Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Beautiful Javascript

How to guide students to create good and elegant code

Harrie Passier, Sylvia Stuurman, and Harold Pootjes

Faculty of Management, Science and Technology

Department of Computer Science

Open Universiteit Nederland

NIOC 2015

Open Universiteit
www.ou.nl

Thema

- Script-talen zoals Javascript zijn stimulerend voor creativiteit
  - Geen dynamische typecontrole
  - Meteen/snel resultaat
- Nadeel: er kan gemakkelijk verschrikkelijke code ontstaan

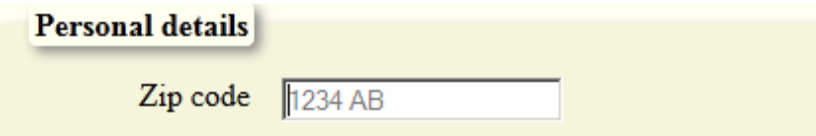
Open Universiteit
www.ou.nl

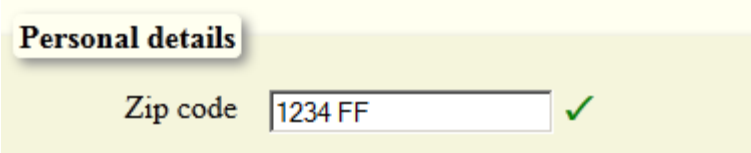
Context

- Algemeen beeld boeken voor Javascript:
  - Veel ‘syntax en voorbeelden’
  - Minder ‘hoe ontwikkelen we goede code?’
- Studenten vinden het moeilijk een grotere applicaties te ontwikkelen. Bevindingen bij een programmeeropdracht o.a.:
  - ‘Optimale’ oplossing: ~ 200 regels code (leesbaar/uitbreidbaar)
  - Student oplossingen: tussen de 400 en 800 regels code
  - Redundantie in code is een belangrijke indicator

Voorbeeld: Ontwikkel een validatiefunctie voor het volgende formulier

```
<fieldset>
 <legend>Personal details</legend>
 <label class="heading">Zip code</label>
 <input id="zipcode" class="input" type="text"
        name="zipcode" placeholder="1234 AB“
        tittle=“The correct format is 1234AB"
   required>
 <span class="feedback"></span>
 ...
</fieldset
```





Open Universiteit
www.ou.nl

Voorbeeld: Oplossing van een student?

Redundante code door specifieke functies

```
function isZipCodeCorrect() {
 var isCorrect = TRUE,
     element = $(ZIPCODE),
     fbField = element.next(CLASS_FEEDBACK),
     value = element.val(),
     pattern = “^[1-9]\d{3}\s?[a-zA-Z]{2}$”,
     regex = new RegExp(pattern),
     title = element.attr(TITLE);
 if (value) {
   isCorrect = regex.test(value);
   if (isCorrect) {
     fbField.removeClass(INVALID).addClass(VALID).html(OK);
   } else {
     fbField.removeClass(VALID).addClass(INVALID).html(title);
   }
 }
 return isCorrect;
}
```

Inlezen gegevens

Verwerking

Wegschrijven resultaten

Lage cohesie, slecht herbruikbaar, slecht uitbreidbaar, …

Oorzaken

- Volgens ons is dit een algemeen probleem: de meeste boeken over programmeren (Java, XML verwerking Web applicaties, Apps, …) gaan vooral over het *wat* (syntax en voorbeelden) en minder, of zelfs niet, over *hoe* (welke stappen, regels, ...).
- Er zijn uitzonderingen, bijvoorbeeld: How to design programs (Felleisen)
- **Feit**: studenten blijken slecht te programmeren na de eerste programmeercursus [McCracken (2001), Ma (2007), Whalley (2006)]
- **Hypothese**: Dit komt door de nadruk op het *wat* en het gebrek aan het *hoe*

Enkele citaten

… Students were provided with assignments and access to computers and were expected to learn through trial and error and unguided discovery … (Sloane and Linn, 1988)

Programming instructors often assume that students can take their general problem-solving skills and discover specific programming design skills on their own. Thus, students learn program design through unguided discovery ... (Linn and Clancy, 1992)

