



© 2004 Valtech – All rights reserved

Pragmatic approach to interconnected requirements, analysis and design with UML

A Pragmatic approach to interconnected requirements, analysis and design with UML

- Your system is not “too complicated to design”
- You cannot reliably go from:
 - Discussion to Code.
 - Use Case to Code.
 - Use Case to technical design.

(What about business flows and modeling?)
- There is a very clear path from Use case to Deployment
 - Every Artifact / document builds on each other
 - Design only what is necessary – don’t design for design’s sake
- **“The significant problems we face cannot be solved by the same level of thinking that created them.” – Albert Einstein**

- **We design so that business and technical people can use a language to communicate with each other (UML)**
 - Most UML diagrams make sense at first glance with little explanation

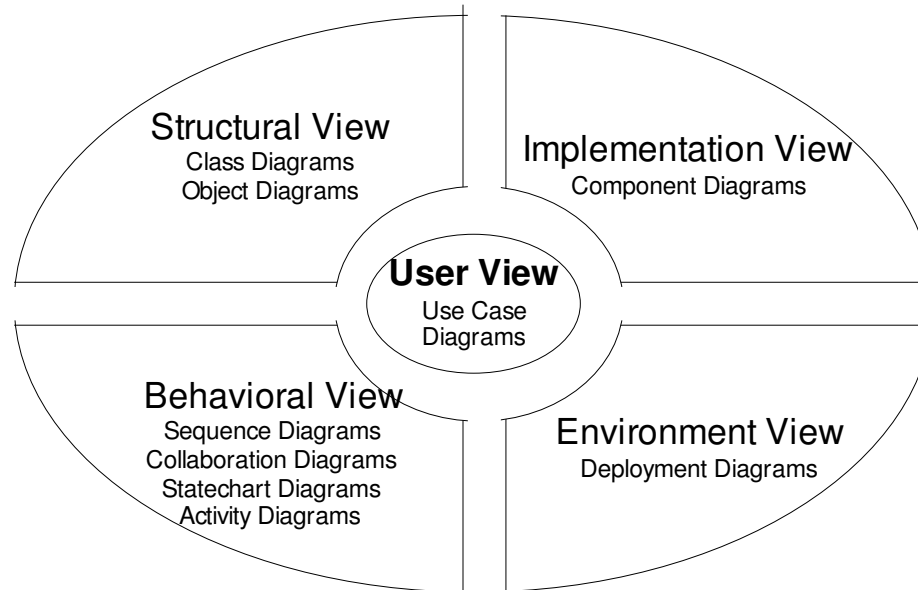
- **We design because it helps us convert**
 - Requirements into tangible business flows and conceptual Models (Behavioral view)
 - Business flows and Conceptual Models into code level diagramming (Structural views)

- **We design because it Documents our system for**
 - Maintenance
 - New team members
 - Change requests
 - Writing test cases
 - Team collaboration

- Enough fluff –
- We are going to design an ATM banking system from vision to deployment (from an ARCHITECT's point of view)
- Demonstrate how each piece of requirements and design build upon each other
- Explain each type of diagram as we approach it
 - Where to use it
 - What it is used for
 - When to use it
 - How to use it

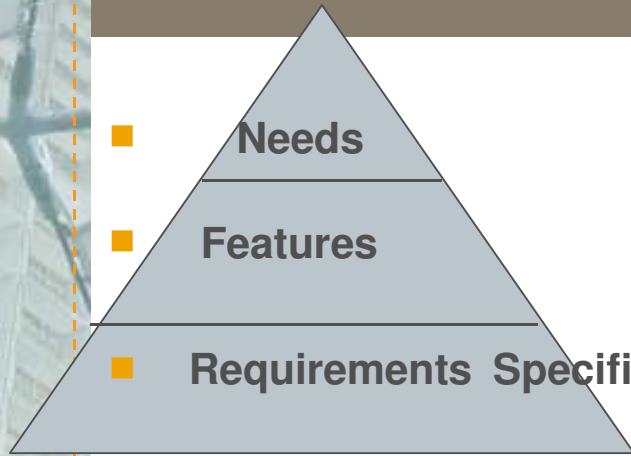
“If you can't describe what you're doing as a process, you don't know what you're doing.” – W. Edwards Deming

- From O'Reilly's book – UML in a Nutshell – 1999



Unclear – When defining the behavior of the system,

- *Where do we represent the business flows?*
- *Do we go straight to code design?*
- *Where is the demarcation between Business and technical? Is it a gray area?*

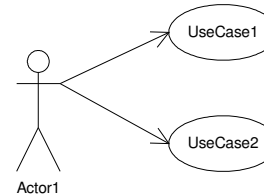


Vision of business

make money, save money, regulation, etc

Broken down into a list of “what” is wanted

➤ High Level Use Case Diagram
and Business Rules

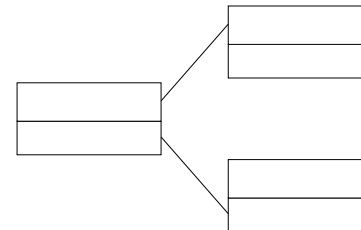


➤ Elaborated Use Case

Main Flow

- 1) Actor does something
- 2) System does something

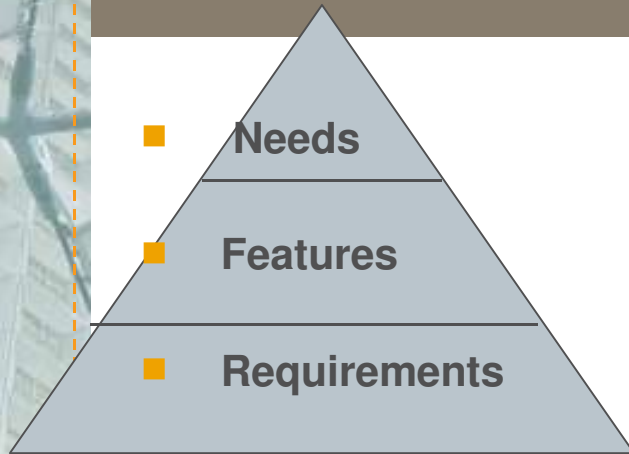
➤ Conceptual Model



Alt Flow1 – (etc)

