Exploratory Modeling

Andreas Tönne – Georg Heeg eK

ESUG 2007

Lugano, August 27th 2007

Overview

- **Part 1**
  - **Successful Smalltalk projects and their reasons**
- Part 2
  - (A little bit) modeling theory
  - Example project (-> SAP)
  - Modeling challenges
- Part 3
  - Exploratory Modeling

Project Woes

- Chaos Report 1995 (Standish Group)
  - 30% canceled
  - 52% cost more than 190% of estimate
  - 16% on time and budget
- It got better the following 10 years!
  - 15% canceled
  - 50% cost more than 43% of estimate
  - 51% challenged (budget, time, features)

Successful Smalltalk-Projects

- Authors personal experience (1997-2007) with Georg Heeg
  - 0% canceled
  - 33% (5/15) challenged (mostly budget and time)
    - Feature driven challenges
- Very satisfied customers
  - Trust that the solution will be good
  - Believe that no showstopper problem exist

Successful Smalltalk-Projects

- A few names…

- ABB
- AMD Dresden (-> Taylan’s talk)
- Commerzbank
- Debeka
- GEFA
- SAP (-> Ralf Ehret’s talk)

Successful Smalltalk-Projects

- Common characteristics:

- Very long living solutions
- High resistance against replacement
- High customer satisfaction
- Solves „special“ customer problems

Customer Satisfaction

- Intuitive reason: good models
  - Models capture customer requirements correct and complete (deep models)
- Excitement created by:
  - Combination of models
    - Unforeseen additional value
  - Robustness of models
    - Models stable against feature changes
    - Features change often, correctly modeled domain concepts rarely
    - No “can’t do that”, “this will be expensive”, “the design does not support this”

Good Modeling in Smalltalk

- Observation:
  - Smalltalk proved to be extremely good at expressing domain concepts in model implementations! Why?
- Simple answer:
  - Because the language is so nice
  - Conceptual classifications (abstraction, generalization) are immediately expressible
  - No implementation/usage separation by class/interface
  - Little technical overhead like types, technical declarations etc.
  - Powerful environments and tools
- This explains why **we** are happy with Smalltalk

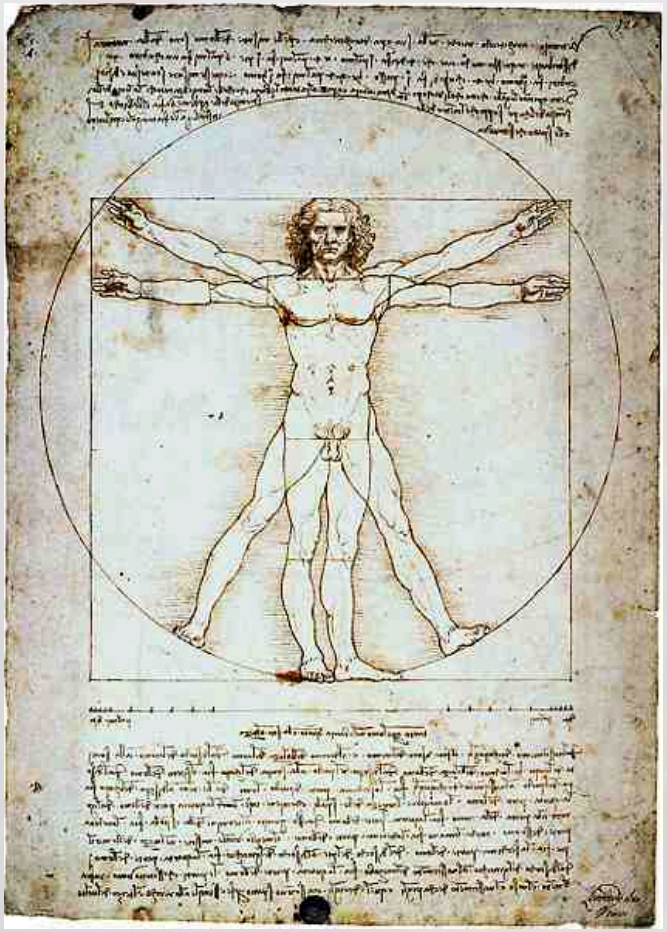
Overview

- Part 1
  - Successful Smalltalk projects and their reasons
- **Part 2**
  - **(A little bit) modeling theory**
  - **Example project (-> SAP)**
  - **Modeling challenges**
- Part 3
  - Exploratory Modeling

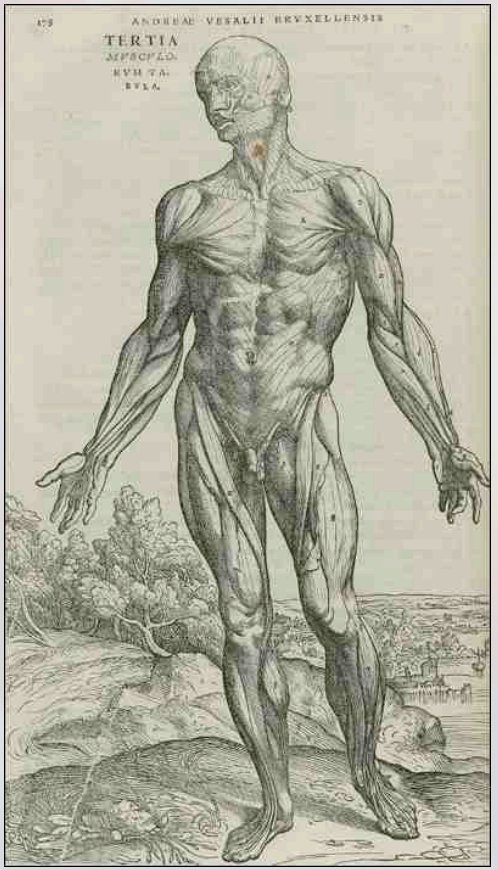
A Little Theory of Modeling

- A **model** abstractly represents a **phenomenon**
- (Stachowiak) Model criteria:
  - Mapping: original mapped to model
  - Reduction: emphasizing the important aspects by deleting irrelevant aspects
  - Pragmatic: models serve a purpose
- Models are targeted at one or more receivers
- Views of a phenomenon yield different models (different purposes of the models)