We need to look seriously at how we teach programming. The purpose of an education should not simply be to pour facts into students, but rather to teach them to think. (Gries, 2006)

Ten Steps to Complex learning (Merrienboer, 2009)

- Taakbeschrijving**
  - Wat moet de student doen, wat moet worden bereikt?
- Ondersteunende informatie**
  - Conceptuele kennis nodig die nodig is om het probleem op te lossen en te weten wanneer een oplossing goed is
- Procedurele information**
  - Een beschrijving van de (vuist-)regels, stappen, procedures om het probleem op te lossen
- Oefening
  - Van voorbeelden tot zelf problemen oplossen, gebruikmakend van ondersteunende en procedurele kennis
  - Authentieke taken

Taakbeschrijving

- Wat is form-validatie?
  - Controle op aanwezigheid, volledigheid en consistente van gegevens (bijvoorbeeld gebruiker-paswoord combinatie).
- Waarom form-validatie?
  - Gebruiksvriendelijkheid, gegevensbetrouwbaarheid en security redenen
- Vier dimensies
  - Autorisatie, tijdigheid, **correctheid** en **compleetheid**
    - Compleetheid: er is invoer
    - Correctheid: de invoer is correct - String (structuur), getal (interval)
- Definitie validiteit: compleet <u>en</u> correct
  - Voor sommige studenten is dit een goede start voor een top-down benadering

Ondersteunende informatie

- Algemene ontwerpprincipes
  - Abstractie, koppeling, cohesie, decompositie, hergebruik, …
  - Expliciet evalueren a.d.h.v. deze principes (tussentijds en eindproduct)
  - Zou meer moeten worden geoefend (voorbeelden en opgaven)
- Specifieke ontwerpprincipes voor validatie
  - Serverside validatie moet
  - Genoeg is genoeg (ondersteun de gebruiker met het invullen van het formulier)
  - Robuustheid: confronteer de gebruiker niet met interne representaties (zowel 1234aa als 1234AA zijn correct) en geef duidelijke feedback
  - Leid de gebruiker (door zowel impliciete alsook expliciete feedback)
  - Zorg voor herbruikbaarheid en uitbreidbaarheid
- Regels voor Javascript
  - Declareer constanten en variabelen bovenaan, …, etc.
  - Stop gerelateerde functies (etc.) in een module (modulepatroon)

Procedurele informatie (I)

De GUI laag (HTML)

- Specificeer de gegevens die moeten worden ingevoerd
  - Bijvoorbeeld: enumeratie, string (structuur), getal (range)
- Specificeer gebruikers ‘guidance’
  - Voorbeeldwaarden, labels, groepeer gerelateerde invoervelden
  - Denk na over duidelijke feedbackberichten
- Bepaal geschikte HTML(5) inputelementen en gebruik de attributen voor het declareren van invoereisen (type, pattern, min en max):
  - Enumeratie: radio button, drop down list
  - String met structuur: text field + placeholder + pattern attribuut

Procedurele informatie (II)

De domeinlaag (Javascript)

•Specificeer en implementeer een event handler voor controle op compleetheid (is er invoer) → dit levert naïeve code op

•Specificeer en implementeer event handlers voor controle op correctheid (is de invoer correct) → dit levert naïeve code op

•Check code met JSHint, creëer tests en test de code

•*Controleer zowel op generieke alsook op specifieke ontwerpprincipes*. Zo nodig:

- *Refactor*
- Creëer modules door toepassing van het modulepatroon

•Evalueer en documenteer

Procedurele informatie (III)

Refactor

- **Verwijder redundante code**
- **Beschouw mogelijke veranderingen en uitbreidingen**
- Controleer op cohesie, compleetheid, …
- Controleer op ‘separation of concerns’.
- Controleer op ‘information hiding’ en ‘separation of interfaces and implementation’ (creëer modules)
- Controleer op koppeling (tussen modules)

