MSO
CVA: Commonality and Variability Analysis
The Matrix

Wouter Swierstra, Hans Philippi

October 16, 2018

This lecture

- Back to analysis
- We have seen some techniques for analysing use cases and requirements, notably the noun-verb analysis . . .
- . . . but you cannot apply this technique without care
- Are there other approaches?

Drawbacks of noun-verb analysis

Which of the following concepts should be in the domain model:

- The *driver* has *qualifications*, recorded in a *database system*
- *Z-tram* has twelve *trams* that can be assigned to three *lines*

Noun-verb analysis

Pros:

- easy to do
- forces you to **not** think about implementation

Cons:

- imprecise
- may create concepts overload in the domain model
- too far separated from object-oriented design principles

A better approach. . .

1. Identify the concepts in your domain (commonalities) together with their different realisations (variabilities)
2. Then specify the interface for the abstraction that encapsulates the variation, to which each commonality adheres
3. Derive this interface by considering how the implementations will be used

A case study: CAD/CAM

What are the commonalities and variations:

- **Different versions of the CAD/CAM software** – version 1 and version 2
- **Different features** – slots, holes, cutouts, . . .
- **Different models** – based on the different versions

Analysis table

Commonality	*Variations*
CAD/CAM system	Version 1
	Version 2
Features	slot
	hole
	cutout
	...
Model	V1-based
	V2-based

Searching for commonalities/variations

- This is ‘easy’ in this domain – you already know the solution
- In general, you can do a brute-force search for commonalities and variations
  1. Look at all possible pairs of entities (X,Y) in your domain;
  2. Ask yourself is X a variation of Y? Or is Y a variation of X? Or are they both variations of something else?
- Restrict yourself to one issue per commonality: don’t introduce V1Holes, V2Holes, V1Slots, V2Slots, etc
- Combining issues leads to weak cohesion

Analysis table

Commonality	*Variations*
CAD/CAM system	Version 1
	Version 2
Features	slot
	hole
	cutout
	...
Model	V1-based
	V2-based

How do we turn this into a design?

What did we do?

1. Identify commonalities
2. Create abstractions for these commonalities
3. Identify derivations with the variations of the commonalities
4. See how these commonalities relate

Designing in two different ways

Two different design strategies produce similar results:

- Design by applying design patterns, creating context
- Design by CVA, establish context by focusing on abstractions

Two complementary styles

- CVA starts focusing on abstractions – try to find the useful abstractions first
- Design patterns – focus on relations between entities

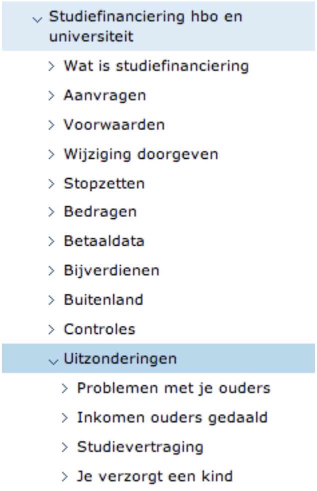
More analysis

We have seen different techniques for analysing a problem domain:

- noun-verb analysis
- commonality variability analysis

We will add one more tool to the toolbox: the *analysis matrix*

Variations, variations, variations. . .



- Studiefinanciering hbo en universiteit
  - Wat is studiefinanciering
  - Aanvragen
  - Voorwaarden
  - Wijziging doorgeven
  - Stopzetten
  - Bedragen
  - Betaaldata
  - Bijverdienen
  - Buitenland
  - Controles
  - Uitzonderingen
    - Problemen met je ouders
    - Inkomen ouders gedaald
    - Studievertraging
    - Je verzorgt een kind

Handling variation

- A lot of systems grow complex quickly because they need to handle a lot of special cases, variations, etc
- Look out for examples the next time you fill in a tax form
- This creates headaches for analysts
- How can we design software that can handle this well?