A Little Theory of Modeling



Leonardo da Vinci's Vitruvian Man
(1492). Pen and ink with wash over metalpoint on paper.



De Humani Corporis Fabrica...
Basel, 1543. Woodcut. National Library of Medicine.
Andreas Vesalius
(1514-1564)

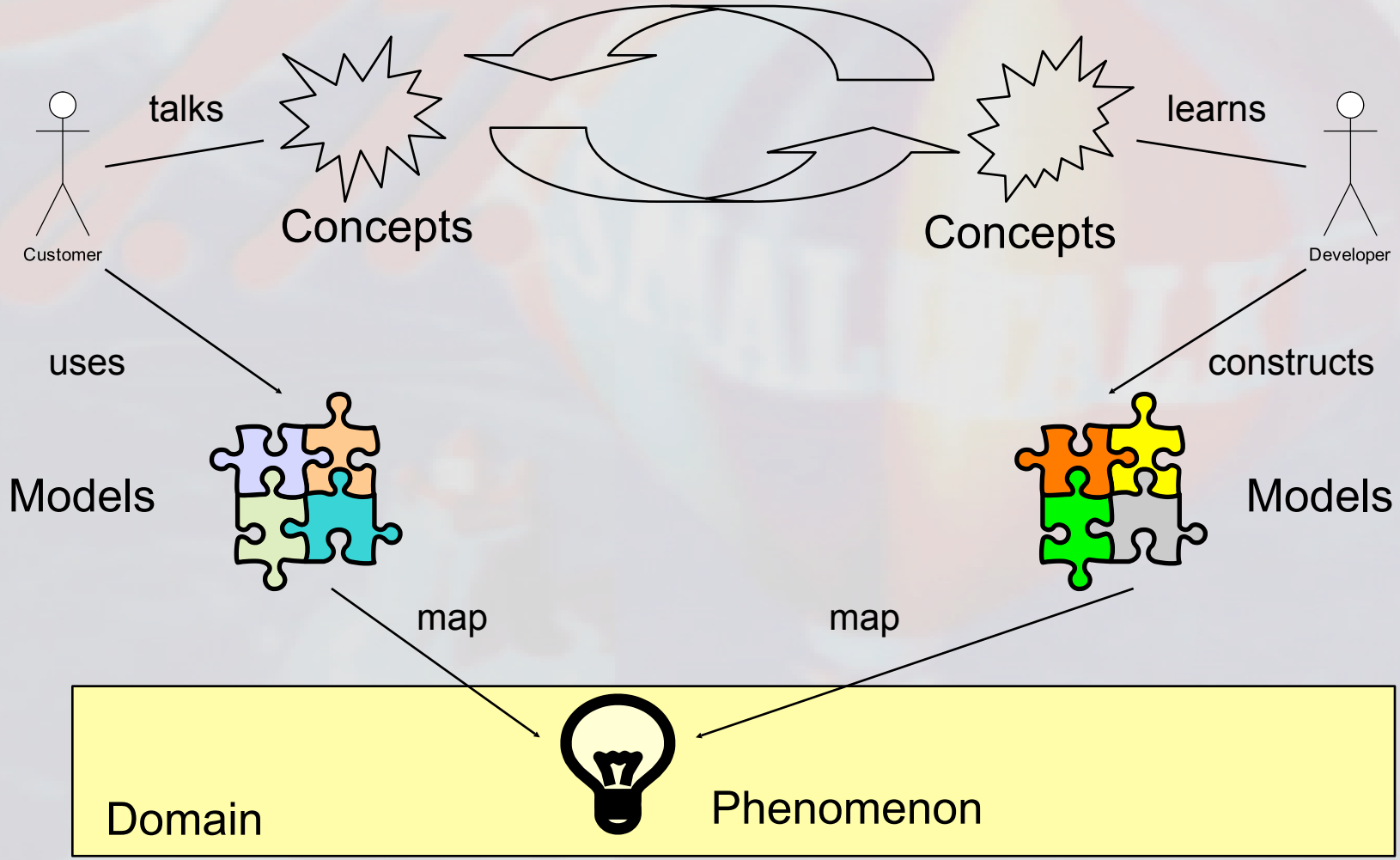
Ontleding des menschelyken lichaams...
Amsterdam, 1690. Copperplate engraving with etching. National Library of Medicine.
Portrait of Govard Bidloo (1649-1713) by **Gérard de Lairesse** (1640-1711).

