Use Cases and Services–
Practical Techniques for Functional Requirements

A special one-day class assembled for ING Belgium by Adept Events and Clariteq Systems Consulting

Alec Sharp
Consultant
Clariteq Systems Consulting Ltd.
West Vancouver, BC, Canada
asharp@clariteq.com
www.clariteq.com

ING BANK

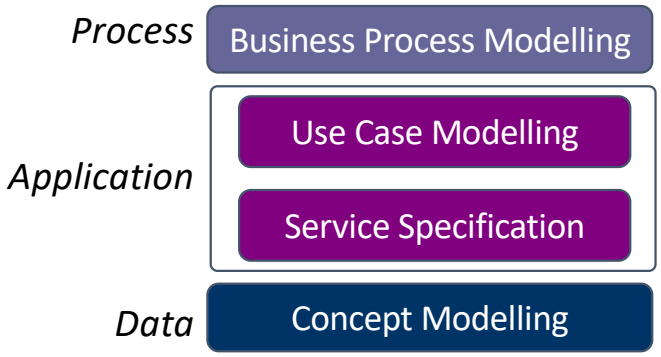
AdeptEvents
www.adeptevents.nl

CLARITEQ
www.clariteq.com

Developer/instructor background…

Alec Sharp, Clariteq Systems Consulting – *asharp@clariteq.com*

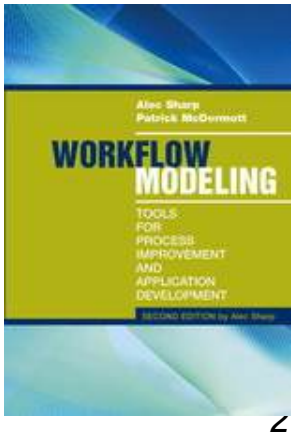
- 45 years experience as an independent consultant:
  - Business Process Change – discover, model, analyse, and design/redesign processes
  - Concept Modelling (Business-friendly Data Modelling)
  - *Application Requirements Specification*
    +
  - Facilitation & Organisational Change
  - Project Recovery



- Consulting, teaching, speaking globally

- Awarded DAMA's global Professional Achievement Award for contributions to "human-friendly" data modelling

Check out the nice reviews on Amazon - http://amzn.to/dHun1o



- Author of "Workflow Modeling"
  - best-selling book on process modelling & improvement
  - second edition – 2009 (sole author, complete re-write)

Clariteq – small, husband & wife company, global clients

ABB (ASEA Brown Boveri)
Aflac
American Honda
AMP (Australia Mutual Provident)
BackOffice Associates
Bank of Finland
Bellrock
Booking.com
Brisbane City Council (Australia)
Canadian Natural Resources Ltd.
City of Seattle
Civica UK
Clearwater Paper
Corvias
Dell
DHL Express
Dutch National Bank
Elisa
Ericsson
Essity
Eurojust (European Justice Comm.)
European Central Bank
Fortum
Gofore
Helse Vest - Norway
HM Land Registry - UK
Home Depot
Idaho Transportation Dept.
Intel
ISO New England
ING Bank
JP Morgan
Kal Tire
KONE
LGM Financial Services
Liberty Mutual
Livestock Improvement Corp.
MacDonald Dettwiler
Manitoba Public Insurance
Marathon Pipe Line
Microsoft
Ministry of Defence - UK
Ministry of Defence - NL
Ministry of the Interior - Slovakia
MTS Allstream
Nexen
Novo Nordisk
Nusenda Credit Union
OP Bank
Partner Reinsurance
Ritchie Brothers
Phillip Morris
Roche Diagnostics/Pharmaceuticals
Salt River Project
Saudi Aramco
Serco
Shell
Sparta Consulting
State Street Bank
SunGard
SVB (NL)
Synechron
Sysdoc
Talent Base
Teck
The MUSIC Group
The Seattle Times
UK Government
University Med Ctr Groningen
YIT(FI)
Washington Gas & Light

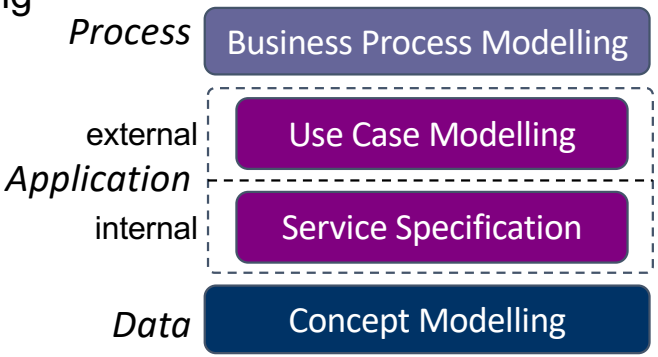
– Higher Education –
Carnegie Mellon University
Cornell University
Douglas College
Gonzaga University
Humboldt State University
The Jackson Laboratory
The Ohio State University
Portland State University
Salt Lake Community College
Southern NH University
University of Arkansas
University of British Columbia
University of the Fraser Valley
University of Maryland
University of Utah
University of Washington
Utah Valley University

Outcomes and topics

Learning Outcomes

This course is a foundation for those who perform Business Analysis or work with BA deliverables:

- Understand the resurgence of interest in *model-based techniques* and how they provide different *lenses* (*perspectives*) to form an *integrated BA framework.*
- Learn a common language for Business Analysis work
- Learn the core techniques for discovering, verifying, and documenting Functional Requirements with
  - Service Specifications
  - Use Cases or User Stories



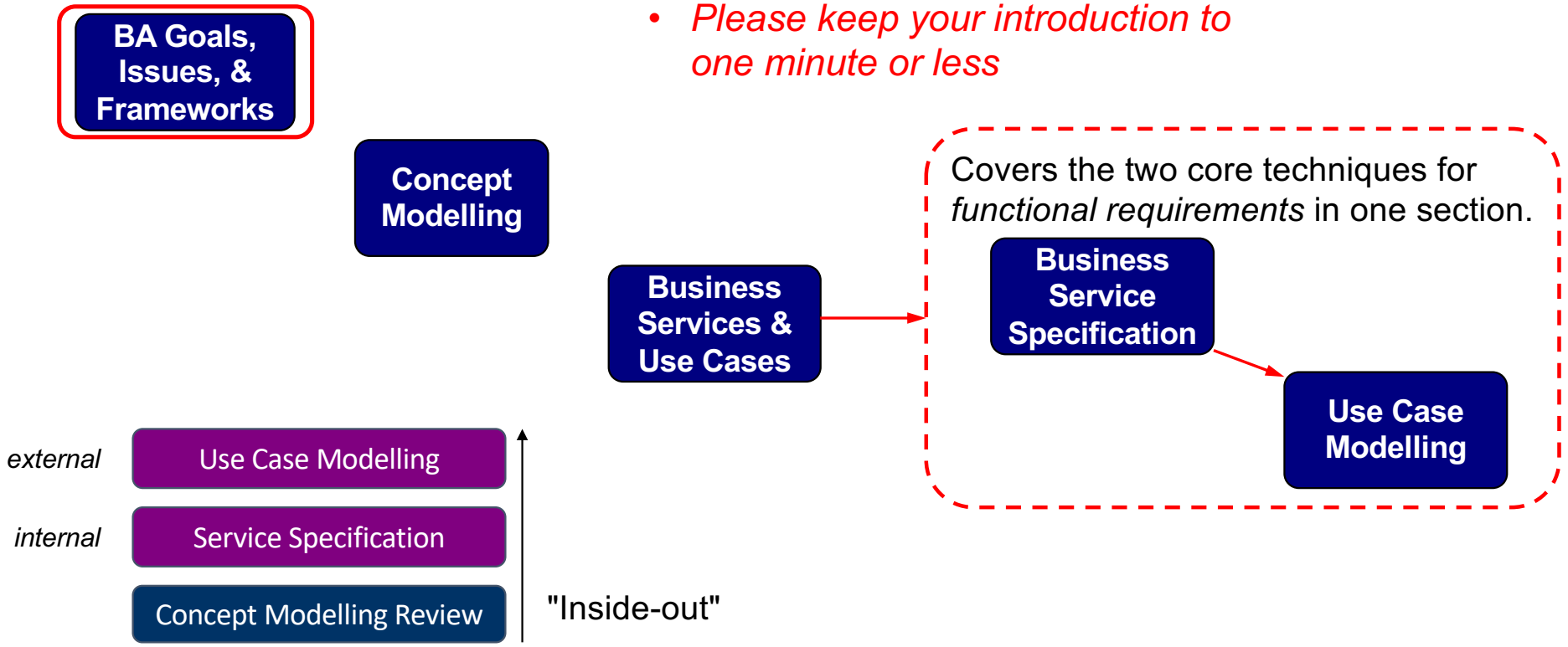
Topics and techniques include:

- *Business Analysis* –
  goals, issues, and an integrating framework made up of four techniques
- For review - a case study illustrating the critical role of *Concept Modelling* and how it supports other Business Analysis techniques
- *Service Specification* –
  capturing validation, business rules, data updates, and other ***internal*** behaviours
- *Use Cases & User Stories* –
  discovering user expectations about a system's ***external*** behaviour

Course overview – an "inside out" flow

And finally, let's get introduced:

- Name (how should I address you?)
- Role / job title, organisation, brief description of your work
- Is there a topic you are most interested in?
- *Please keep your introduction to one minute or less*



Back to basics – what is a "requirement?"

Requirement:
a *capability* or *behavior* that a system must have in order to…

- Support business goals and objectives
- Meet the needs of users –
  support tasks and decisions whether ad hoc
  or within a defined business process

Many categories:

Operational characteristics	99.95% uptime 24x7
Integration or interfaces	must use our Workday HR System
Deployment platform	AWS

Functional Requirements

- (What business processes the system supports	*– Business Process Model)*
- What the system knows	*– Concept Model*
- What the system does	*– Use Cases & Services*

Functional Requirements w. User Stories / Use Cases & Services

"Business Analysis" gets criticised because of the extremes

Simplistic methods at one extreme: can do as much harm as good	The goal lies in the middle ground:	Overly complex methods at the other extreme: difficult for businesspeople to verify

List-form requirements, typically a Business Requirements Document – "*context-free requirements*"

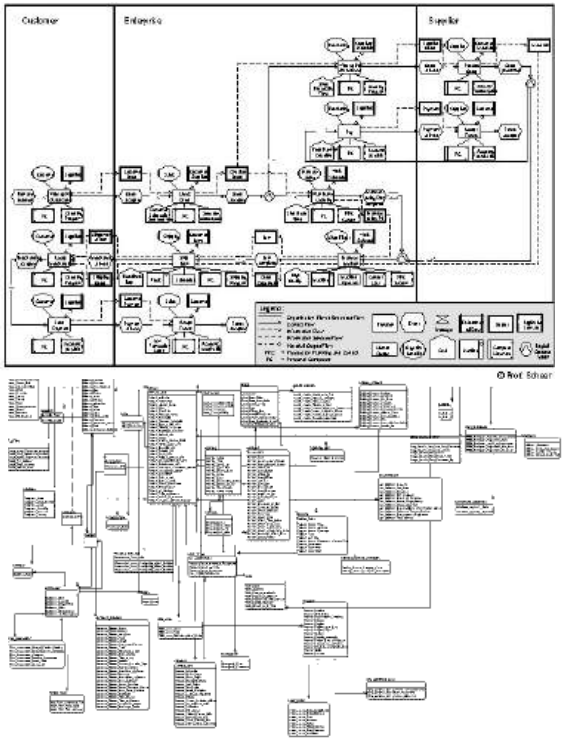
ID#	Business Feature	Requirement Type	Business Unit(s) Affected	Potential Application(s) Impacted
BRQ025	files that are available for the selected day.		Readiness	
OMSPI-BRQ026	System shall include all outage status in the Transmission Outage report.	Core	Operation Readiness	WebOMS
OMSPI-BRQ027	System shall display consistency in the format of output data in the Transmission Outage report when using pipe-delimited feature as follows: For the same row of output data, all data elements in the same position in any field must correspond to each other. Example of existing Transmission Outage report where there are two inconsistencies in the output data format: 1. Report shows one Outage ID, three Substations, and four Equipment Names. 2. First listed Substation does not correspond to the first listed Equipment Name. Outage ID: 3042750 | Substation: HUNTERS POINT PP P / MISSION X| LARKIN Y / POTRERO PP A (PGAE)| MISSION X | Equipment Name: A-Y 2| BNK-2| P-X 1| P-X 2	Core	Operation Readiness	WebOMS
OMSPI-BRQ028	System shall allow the format of the Transmission Outage report published periodically automatically to support the following formats: 1. PDF 2. HTML 3. MS Word	Core	Operation Readiness	WebOMS
OMSPI-	System shall allow admin user to configure the number of days in the Transmission	Core	Operation	WebOMS

Client – *understandable, and therefore verifiable.*

Analyst – *doable, within Agile timeframes.*

Developer – *unambiguous, complete, actionable*

Thinly-disguised, implementation-level design methods – *not* useful for discovering stakeholder needs



Discussion – the problems with list-based requirements

Simplistic methods at one extreme:

An actual example, one in a list of 451 individual requirements for the "Provide Scientific Evidence" process at a national forensic science laboratory:

#49 -

The system shall provide a visual mechanism through which to view or amend the sequencing of items for a previously selected case or allocations thereof.

WHAAAT???!!!

List-based approaches to business analysis quickly break down – no way to ensure *completeness, accuracy, consistency, …*

So… what's wrong with this as a requirement? What does it NOT tell us?

What are they really trying to say?

Who? Senior Scientist

What? Schedule a Test (an Allocation) on a Sample from an Item

When? At Item Submission

How? By viewing upcoming workload

Why? To provide a completion date to the Customer (the Police)

Essentially, a Use Case or *User Story*:

As a Senior Scientist, I need the ability to view upcoming workload and schedule a Test on an Item, so I can provide a completion date to the Customer.

We will also use

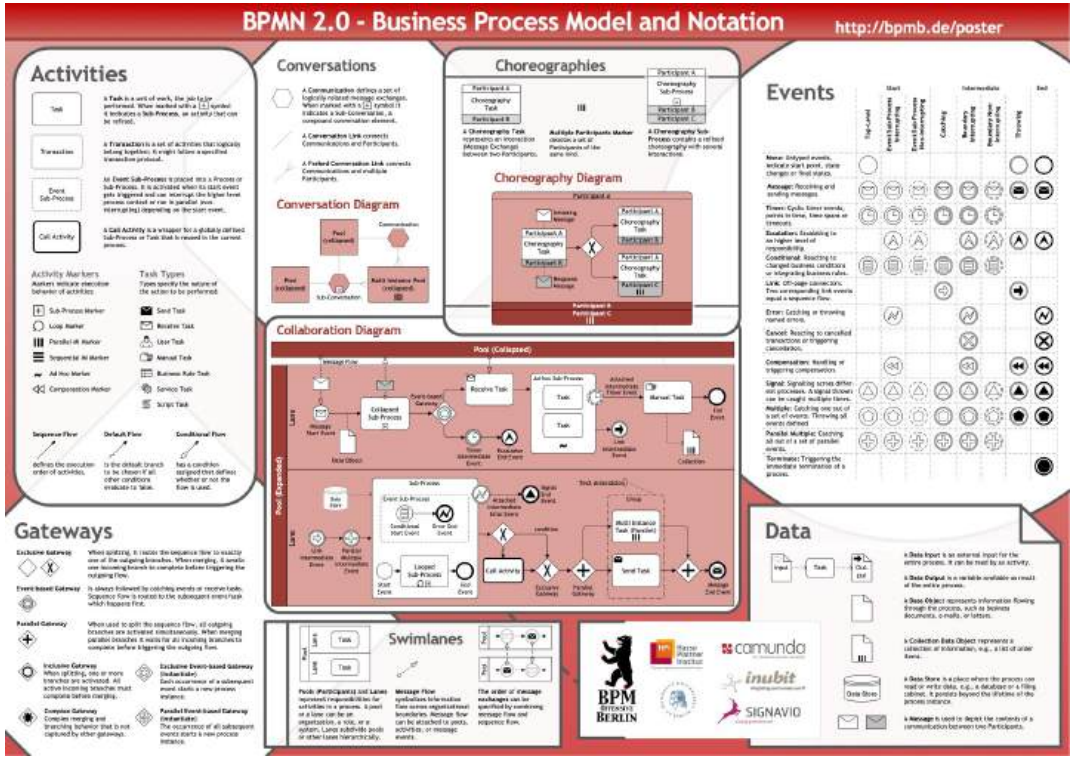
- *Business Process Models* to show where this fits in the end-to-end process
- *Concept Models* to show the required information

Complicated methods at the other extreme

"Can we use UML for Business Analysis?" As the late Michael Hammer said: “You could, but it will be like eating rice with a steak knife – messy, and someone’s going to get hurt.”

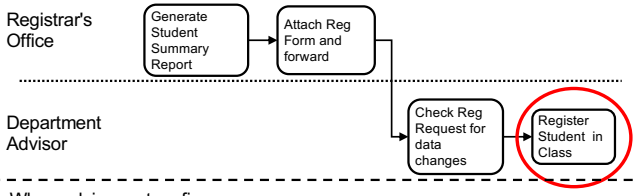
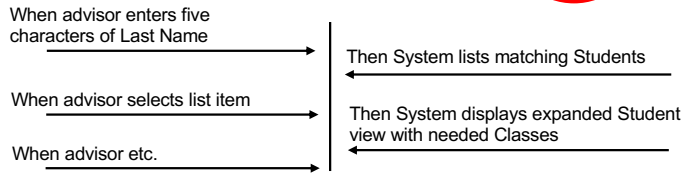
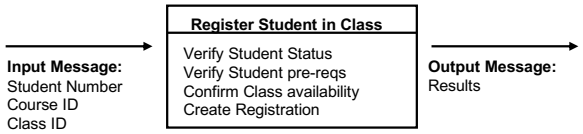
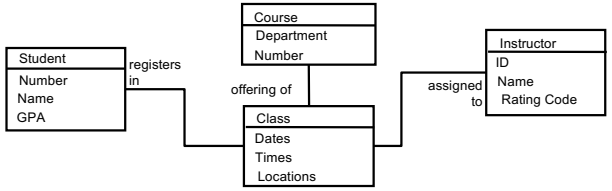
From the original UML specification: “The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the *artifacts of a software-intensive system*.”

Same story for full BPMN (Business Process Model & Notation) – a platform-independent *visual programming language* for specifying automated workflows.



An integrated, model-based framework

The Clariteq Framework for Business Analysis

Framework Layer	Technique sample	What it covers
Business Goals & Objectives	The university is initiating the "Strategic Enrollment" program to raise Student graduation rates in part by ensuring Classes are available for Student registration when needed.	✓ ***Project Charter*** – documents the rationale, objectives, scope, and success measures for the project
Business Process		✓ ***Process Model*** - shows "what" in a Scope Model, then "who & how" in a Workflow Model – the steps done by the actors in the process
Presentation Layer (user interface)		✓ ***Use Case*** – models how an actor interacts with a system to obtain (trigger) a service, typically to complete a step in a process
Application Layer (rules & logic)		✓ ***Service Specification*** - describes a service – a package of rules and logic – that is triggered to complete or respond to a business event
Data Layer (data & storage)		✓ ***Concept Model*** - depicts the things and the facts about things the organisation needs to record; the things (the entities) are what processes and solutions act on.

