Verontschuldigd:

Korte stand van zaken

Wat is er gerealiseerd sinds vorige vergadering:

- Datacontract adressen zonder ID
- Datacontracten algemeen: geen entity's meer

Validatie aan UI- en aan businesskant

Het probleem

Zowel aan de kant van de user interface als aan de kant van de businesslaag willen we eenvoudige validatie doen (lengte van een string, verplichte velden). Op de vorige vergadering beslisten we echter dat we entiteiten niet meer zouden exposen over de service. (Zie ticket https://develop.chiro.be/trac/cg2/ticket/433, ondertussen gesloten.)

Aan de UI-kant zitten we met datacontracts, aan de businesskant met entity's. Het is dus niet zo vanzelfsprekend om aan beide kanten hetzelfde te gebruiken voor validatie.

Wat is er al van validatie?

Enerzijds gebruiken we een validatiemechanisme op basis van attributen uit System.ComponentModel.DataAnnotations. Aan de kant van de user interface, weet Controller.ModelState dan 'automagisch' of aan de requirements van de attributen voldaan is. Aan de 'businesskant' deden we hier niets mee, en voorlopig kan dat ook niet meer echt, omdat de attributen in kwestie nu op de datacontracts staan.

```
[DataContract]
public class PersoonInfo
{
        [Verplicht]
        [DisplayName(@"Voornaam")]
        [StringLengte(60), StringMinimumLengte(2)]
        [DataMember]
        public string VoorNaam { get; set; }

        [Verplicht(), StringLengte(160), StringMinimumLengte(2)]
        [DisplayName(@"Familienaam")]
        [DataMember]
        public string Naam { get; set; }

        [DataType(DataType.Date)]
        [DisplayName(@"Geboortedatum")]
        [GeboorteDatumInVerleden]
        [DataMember]
        public DateTime? GeboorteDatum { get; set; }

        // enz…
}
```

Anderzijds hebben we in Chiro.Gap.Validatie de klasse Validator<T>, met 1 method: 'Valideer(T)'. Hiervan hadden we 1 implementatie: CommunicatieValidator: Validator<CommunicatieVorm>, die zowe in UI als in backend gebruikt werd:

```
public class CommunicatieVormValidator : Validator<CommunicatieVorm>
{
	public override bool Valideer(CommunicatieVorm cv)
	{
		Debug.Assert(cv != null);
		return cv.Nummer != null && Regex.IsMatch(
		   cv.Nummer,
		   cv.CommunicatieType.Validatie);
	}
}
```

Maar dat lukt natuurlijk niet meer als CommunicatieVorm niet meer beschikbaar is voor de UI. Ik heb hier nu rond gewerkt door de CommunicatieValidator op een interface te laten werken:

```
public class CommunicatieVormValidator : Validator<ICommunicatie>
{
	public override bool Valideer(ICommunicatie cv)
	{
		Debug.Assert(cv != null);
		return cv.Nummer != null && Regex.IsMatch(
		   cv.Nummer,
		   cv.CommunicatieType.Validatie);
	}
}
```

Waar:

```
public interface ICommunicatie
{
	string Nummer { get; set; }
	string CommunicatieTypeValidatie { get; set; }
}
```

Zodat zowel CommunicatieVorm als CommunicatieDetail implementeren ICommunicatie

Hoe willen we verder met validatie?

Het voorstel:

- Ivm attributen:
  - Validatie-attributen sowieso op datacontract
  - Controleren op niveau van UI (MVC doet dit voor ons, soms zelfs al via javascript), maar ook door service bij het 'binnenkrijgen'.
  - Indien nodig/nuttig attributen ook aan de businesskant gebruiken op entity's, zoals dat vroeger gebeurde.
- Systeem van interface voor gebruik van validatorklasse: OK indien het niet met attributen opgelost kan worden.

Resharper, StyleCop, coding guidelines

Sinds kort gebruiken we resharper, en Bart laat ook geregeld StyleCop eens los op onze code. Deze tools doen soms andere dingen dan vastgelegd in onze coding guidelines. We zullen moeten bekijken waar we de guidelines aanpassen, en waar we de tools anders gaan configureren. Soms brengen die tools ook zaken opnieuw in de aandacht, die wel in onze guidelines stonden, maar die we 'vergaten' te doen. Da's natuurlijk ook interessant.
Een aantal zaken die mogelijk herzien kunnen worden (hoeft natuurlijk niet):

Automatische properties, eenvoudige getters/setters, lege functies

Guideline 41:

- *Eenvoudige getters en setters worden op 1 lijn gedefinieerd, inclusief accolades.*

• Abstracte en automatische property's worden op 1 lijn gedeclareerd, inclusief accolades.
• Lege functies (bijv. constructors) worden op 1 lijn gedefinieerd, inclusief accolades, bijv. EenKlasse(string naam): base(naam) {}.

StyleCop denkt daar blijkbaar anders over:
https://develop.chiro.be/trac/cg2/changeset/934#file69

Dit komt niet van stylecop, maar is vermoedeiljk ergens een instelling in Visual Studio. De constructie
Int BlaBla { get; set; }
Mag alleszins op 1 regel. Veel andere 'eenvoudige' getters/setters komen we niet tegen.

Het gebruik van 'var'

Guideline 120:
Gebruik geen ' var ' voor een basistype zoals int of string .
Resharper is het er niet mee eens.

We behouden de regel: Als het duidelijker is met 'var' dan met iets anders, dan gebruiken we zeker 'var'. Voor 'string' en 'int' weten we het nog niet.

LINQ

Voorstel nieuwe guideline:
- Gebruik steeds LINQ om in een collectie te zoeken

OK

Datacontracts

Voorstel nieuwe guideline:
- Naamgeving datacontracts (-info, -detail)

Dit wisten we al; alleen nog formaliseren.

Ternaire operators

Guideline 134:
Een ternaire (? :) of 'coalescing' (??) operator mag enkel gebruikt worden in statements die op 1 lijn passen. Converteer ze naar een if -structuur als dat niet lukt.

We proberen dit: We behouden de guideline. Als resharper een te lange lijn maakt met ?: of ??, dan kijken of het niet op een andere manier vereenvoudigd kan worden. (Bijv: Extra variabele introduceren die een deel van het resultaat bevat.)