A Little Theory of Modeling

- Various views/models by intention
  - For domain understanding or implementation
  - Level of detail from IT-landscape to algorithm
- Various views/models by receiver
  - Blackbox-view by functional behavior for customer
  - Whitebox-view by inner structure of concepts and their interaction for developer

A Little Theory of Modeling

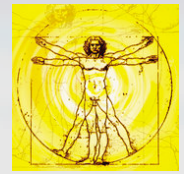
Project modeling szenario



A Little Theory of Modeling

- Primary purpose of modeling: **COMMUNICATION**
- Achieve a common understanding of the concepts
  - Vocabulary and Imagination
- Verify the model for completeness and correctness
  - **This is a hard problem!**

Practical Modeling



- What to use: UML or UML?
- Our point: UML is a good thing carried too far
  - Cannot cover communication and verification with the customer easily (details later)
- Others recognize this as well
  - Microsoft's domain-specific languages

Modeling using Smalltalk

- Smalltalk is extremely good at expressing domain concepts in model implementations! Why?
- Modeling answer:
  - concepts are represented 1-1
  - **Colloquial use of concepts is expressible in the model implementation**
- Important consequence:
  - **Complexity of concepts and model implementation match**
  - Complexity of change to concept and model implementation match
- View Smalltalk as a kind of generic domain-specific language

Customer Satisfaction (rep.)

- Excitement factor: good models
  - **Models capture customer requirements correct and complete**
- Excitement created by:
  - **Combination of models**
    - Unforeseen additional value
  - **Robustness of models**
    - Models stable against feature changes
    - Features change often, correctly modeled domain concepts rarely
    - No “can’t do that”, “this will be expensive”, “the design does not support this”

Overview

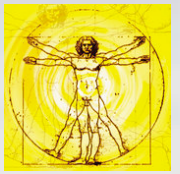
- Part 1
  - Successful Smalltalk projects and their reasons
- **Part 2**
  - (A little bit) modeling theory
  - **Example project (-> SAP)**
  - **Modeling challenges**
- Part 3
  - Exploratory Modeling

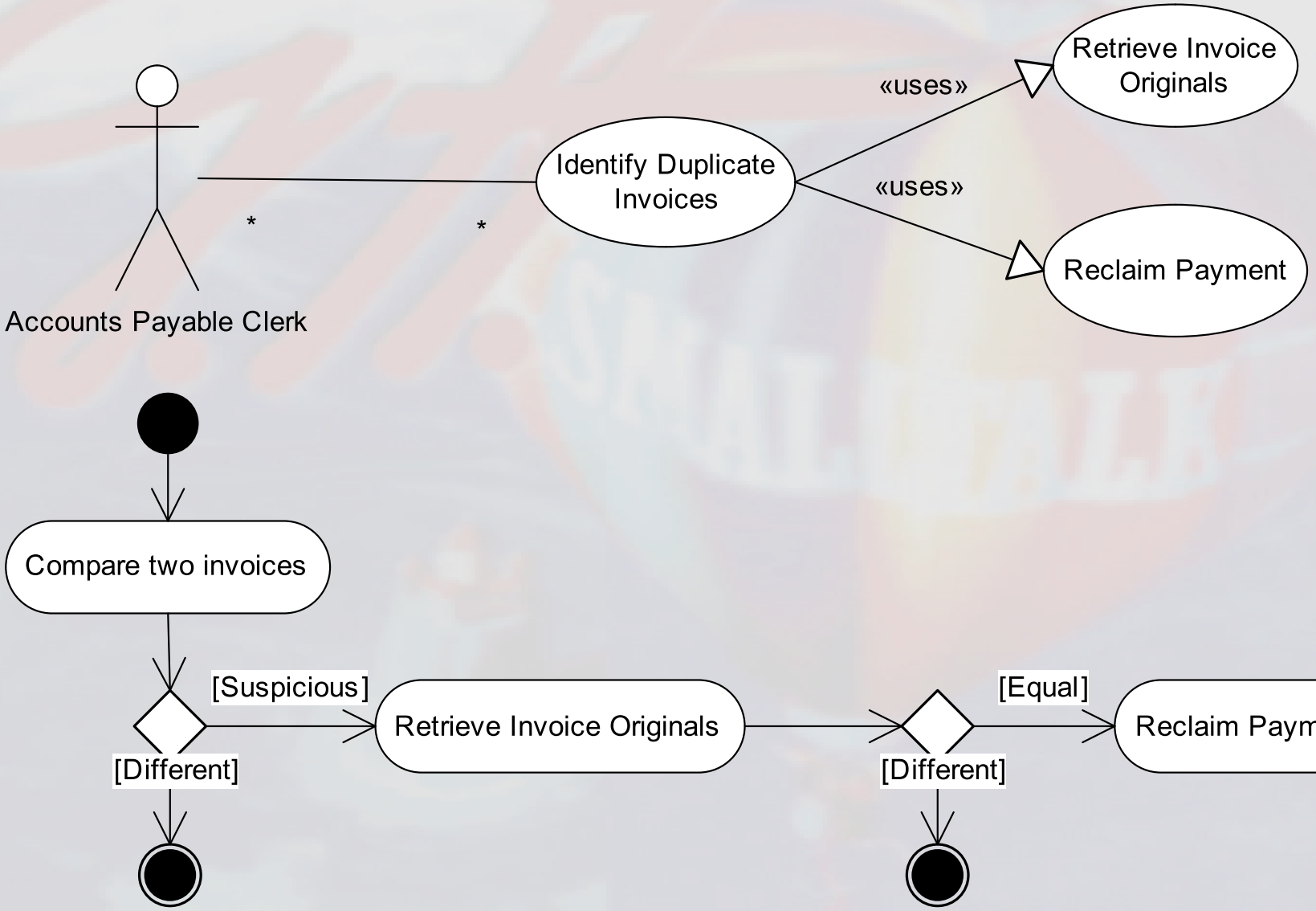
Example: Duplicate Analyzer

- Situation: Company with invoice handling based on standard software
- Expert: Accounts payable clerk
- Task: Identify duplicated paid invoices
- Goal: Reclaim unwarranted payments

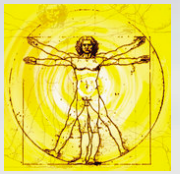
- Show of hands: Is this an easy task?