This is not a sequence!

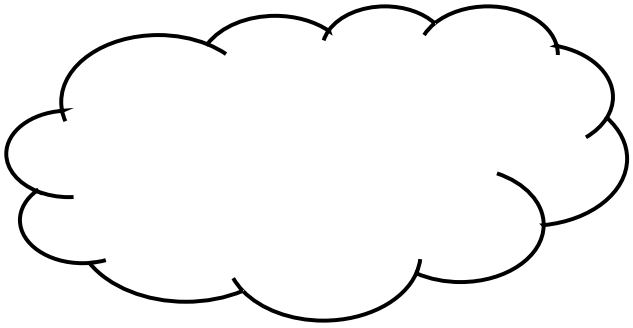
Business Process: gives great *context* for *Business Analysis*

Use Cases and Services: where we capture *Functional Requirements*

Concept Model / Data Model: a great *platform* for *Business Analysis*

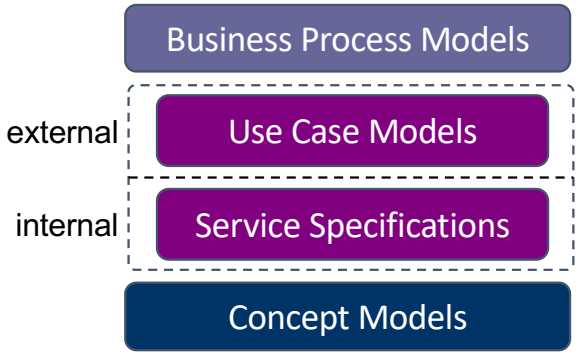
These techniques supports the Business Analyst's role – current to future state

Current state



Iteration

Future state



The current *(as-is)* state may be incompletely understood, with multiple perspectives – some *as-is* modelling will clarify it.

The future (*to-be*) state, described with a rigorous set of interrelated models – *easier* than you might think

- A "best practice" solution – *"integrated business modelling techniques"* which build graphic and narrative models highlighting different aspects of the *as-is* and *to-be* states
- Key ideas – "business-friendly," "suitable for mere mortals," "progressive detail," "high context," and "do-able within your natural lifetime"

We'll start *(surprisingly!)* with the Concept Model / Conceptual Data Model

Case study review – Concept Model, Services, Use Cases, Business Processes

Client –

- Regulatory agency ensuring the safe design, installation, and use of technical equipment
- Natural gas systems, electrical systems, boilers and pressure vessels, elevating devices, & many more

Goal –

- Shift from an inspection-based model (~800 inspectors!) to client-managed safety programs
- Clients will apply for a *Client Safety Management Program Authorisation (CSMP Authorisation)* - must show effective processes and accurate record-keeping
- Clients will pay a fee for managing *their own safety programs!* Still beneficial!

Case study – Concept Model, Services, Use Cases

- Business Development chooses Pilot Program –
  boilers and pressure vessels in Oil & Gas fields



- Current systems won't support CSMP, time-consuming and expensive to change them –
  IT and Finance suggest 18 – 24 months of work
- BD is unimpressed by IT and Finance objections ("You're being mindlessly obstructionist!")
  and proposes work-around procedure. *Guess which tool they intend to use?*
- I'm hired to identify end-to-end implications –
  "Design a process and determine IT requirements that will allow this procedure to work."
- *Concept Modelling was a critical tool in understanding the underlying policies,*
  and developing the process & requirements

Building your initial Concept Model, step-by-step

Identify and define "Things"

1. Collect terms
- 1:1 interviews
- survey (e.g., email)
- group brainstorm
- analyse documents

2. Isolate "things"

Ask *Is this…*
- a thing?
- a fact about a thing?
- or "other stuff?"

3. Identify synonyms
- select a term to use
- as general as possible
- just for this initiative, not the entire enterprise

4. Define each thing
- "good enough for now"
- first, identify "anomalies, sources of confusion, and valid differences of opinion"
- select which to include

Develop initial Concept Model

5. Organise things
- independent things across the top
- then laid out top-down by dependency

6. Draw relationships
- show dependency
- parent-child drawn bottom-to-top
- otherwise, side-to-side

7. Name relationships
- in both directions
- active verb-based!
- not mushy – *has*
- not meaningless – *related to*

8. Add cardinality
- use words first
- 1:1 is probably wrong
- 1:M (one to many)
- M:M (many to many)

Refine Concept Model

9. State assertions
- forcefully, for each relationship
- challenge the assertions!
- restate the assertion & why it changed, if it did

10. Redraw the model
- shows revised assertions
- e.g., 1:M becoming M:M
- e.g., dependent things becoming independent

11. Collect attributes
- a few for each thing
- not *all* attributes
- don't worry about normalisation

12. Move to identifying:
1. events / services
2. use cases / user stories

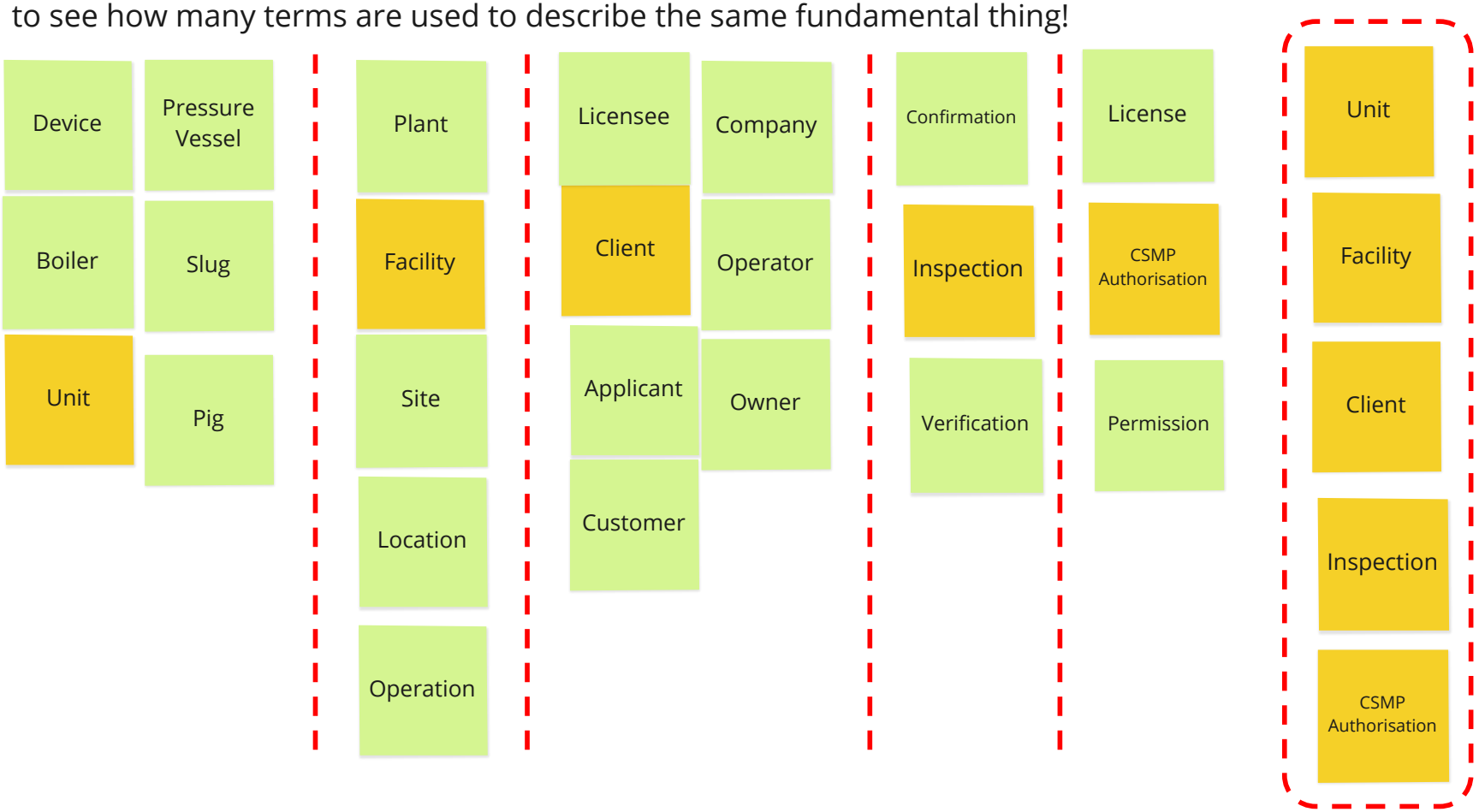
Always start with terminology (the “things”)

From one-on-one interviews with 8-10 key stakeholders we gathered ~200 terms related to CSMP (Client Safety Management Program) – "anything that went by a name."
Here are 24 that met the criteria to be a "thing"– the candidate *Entities.*

Identify synonyms and select one term.
How do these relate to one another?
What do you need to know about each?

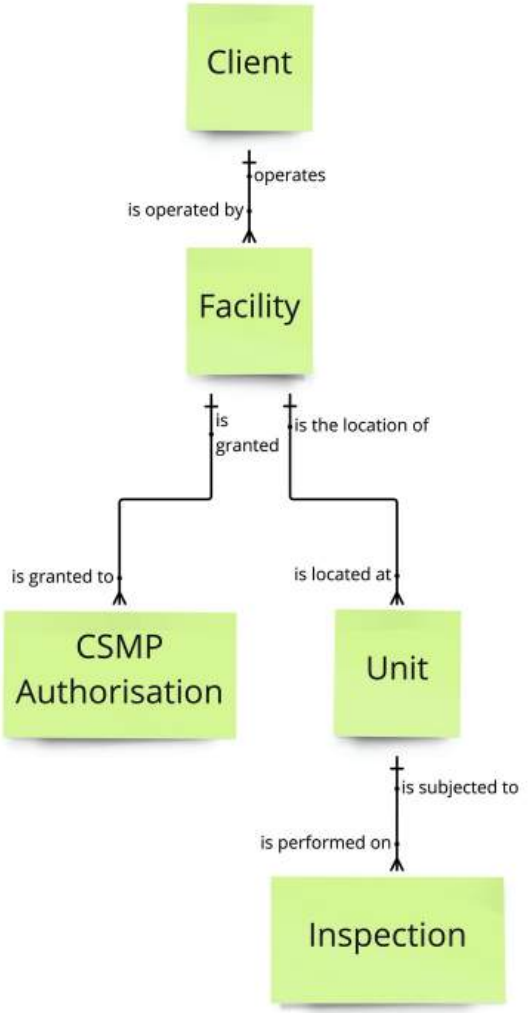
Review of a Miro example – Terminology Analysis

Terminology analysis (continued):
Let's arrange these terms into columns of synonyms. It's always a surprise for the business to see how many terms are used to describe the same fundamental thing!



Concept Model Version 1 – not perfect, but a good start

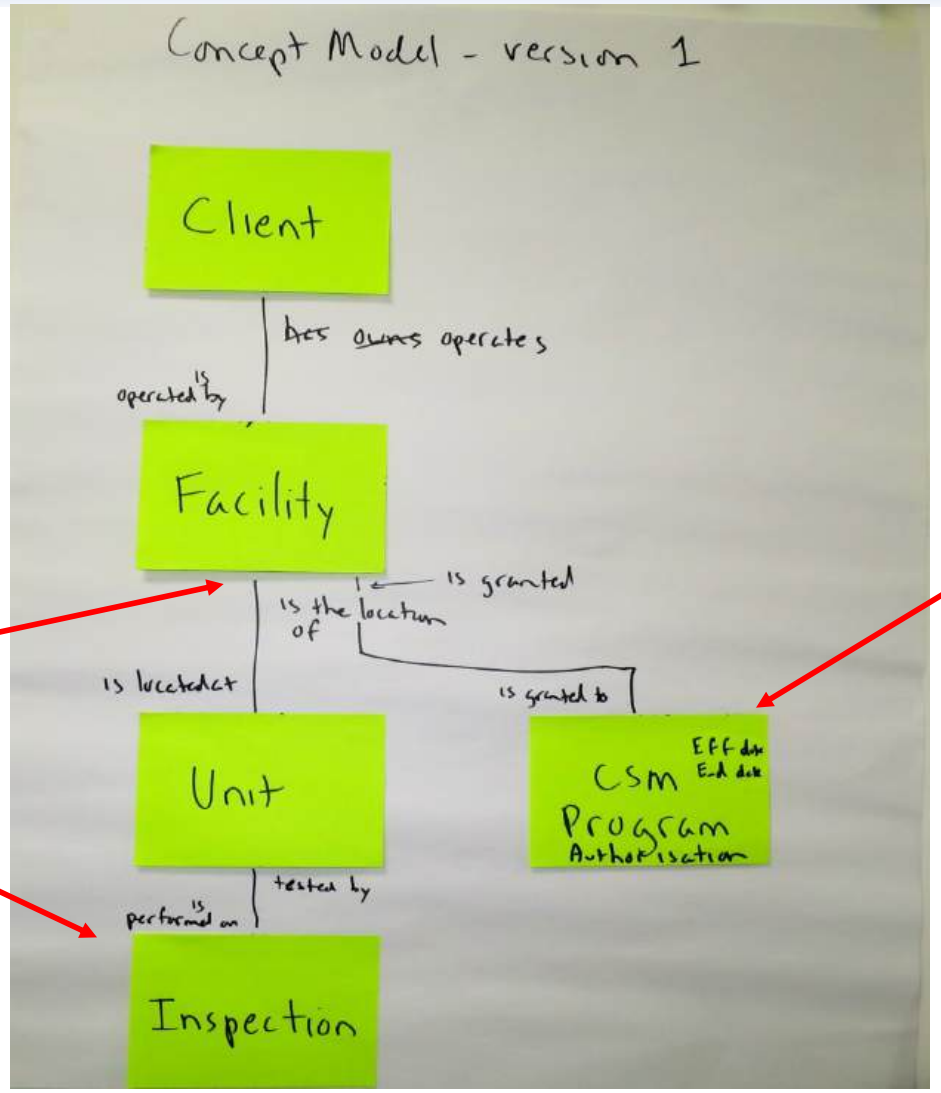
1. We arranged the entities / business objects by dependency
2. Then we drew relationship lines
3. Then we added a relationship name in each direction
4. Only then did we state (in words) the cardinality (1:1, 1:M, M:M) and then update the diagram with hash marks (⟊) and crowsfeet (⅄)



Definition -
A CSMP Authorisation is a permission (or license) to operate a self-managed safety program (a Client Safety Management Program) at a specific Facility, for a specified time period, usually 1, 2, or 5 years.

The CSMP Authorisation is "all or nothing" - it covers ALL the Units at a Facility.

Just boxes and lines, but raises important questions



Are Units permanently part of one Facility?

What do we Inspect?

What do we issue the Authorisation to?

Concept Model Version 1 – state Assertions and challenge them

Now, state the relationships ***emphatically*** as Assertions. ***Each*** Client operates ***one or more*** Facilities! Then, ***challenge*** them!
Again, don't worry yet about ***optionality*** – whether the relationship ***must be*** or ***may be*** be present.
We only care now about the ***maximum*** – each ObjectA is related to a ***maximum*** of ***one*** or ***one or more (or many)*** ObjectB.

Assertion:
Each Client operates ___

Assertion:
Each Facility is operated by ___

Assertion:
Each Facility is granted ___

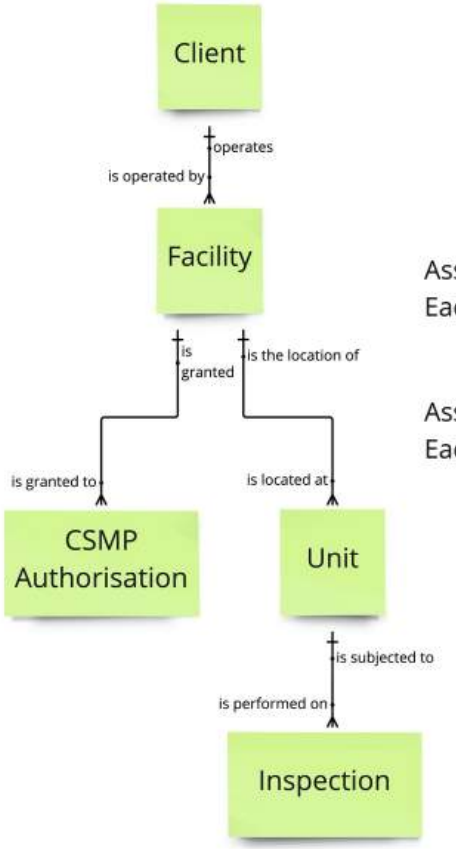
Assertion:
Each Facility is the location of ___

Assertion:
Each Unit is located at ___

Assertion:
Each CSMP Authorisation is granted to ___

Assertion:
Each Unit is subjected to ___

Assertion:
Each Inspection is performed on ___



Concept Model Version 1 – revised Assertions from challenges

Now, state the relationships ***emphatically*** as Assertions. ***Each*** Client operates ***one or more*** Facilities! Then, ***challenge*** them!
Again, don't worry yet about ***optionality*** – whether the relationship ***must be*** or ***may be*** be present.
We only care now about the ***maximum*** – each ObjectA is related to a ***maximum*** of ***one*** or ***one or more (or many)*** ObjectB.

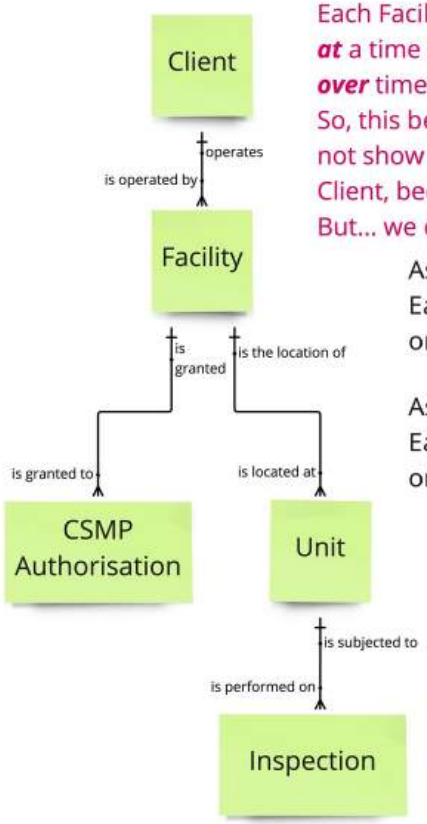
Assertion:
Each Client operates
one or more Facilities

Assertion:
Each Facility is operated by
one Client

Assertion:
Each Facility is granted
one or more CSMP Authorisations

One CSMP Authorisation ***at*** a time,
but one or more ***over*** time

Assertion:
Each CSMP Authorisation is granted to
one Facility



Each Facility is operated by one or more Clients
at a time (Joint Ventures) and
over time (changes in Ownership or Lease.)
So, this becomes a M:M relationship, and we should not show a Facility as being dependent on a single Client, because a Facility is an independent thing.
But... we don't always get our way!