- 1) Actor does something
- 2) System does something

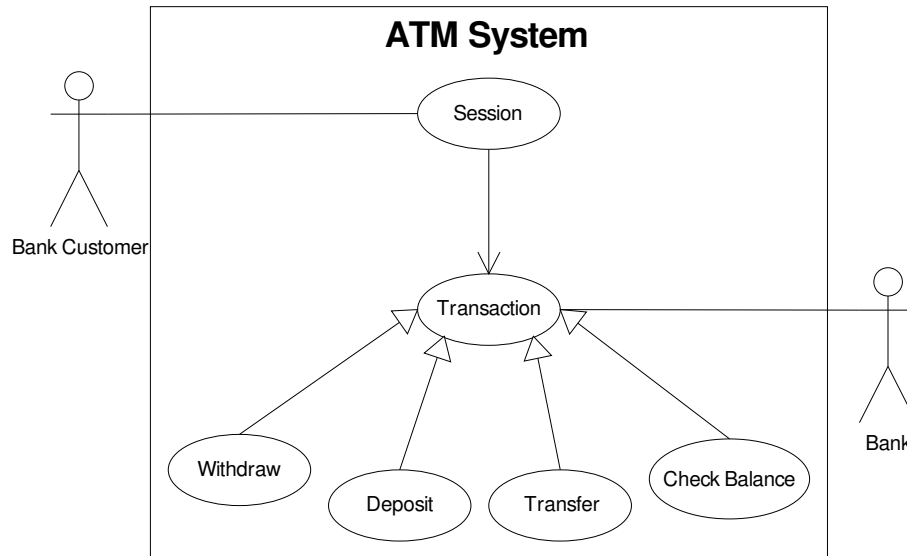
Requirements of ATM



Vision: We need an ATM system that our bank customers can use for drive-up or walkup banking outside the bank.

Features: Provide a way for bank customers to:

- Withdraw cash from machine
- Login with ATM Card and remain connected until card removed
- Deposit cash to machine
- Transfer funds from accounts
- Check balance of accounts



Requirements:

- High level use case diagrams to identify use cases.
- Initial use cases are usually a pretty close match to features.
- Identify Actors – Bank customer / Bank

Business Rules:

- Bank Customer cannot withdraw more than \$200/day from ATM

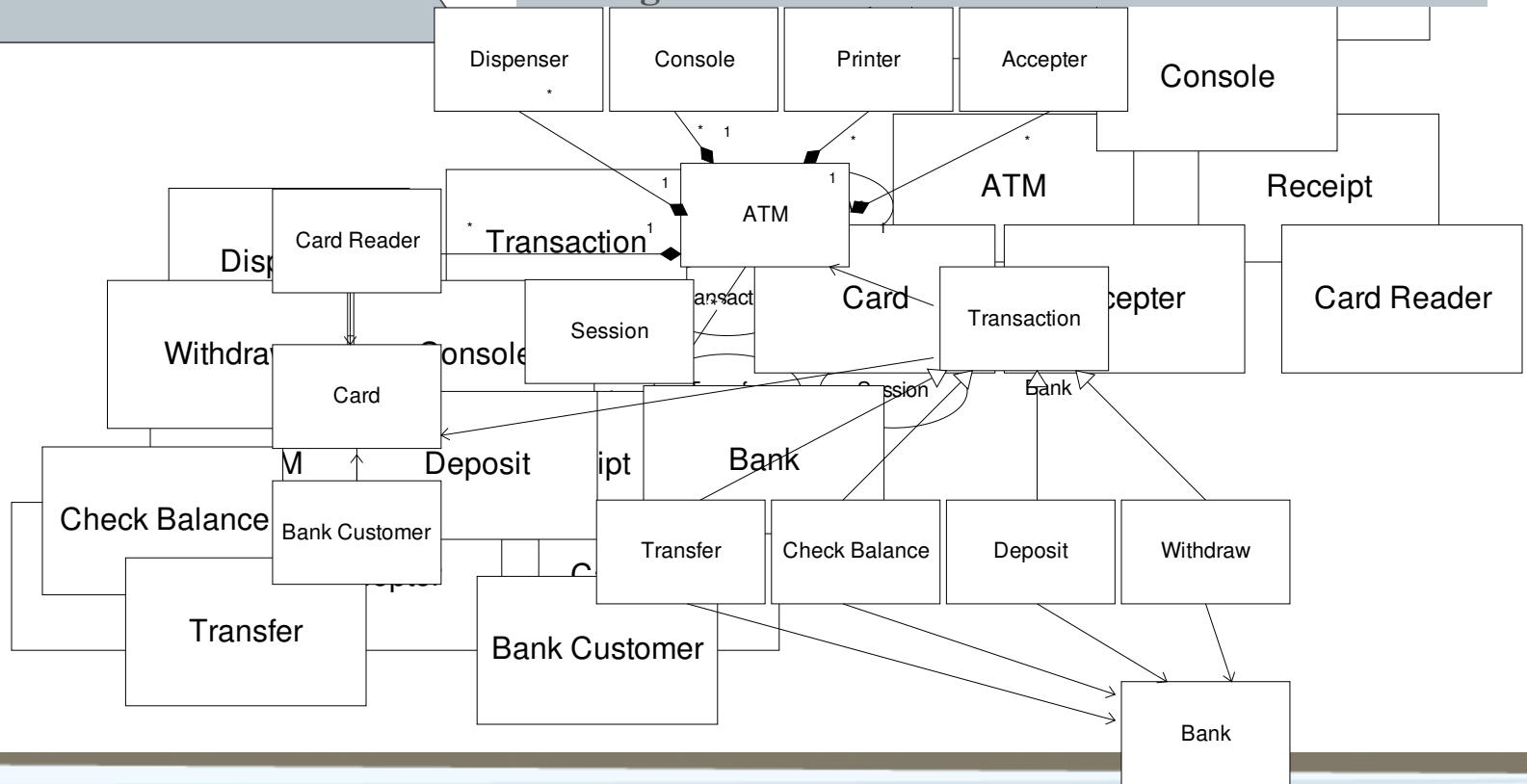
Requirements of ATM

- Needs
- Features
- Requirements

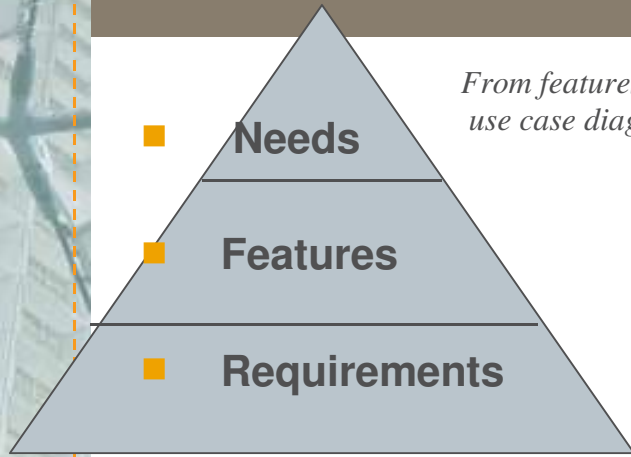
Conceptual Model: What are the “Parts” of the system which make up the whole?

Tip) Nouns and Verbs:

Take the nouns from the requirements as domain objects, take the verbs as sequence diagram messages.

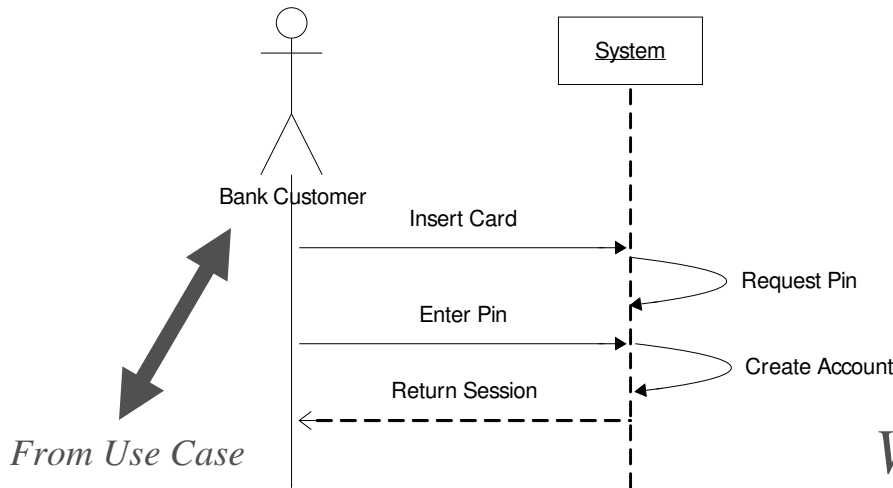


Elaborate Requirements:



From features, business rules, use case diagrams to →

YES they match!



For each use case in diagram . . . Happy path!

“Create Session UC1” – flow of events

3. Bank Customer (BC) inserts card
4. System request pin
5. BC enters pin
6. System creates session per the users account

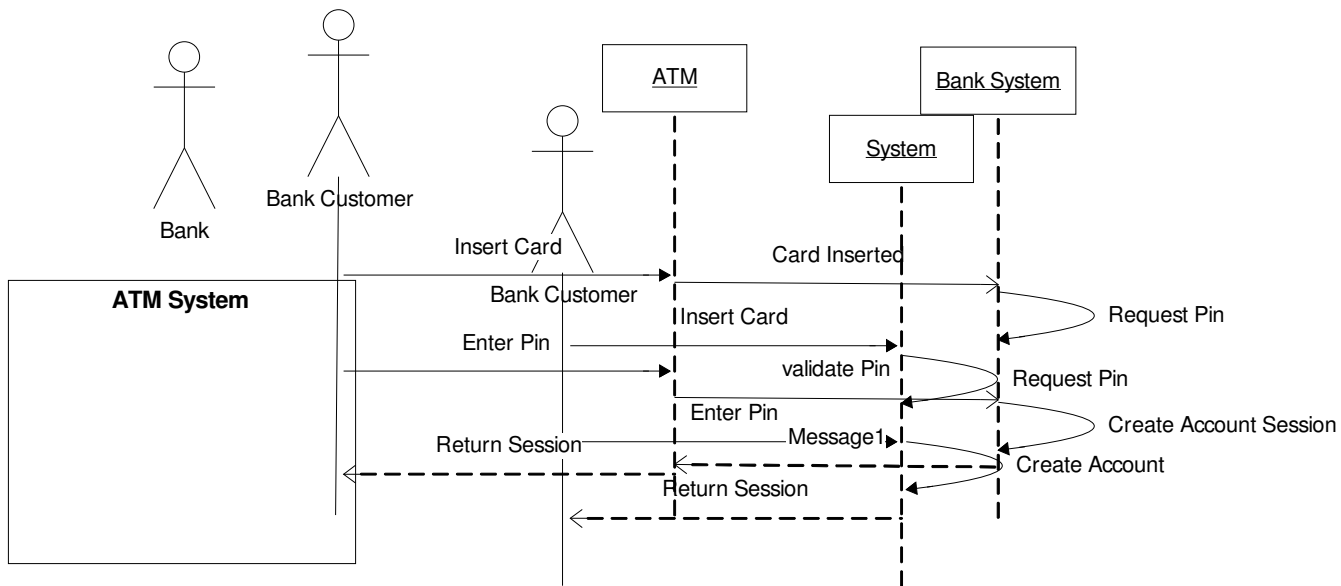
Identify Alternate flows - where they begin and end

“Create Session UC1” – flow of events

3. BC enters pin
4. System identifies invalid pin and ejects card
5. UC ends

*Why are they the same?
Is this wasted Work?*

- Start with your System Sequence Diagram from the “flow of events”
- Add your “ATM System” from the use case (ATM), and the external actor (“Bank” system)



- Add the few things that related to the “ATM” and “Bank System”

Recap – can we code yet?