Duplicate Analyzer – First Model

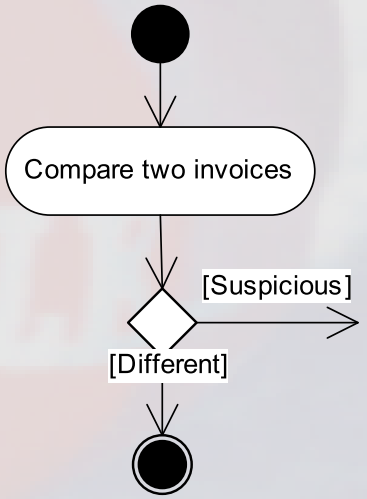




Duplicate Analyzer – First Model



- Too simple analysis model
- Does not even describe the task correctly
- Hidden (unspoken) goals



- Use-case needs refinement

Duplicate Analyzer – Fake Interview

-> To start with, what is an invoice and how does it differ from the invoice originals? Can't you always work on the originals?

<- That would be too time consuming. And the originals are just scans.

-> So what is an invoice and how do you get it?

<- The invoice is coming from the finance system.

-> What is this invoice?

<- "shows a dialog of the finance systems"

-> By what criteria do you choose invoices for comparison?

<- ?

-> You get a list of invoices and do what?

<- Oh! I suppose I have to compare them one by one. The software should do this.

-> That is the point of this project! So you need to compare all invoices in pairs or do you compare more than two in one go?

<- No I compare two at a time.

-> Do you compare all invoices with all other invoices?

<- That takes too much time. I only compare those that are interesting and potentially duplicate.

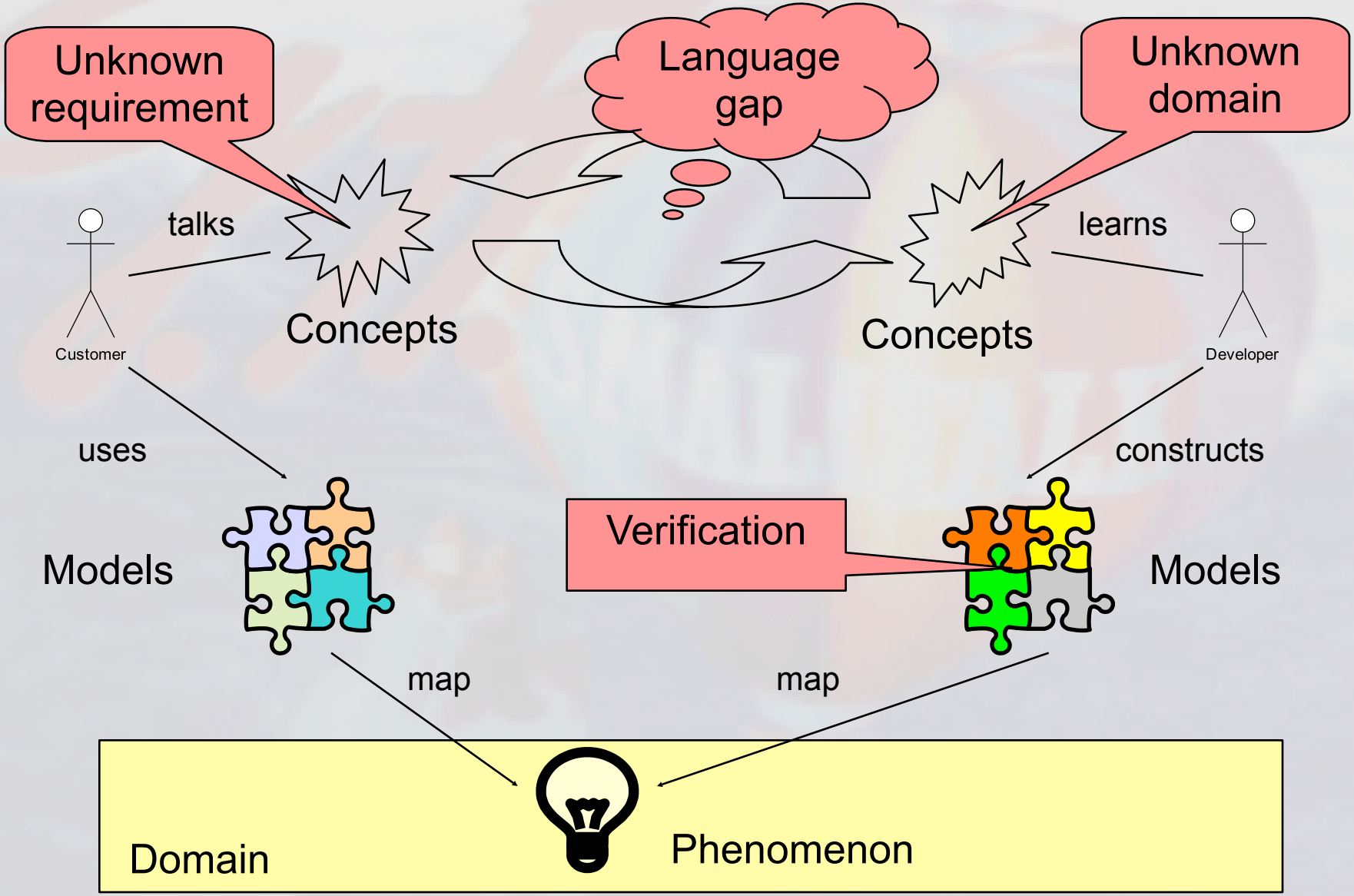
-> How do you determine these invoice pairs?

