderling zijn vorderingen op de **sprint retrospective**. De aan taken toegekende story points worden vergeleken met de werkelijke hoeveelheid werk, zodat de volgende keer efficiënter kan worden geschat. Ook eventuele algemene verbeterpunten worden besproken.

Na de sprint review is het mogelijk dat de klant wijzigingen vraagt aan bepaalde gerealiseerde features, en uiteraard kunnen er bugs gevonden worden. De volgende sprint zal dan deels bestaan uit het aanpakken van deze aanpassingen en bugs, en deels uit een aantal bijkomende features.

4.4 Beheer

Voor het beheer van de features en taken, maar ook ontwerpdocumenten, code, gebruikersdocumentatie enzovoort, maak je gebruik van de **projectbeheersite** *http://project.tiwi.be/*. Gebruik je UGent-account om in te loggen.

Op deze website voer je alle features en taken in, opgedeeld per sprint. Je kunt onder andere het geschatte aantal story points opgeven, de toestand (*in progress*, *done* …), de effectief gespendeerde tijd en de betrokken ontwikkelaars. Bij Scrum wordt doorgaans een zogenaamd *Scrum board* gebruikt: een bord met kleefbriefjes, dat meteen een overzicht geeft van de toestand van de features en taken. De projectbeheerwebsite emuleert dergelijk bord, zodat je er ook van thuis uit toegang toe hebt.

Houd deze informatie nauwgezet bij! Bij de evaluatie van het project wordt deze gebruikt om na te gaan of elk teamlid voldoende heeft bijgedragen tot het project.

5 Infrastructuur

Een belangrijke pijler van Scrum is dat er **op elk moment** een werkende versie van de code moet zijn. We voorzien daarom voor elk team alle werkingsmiddelen om dit te realiseren. Naast de hierboven beschreven *projectbeheersite* beschikt je team nog over:

- de Git-**versiebeheerserver** *github.ugent.be*;
- de Java EE-**applicatieserver**;
- de MySQL-**databankserver** *db.vop.tiwi.be*;
- specifieke **harde schijven** in de labolokalen.

Zowel bij de applicatieserver als bij de databankserver maken we onderscheid tussen twee omgevingen. Technologisch zijn beide identiek, maar hun doel verschilt. De klant is enkel geïnteresseerd in de **stabiele omgeving**, waar (idealiter) een foutloze versie van de applicatie draait. Om te vermijden dat niet-werkende code in deze omgeving belandt, test je *elke versie* steeds uit in de **testomgeving**. Werkt deze naar behoren, dan pas kun je ontplooien naar de stabiele omgeving.

De technische details van al deze systemen, alsook de configuratie- en logingegevens, ontvang je spoedig per e-mail.

6 Analyse en ontwerp

Bij een middelgroot project als dit is implementatie – het eigenlijke programmeerwerk – slechts een van de stappen die je team doorloopt. De ervaring leert dat teams die onvoldoende aandacht besteden aan analyse en ontwerp (of aan de andere stappen), weleens geconfronteerd worden met de onaangename gevolgen daarvan.

Enerzijds zijn analysedocumenten bedoeld zijn voor het team zelf, anderzijds worden ze voorgelegd aan de klant (behalve heel technische diagrammen). Ook de coaches zullen regelmatig de analyse onder de loep nemen.

Zoals vermeld, vertrek je steeds van de sprint features, en distilleer je hieruit in overleg met de klant use cases, scenario's en UI-schetsen. Pas wanneer een eerste analyse en modellering van een feature voltooid is, kun je beginnen te programmeren.

Naarmate nieuwe features worden toegevoegd of bestaande features wijzigen, zal vaak gerefactord moeten worden. Agile methodologieën voorzien dit ook: *prepare for change*. Indien de code wijzigt, moeten de analysedocumenten systematisch mee aangepast worden, zodat er altijd een actuele versie is. Documentatie kan op de projectbeheersite bijgehouden worden (bij *Wiki*, *Documents* of *Files*), of eventueel in de Git-repository.

Welke UML-diagrammen er gemaakt moeten worden, hangt af van de applicatie. Voor de structuur is doorgaans een *class diagram* nodig, aangevuld met bijvoorbeeld een *component diagram*, *package diagram* en/of *deployment diagram*. Het gedrag kun je modelleren aan de hand van een *activity diagram*, *state diagram* of *interaction diagram*.

Maak hoe dan ook *zinvol* gebruik van UML: maak zeker niet te weinig diagrammen, maar ook niet te veel, en kies steeds voor het diagram dat het ontwerp het duidelijkst vertolkt. Be-

perk je bovendien niet tot enkel een klassendiagram van het model. Ook diagrammen die de lagenstructuur verduidelijken of de toepassing van design patterns in de verf zetten, zijn aan te bevelen.