Assertion:
Each Facility is the location of
one or more Units

YES, but one or more Facilities ***over*** time, because Units can move between Facilities. So, this becomes a M:M relationship, and we cannot show a Unit as being dependent on a single Facility, because a Unit is an independent thing

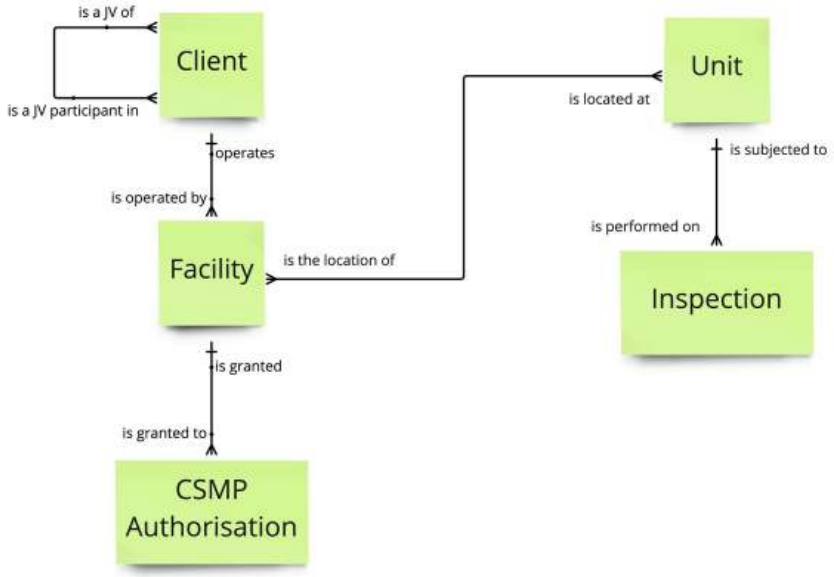
Assertion:
Each Unit is located at
one Facility

Assertion:
Each Unit is subjected to
one or more Inspections

Assertion:
Each Inspection is performed on
one Unit

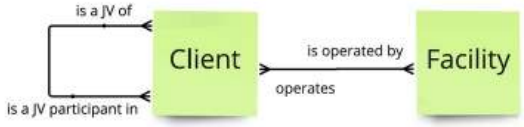
Concept Model Version 2 – revised from challenging Assertions

Now we will re-draw the initial Concept Model based on changes that came from challenging the Assertions in Ver. 1.

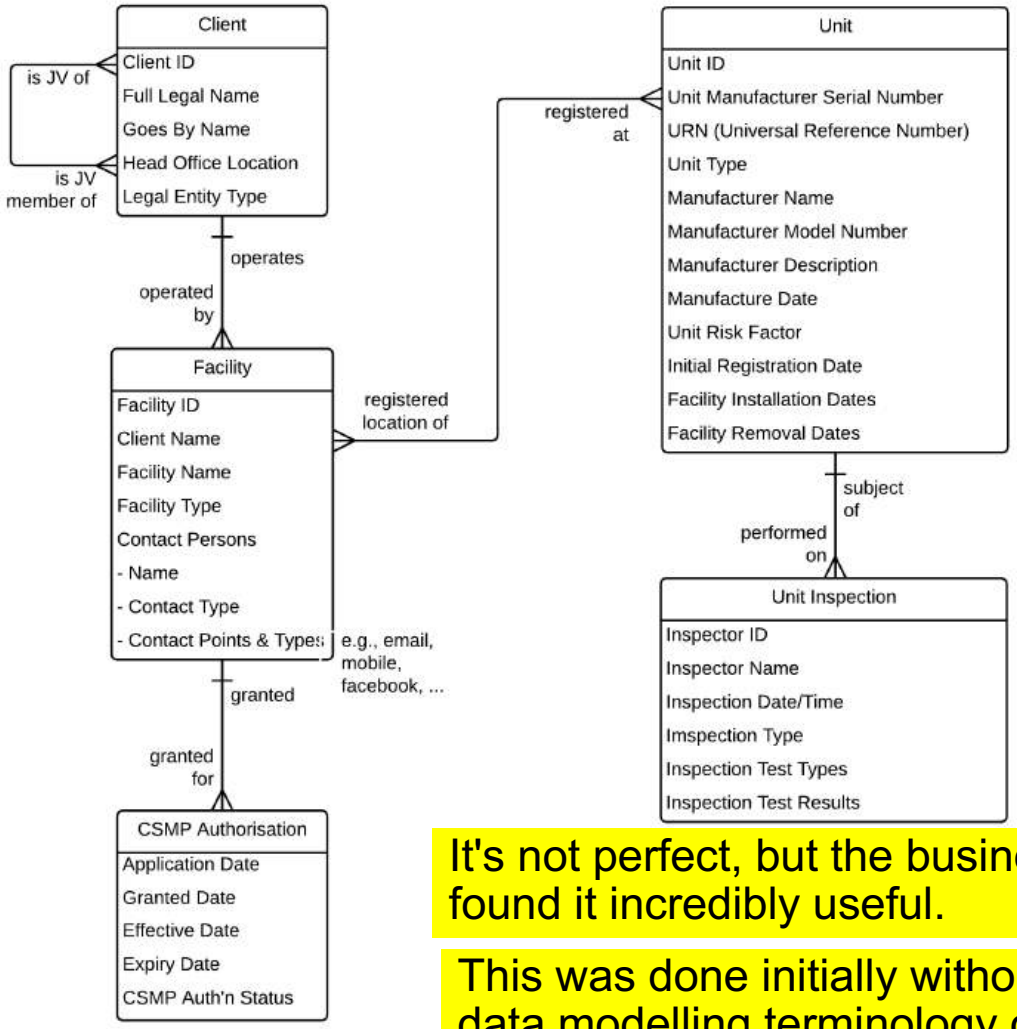


Note:
You don't always get what you *want* or what you think is the *right* thing in Concept Modelling. In this case the client (the Regulator) said they always wanted a Facility to be operated by ONE AND ONLY ONE Client.
If a Facility was operated by multiple Clients, they would require the Clients to form a new Joint Venture Client. This was to ensure that if there were legal difficulties, there was only ONE Client to go after.
Or, as they put it, "one throat to choke."

Later in the project, they realised they needed a history of the Clients that had operated a Facility, so the Client-Facility relationship became Many-to-Many, and Facility was modelled (correctly) as an independent Entity, as shown here:



"What do you need to know about the things in the Concept Model?"



Sketching this out was *fast, and* raised many questions that had not occurred to the client…

- Is there one CSMP per Client, per Facility, or some other basis?
- Do Units frequently relocate, or even turn up at another Client?
- What is inspected – the Facility or the Unit?
- Does the CSMP cover all or some Units at a Facility?
- …and MANY more…

It's not perfect, but the businesspeople found it incredibly useful.

This was done initially without any data modelling terminology or symbols!

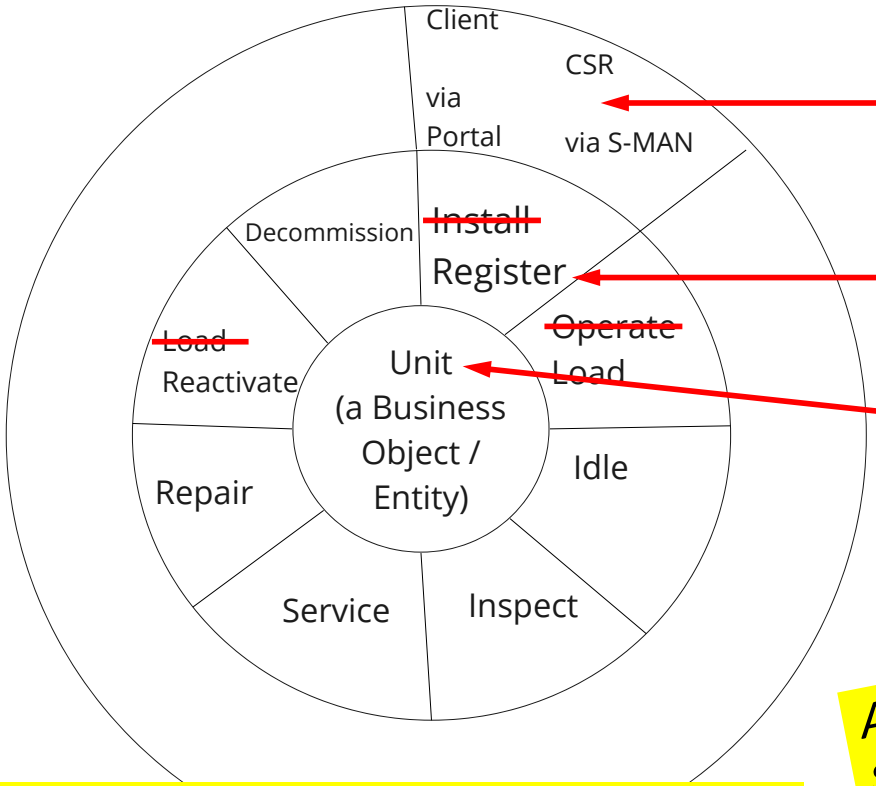
Model took ~90 minutes

Identify Services (Events) then Use Cases / User Stories

Finally, we'll identify the Services (verb - noun pairs) we need, and the Use Cases / User Stories by which the Services will be accessed

What events happen to a Unit - what are the needed services? (Verb - Noun)

- ...
- ...
- ...
-



Who needs access to each Service, and How?

Use Case	*Use Case or User Story - add Who and How*
Service Specification (Events)	*Service (or Event) - add a Verb to the Noun*
Concept Model	*Entity or simply a "thing" - a core Noun*

A Concept Model is a great starting point for discovering your Services and Use Cases (User Stories)

Supports *Service-Oriented Business Analysis*

Note – "User Story" and "Use Case" are not so different

Different format and detail, but the same basic concept.
Initially, at the Scope level, they're much the same:

User Story (who – what – *why*):
"As a Client, I need the ability to Register Unit(s,)
so I can maintain compliance with my CSMP Authorisation"

Use Case: (who – what – *how*):
"Client Register Unit via Portal"

When we add detail at the Concept level, they become identical:
- User Story / Use Case abstract
- Main success sequence – dialogue in "when-then" format
- Alternate sequences – variations, exceptions, errors

Develop high-level use cases and services

Service: *Register Unit*

- Check for presence of properly formatted UR Number
- Determine if Unit UR Number is previously known
- If known, has it (a) moved (b) changed ownership (c) …?

Use Case: *CSR Registers Unit via S-MAN*

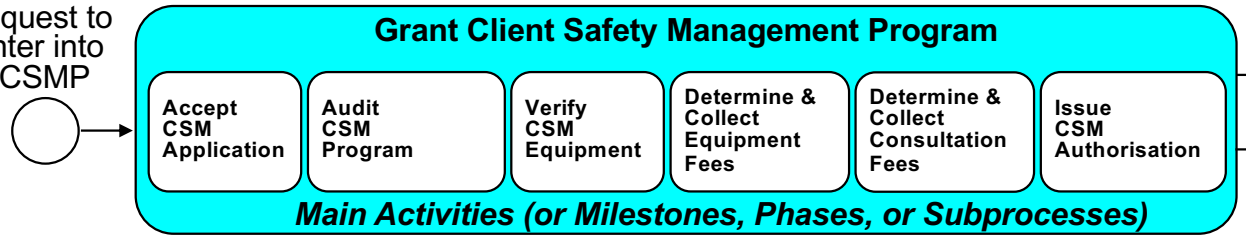
- CSR will select "spreadsheet" of all Units covered by CSMP app
- S-MAN will highlight all that can proceed immediately
- For each category of Units requiring intervention…

Note:

Services and Use Cases at the "upper conceptual" level to provide vendor with key elements of requirements and avoid the usual bulleted list requirements document.

Clarify scope of the new process and identify participants

Trigger:
Client submits request to enter into a CSMP

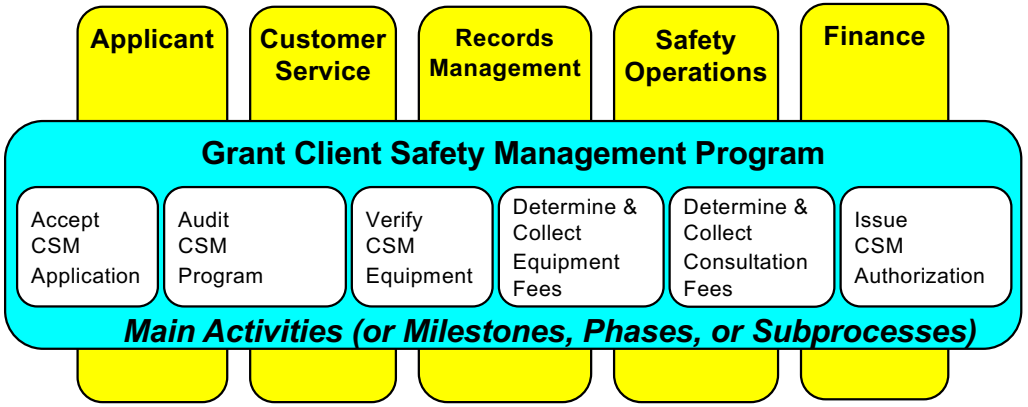


Client Result:
Approval granted for a self-managed safety program.

Agency Result:
Revenue collected.
New participant in CSMP; confirmation that regulations are satisfied

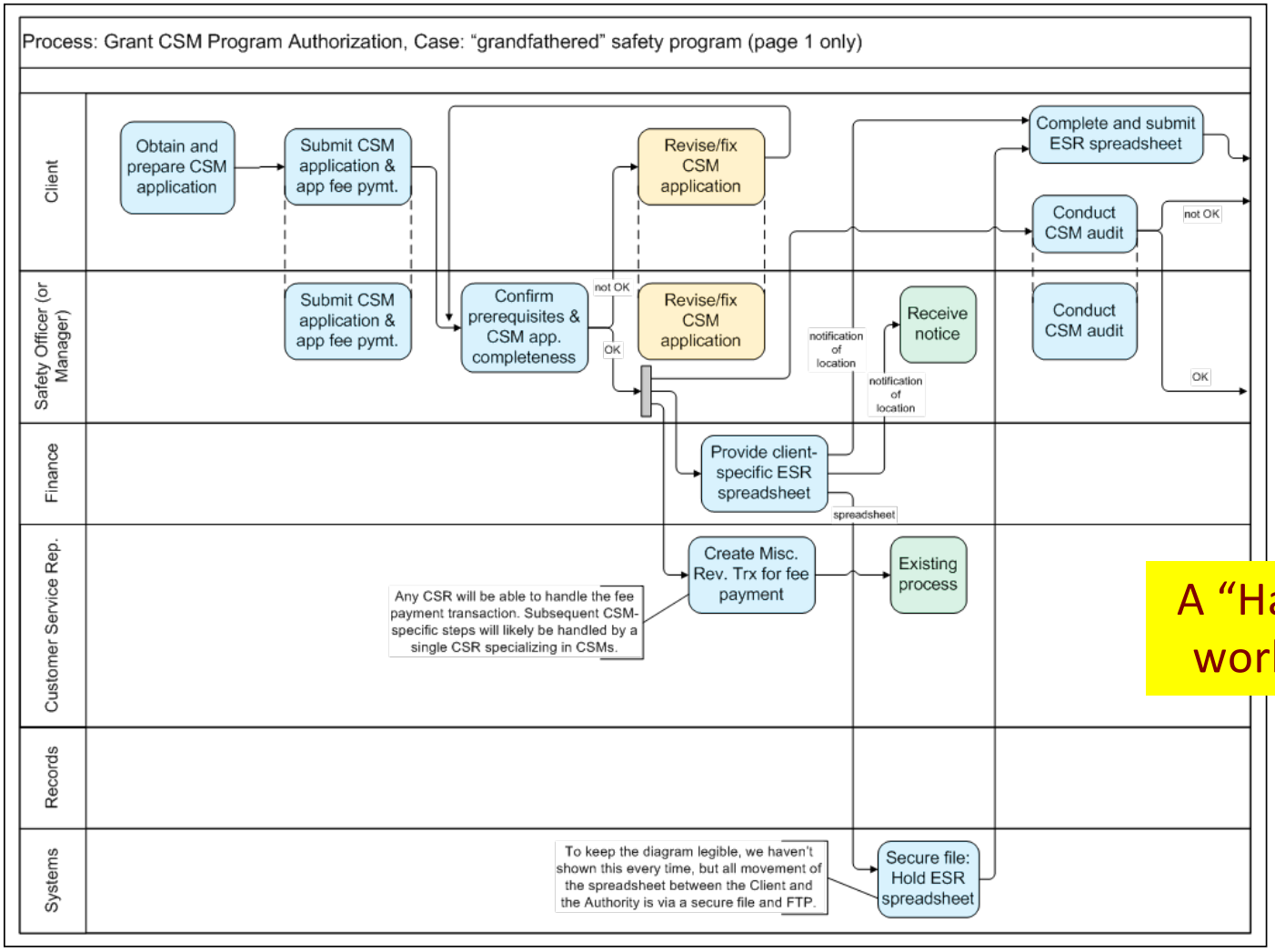
Cases:
- New
- Grandfathered
- Ownership Change