YES!!

Few disclaimers

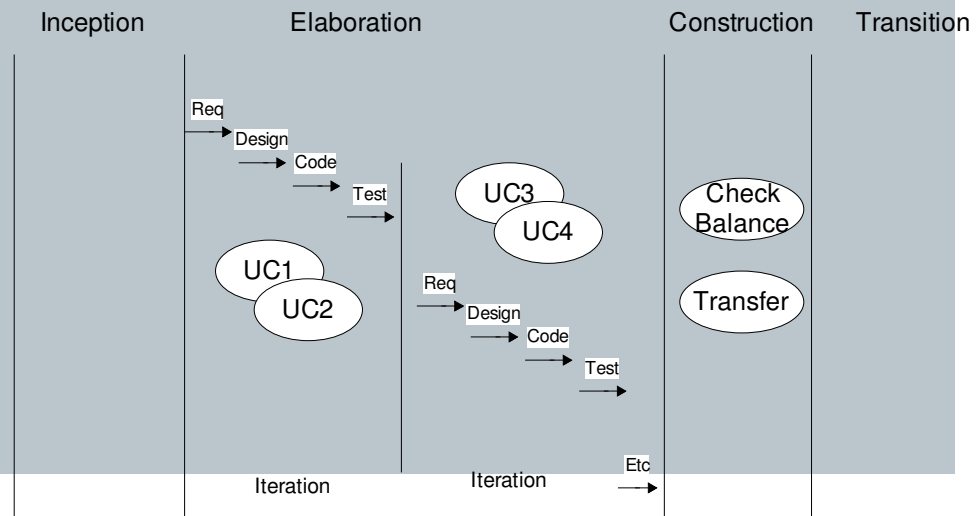
- > Based on something like the Unified Process, there are phases like:
Inception, Elaboration, Construction, Transition
And TIMEBOXED iterations in those
- > When we started, we created high level use cases (UC diagram) to get our arms around the project (and maybe guess at the estimate)
- > Then took the highest risk use cases (biggest problems, might not work, etc.)

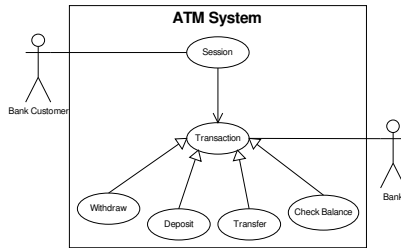
> A few UCs at a time

- Elaborate Requirements
- Design them
- Code them
- Test them

> Why iterate?

- Things change
- People change their mind
- Some things don't work



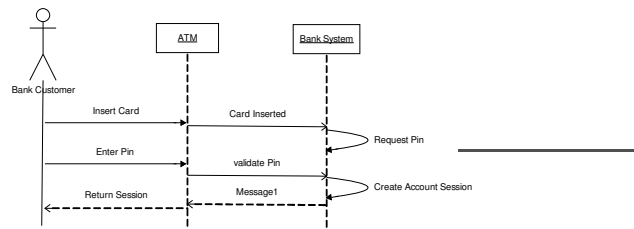
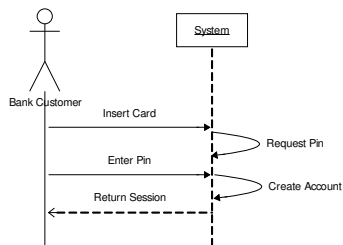
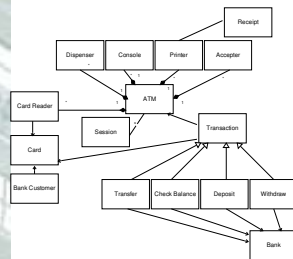


Main Flow

- 1) Actor does something
- 2) System does something

Alt Flow1 – (etc)

- 1) Actor does something
- 2) System does something



Where are we at?

- UC diagram(s)
- Elaborated UC (1st one)
- Conceptual diagram of whole system
- System Sequence diagram of 1st UC (and elaborated system sequence diagram)

- We could start coding right now, judgment call of Architect -

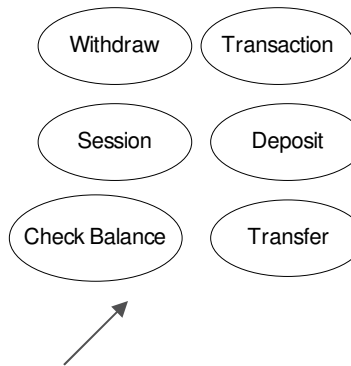
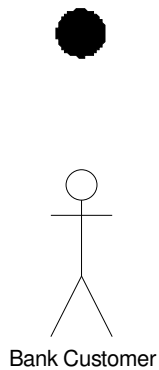
Need more information?

Do an activity diagram to show detailed business flow with decisions for certain use cases.

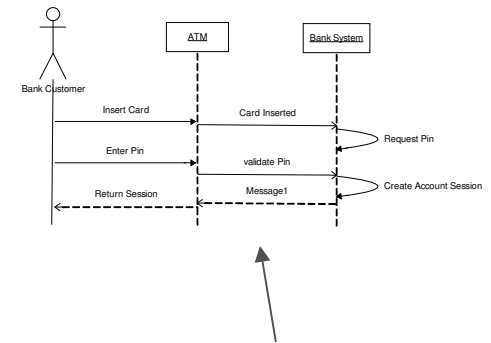
Do a state diagram to show state of ATM at any time

Let's go!

Activity Diagram – Gather the pieces!