Entity-relationship diagrams kunnen interessant zijn, maar niet als eerste keuze. Als je een ER-diagram maakt, leid het dan af van een *class diagram*; dit biedt immers meer abstractiemogelijkheden (zoals overerving).

Het gebruik van UML-tools, zoals *Visual Paradigm for UML*, laat toe om elegante diagrammen te maken, maar is geen absolute must. Heb je een diagram (verzorgd) uitgetekend op een stiftenbord of een velletje papier, dan is een duidelijke foto daarvan even nuttig.

7 Implementatie

De applicatie moet gestructureerd worden in drie **tiers**:

- De **presentation tier** bestaat uit een aantal clientapplicaties. Een gedeelte moet een desktopapplicatie met **Java Swing** worden, een ander gedeelte een webapplicatie met **servlets**. Eventueel zijn er ook consoleapplicaties.
- De **business logic tier** is een serverapplicatie die bestaat uit één of meerdere servlets die *HTML* produceren én één of meerdere servlets die *JSON* produceren (en aldus een *REST-webservice* aanbieden).
- De **data tier** zorgt voor persistentie van de gegevens, in eerste instantie via een relationele databank.

Alle communicatie van de clients met de server gebeurt via *HTTP*, en clients kunnen niet rechtstreeks met elkaar communiceren. Wanneer clients tóch contact met elkaar opnemen, gebeurt dat bijgevolg onrechtstreeks via de server (en niet via de data tier).

Alle code wordt in **Java** geschreven, aangevuld met **SQL**, **Perl** en **JavaScript**. Andere talen, zoals C# en PHP, zijn dus niet toegestaan.

7.1 Algemeen

Gebruik **Apache Maven** om je code te organiseren. Maak een hoofdproject met submodules voor de verschillende componenten (GUI, model, datalaag enzovoort). Houd je telkens aan de geijkte bestandsstructuren en naamgeving. Maven is geïntegreerd in recente versies van NetBeans IDE en Eclipse.

Zowel lagen als tiers moeten eenvoudig **inwisselbaar** zijn. Zo kan het vervangen van de data tier door een *in-memory data store* het testen vergemakkelijken, net als het vervangen van een grafische gebruikersinterface door een tekstuele. Zorg dat dit eenvoudig in te stellen valt, bijvoorbeeld via een configuratiebestand. *Dependency injection*, bijvoorbeeld aan de hand van *Spring*, kan hierbij helpen.

Maak **unit tests** (met *JUnit* of *UnitNG*) voor modelklassen en business logica. Gebruik *DbUnit* voor databanktesten. Voorzie ook **user-interface-testen** (*integration testing*), met *UISpec4J* voor de GUI en *Selenium Web Driver* voor het webgedeelte. Gebruik een mock-framework zoals *JMockit* om **mock-versies** voor bepaalde componenten te genereren.

Voorzie doorheen de applicatie **logging** met *SLF4J*. Produceer duidelijke en consequente logberichten. Zorg ervoor dat de gebruiker kan instellen welke ervan worden gelogd.

Indien de klant een meertalige applicatie wenst, gebruik je de gekende Java-technieken voor **internationalisation**. In JavaScript en/of Perl rekenen we op je creativiteit.

7.2 Presentation tier

Webpagina's maken gebruik van **HTML5** en *JavaScript* (met *jQuery*) en spreken de REST-webservice aan met behulp van *AJAX*.

Indien er een applicatie voor een **mobiel apparaat** gevraagd wordt, dan moet die als web-applicatie worden gerealiseerd; *native apps* zijn dus niet toegestaan. Je mag eventueel gebruikmaken van mobiel georiënteerde toolkits zoals *jQuery Mobile* of *Bootstrap 3*.

Verzorg de lay-out van je webpagina's. Zorg voor een scheiding tussen presentatie en structuur; maak hiervoor gebruik van *CSS3*.

Java-applets willen we (gelukkig) niet zien. Je mag wel *Java Web Start* gebruiken.

7.3 Business logic tier

De REST API kun je eventueel met klassieke servlets implementeren. Het verdient echter aanbeveling om *JAX-RS (Jersey)* te gebruiken.

Om JSON-code te genereren en te verwerken, zijn vele bibliotheken beschikbaar. Wij geven de voorkeur aan *Gson* van Google.

7.4 Data tier

De relationele databank dient geïmplementeerd te worden met **MySQL**. Maak zinvol gebruik van *stored procedures* en *before-* en *after-triggers*.

Het is aangewezen om scripts te maken voor de initialisatie van de databank, zodat de databank makkelijk in een andere omgeving kan opgezet worden. Daarnaast vul je, met behulp van één of meerdere **Perl**-scripts, de databank op met een grote hoeveelheid realistische gegevens, die representatief zijn voor de gegevens waarmee de klant uiteindelijk zal werken. De richtlijnen voor deze Perl-scripts staan beschreven aan het einde van dit document.