Process Scope Model – pure "what"…



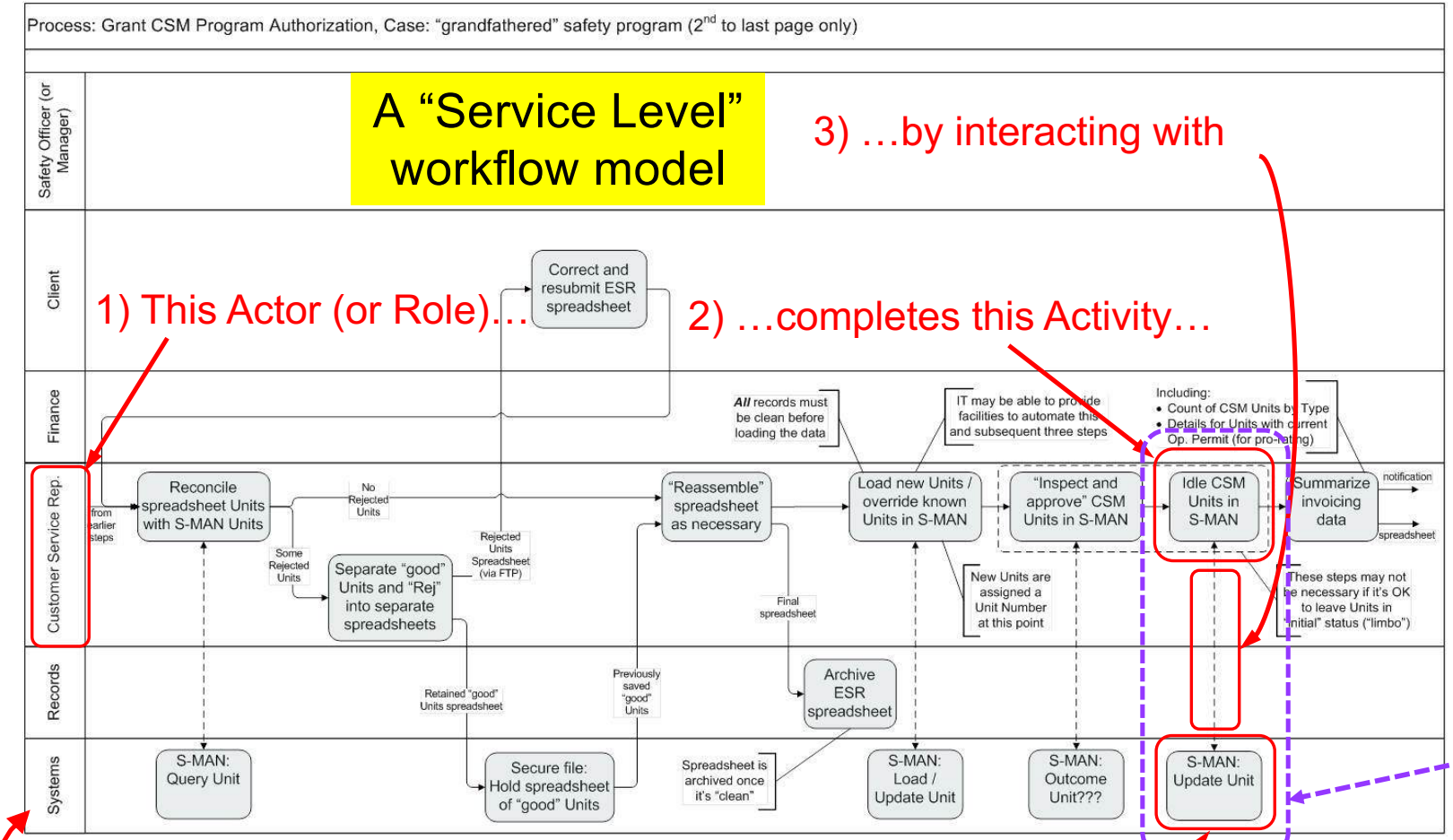
Process Summary Chart – simplified "what," plus "who"

The initial, business-friendly workflow model



A "Handoff Level" workflow model

Then detail showing where use cases & services fit



That's a Use Case!
- an actor
- interacting with a system
- to obtain a service
- to help them complete a task or obtain information

is what we mean by a Use Case (which may begin as a User Story)

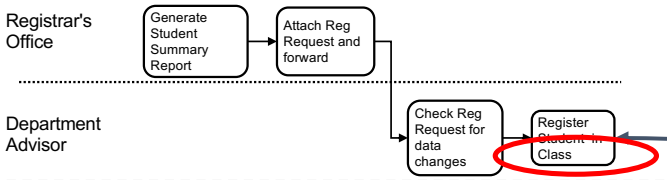
Everything relies on the Concept Model / Data Model

All use the language and constraints of the Concept Model (the "thing model") – the ultimate "what"

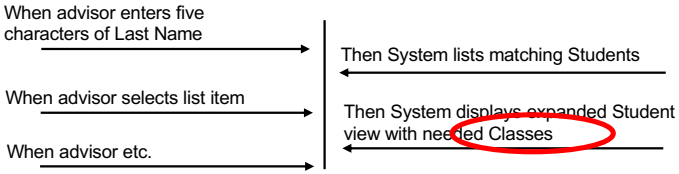
Business Goals & Objectives

The university is initiating the "Strategic Enrollment" program to raise Student graduation rates in part by ensuring Classes are available for Student registration when needed.

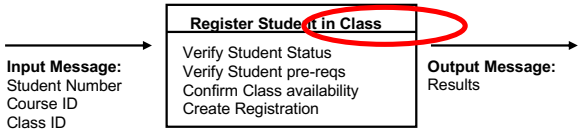
2 Business Process



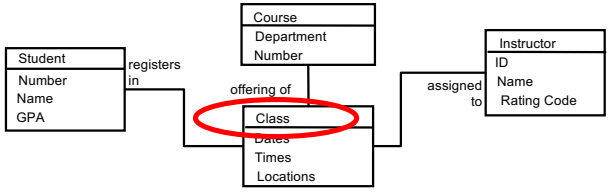
4 Presentation Layer (user interface)



3 Application Layer (rules & logic)



1 Data Layer (data & storage)



My usual sequence

Use Case
actor + service + platform:
Advisor *Register Student in Class* via SRS

Service
verb + noun (+ noun):
Register Student in Class

Entity
noun:
Class

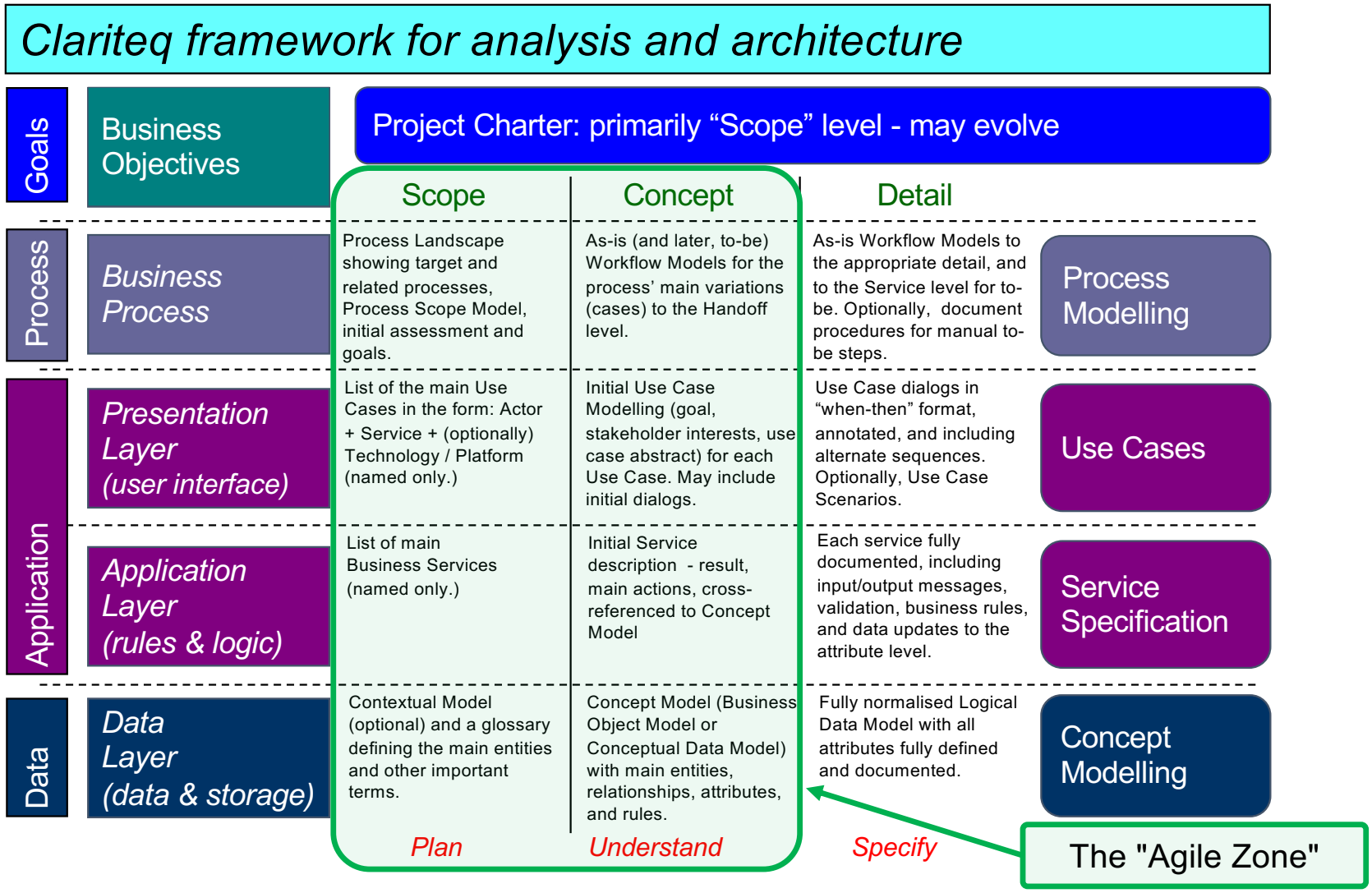
Use Cases/User Stories:
- Who (Actors) needs access to the Services, and how (Platform)?

Verb-Noun pairs:
- The *Services* (event-handlers) that are at the heart of a *Service Oriented Architecture.*
- *Also* "building blocks" of Business Processes

The core *Nouns or* Things in your enterprise. Also known as *Business Objects.*

Bonus – great starting point to discover your Events/Services and Use Cases/User Stories

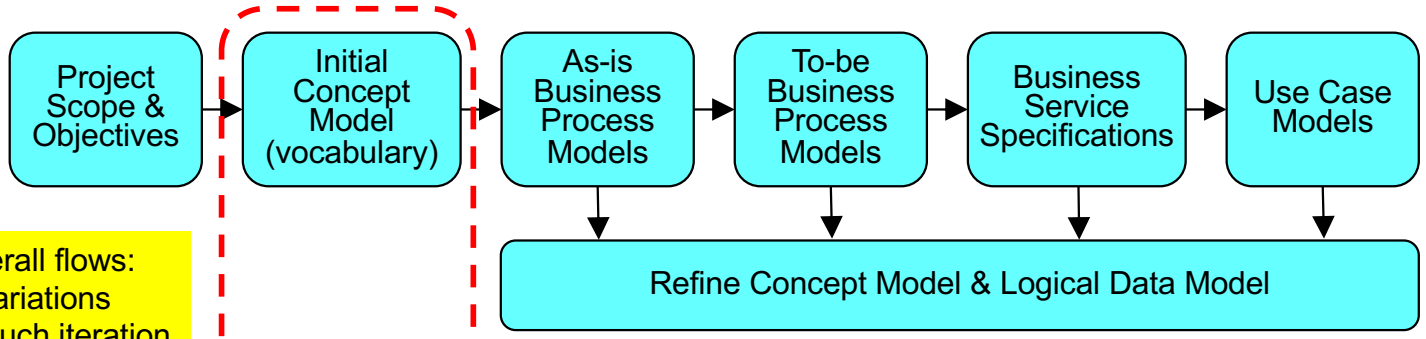
With progressive detail, Concept Modelling supports Agile



Techniques and methodologies

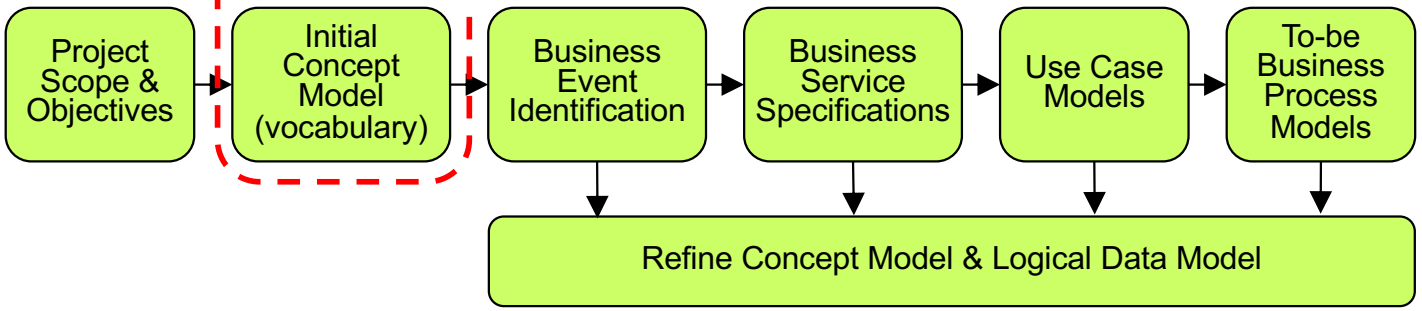
- The same techniques are used in different sequences, with different emphasis, in different situations.
- Concept Modelling to clarify language is a great starting point.

Larger project (enterprise-wide, cross-functional): process-oriented / "outside-in" –



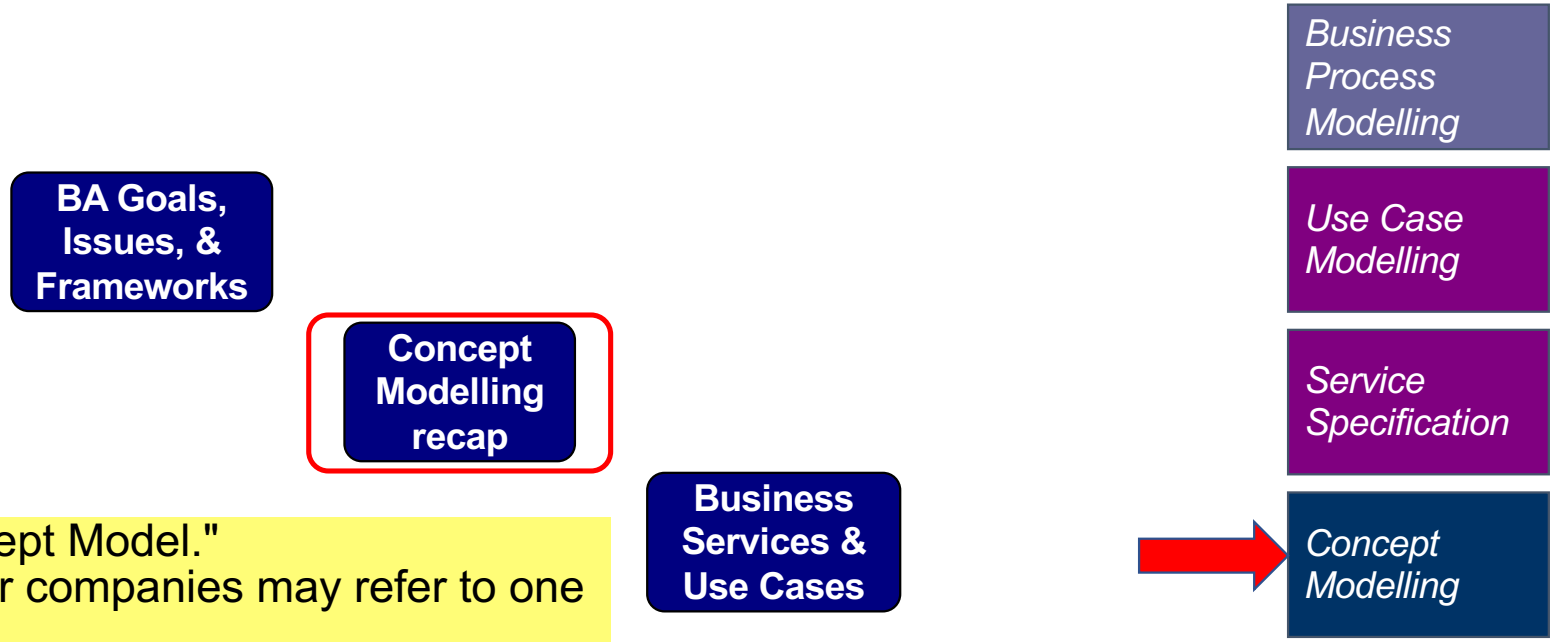
These are typical overall flows:
- there are many variations
- there is always much iteration

Smaller project (local, departmental): service or use case-oriented / "inside-out" –



This slide left blank by accident…

Concept Modelling recap



We use the term "Concept Model."
Your colleagues at other companies may refer to one of these as a:

- Business Object Model
- Conceptual Data Model (and Logical Data Model)
- Domain Model
- …

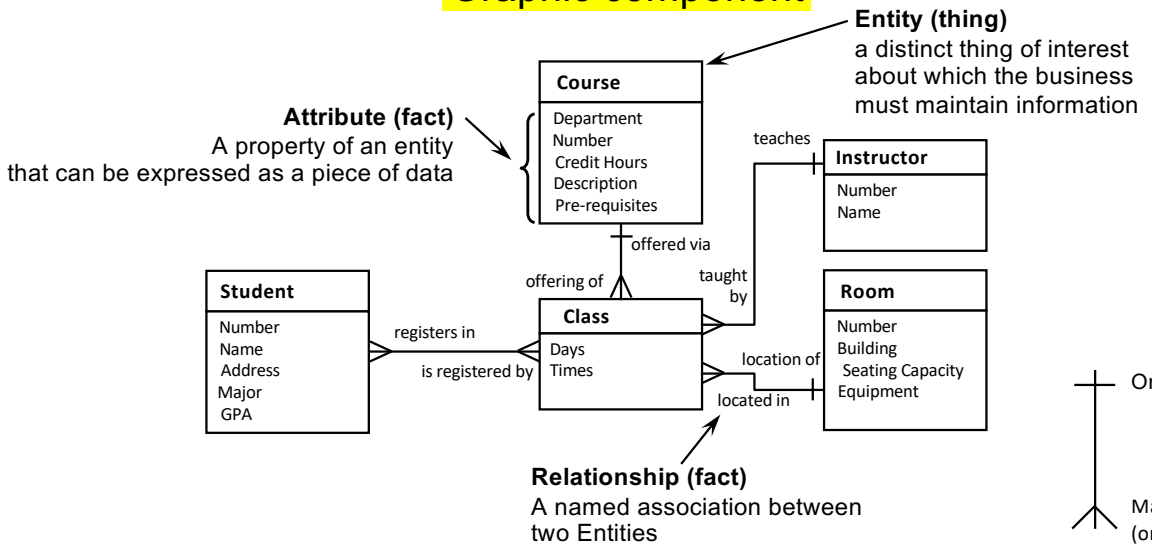
and instead of "Entities" they will refer to " Business Objects." I will do that sometimes too…

What is a Concept Model / Business Object Model / Domain Model…?

- A *description of a business* in terms of
  - **things** it needs to maintain records of – *Entities*
  - **facts about those things** – *Relationships & Attributes*
  - **policies & rules** *governing those things and facts*
- Models a view of the **real world**, not a technical design (therefore, stable and flexible)
- Can be comprehended by mere mortals (at least initially)
- Graham Witt – "A narrative supported by a graphic"

"Things" first, data later!