<- ? I look at them and see.

-> So you have some rules by which you know it is worthwhile looking a bit closer?

<- ?

Challenges for Modelling



Challenges - The Unknown

- Requirements are expressed in terms of needs and examples
  - Lack of formal coherence
  - Lack of abstraction
- Understanding the domain
- Moving targets
  - Fixing requirements and understanding changes the goals.

Challenges - The Unspoken

- Language gap!
  - Different way to express models
  - Developer: formal, abstract, seeking generalizations
  - Customer: more informal, example (process) based, individual use cases
- Need a common language for the concepts and for talking about the modeling process
- UML considered not appropriate!
  - Formally adequate for the developer
  - But not matching the language of the customer

Challenges – Language gap

Two models of a woman with a hat. Seriously clashing language!

Challenges – Customer Assurance

- Often overlooked: achieve a mutual agreement on the model
  - Captures the customer needs
  - Can be handled by the developer
- Why does the customer agree that a model captures his needs?
- **VERIFICATION**!

Challenge Summary

- Find a modeling process that
  - Expresses the domain in enough formal rigor
  - Can be understood by the customer
  - Produces agreeable proof of the model
  - Is fast enough to take place at the speed of communication
- Get right what the model does instead of how it is written up

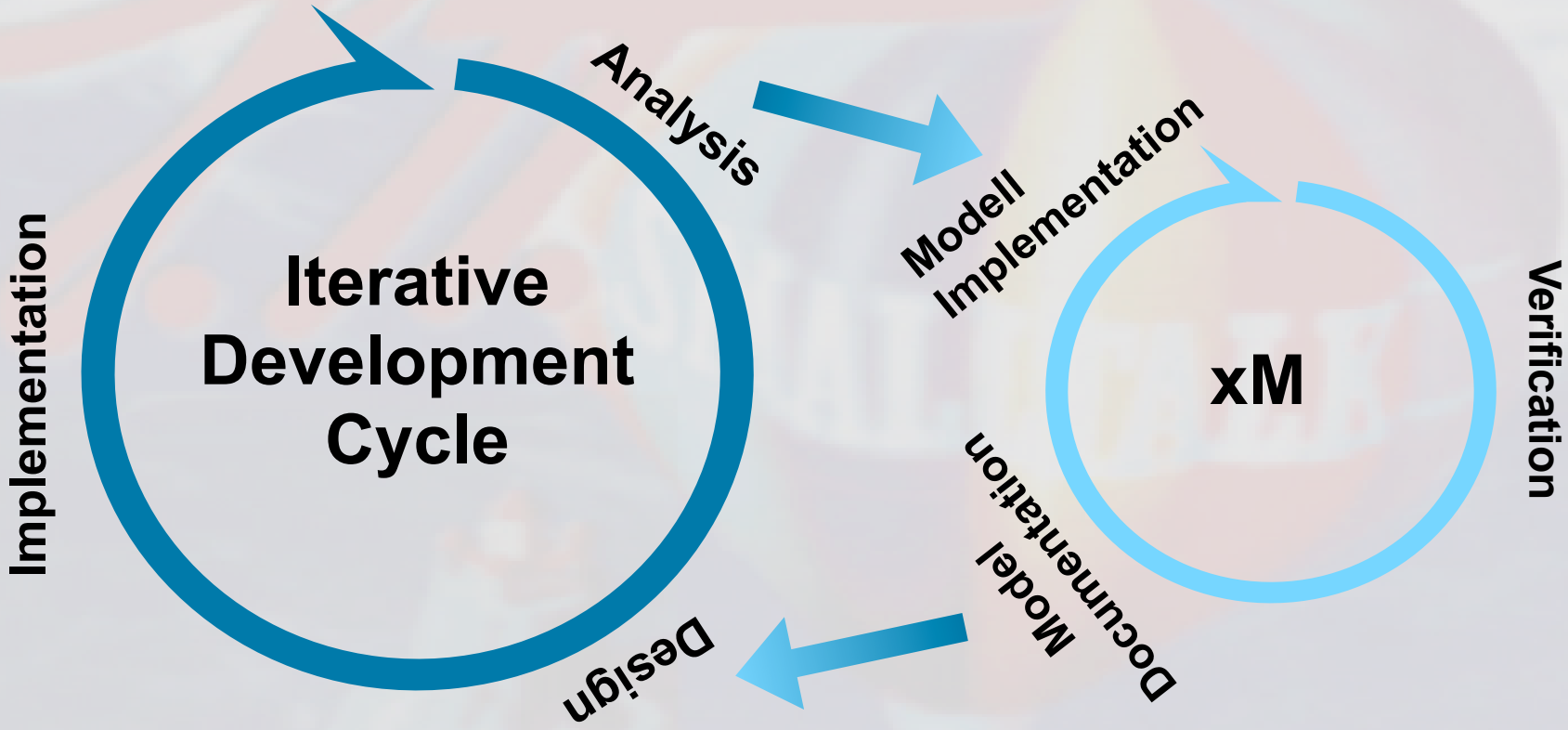
Overview

- Part 1
  - Successful Smalltalk projects and their reasons
- Part 2
  - (A little bit) modeling theory
  - Example project (-> SAP)
  - Modeling challenges
- **Part 3**
  - **Exploratory Modeling**