De serverapplicatie communiceert met de databank via **JDBC**. ORM-tools (zoals *Hibernate*) zijn niet toegelaten; je schrijft of genereert een SQL-query dus steeds zelf.

7.5 Opmerking

Nogmaals: wens je gebruik te maken van tools, bibliotheken of frameworks die niet expliciet in dit document vermeld staan, dan leg je dit **altijd** vooraf voor aan de coaches.

8 Evaluatie

Bij een project van dergelijke omvang is het niet eenvoudig om tot een representatieve en objectieve quotering te komen. In ieder geval trachten we zo transparant mogelijk tewerk te gaan bij de beoordeling. We geven daarom alvast de grote lijnen mee:

- Het verloop en eindresultaat van elke sprint worden zowel door de klant als de coaches beoordeeld. Ze geven elk (ongeveer) de helft van de punten.
- Elk teamlid krijgt een individuele quotering. Het kan dus voorvallen dat één of meerdere teamleden beduidend hoger of lager scoren, of al dan niet slagen voor het opleidingsonderdeel.
- Het is de bedoeling dat elk teamlid met elk onderdeel van het project in contact komt; tenslotte is het een *vakoverschrijdend* project. Soms valt dit moeilijk te realiseren, maar er wordt verwacht dat het team zelf voldoende variatie aanbrengt.
- Het volledige project wordt in rekening gebracht: analyse, ontwerp, implementatie, testen, methodologie, teamwork, communicatievaardigheid, inzet, stiptheid enzovoort. Vanzelfsprekend weegt niet elk aspect daarbij even zwaar door.
- De docenten evalueren permanent de werking van het team. Dit kan bijvoorbeeld door (discreet) toe te kijken tijdens een daily standup, maar ook door een blik op de projectbeheerwebsite of door consultatie van individuele teamleden.
- Tussentijds worden op regelmatige basis beknopte *peerevaluaties* uitgevoerd, om na te gaan hoe de samenwerking binnen het team verloopt. Na afloop van het project volgt nog een uitgebreidere evaluatie.
- Na elke sprint ontvangt elk team een tussentijdse beoordeling onder de vorm van een groen, oranje of rood licht. Dit wordt vergezeld van een korte toelichting.

Het project is hoe dan ook een dynamisch gegeven. De docenten behouden zich daarom steeds het recht voor om van het bovenstaande af te wijken.

9 Contact

De verschillende vergaderingen met de klant werden reeds opgelijst. Voor contact met de docenten wordt geen agenda voorzien; dit gebeurt wanneer nodig. Enerzijds kan een team of teamlid zelfstandig contact opnemen met de docenten, hetzij mondeling, hetzij via e-mail; anderzijds kunnen de docenten een team of teamlid uitnodigen voor een vergadering.

Aarzel niet om ons te contacteren bij persoonlijke problemen of problemen in je team!

Om contact op te nemen met de **coaches**, stuur je een e-mail naar *vop@tiwi.be*. De **klanten** daarentegen contacteer je op het adres dat overeenstemt met je onderwerp:

- *vop-racing@tiwi.be*
- *vop-stambomen@tiwi.be*

Daarnaast kun je de individuele docenten ook bereiken op hun persoonlijke e-mailadressen. In de meeste gevallen is het echter aan te raden om bovenstaande adressen te gebruiken.

Richtlijnen Perl

Bij het VOP gebruik je **Perl** (versie 5.12 of nieuwer) om de databank te vullen met een grote hoeveelheid testgegevens. Aan de hand van die gegevens kun je de schaalbaarheid van je applicatie(s) testen. Het vullen van de databank gebeurt op twee manieren:

- **Scraping.** Je script haalt de HTML-code van een pagina af, filtert aan de hand van reguliere expressies de nuttige informatie eruit, en stopt deze in de databank. Vaak zal de gewenste inhoud zich verspreid over meerdere pagina's bevinden; in dit geval volgt het script de links en filtert het de inhoud uit de verschillende pagina's.
- **Dummy data.** De gegevens die je hebt verworven aan de hand van scraping vul je aan met willekeurig gegenereerde gegevens. Een triviaal voorbeeld is het genereren van "Gebruiker 1" tot en met "Gebruiker 100.000", maar je kunt ook creatiever te werk gaan. Heb je bijvoorbeeld gebruikersinfo gescrapet, dan kun je nog meer fictieve gebruikers genereren door hun voor- en familienamen onderling te combineren.