Graphic component



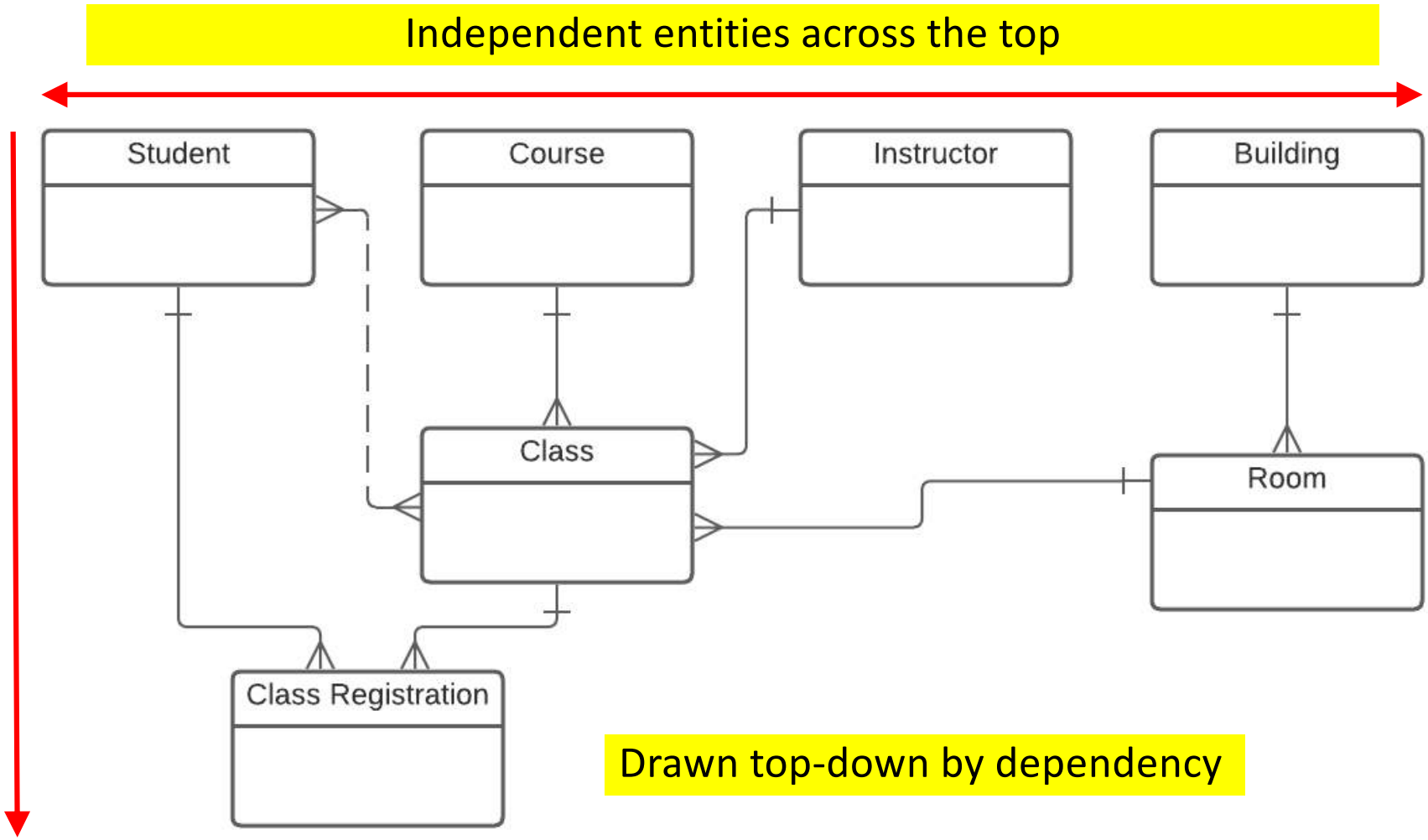
Narrative component

Student definition:
A Student is any person who has been admitted to the University, has accepted, and has enrolled in a course within a designated time. Faculty and staff members may also be Students

Plus "Assertions" (policies & rules)
- Each Course is offered through one or more Classes
  Each Class is an offering of a single, specific Course
- Each Instructor teaches one or more Classes
- Each Class is taught by one Instructor
  (which may or may not be true…)

Many rules can't be shown on the diagram…
- A Student can not register in two Classes of the same Course in the same Academic Term

A better looking version of the model on the previous slide



The basics: ERA – Entities

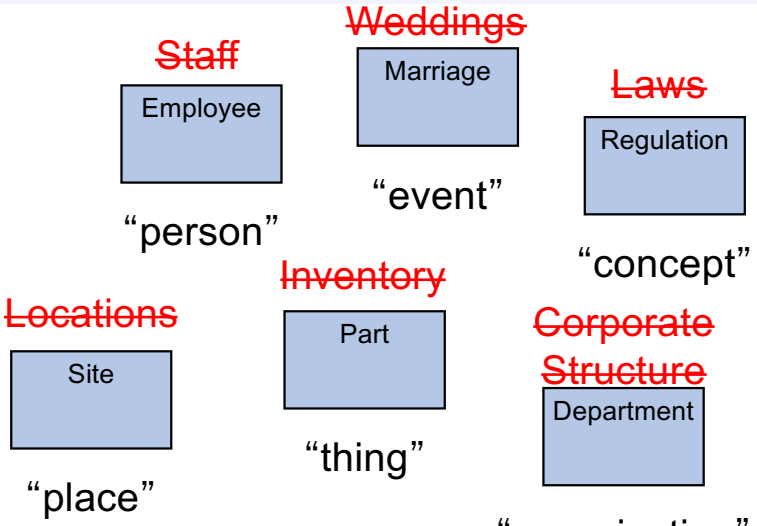
An *entity* is a distinct thing the business *needs* to know about - a *person, place, thing, event, concept,* or *organisation,* and…

- is named with a *singular noun* that implies a single instance
  - not a plural or collective noun, list, set, collection, report, etc.
  - we can discuss "one of them," e.g. "Weather" is not a good name
- has multiple occurrences (or instances)
- we *need* to and *can* keep track of (differentiate) each occurrence
- has *facts* that must be recorded, e.g.
  - *Student* attributes: Number, Name, Birth Date, Major, GPA, …
  - *Student* relationships: "majors in" *Subject*, "enrolls in" *Section*
- is acted on by *processes*, so they make sense in a "verb-noun" pair
- refers to the *essence*, not the implementation – *the most common error is to identify artifacts (forms, reports, spreadsheets, …) as entities!*

Named - a business-oriented noun / noun phrase

Defined - "What *is* one of these things?" or "What do you mean by _____?"

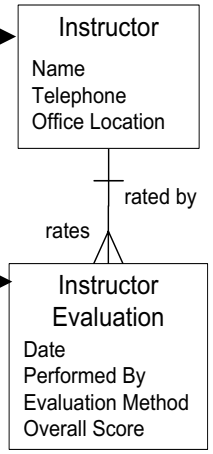
These are the ones our business partners often struggle with.



Two basic types

Independent
- can stand alone
- no relationships "on top" (no parents)

Dependent
- must have one or more parents – one or more relationships "on top" to parent(s)



Naming and definition – the essence of Concept Modelling

Agreement on naming is essential – entities are the *nouns* at the heart of business language. They are what processes act on, applications manipulate, databases record, and BI & analytics tools provide info about.

So, organisations need a *common language* more than ever, for…

- Data integration (data lake, data mesh, data fabric, data virtualisation, data warehouse, operational data store, …)
- Mergers/acquisitions/partnerships/…
- Performance measures, e.g., KPIs
- ***Business Analysis*** *– most requirements can't be stated without using a term from the Concept Model*

Note – it works best if you don't start by talking about *Concept Modelling* or *Data Modelling…*

"Now! That should clear up a few things around here!"

Summary – three types of data models

Different levels of detail support different perspectives

1 Contextual (Scope)

- ✓ Optional - *Context model*
- ✓ Agreement on "big picture," context, and some vocabulary
- ✓ A block diagram of "subject areas," higher level than individual entities
- ✓ Shows the scope or "footprint"

2 Conceptual (Overview)

- ✓ *Concept Model*
- ✓ Agreements on basic concepts, vocabulary, and rules

3 Logical (Detail)

- ✓ *Logical Data Model*
- ✓ Complete detail for physical design

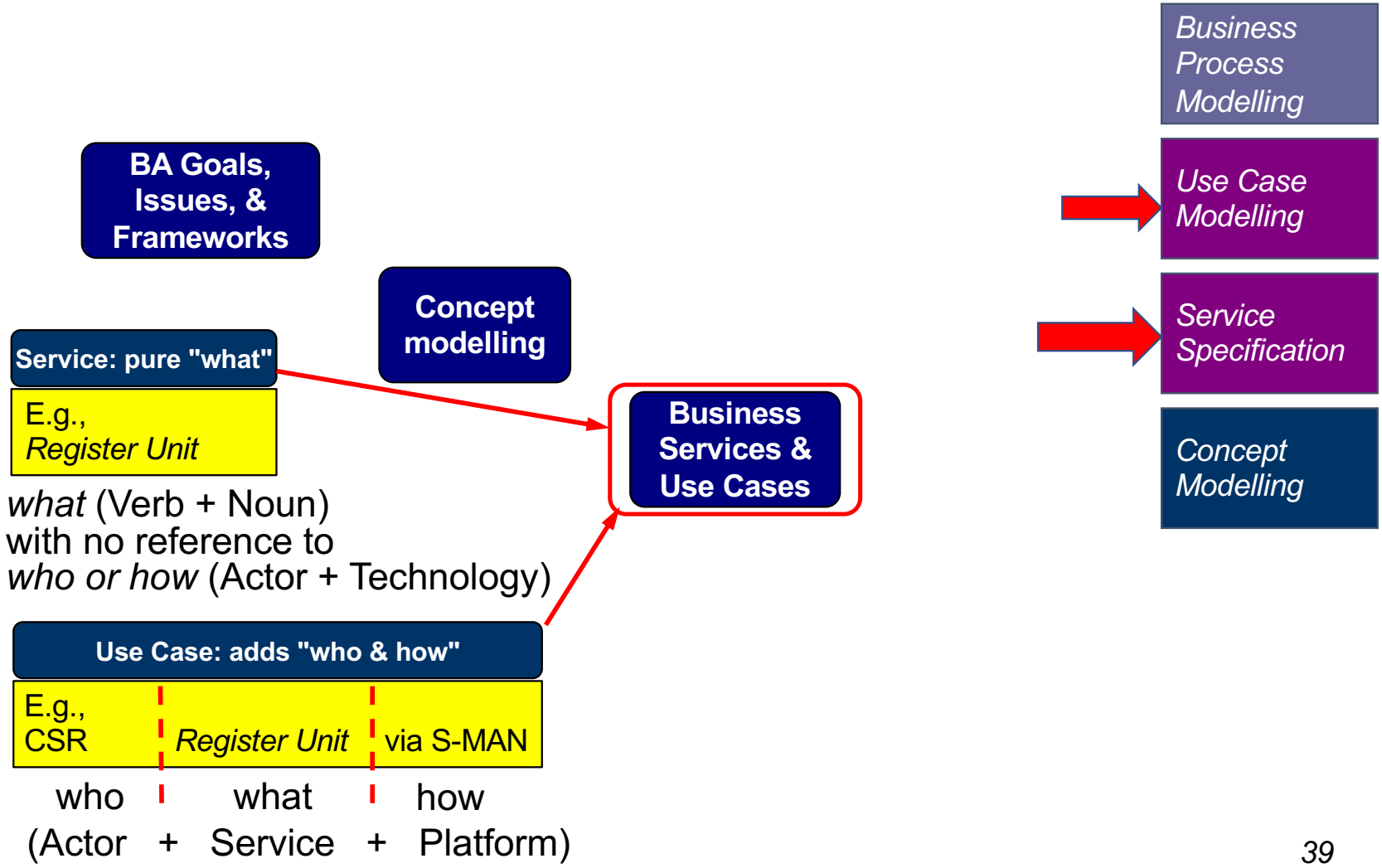
Based on my most plagiarised slide ever!

Some important differences

- ✓ Main ("recognisable") entities only - a singular noun used daily
- ✓ Main attributes only, many are non-atomic
- ✓ M:M relationships
- ✓ Doesn't show keys
- ✓ Not normalised
- ✓ A "one-pager"

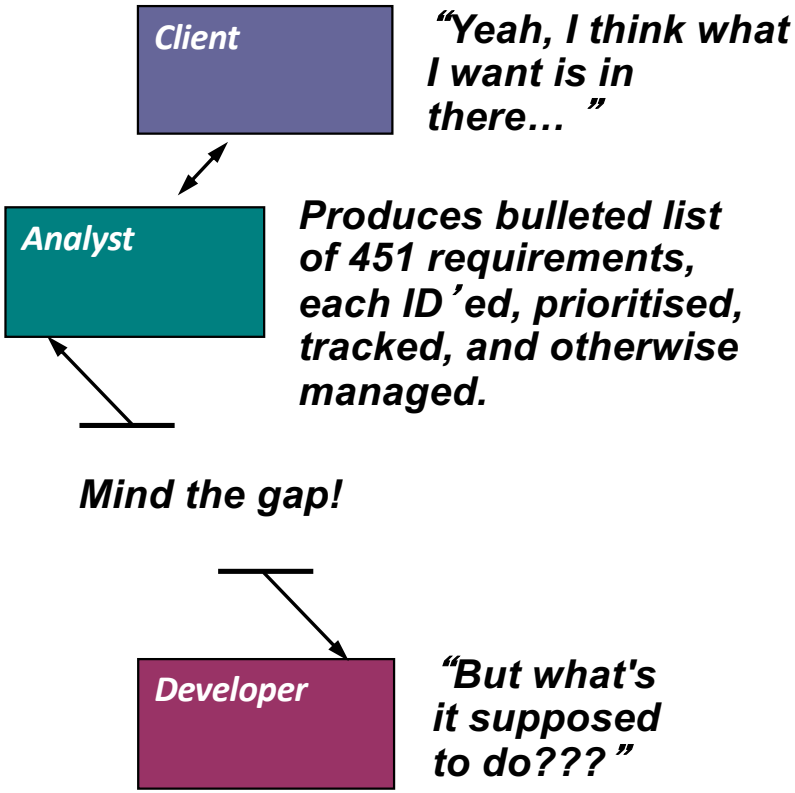
- ✓ All granular entities – many too detailed to come up daily
- ✓ All attributes included, all are atomic
- ✓ All M:M resolved
- ✓ Shows primary & foreign keys
- ✓ Fully normalised
- ✓ Five times as many entities

Business Services & Use Cases – the "what" and the "who & how"

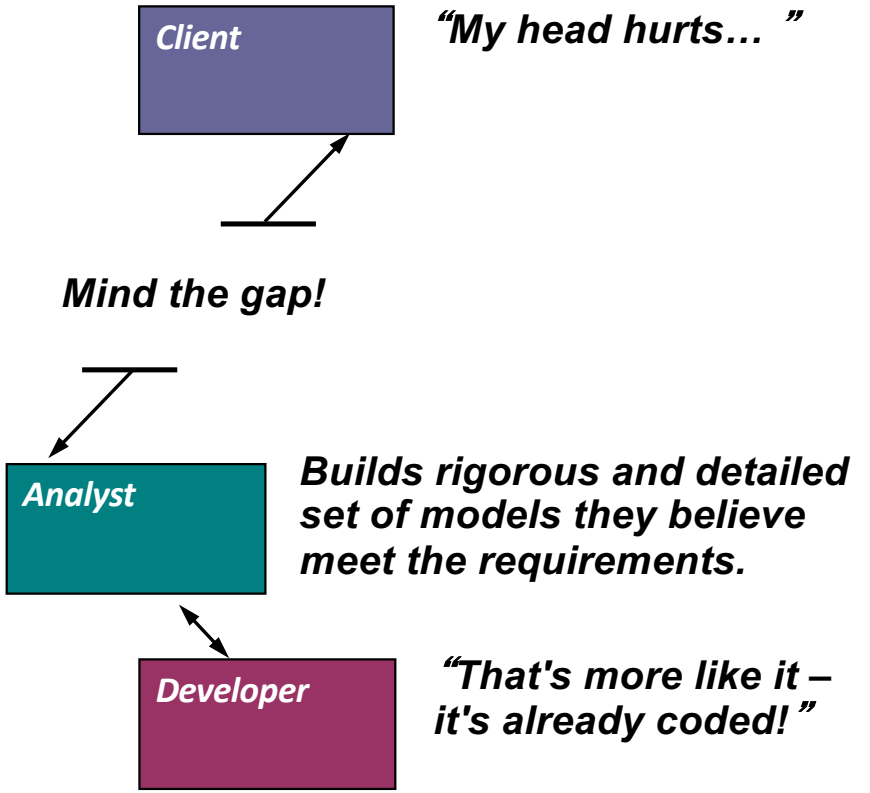


Gaps – business client, business analyst, developer

Business Analyst, new to the field, takes a *Requirements Management* course…



Business Analyst, annoyed, decides "I'll show him!" Learns UML, OCL, ORM, BPMN, and other complex methods…



Use cases to the rescue?

Use case - a description of a specific case in which an actor will use a system to complete a task or obtain a service

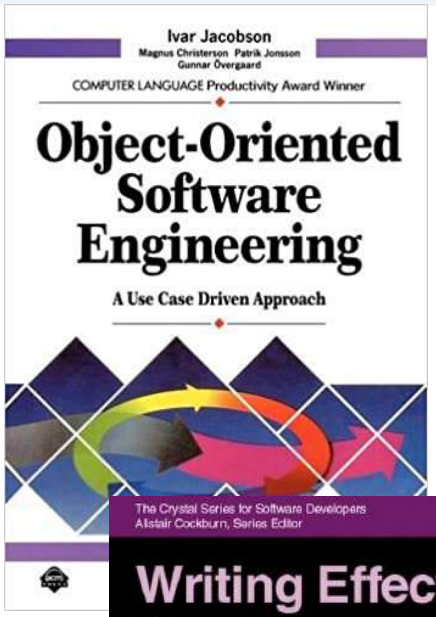
The idea – appealing in its simplicity

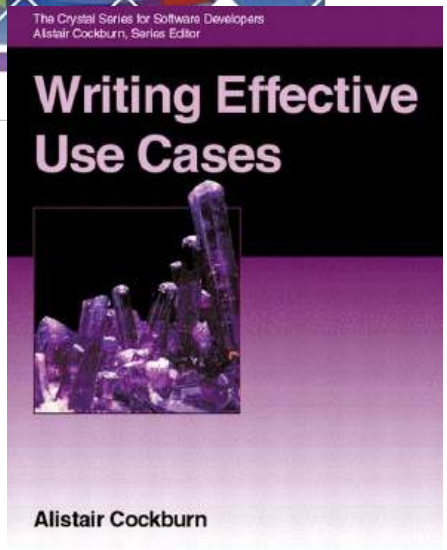
- Recognizable *tasks* provide *context.*

- *"Use cases are wonderful but confusing."* Alistair Cockburn
- *"A use case seems to be anything anyone wants it to be."* Charlie MacLachlan

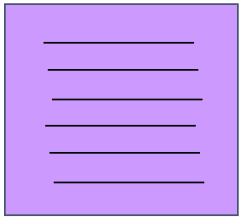
The reality – plenty of grief and confusion

- Granularity, form, content, perspective, used for, …?
- How many use cases?
- Complete, self-contained methodology?
- Excessive complexity of some approaches

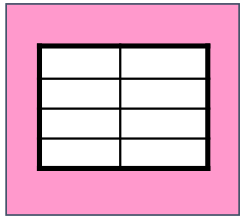




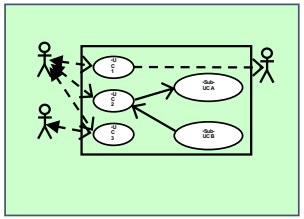
- Will the real Use Case please stand up? -



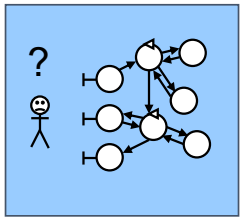
Meandering Narrative



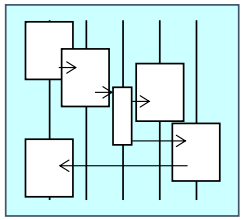
Tabular



UML Use Case



Ideal Object Model



Sequence Diagram

Why the difficulty?