Exploratory Modeling - Idea

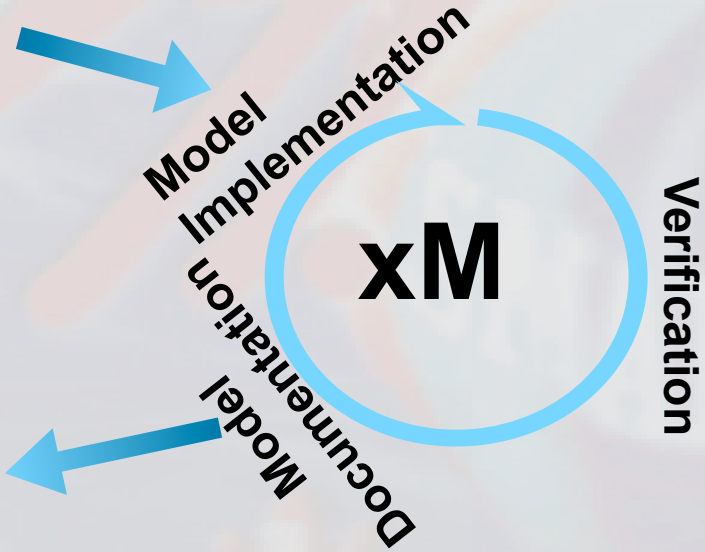
- Models should communicate to the customer **and** the developer
- Whitebox models in UML
  - Too technical for many customers
- Blackbox models like prototypes
  - Not formal enough for developer
- We like to have the cake and eat it too!

Exploratory Modeling



Exploratory Modeling in Detail

1. Build model in suitable programming language



2. Conduct experiments for verification

3. Document model results

➔No more “this is not what we intended”

Exploratory Modeling Step 1

- Express problem domain in a suitable programming environment
- Rules:
  - Use the language of the customer
    - Simplest possible implementation
    - As “non-technical” as possible
  - Make the model executable
    - Add experimentation environment

Exploratory Modeling Step 2

- Model is implemented for experiments
  - Formal rigor given by implementation
  - Experiments verify consistency
  - Experiments conducted by implementer and customer together
  - Customer has immediate feedback that the model is right
  - Experiments achieve deepening of the model!

Exploratory Modeling Step 3

- Create model documentation
  - Implementation expresses model exactly
  - Less technical, visual documentation is better
  - UML is fine for documentation

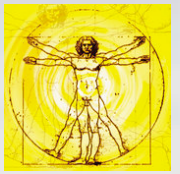
Exploratory Modeling Result

- No prototyping!
  - Running program is not enough
  - The model should be expressed in the implementation
  - Allows for modeling cycles
- Blackbox view: customer can verify the correctness of the model
- Whitebox view: formal model in UML for the developer
- Rigorous application of xM assures that both views are in sync

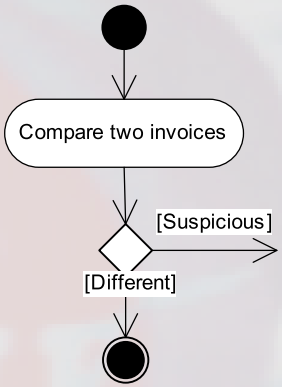
Exploratory Modeling Language

- Need programming environment with certain agile qualities
  - Non-technical, barrier free, scripting style
  - Meta-programmable
  - Concept-oriented
  - Interactive
- Only a few languages are flexible and powerful enough for exploratory modeling
- Smalltalk is optimal for its modeling powers