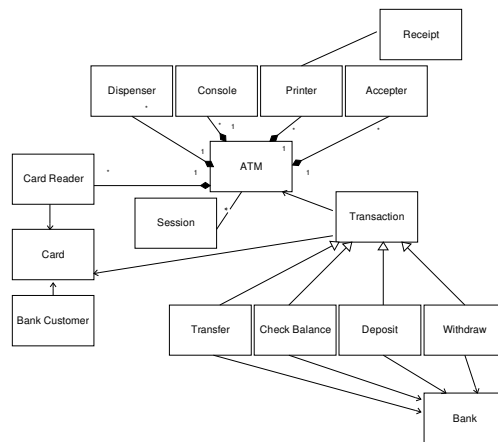
Use cases



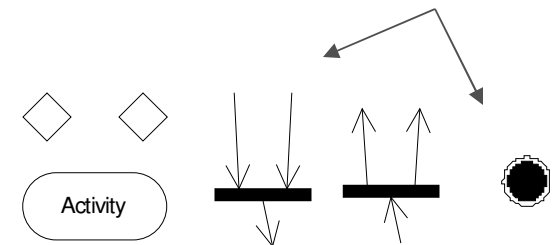
System Sequence

swimlanes

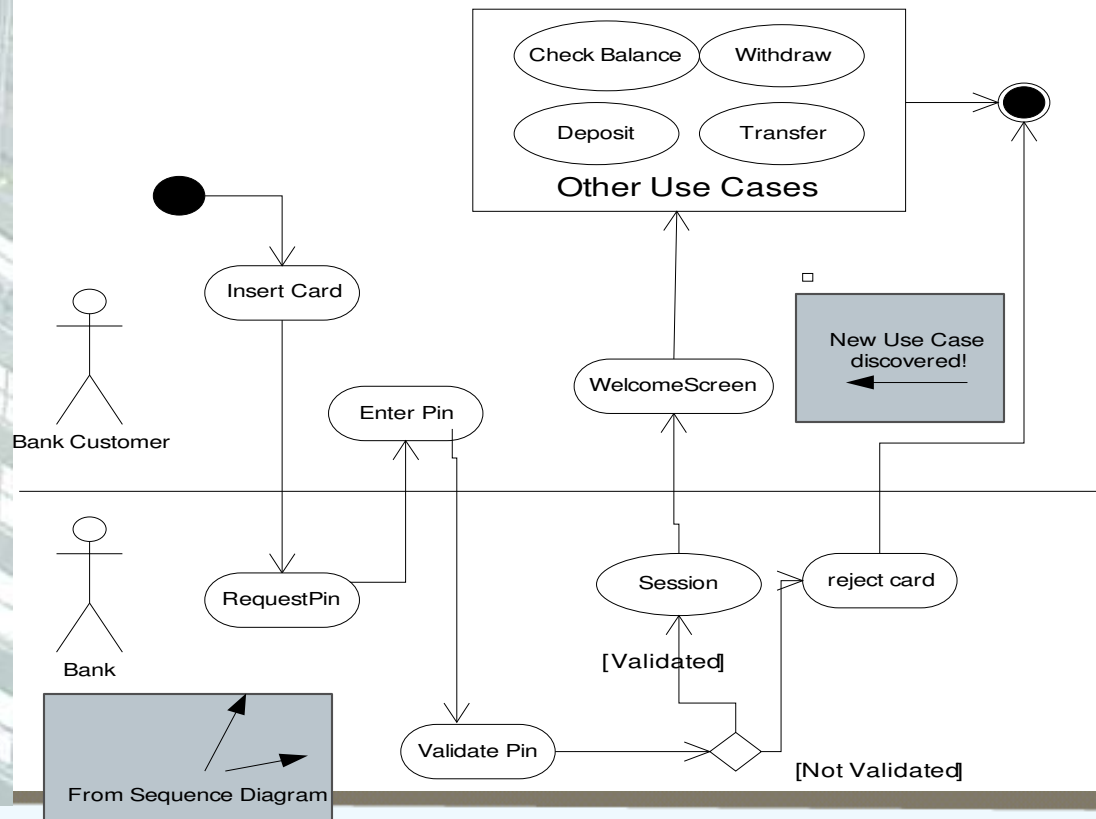
Conceptual



Activity Pieces



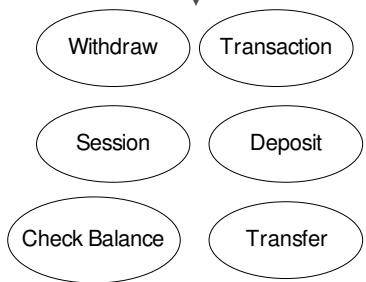
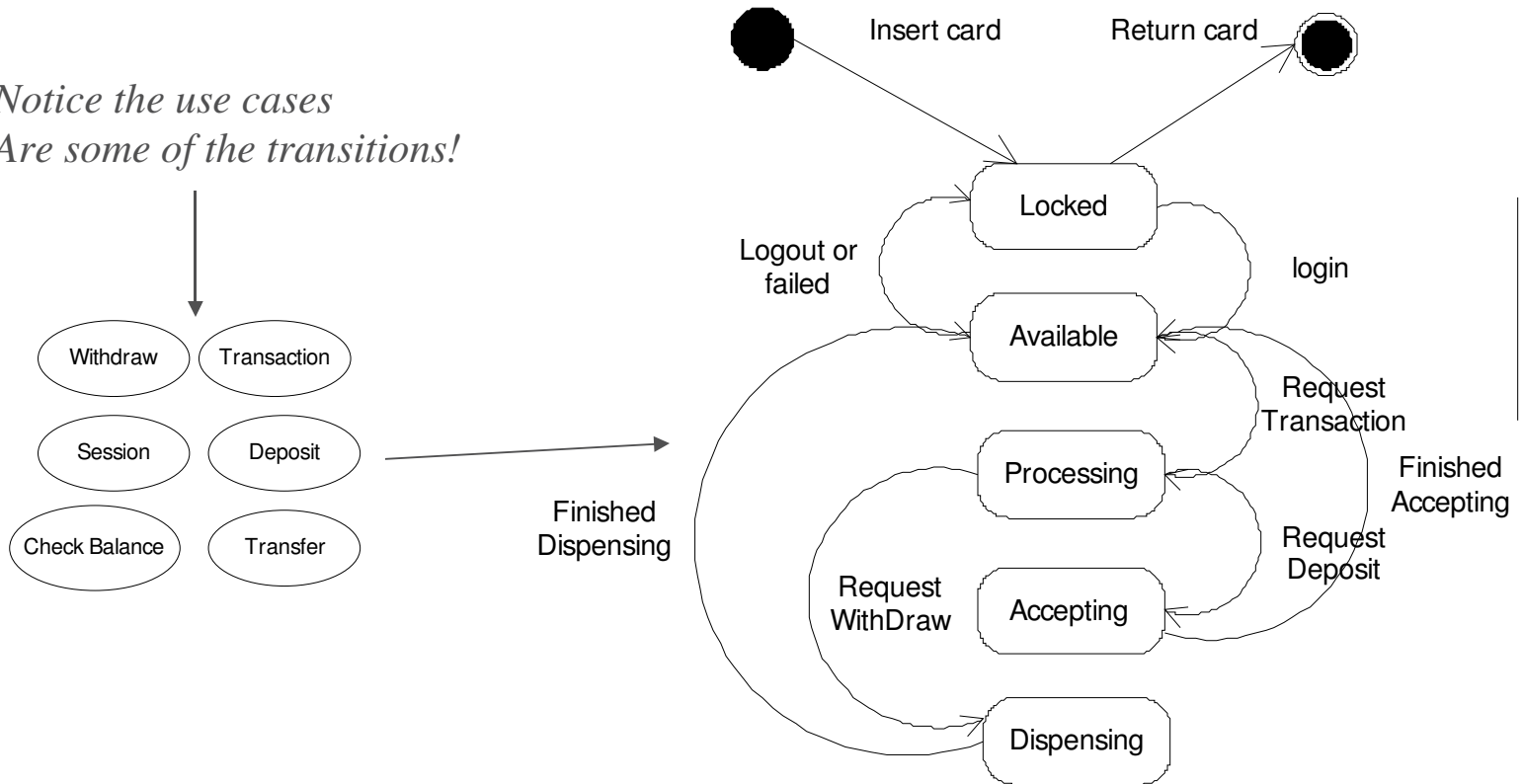
- Let's walk through it and see where we drew conclusions.
- Notice many of the messages from Sequence diagram.
- New Use Case Discovered!! - "Welcome Screen"
- Notice using "Session" use case in diagram, also referring to others.



State Diagram

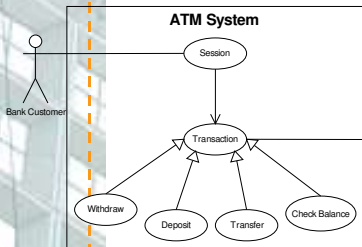
- State diagram shows the states and transitions that the “System” goes through.