Perspective of many early use case proponents:

"They're all you really need!"

Usually based on three false premises:

1. You don't need to model business processes because *"a business process is just a very large use case"*

2. A concept model / data model is a waste of time because:
   (a) Data requirements will *"fall out of"* the use cases
   (b) No one understands your stupid data model anyway

3. A use case (or user story) is the *best* and *only* mechanism for capturing functional requirements.

Traditional use cases can't do it all, part 1

Process

Business Process

- Process models (e.g., scope models and swimlane diagrams) are well understood
- Provide *context* – actors, tasks, sequence
- Help to ensure we aren't automating bad processes or unnecessary tasks.

Application

Application requirements

The central question –
are *use cases* the best mechanism for discovering, documenting, and verifying *all* application requirements?

NO!!!

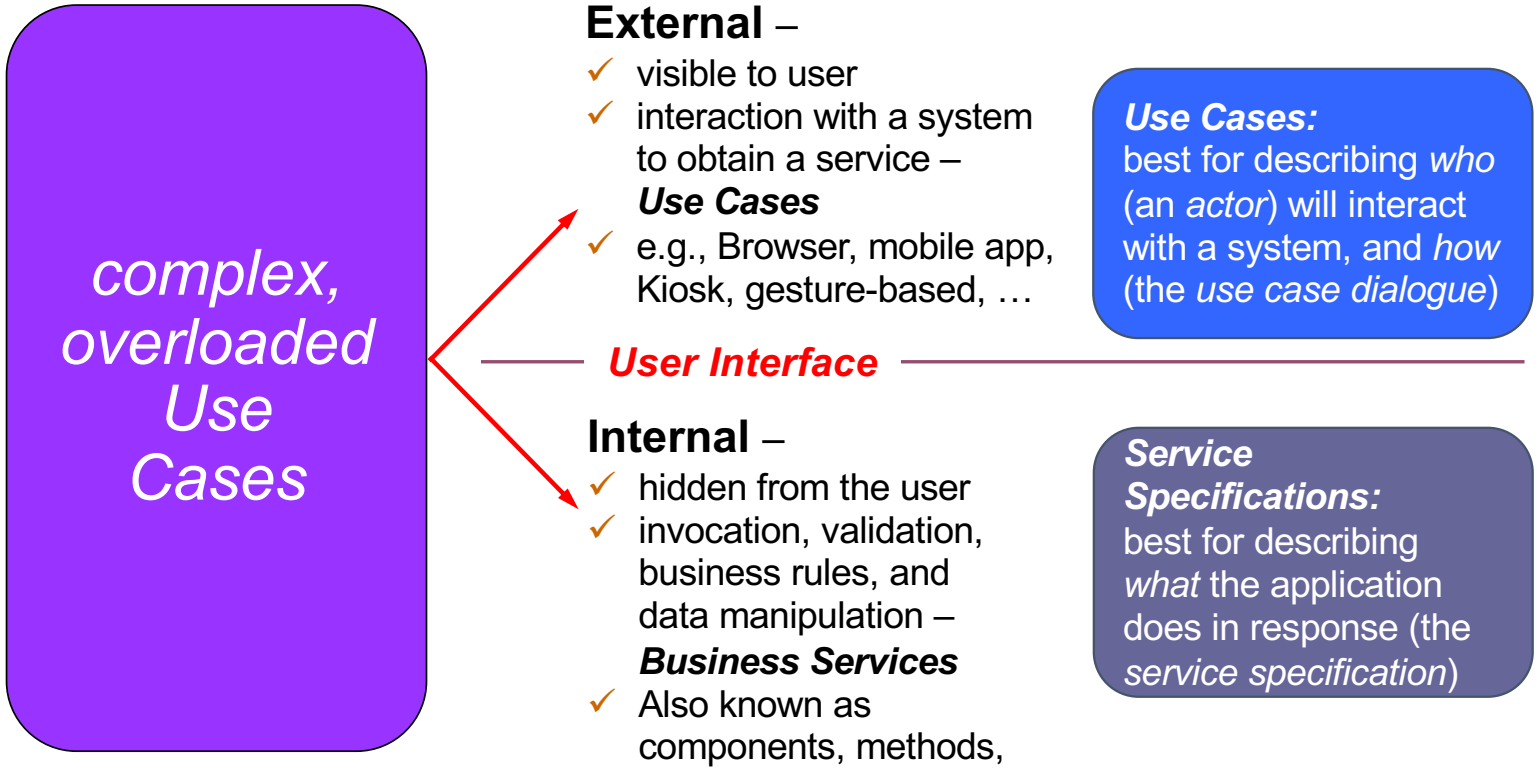
Data

Data Management

- Concept Models *can* be well understood, with care
- Provide a *platform* – vocabulary, structure, dependency
- Help to ensure we're all using the same terminology, and employing the same basic concepts and rules

Traditional use cases can't do it all, part 2

The central problem with many use case methods is a failure to recognise that applications have external and internal views –
"techniques that work for one don't work for another"



Services, Use Cases, Use Case Scenarios

Review, Check, Monitor, Track, Analyze, Enable, Handle, Process, Manage…

No mushy verbs!!!
"Noun is Verbed" (Order is Placed) must be an essential event.

Business Service:
Place Order

- ✓ Abstract or "essential" - no reference to "who or how"
- ✓ Action verb + noun (+ noun)
- ✓ Helps us to focus on the *essence* of *what* must be accomplished to operate the business
- ✓ Often surprising to a business to see what it really does, stripped of all procedural overhead

Use Case:
Customer Places Order via Web

- ✓ Generalised (or "abstract")
- ✓ Actor + service (or goal) plus (usually) technology (browser, purpose-built kiosk, IVR, …)
- ✓ Helps us document different situations
- ✓ Same service can be accessed via multiple use cases
- ✓ Demonstrated in multiple UC scenarios

Use Case Scenario:
Joe Bloggs, a Platinum customer, places a complex order involving four ship-to addresses…

- ✓ Specific (or "concrete")
- ✓ A scenario comprising a "worked example" of one or more linked Use Cases
- ✓ Scenario – a story or "vignette" including named actors, specific data values, and predefined decision outcomes.
- ✓ Helps put UCs in context, so users can contribute / verify.

Relationships among Services, Use Cases, Use Case Scenarios

Key Point

We follow an "inside-out" approach – services first, then use cases

A single indivisible capability that must be carried out in order for the business to operate. It is the system's response (rules and data updates) to some trigger, and results in a business event

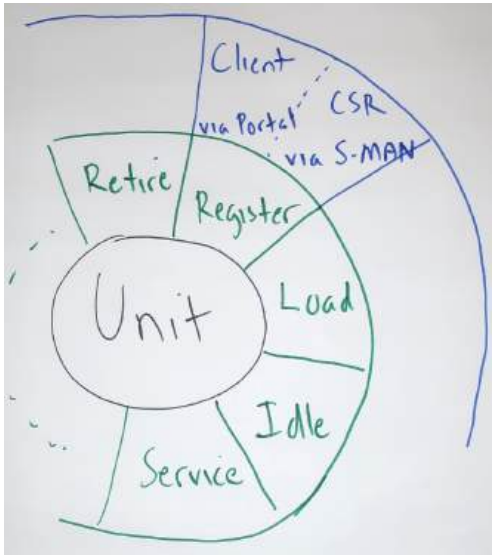
How a specific actor, with a specific technology, will interact with the system to obtain the service.

A "worked example" – a walkthrough of an actual session, encompassing one or more use cases, using named actors, predetermined data values, and predetermined decision outcomes. It is a single sequence of interactions with no branching or alternative flows.

Discussion – one Business Service, one or more Use Cases

One Service

	Who	What (the Service – verb + noun)	How
Multiple Use Cases	Client	Register Unit	via Portal
	Customer Service Rep (CSR)	Register Unit	via S-MAN (the ERP)
	Client	Register Unit	via Mobile App
	???	Register Unit	???



What is the value of separating the Service from the Use Case and documenting it only *once?*
(This is a unique feature of our approach.)

- …
- …
- …

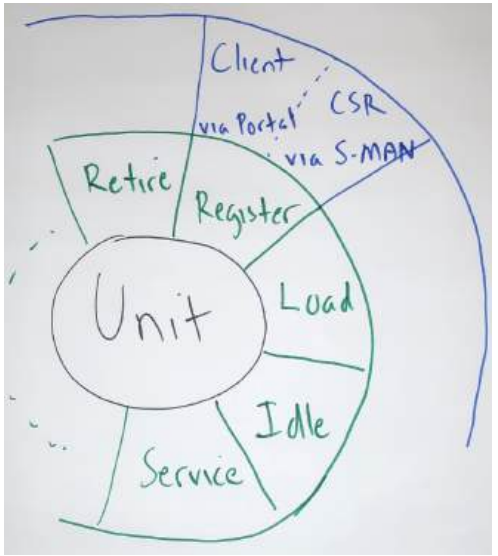
Why would we make a *single* Service available via *multiple* Use Cases?

- …
- …
- …

Discussion – one Business Service, one or more Use Cases

One Service

	Who	What (the Service – verb + noun)	How
Multiple Use Cases	Client	Register Unit	via Portal
	Customer Service Rep (CSR)	Register Unit	via S-MAN (the ERP)
	Client	Register Unit	via Mobile App
	???	Register Unit	???



What is the value of separating the Service from the Use Case and documenting it only *once?*

- re-use of the asset, and therefore higher consistency
- better chance of getting it right – higher value from less effort
- if it's implemented as a single service, easier maintenance – it's in ONE place.

Why would we make a *single* Service available via *multiple* Use Cases?

- different actors need different "navigation and hand-holding," e.g., casual vs. expert users
- different technology platforms have different capabilities, e.g., mobile phone vs. touch-screen kiosk

Are services important? – ask Jeff Bezos

The power of an SOA platform

In 2002, Jeff Bezos, CEO of Amazon, determined that its business strategy was to transition its website to an online retail *platform*. To that end, he sent the following email to all employees.

1. All teams will henceforth expose their data and functionality through ***service interfaces.***
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of inter-process communication allowed:
   no direct linking, no direct reads of another team's data store,
   no shared-memory model, no back-doors whatsoever.
   The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pub/Sub, custom protocols — doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be *externalisable*. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.

Multiple discovery approaches

Choose *one* as your *primary approach,* use *others* to verify & extend – there is no *one best way*

We did this on Day 1

Good for completeness

Entity based

1. ID core entities
2. List primary services (or events) for each
3. Describe service rules and actions, and optionally state diagrams and other service logic
4. Allocate services to actors, yielding Use Cases
5. Proceed with Use Case development

Good for big picture, context

Workflow based

1. Design new workflow
2. Isolate "system supported" steps, which are Use Cases for that actor
3. Proceed with Use Case development

Discovering Business Services and Use Cases

Good for relevance, buy-in

Actor based

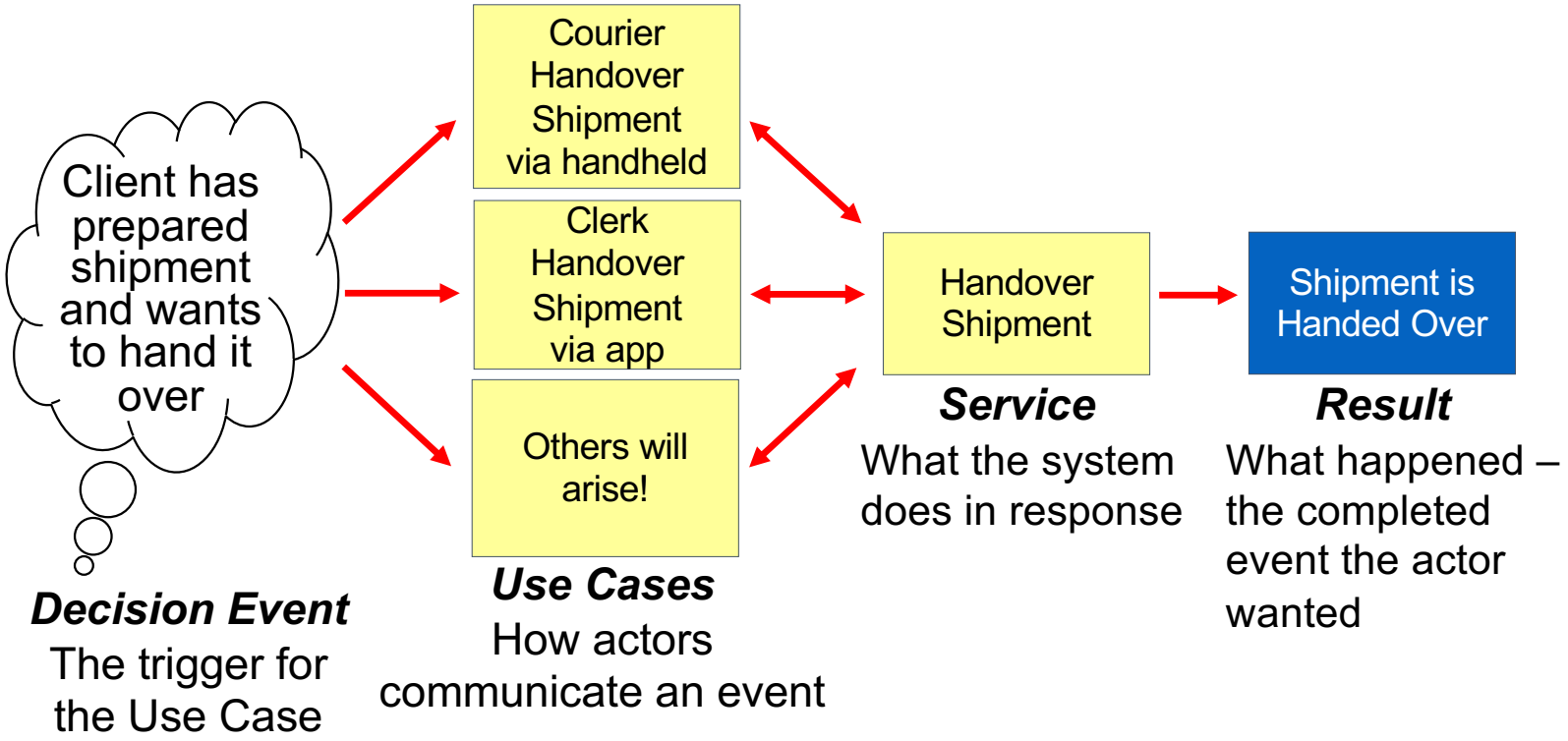
1. ID actor
2. ID services or goals (use cases) for each actor
3. Proceed with Use Case development

Good for finding outliers

Unstructured

1. What else happens that the system must respond to?

Use cases, services, events



Decision Event
The trigger for the Use Case

Use Cases
How actors communicate an event

Service
What the system does in response

Result
What happened – the completed event the actor wanted

You can look at a use case as a mechanism for:

- ✓An actor to notify the system that the actor *wants* the event to happen, e.g., Place Order
- ✓An actor to notify the system that an event *has* happened, e.g., Change Address and they want the system updated (after the fact)

Use Cases at an ATM

We'll use the ATM as an example –
it's familiar and demonstrates key principles.

1) What actors interact with an ATM?

2) What Services would each of the actors like to access at an ATM?

Customer

-

-

-

…

Use Cases at an ATM

We'll use the ATM as an example –
it's familiar and demonstrates key principles.

1) What actors interact with an ATM?

2) What Services would each of the actors like to access at an ATM?

Customer –

1. Withdraw Funds
2. View Balance
3. Deposit Funds
4. Transfer Funds
5. Change PIN (or Call Tech Support)

Bank Employee (Teller) – Replenish Supplies

Bank Manager – Add/Drop Access Permission

Technician – Update Software

Service Provider (Brinks) – Replenish Cash

Core Banking System – Check Connection, Stop/Start ATM

Others? – YES! Manufacturer, Regulator, Other Bank, Scammer/Thief

Customer

Bank Employee (Teller)

Bank Manager

Technician

ATM Service Provider (Brinks)

Core Banking System

…?

Three kinds of events trigger services

Action Event or Decision-Based Event	*Temporal Event or Time-Based Event*	*Conditional Event or Data-Based Event*

Raised by an actor deciding to do something
 	
E.g.,
• Decision to Place Order: actor Place Order
• Decision to Raise Employee Salary
• Decision to Submit Complaint
	Raised by the system when a predetermined date/time is reached
	
	E.g.,
	• Time to Place Recurring Order: triggers Place Order
	• Time to Pay Employee
	• Time to Submit Financial Statements
		Raised by the system when a predetermined threshold is reached
		
		E.g.,
		• Inventory Reorder Level is Reached: triggers Place Order
		• Temp >0C & <40C:
		  • High temp threshold hit
		  • Low temp threshold hit

Always introduces new data to the system	*Does not* introduce new data to the system	*Might* introduce new data – the measurement and time
A *Real* Use Case A Use Case for a human actor to convey the event to the system	A *System Use Case* Also needs a *Real* Use Case to “set” the alarm	A *System Use Case* Also needs a *Real* Use Case to “set” the data threshold

Business Service perspectives

A Business Service is a package of validations, business rules, operations, and updates to stored data, triggered by a message.

- Business Perspective -	- Systems Perspective -
✓ A discrete, logically complete unit of work – can't be broken into smaller units that would still yield a complete result	✓ The building blocks of an application – implemented as components, methods, services, etc.
✓ Focus on "what" rather than "who or how" provides a new perspective for the business	✓ Focusing on "what" rather than "who or how" simplifies analysis
✓ Understand and participate in consensus on rules	✓ Re-usable, independent components provide integrity and flexibility

Business Services

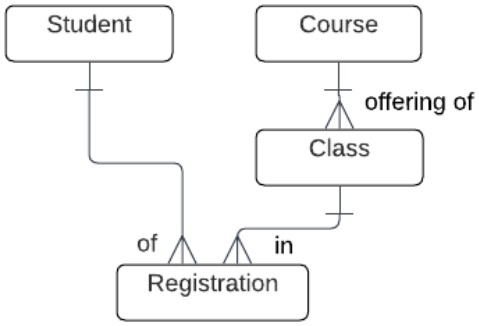
What, not *who* or *how* – completely independent of actor and user interface.