Beide methodes dienen geïmplementeerd te worden, onder de vorm van één of meerdere scripts. Voor beide zijn de mogelijkheden legio: bij welke website(s) je gaat scrapen, welke gegevens je genereert, hoe realistisch de gegevens zijn … hangt sterk af van de scenario's. Belangrijker zijn:

- **Schaal.** Hoe meer gegevens, hoe beter. Een databank met 100.000 records, verspreid over de verschillende tabellen, is geen uitzondering.
- **Bruikbaarheid.** Ook al zijn de gegevens fictief, ze moeten representatief zijn voor de échte gegevens, die uiteindelijk in de databank zullen belanden. Overleg daarom goed met de klant!

Het VOP bouwt verder op onder andere het opleidingsonderdeel *Besturingssystemen II*. Je past dus uiteraard de daarin vergaarde expertise toe. Beperk je echter niet tot een triviaal Perl-script van enkele tientallen regels. De coaches rekenen op je creatieve inbreng, die de geziene inleiding tot Perl overstijgt. Tegelijkertijd leggen we echter volgende beperkingen op:

- **Werkwijze.** Zoals vermeld, voert je Perl-code in essentie drie stappen uit:
    1. **HTML-code afhalen.** Dit gebeurt verplicht aan de hand van *libwww-perl (LWP)*. Om (correcte) URL's op te bouwen, mag je de module *URI::Escape* gebruiken.
    2. **HTML-code verwerken.** Op de afgehaalde code pas je (zelf) *reguliere expressies* toe om de nodige informatie te filteren. Voor de conversie van HTML naar platte tekst kun je een beroep doen op de module *HTML::Entities*. Het gebruik van Perl-modules die het scrapingproces verder vereenvoudigen, zoals bijvoorbeeld *Web::Scraper*, is **niet** toegestaan.
    3. **Gegevens opslaan.** Eens je script over de nodige waarden beschikt, voegt het deze zelf toe aan de databank. Je maakt hiervoor gebruik van de module *DBI*, logischerwijs met *DBD::mysql* als databankdriver.

- **Modules.** Het gebruik van modules die hierboven niet werden vermeld, is niet toegestaan. Wens je toch een beroep te doen op een bijkomende module, bijvoorbeeld om afbeeldingen te verwerken, archiefbestanden aan te maken …, dan leg je dit eerst voor aan de coaches.

- **Stijl.** Alle scripts moeten het *strict-pragma* van Perl respecteren. Daarnaast zorg je voor leesbare code: structureer je code op een zinvolle manier, maak consequent gebruik van indentering en voeg commentaar toe waar nodig.

Er zijn vele manieren om toegevoegde waarde te creëren ten opzichte van wat werd behandeld in het opleidingsonderdeel Besturingssystemen II. Hieronder geven we er enkele mee; deze lijst is echter niet limitatief.

- **Analyse.** Ook Perl-programmeurs maken weleens een UML-diagram. Deins hier dus niet voor terug. En zoals je weet: wacht er niet mee tot ná het programmeren!

- **Subroutines.** In Perl heten methodes *subroutines*. Om je code te structureren, is het gebruik hiervan sterk aan te raden. De documentatiepagina *perlsub* beschrijft het gebruik en de werking ervan.

- **Includes.** Je kunt je code structureren door (bijvoorbeeld) eigen subroutines logisch te verspreiden over verschillende scriptbestanden, die dienstdoen als bibliotheken. Om deze in te laden in het huidige script, gebruik je de functie *require*.

- **OGP.** Terwijl Perl geen zuiver objectgeoriënteerde taal is, ondersteunt ze toch de meest elementaire concepten ervan. De documentatiepagina's *perlootut* en *perlobj* bespreken het gebruik van klassen en objecten in Perl. Een iets uitgebreidere, maar meer natuurlijke manier van objectgeoriënteerd programmeren wordt voorzien door het pakket *Moose*; het gebruik hiervan is eveneens toegestaan.

- **Foutafhandeling.** Er kan heel wat mislopen bij scraping: een website of -pagina is niet beschikbaar, de HTML-code ervan is niet zoals verwacht, de databank is offline … Tracht hier dan ook zo goed mogelijk op in te spelen.

- **Hervatten.** Aansluitend op foutafhandeling, kun je ervoor zorgen dat het scrapingproces steeds bijhoudt hoever het gekomen is, zodat het in geval van een fout niet weer van voor af aan hoeft te starten. Of je daarbij een beroep doet op hulpbestanden, de databank of een nog andere opslagplaats, maak je zelf uit.

- **GUI's.** Ook in Perl kun je volwaardige grafische gebruikersinterfaces programmeren. Het kan bijvoorbeeld aangenaam zijn om tijdens het scrapen een voortgangsbalkje te laten zien, of knoppen te voorzien om scrapingscenario's in te laden. Overdrijf echter niet, want op een zinloze GUI zit de klant ook niet te wachten.