*Notice the use cases
Are some of the transitions!*



How to transition to code?

- Some call this the Technical Design.
- Generally transition
 - Conceptual model to class diagram – on a new system, the first try might be a one to one match!
 - Use the classes from the Class diagram, and the UC Actors and create a sequence diagram.
 - Can also use those classes to create collaboration diagrams (very similar to sequence diagrams)

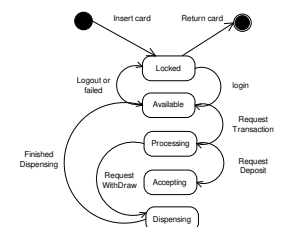
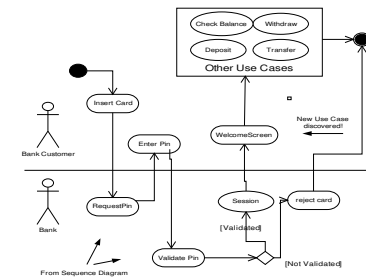
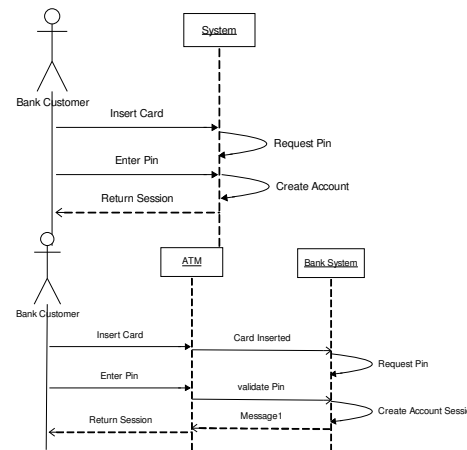
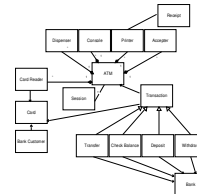