An *indivisible*, *atomic, all or nothing* business event.

Meets *ACID* criteria

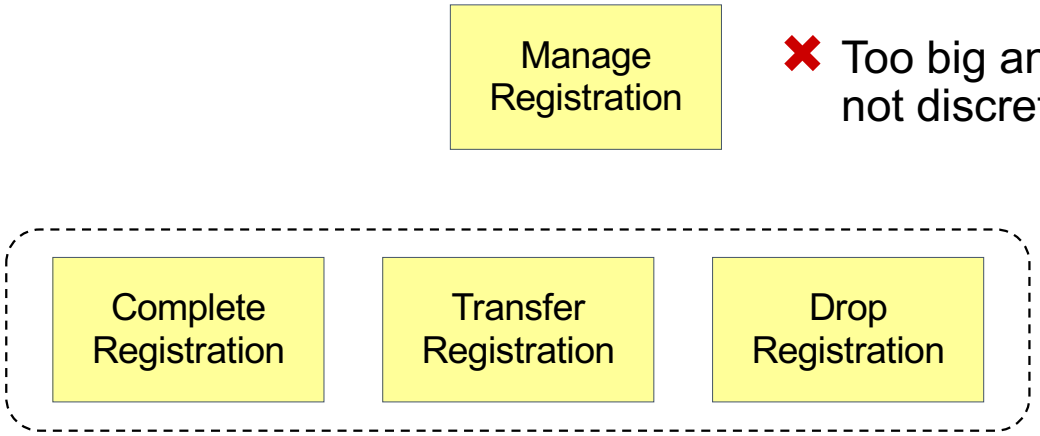
- *atomic*
- *consistent*
- *isolated*
- *durable*

Acts on the entities in the *Concept Model*



Granularity is everything

A Business Service completes a single, discrete, *business–oriented* unit of work – a business event



Manage Registration

✖ Too big and mushy – not discrete!

✓ *Just right! – real business events*

Create Registration

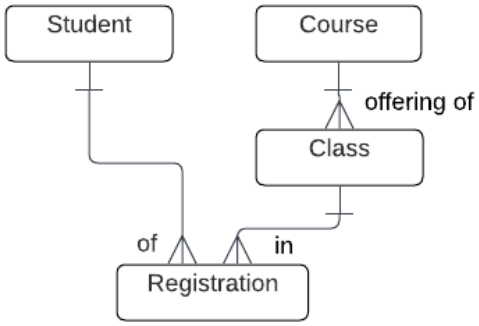
Read student

Update class

Delete Registration

✖ Too small, too internals-focused – "CRUD" (Create, Read, Update, Delete)

Discussion – which Services do we need?



Complete Registration

Transfer Registration

Drop Registration

It is often suggested we don't need "Transfer Registration." Instead, we could:

- "Drop Registration" to delete a Registration in the Class the Student is leaving
- "Complete Registration " to create a Registration in the Class the Student is transferring to.

What are the flaws in this logic?

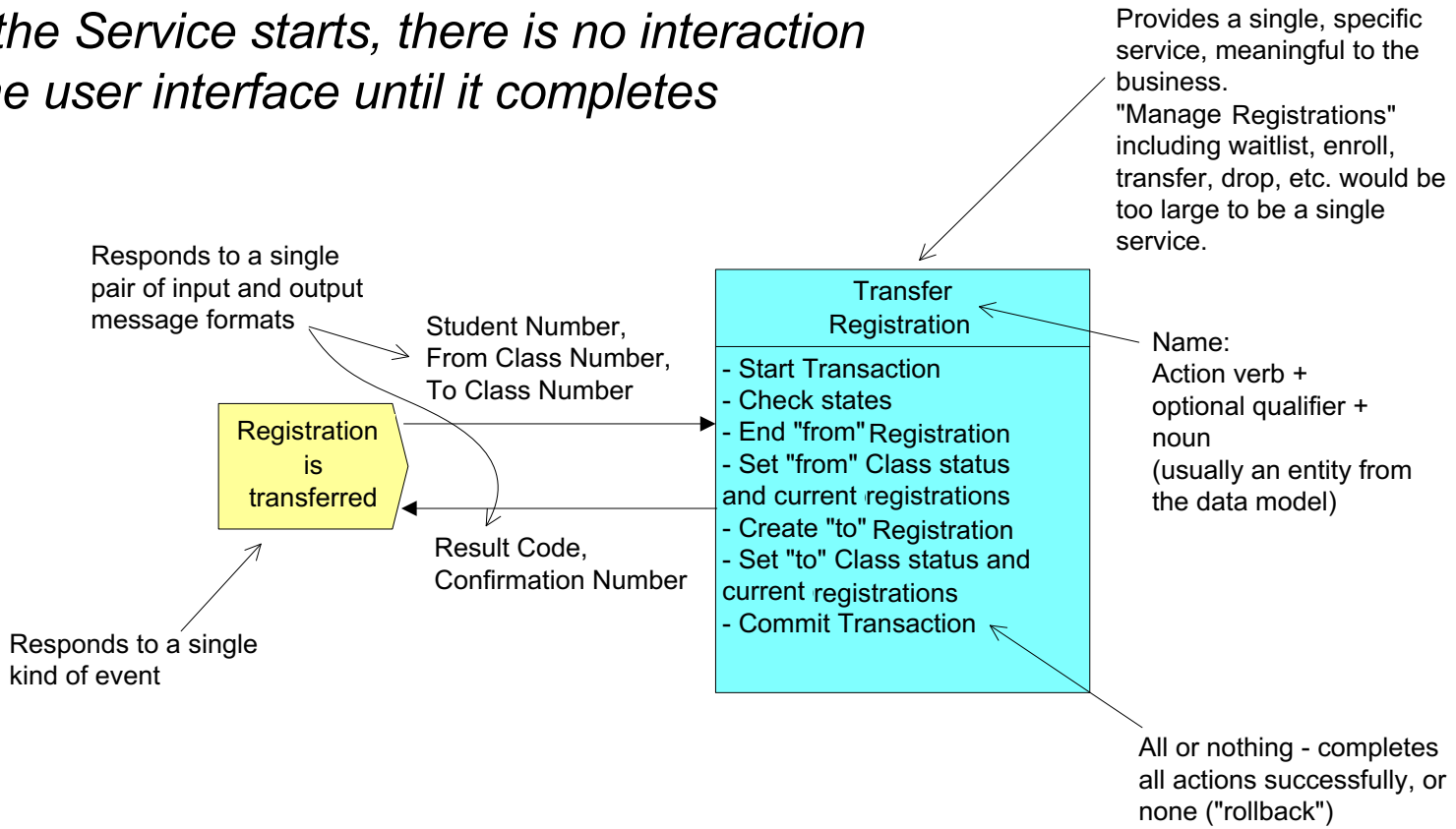
- Cannot enforce rules specific to "Transfer" – fees, limits, dates, …
- Lose data on "Transfer" events – who, how often, why, …?
- Student might lose the last space between "Drop" and "Complete" – another Student gets the last space in the "To" Class

Services *must* represent real business events!

Business Service guidelines

"What", not "who" or "how" – *completely independent of actor and user interface.*

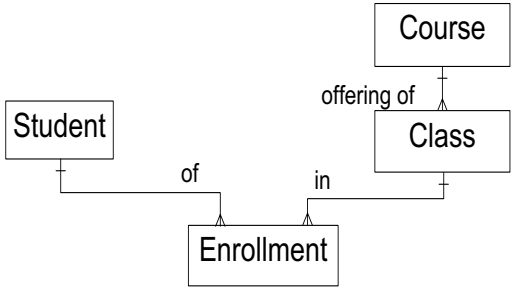
Once the Service starts, there is no interaction with the user interface until it completes



Initial (concept level) service specification

	Component	Description	Notes
1	Name	The Business Service name	• Typically "action verb + noun" or "action verb + noun + noun"
2	Result	A short (1 – 3 sentence) description of how the world (and therefore our records of it – files or databases) are changed by successful completion of the service	• Must use the language of the Concept Model and any other pre-defined artifacts (e.g., standard calculations like metrics) • Must make sense to both business and technical audiences
3	Action	5 +/- 2 (give or take) bullet points describing the key steps that comprise the service	• Again, uses the language of the Concept Model and any other pre-defined artifacts, and makes sense to both business and technical audiences • Focus is on "what, not how" and successful completion (not all the exceptions) • Will describe essential validation, and the core operations and data updates • Corresponds, in part, to "acceptance criteria" in a user story
4	Notes	Any additional requirements, assumptions, or questions that arise	• May include requirements (e.g., constraints or business rules) that will later be captured in the detailed service spec, or elsewhere, e.g. the use case or concept model

Initial (concept level) service spec example



Component	Description
1 **Name**	Complete Registration
2 **Result**	Enrolls a qualified Student in a single Class by creating a Registration record linked to Student and Class
3 **Action**	• Validate Student status and prerequisites • Confirm space in Class • Create Registration, link to Student and Class • Generate confirmation number • Revise remaining space in Class • Apply fee to Student Account
4 **Questions & Assumptions**	• Does this need to be able to create a Registration in "waitlist" status?

Now, review main actions with subject matter experts (SMEs) and ask:

- "Would you usually do *more* or *less* than this?"
  That is, is the service too *small* or too *big*?
- "Have we missed any important Actions?"

Discussion – build an Initial Service Spec

	Component	Description
1	Name	Transfer Funds (or "Complete Transfer Transaction")
2	Result	
3	Action	
4	Questions & Assumptions	

Discussion – build an Initial Service Spec for Transfer Funds

	Component	Description
1	Name	Transfer Funds (or "Complete Transfer Transaction")
2	Result	Moves a specified amount of money from one of a Customer's Accounts to another of the same Customer's Accounts, either immediately or at a future date and time.
3	Action	• Confirm Customer & Account existence and status; • Ensure Transfer Amount is within limits; • Confirm adequate balance in "From" Account; • Move funds "From" to "To"; • Generate Confirmation Number. • Update "From" and "To" Account Balances:
4	Questions & Assumptions	• Same Currency or Cross-Currency?

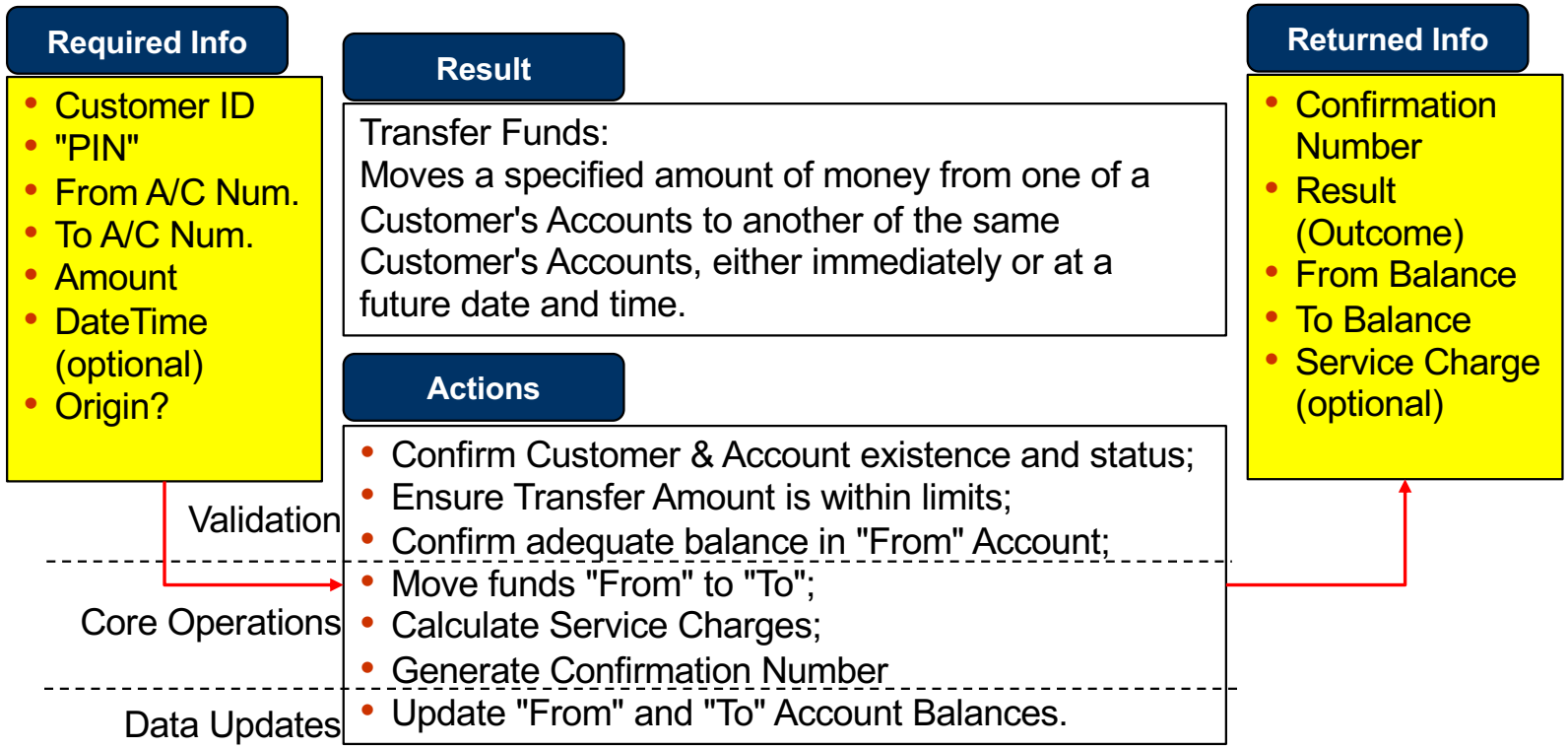
Intent – Determine if the Client and the Analyst share the same basic understanding.

- Anything important missed? (e.g., Calculate Fees, Eligible Accounts…)
- In Agile terms, the points are "Placeholders for a conversation."

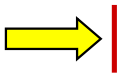
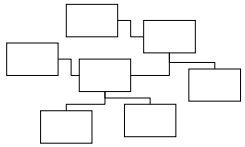
Extending the Initial Service Spec

Sometimes, to clarify our Initial Service Spec we add elements from the Detailed Service Spec:

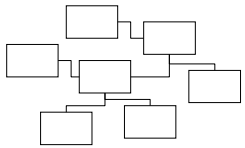
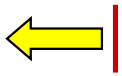
- Break Actions into Validation, Core Operations, and Data Updates;
- Define the Required Information (Input Message) and Returned Information (Output Message) formats.



Final (detail level) service specification – 1

	Component	Description	Example	✓ Tips
1	Name	• Action verb - noun	Transfer Registration	• *Avoid "mushy" verbs*
2	Result 	• A description of how the 'state of the world' has changed on successful completion	A Registration in one Class of a Course is ended, and a new Registration in a different Class of the same Course is established	• *Precisely describe the result both in business terms and with respect to the Concept Model*
3	Required Info 	• The minimal list of attributes necessary to initiate the service	Student Number, From Class Number, To Class Number	• *Different message formats imply different services*
4	Validation 	• The rules that determine whether the service can proceed • Read stored data and check: - entity existence - entity state & type - entity data values	The Registration record for the Student in the "from" Class must exist ("existence") and currently be in "active" state ("state") and the available spaces in the "to" Class >= 1 ("data value")	• *Check other entities "in the neighborhood"* • *Almost all important entities will have a "state" attribute*

Final (detail level) service specification – 2

Component	Description	Example	✔ Tips
5 **Functions** X+Y = Z	• Implements calculations, algorithms, rules, etc. E.g., limits, totals, taxes, charges, etc.	If the transfer request is AFTER the Class start date, APPLY transfer penalty	• *Be careful with time and date ranges, checking for conditions such as contiguous ranges, or gaps or overlaps.*
6 **Data Updates** 	• All of the creates, updates and deletes made to the entities	Set the old Registration record to the status "ended" AND create new Registration record with the status of "active"	• *More common to change the status of a record and date stamp it than to delete it*
7 **Returned Info** 	• The list of attributes sent back to the user interface	Result(s), Confirmation Number	• *The minimum output message is usually a result/ status code*

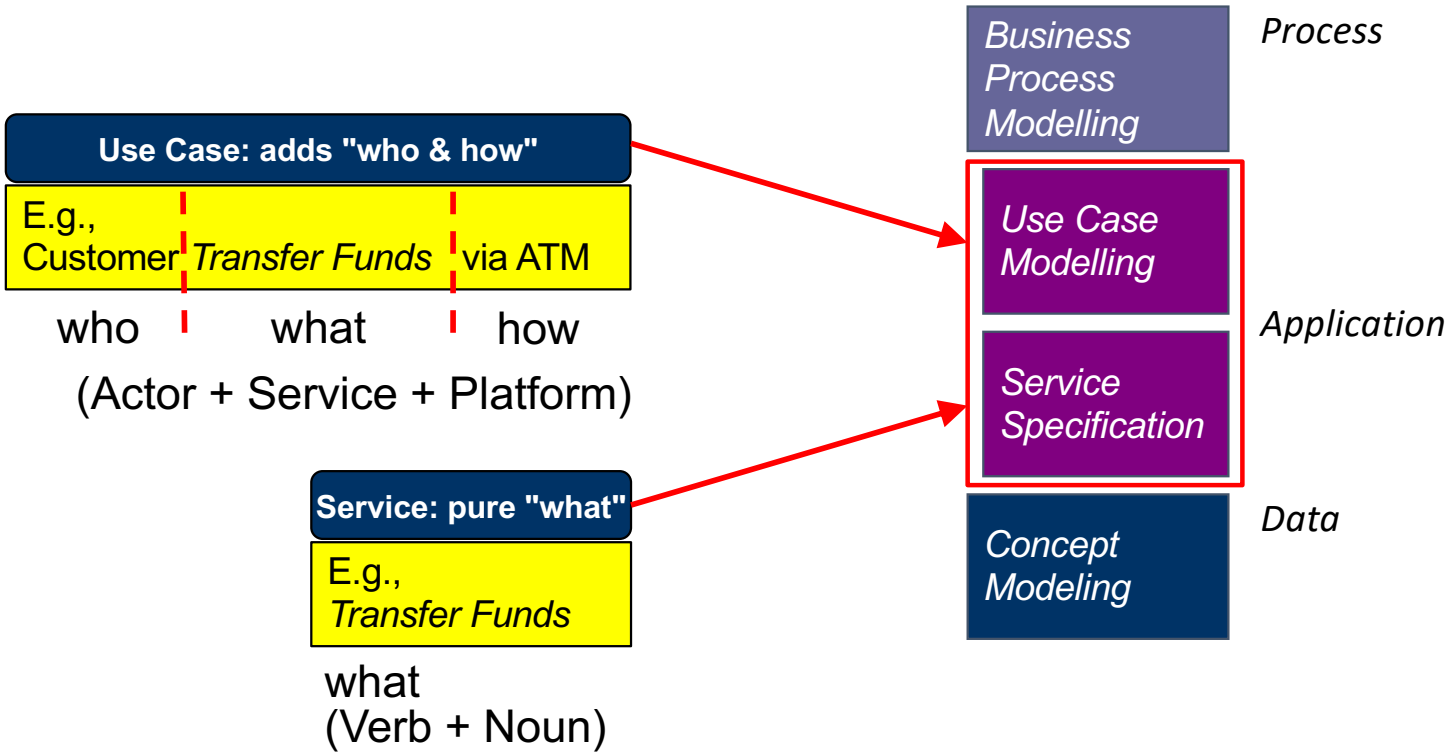
Service specification example

Service Name
Transfer Registration
Result (Description)
A Student's Registration in one Class of a Course is ended, and a new Registration in a different Class of the same Course is established
Input Message
Student Number, From Class ID, To Class ID
Output Message
Result(s), Confirmation Number
Actions—Validation
• Student must be in "registered" state • "from" Class must be in "filled" or "available" state • "from" Class Registration must be in "active" state • "to" Class must be in "available" state • "to" Class Registration must not exist • Transaction date must be on or before Semester last drop date
Actions—Operations and calculations
• If start date has passed, calculate transfer fee and new student account balance • Calculate new registrations count for "from" Class, and set state to "available" if it was "filled" before • Calculate new registrations count for "to" Class, and set state to "filled" if registrations count is now at maximum • Set Result Code • Set Confirmation Number (as per rules in data model) • Etc.
Actions—Updates
• Set "from" Class Registration to "ended" state, and set state change date • Create "to" Class Registration in "active" status • Update "from" Class as determined • Update "to" Class as determined • Update Student Account balance due.
Notes
• Investigate need to provide transfer fee exemption number • Investigate need to provide late transfer approval number