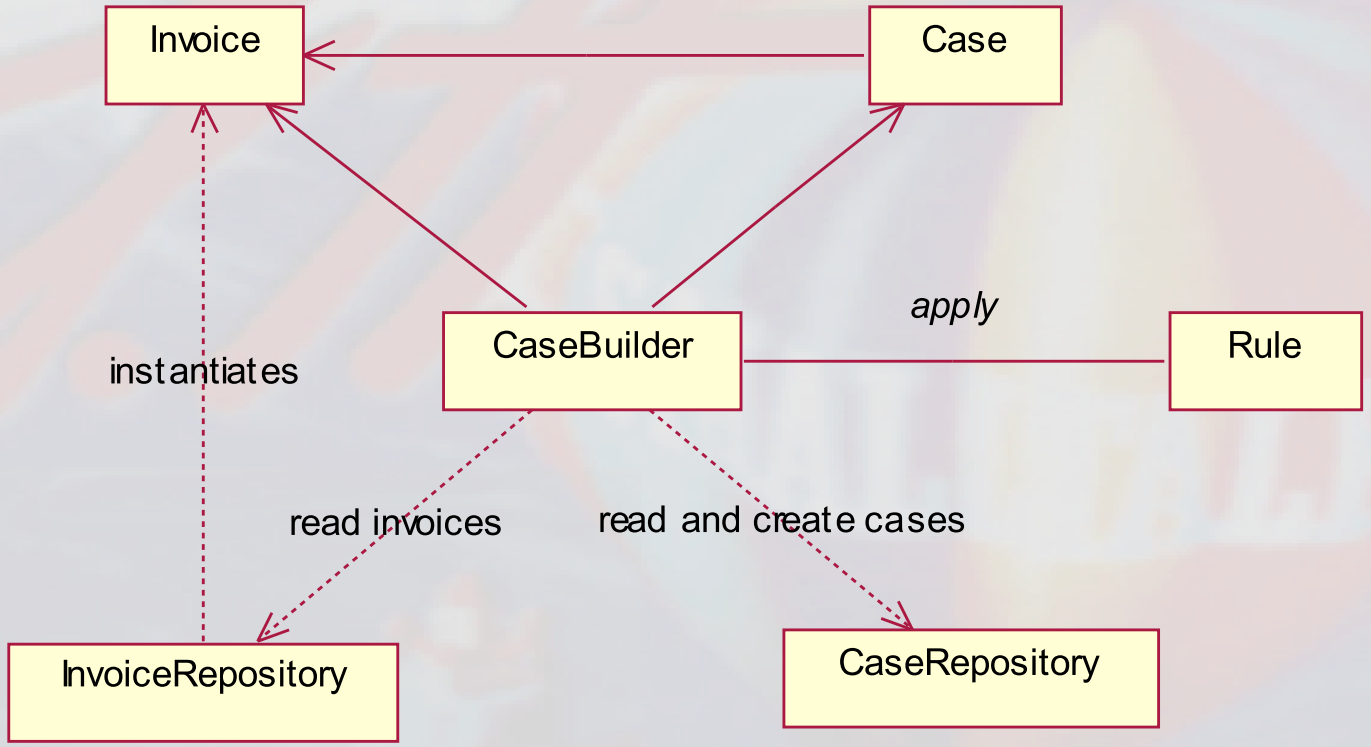
xM Example



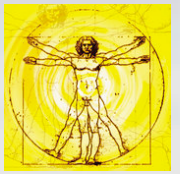
- Three modeling cycles for duplicate analyzer
- Model implementation embedded in an experimentation workbench
- See SAP talk of Ralf Ehret