Main Flow

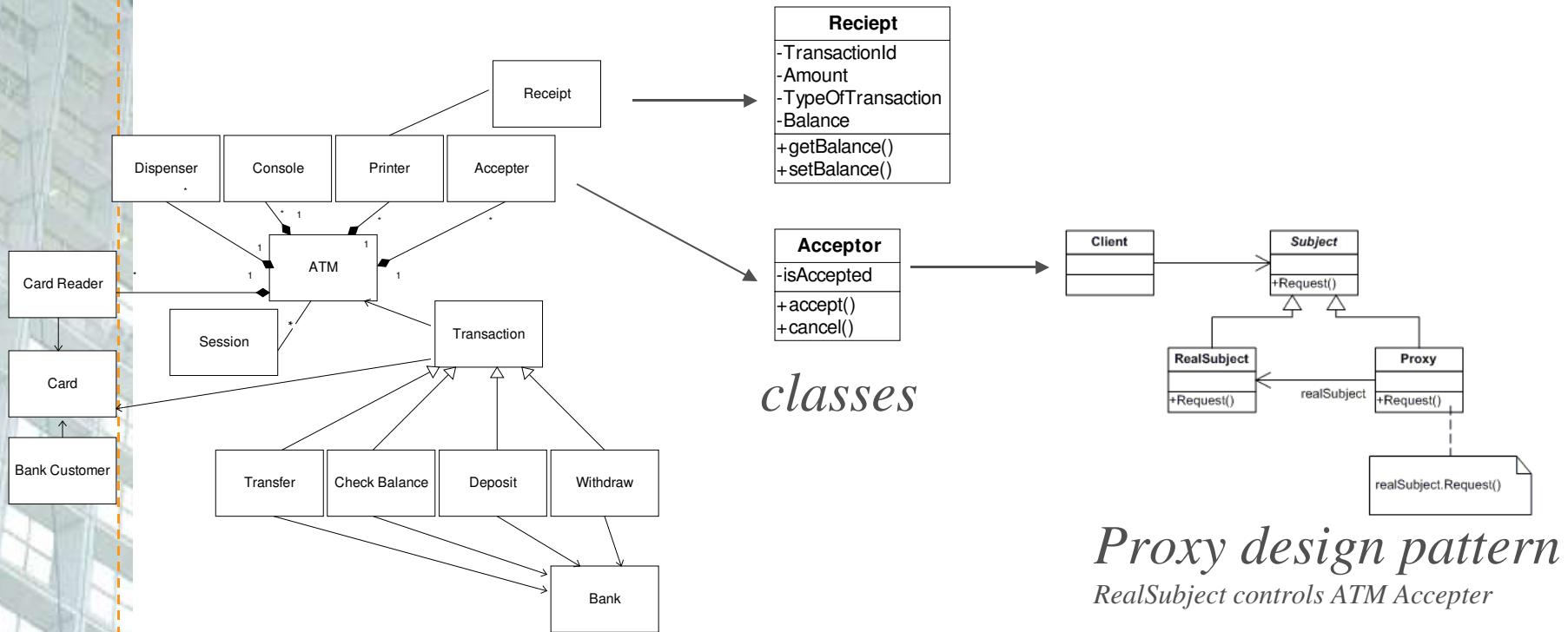
- 1) Actor does something
- 2) System does something

Alt Flow1 – (etc)

- 1) Actor does something
- 2) System does something



- Turn your domain model into a class diagram (works best for new dev)
- Ex) Use the “Acceptor” to connect to the client in the Proxy design pattern to control, yet uncouple your code from the actual hardware