The analysis matrix (scenario based)

1. Identify the most important features in a particular scenario and organize them in a matrix
2. Proceed through other scenarios, expanding the matrix as necessary
3. Expand the analysis matrix with new concepts
4. Use the rows to identify rules
5. Use the columns to identify specific situations
6. Identify design patterns from this analysis
7. Develop a high-level design

Case study: e-commerce requirements

- Build a sales order system for Canada and the USA
- Calculate freight based on the country we are in
- Money will be handled in the country we are in
- In the USA, tax will be calculated by state
- Use US Postal rules for verifying addresses
- In Canada, use FedEx for shipping and Government Sales Tax and Provincial Sales Tax

Identify variation

We have one important distinction to make:

- When the customer is in the USA
- When the customer is in Canada

The analysis matrix

Start identifying the key features of one case: selling in the USA

	US Sales
Calculate freight	Use UPS rates
Verify address	Use US postal rules
Calculate tax	Use state and local taxes
Currency	US Dollars

Growing the matrix

	US Sales	**Canadian Sales**
Calculate freight	Use UPS rates	
Verify address	Use US postal rules	
Calculate tax	Use state and local taxes	
Currency	US Dollars	

Growing the matrix

	US Sales	**Canadian Sales**
Calculate freight	Use UPS rates	
Verify address	Use US postal rules	
Calculate tax	Use state and local taxes	
Currency	US Dollars	Canadian Dollars

Growing the matrix

	US Sales	**Canadian Sales**
Calculate freight	Use UPS rates	Use FedEx rates
Verify address	Use US postal rules	Canadian postal rules
Calculate tax	Use state and local taxes	Use GST and PST
Currency	US Dollars	Canadian Dollars

In practice . . .

- It does not always play out so nicely . . .
- . . . but that is not a Bad Thing!
- The analysis matrix can help point out gaps in the requirements
- For example, in Canada the maximum weight FedEx will handle is 31.5 kg . . .
- . . . now you can ask your customer if there is a maximum weight for US shipments (they probably forgot to mention this)

Extending the design

- As you add new countries, you may uncover further variation, adding new rows
- For instance, shipping to the Netherlands means:
  - handling a new currency
  - but also handling a different date format

Working with customers

- Customers know the problem domain extremely well
- They do not think like Computer Scientists or Information Scientists – they talk about specific examples
- When they say *always*, they mean *usually*
- When they say *never*, they mean *rarely*

Organizing information in this matrix can guide customer interaction

Identifying rules

The different *rows* correspond to different implementations of some general concept:

- Calculating shipping rates
- Formatting dates

Implementing this variation can be done in different ways:

- The class itself may be responsible for the variation, such as formatting Dates
- You may want to introduce a design pattern (such as the Strategy, Bridge, or Decorator) to encapsulate the variation

Isn't this the same as CVA?

- Yes and no
- Both a CVA and Analysis matrix identify variation in your domain, but the focus is very different:
- A CVA is 'domain-model driven' and classifies conceptual classes according to their commonality/variation
- An analysis matrix is 'use-case driven' and structures the available information – and helps identifies missing information
- They are closely related, but complementary techniques

So far for the Analysis matrix

Final project: Mankala

Final project: Mankala

- Mankala is a (family) of board game where two players move pebbles between the various ‘cups’
- There are many variations on this game, each with slightly different rules, board sizes, winning conditions, etc
- The final project consists of designing a game for playing two games, Mankala and Wari, in C#

Final project

This project brings together all the material we have covered:

- **Analyis:** domain models, GRASP principles, CVA, Analysis matrix
- **Design:** UML, design patterns

It is not an easy problem – I think you'll be surprised at how much you've learned

Word of warning

In my experience, students can get bogged down in the details:

- How to display the welcome screen?
- How will the GUI work?
- How will the pebbles get drawn?

. . . while the real issues (finding variation and encapsulating it; coming up with a design that can actually be implemented) are ignored

Separate key design issues from the cosmetic detail

Material covered

- Design Patterns explained: chapter 12-15