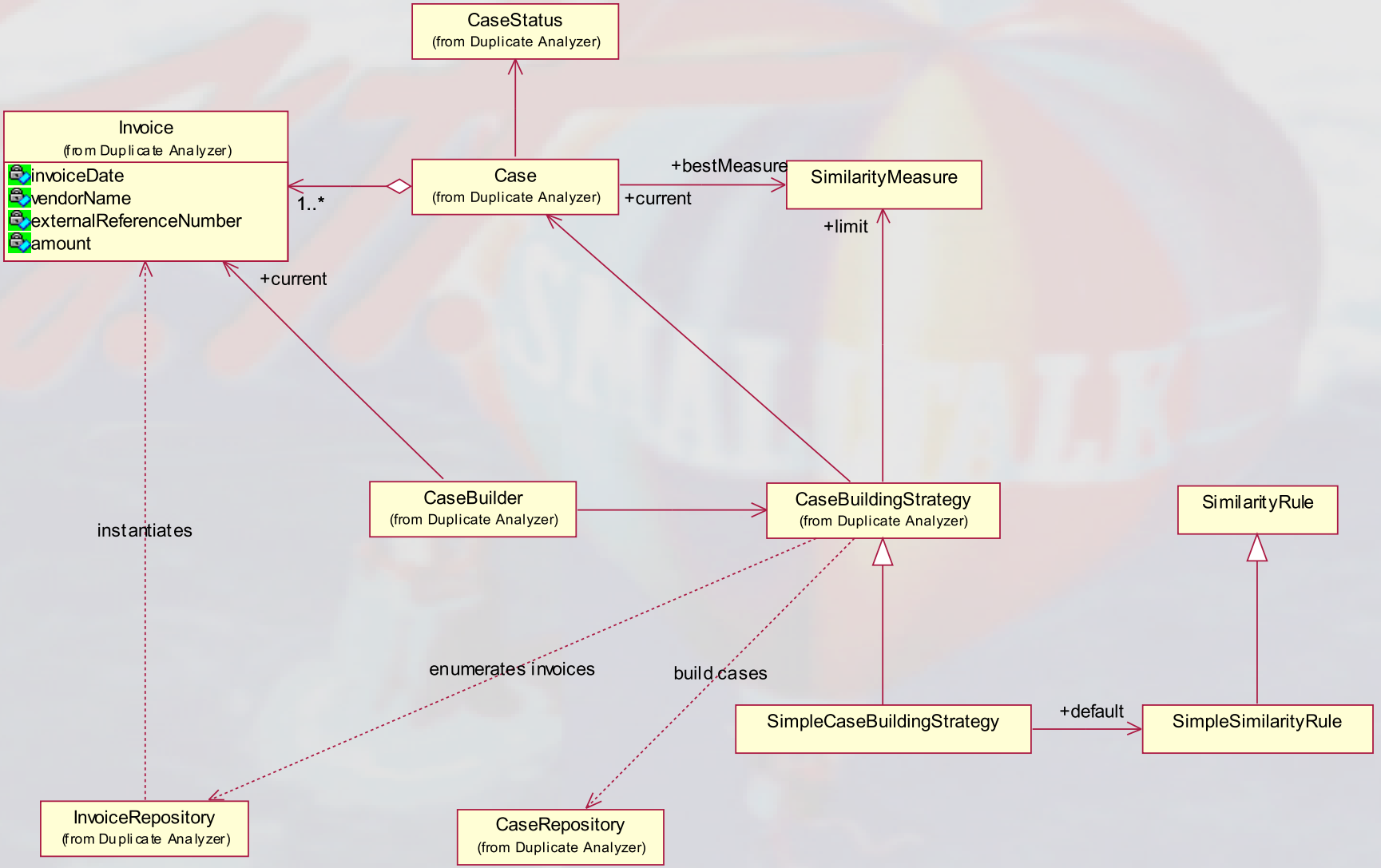


xM Example – Cycle 1

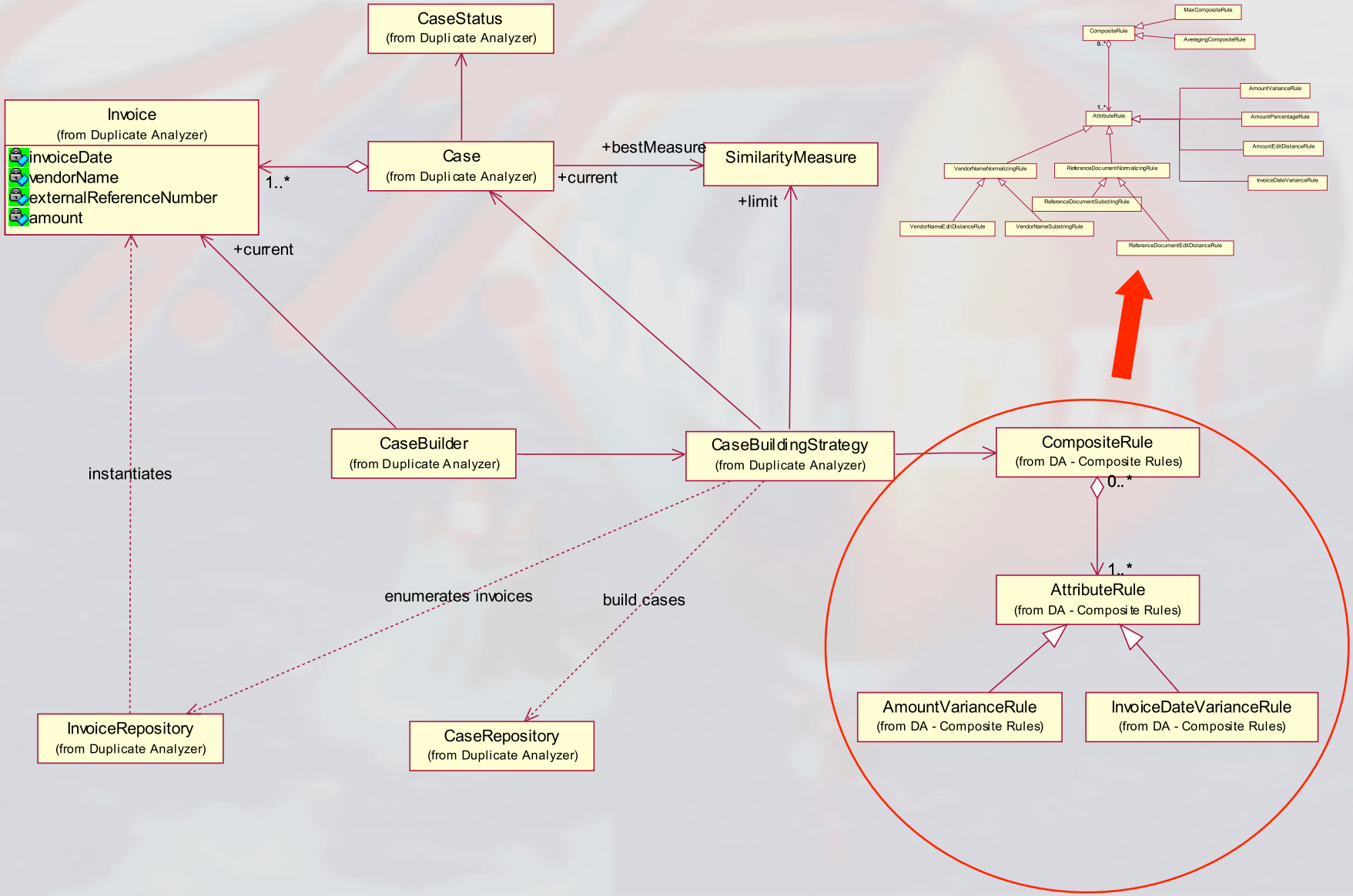


xM Example – Cycle 2





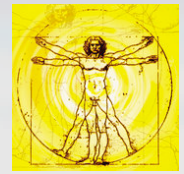
xM Example – Cycle 3



xM Example - Results

- “Modeling experience” means to model the clerks knowledge of reasons for duplicates
- Variety of modeled duplicate reasons
- Finding suspicious invoice pairs by applying customer specific set of duplicate models

xM - FAQ



- When can I apply xM?
  - If there are modeling challenges that are expensive, risky or unsolvable.
- May I keep the xM implementation?
  - A good Smalltalk model implementation is a good start for the product implementation.
- We have to use language X for the product. Can I still use xM?
  - Yes! The model documentation is the produced value. A great opportunity to introduce Smalltalk.

Summary

- xM is a modeling process that combines
  - Smalltalk as a modeling language
  - Continuous experimentation with runable Smalltalk models
  - Model documentation accordingly to the project requirements
- Results in high quality, verified models
- Good customers reassurance early in a project