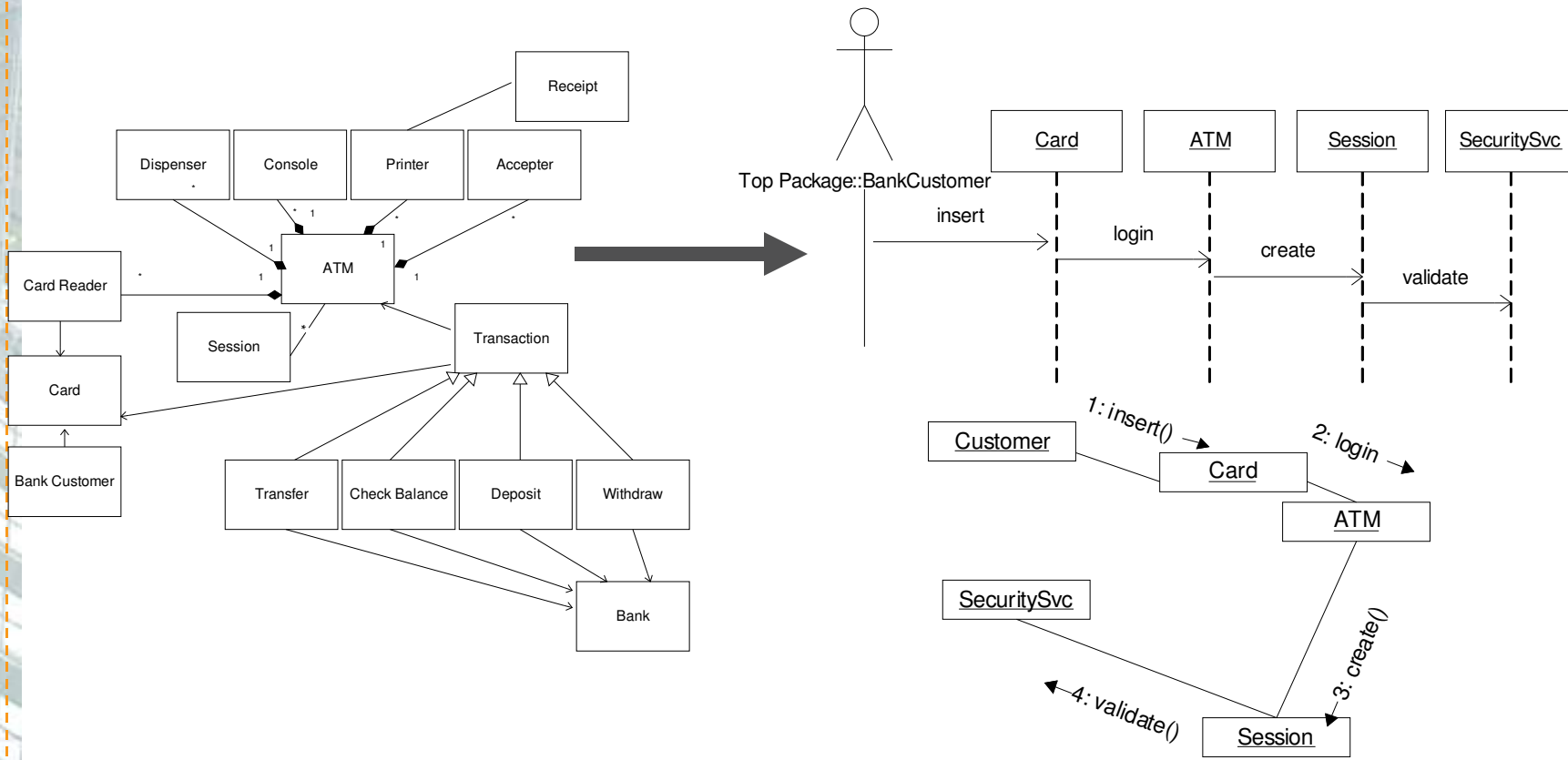


classes

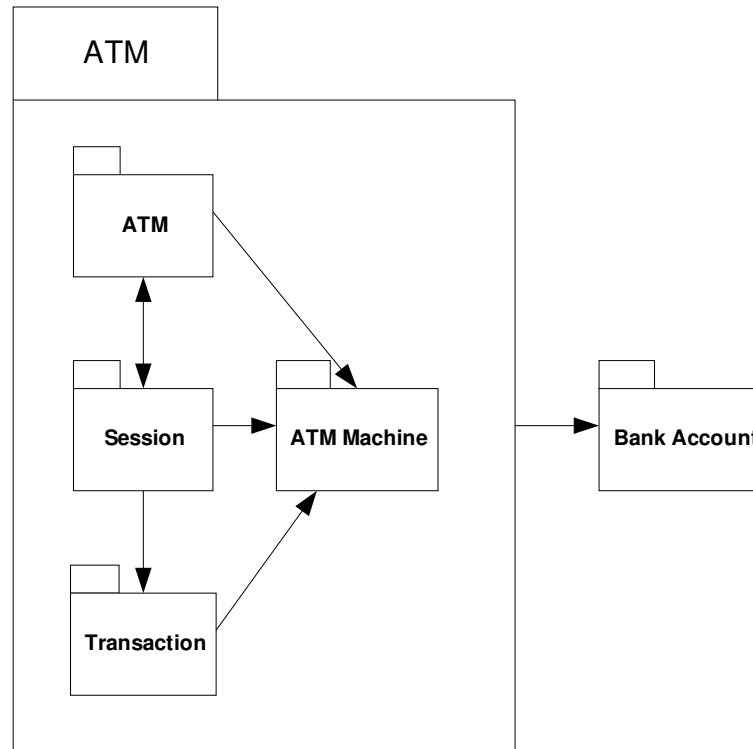
Proxy design pattern
RealSubject controls ATM Acceptor

Interaction Diagrams

- Take your classes from the class diagram you just made and demonstrate flow in your code.
- Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. the diagrams are read left to right and descending.
- Collaboration diagrams show the relationship between objects and the order of messages passed between them. -
 - ie) sometimes you can take a class diagram and turn it into a collaboration diagram.

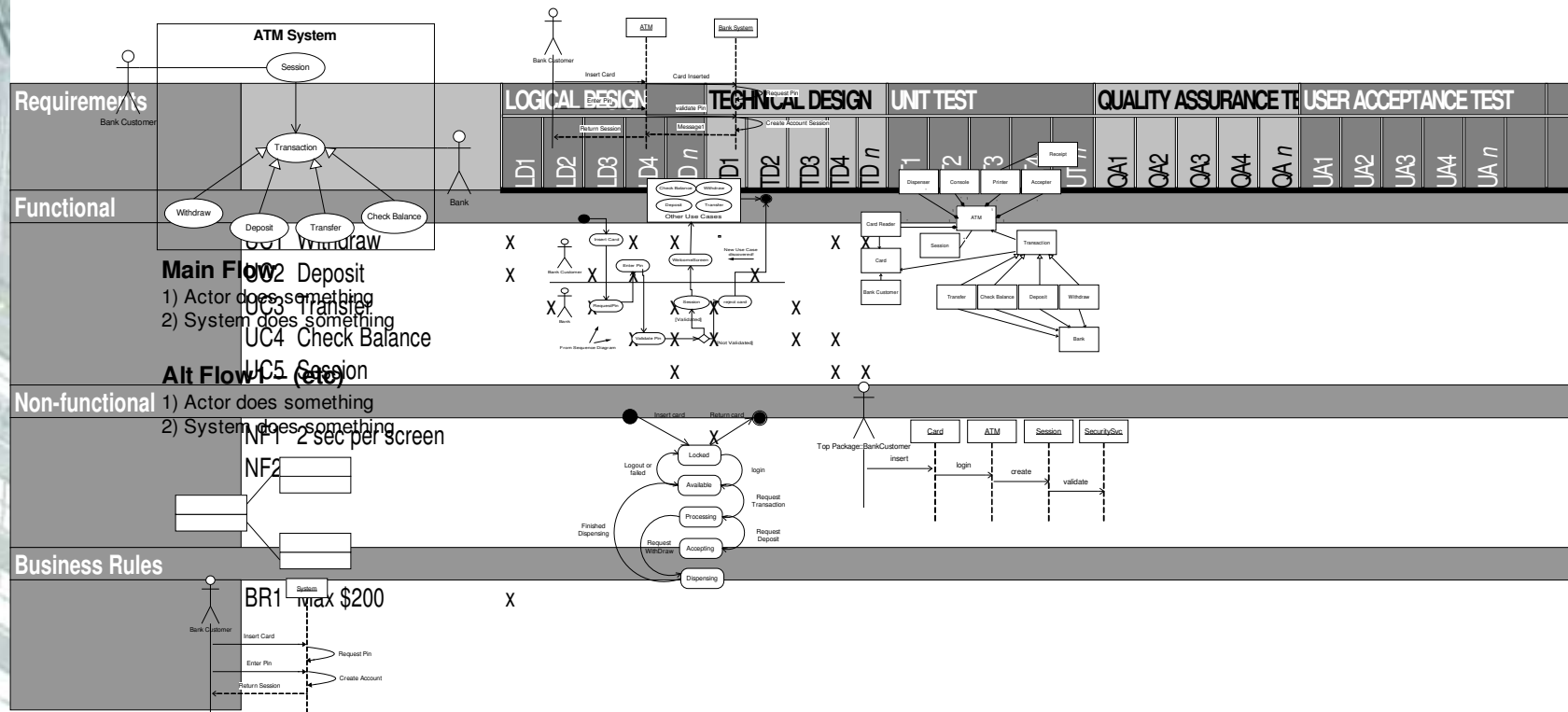


- Sum up your design at a higher level



Did you get it all?

- Trace ability Matrix is the key in the Architect's toolbox
- Make sure that you have an answer for each of the requirements



Questions?

- Questions?



■ Thank You!

■ Thank You!