Open Universiteit
www.ou.nl

Voorbeeld: Redundantie

```
function isZipCodeCorrect() {
 var isCorrect = true,
 element = $(ZIPCODE),
 fbField = element.next(CLASSFEEDBACK),
 value = element.val(),
 pattern = element.attr(PATTERN),
 regex = new RegExp(pattern),
 title = element.attr(TITLE);

 if (value) {
  isCorrect = regex.test(value);
  if (isCorrect) {
   fbField.removeClass(INVALID).
        addClass(VALID).html(OK);
  } else {
   fbField.removeClass(VALID).
        addClass(INVALID).html(title);
  }
 }
 return isCorrect;
}
```

```
function isYearsWithoutAccCorrect() {
 var isCorrect = true,
 element = $(YEARSWITHOUT),
 fbField = element.next(CLASSFEEDBACK),
 value = parseInt(element.val()),
 min = element.attr(MIN) || ...,
 max = element.attr(MAX) || ...,
 title = element.attr(TITLE);

 if (value) {
  isCorrect = value >= min && ...;
  if (isCorrect) {
   fbField.removeClass(INVALID).
        addClass(VALID).html(OK);
  } else {
   fbField.removeClass(VALID).
        addClass(INVALID).html(title);
  }
 }
 return isCorrect;
}
```

Defineer hulpfunctie `handleMessage`

Voorbeeld: Mogelijke uitbreidingen

```
function isZipCodeCorrect() {
 var isCorrect = true,
 element = $(ZIPCODE),
 fbField = element.next(CLASSFEEDBACK),
 value = element.val(),
 pattern = element.attr(PATTERN),
 regex = new RegExp(pattern),
 title = element.attr(TITLE);

 if (value) {
  isCorrect = regex.test(value);
  handleMessage(isCorrect, fbField,
                title);
 }
 return isCorrect;
}
```

```
function isYearsWithoutAccCorrect() {
 var isCorrect = true,
 element = $(YEARSWITHOUT),
 fbField = element.next(CLASSFEEDBACK),
 value = parseInt(element.val()),
 min = element.attr(MIN) || ...,
 max = element.attr(MAX) || ...,
 title = element.attr(TITLE);

 if (value) {
  isCorrect = value >= min && ...;
  handleMessage(isCorrect, fbField, title);
 }
 return isCorrect;
}
```

Dit zijn generieke expressies voor het testen op ‘formaat’ en ‘getal in een interval’, en kunnen keer op keer worden gebruikt.

Bijvoorbeeld: `isTelephoneNumberCorrect` is ook een test op formaat.

Voorbeeld: Mogelijke uitbreidingen

```
function isValidAgainstRegex(element) {
 var isCorrect = true,
 fbField = element.next(CLASSFEEDBACK),
 value = element.val(),
 pattern = element.attr(PATTERN),
 regex = new RegExp(pattern),
 title = element.attr(TITLE);

 if (value) {
   isCorrect = regex.test(value);
   handleMessage(isCorrect, fbField,
                 title);
 }
 return isCorrect;
 }
```

```
function isValidNumber(element) {
  ...
}
```

```
function isCorrect (element) {
 var res = true;
 switch (element.attr(TYPE)) {
  case RANGE :
  case NUMBER : {
   res = isValidNumber(element);
   break;
  }
  default : {
   if (element.attr(PATTERN)) {
    res = isValidAgainstRegex(element);
   }
   break;
  }
 }
 return res;
}
```

Abstraheer van wat we valideren, focus op het type validatie: string (structure), number (range)

Resultaat: validator.js

```
var validator = (function (){

  // private
  const ...
  var isComplete = function(el) {...},
      isCorrect = function(element){...},

  // Private helper functions
      isValidAgainstRegex = function(element) {...},
      isValidNumber = function(element) {...},
      ...;

  // public API
  return {
    isComplete: isComplete,
    isCorrect: isCorrect
  }
}());
```

Conclusies

- Op zichzelf een prachtig voorbeeld voor studenten
  - Product: een (herbruikbare en uitbreidbare) module voor form-validatie
  - Proces: de ontwikkelstappen (iteratief) voor het ontwerpen en implementeren van de module
- Algemeen: Een eerste poging om een methode te ontwikkelen voor het stapsgewijs ontwikkelen van programma's gebaseerd op een didactische aanpak (Complex learning)
- Specifiek: Een methode voor het ontwikkelen van client side validatiefunctionaliteit

- Vraag: Wie wil er meewerken om in de praktijk te testen of dit studenten helpt?