Checkpoint: whether you did Concept or Detail level Service Specifications, you have reduced the unknowns, and now have an excellent platform for Use Case development:

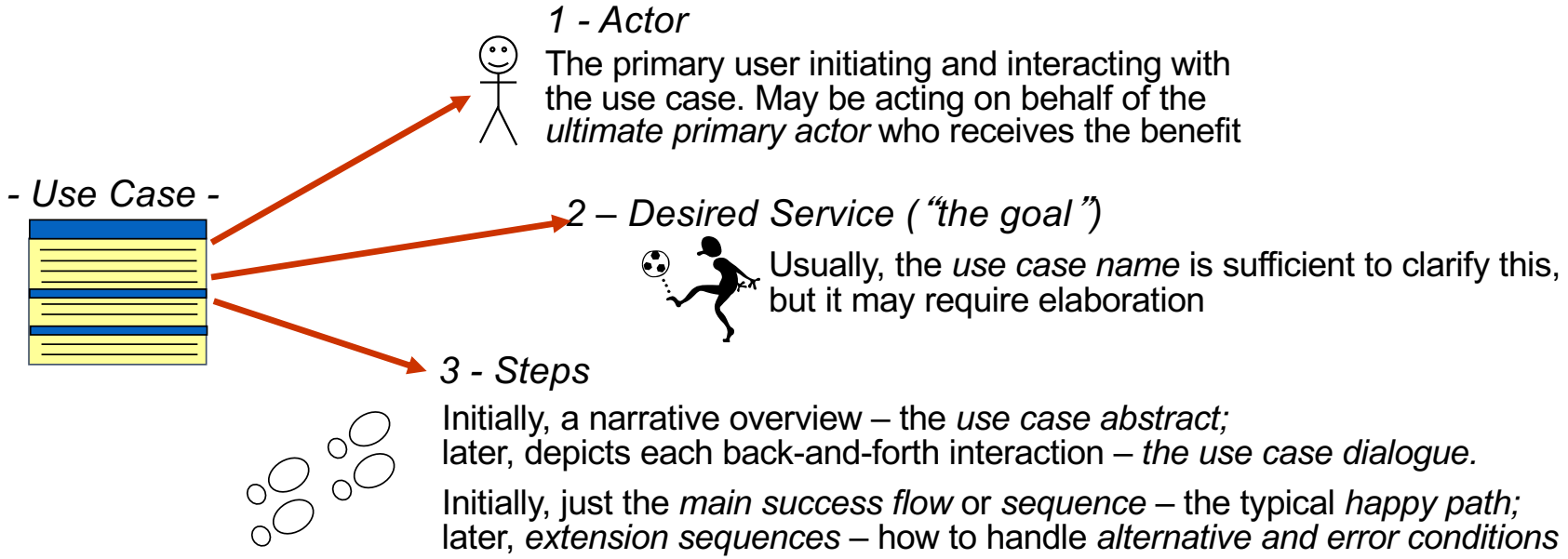
- Granularity
- Expected result
- Validation and actions

Finally, add "who" & "how" to Services to ID Use Cases



Use Cases – essential components

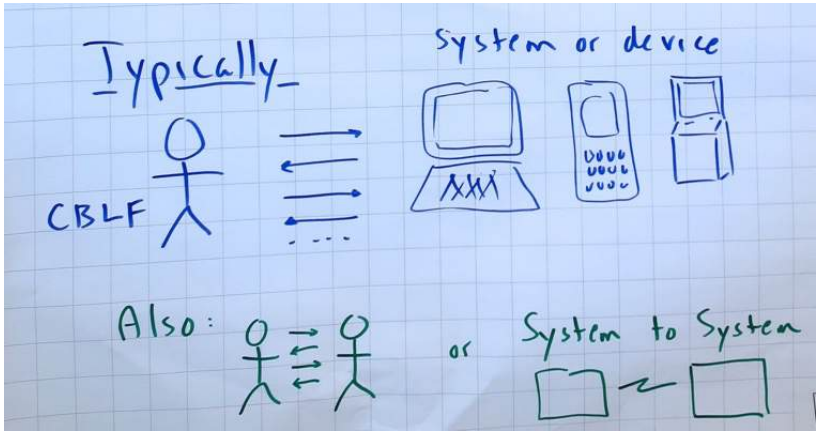
There are lots of detailed, confusing Use Case templates available,
but the essence is straightforward…



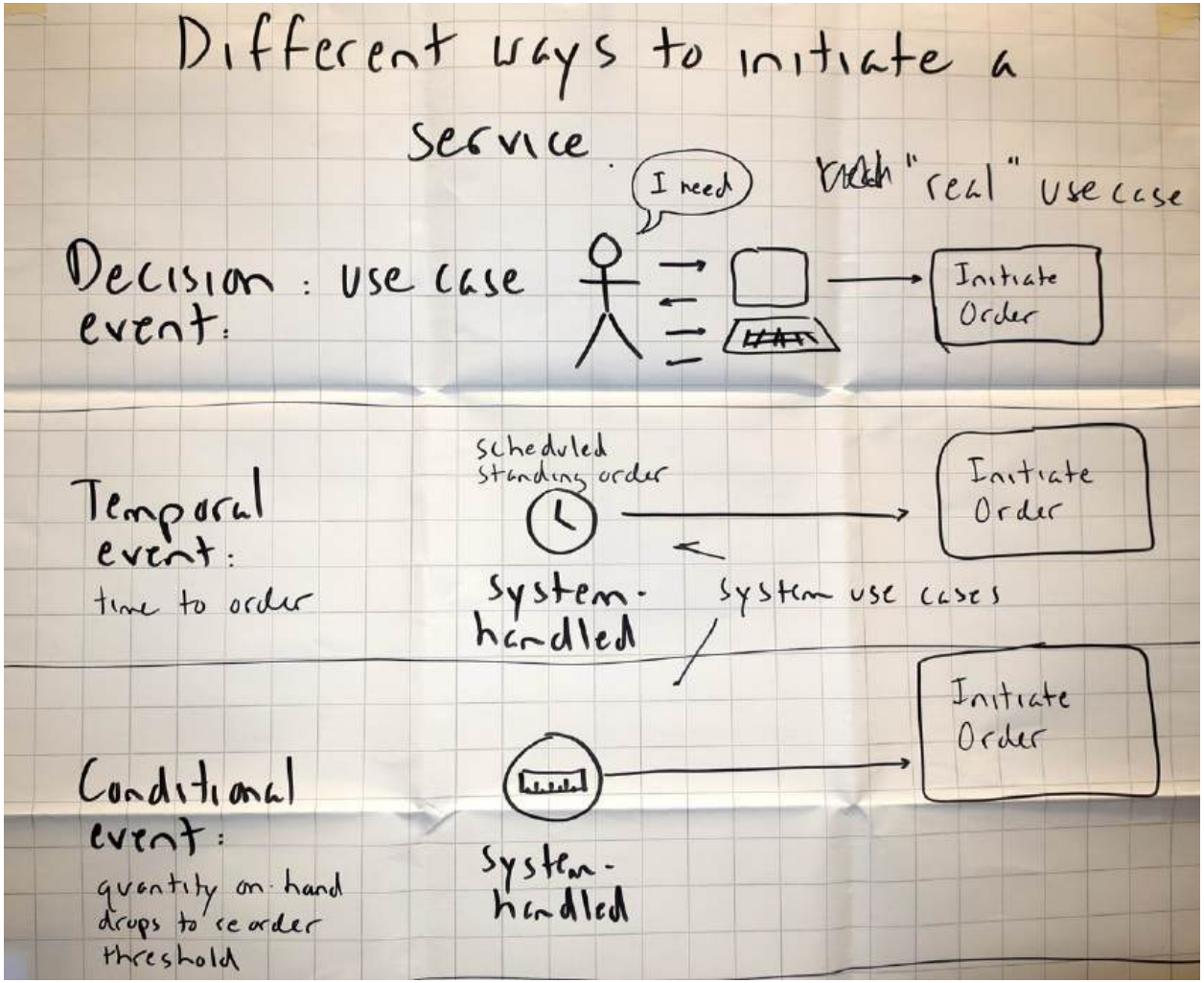
- ✓ Who (the actor), does what (the steps), why (the goal) –
  the other components can help, but these are the essence
- ✓ A "story" – an easy-to-follow narrative
- ✓ A common definition – "A goal-oriented sequence of interactions between external actor(s) and the system necessary to deliver the service to satisfy the actor's goal."

Clarifying "Use Case"

- The term “Use Case” has been overused and misused since Ivar Jacobsen first created it,
  E.g., Gartner Group –
  "Top Five Use Cases for Robotic Process Automation"
- What do *we* mean by a Use Case?
  In general, a Use Case is a description of how an Actor would like to interact with a System to complete a Task:
  - obtain a service
  - receive information
- Typically…
  - a person interacting with a system – a "Real Use Case"
  - interaction within/between systems – a "System Use Case"



Recap – Real Use Cases and System Use Cases



Actors, roles, stakeholders

Actors and Roles

- ✓ Actor – a job title, role, organisation, or system driving the use case
- ✓ A use case generally has one primary actor, but could have other actors – e.g., Sales Rep is primary in the UC "Sales Rep Enters Order via _____," but Sales Assistant and Sales Manager can also be actors
- ✓ You could handle this by creating the role of "Orderer" or…
- ✓ …simply identify the primary actor and list additional actors for the use case

Stakeholders

- ✓ Other interested parties, such as business owners or regulators, that are impacted by use cases but not involved.
- ✓ They have "interests" that must be protected by system
- ✓ The institution itself is best treated as a Stakeholder, not an Actor

Actor Criteria

- ✓ A Use Case must deliver a "service of value" so an actor can "walk away happy"
  - complete a Business Service that changes the state of the world (updates data)
  - view useful information
  - not "Log In" – that's a "sub-Use Case"
- ✓ An Actor is *anyone* or *anything* that interacts with the system to obtain a Service
  - typically a person
  - could be the system itself, or another system (System Use Case)
- ✓ The Actor (e.g., a Customer Service Representative) may be acting on behalf of the *ultimate primary actor,* e.g., the Customer on the phone
- ✓ A Use Case might bring in a "Supporting Actor," e.g., "Click here for chat."

An approximate sequence for Services and Use Cases

We did this on day 1 using the "doughnut model"

Entity-based
- *Identify Services* (Events) for each Entity by adding a Verb to the Noun

Choose one as your primary approach – our ATM example is Actor-based

Develop initial Concept Model

Actor-based
- Identify Actors
- Identify needed Services (Use Cases)
- *Isolate Services*

Write Initial Service Specification

Eliminates unknowns!

Identify and refine Use Cases

Which Actor needs which Service?

Write initial Use Case Description

Focus on the Abstract and Stakeholder Interests

Write Use Case Dialogue

Main flow then alternate sequences

Optional: Develop Use Case Scenarios

A "worked example" then refine Use Cases

Write final Service Specification

Include state changes and newly-discovered requirements

Workflow-based
- Build end-to-end Workflow Model
- Identify steps requiring a Use Case
- *Isolate Services*

A blank slide to maintain balance in the universe

Discussion – ID additional Use Cases

- List known Actors and Platforms
- Create Use Cases by assigning Services to Actors and Platforms
- Identify additional Use Cases using the Actor-Based and Workflow-Based approaches
- Identify Administrative Use Cases (profiles, code sets, etc.)

Tips:

- Services triggered by temporal or conditional events are "System Use Cases"
- If it's not „one Actor at a single place and time" you might be at too high a level – break it down further
- If it's not something of value ("Actor can walk away happy") then it's too low level – bring it up higher

Who	What	How
Customer	Transfer Funds	via ATM
	Transfer Funds	
	Transfer Funds	
	Transfer Funds	
	Transfer Funds	
	Transfer Funds	
	Transfer Funds	

Note – the core banking system is "RBS" – Retail Banking System

Discussion – ID additional Use Cases

Who	What	How
Customer	Transfer Funds	via ATM
Customer	Transfer Funds	Mobile App
Customer	Transfer Funds	web site
Teller	Transfer Funds	RBS
Auth. 3rd Party	Transfer Funds	RBS
Temporal Event (scheduled transfer)	Transfer Funds	RBS
Conditional Event (overdraft protection)	Transfer Funds	RBS

Note – the core banking system is "RBS" – Retail Banking System

Discussion – good Use Case or not?

Which of the following might ***not*** be good Use Cases?

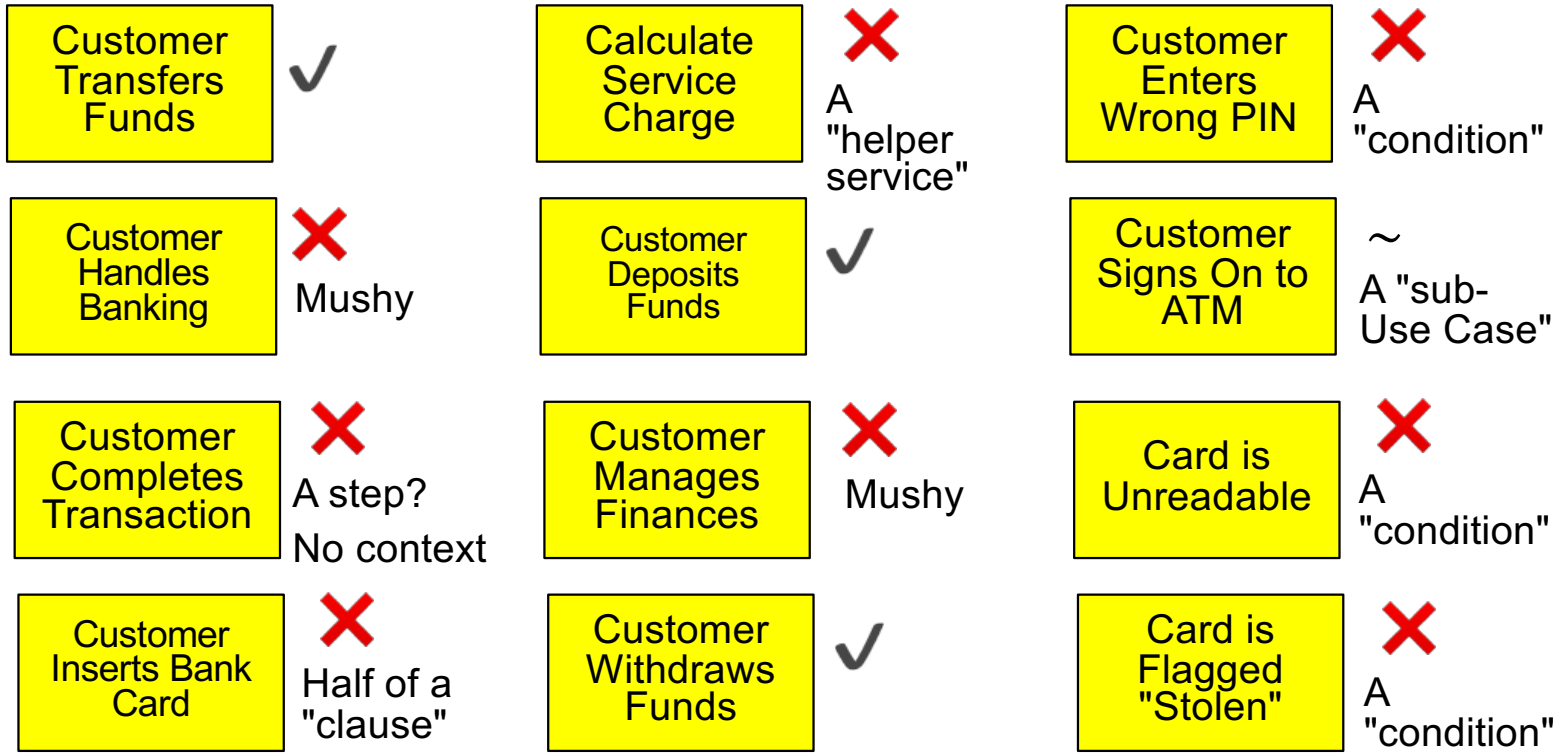
And if not, *why* not?

Customer Transfers Funds	Calculate Service Charge	Customer Enters Wrong PIN
Customer Handles Banking	Customer Deposits Funds	Customer Signs On to ATM
Customer Completes Transaction	Customer Manages Finances	Card is Unreadable
Customer Inserts Bank Card	Customer Withdraws Funds	Card is Flagged "Stolen"

Discussion – good Use Case or not?

Which of the following might ***not*** be good Use Cases?

And if not, *why* not?



Initial (concept level) use case description – 1

	Component	Description	Notes
1	**Actor(s)**	Name of primary and other actors	• Could include "supporting actors" or actors working on behalf of "ultimate primary actor" • Could be primary plus others, or "role" name
2	**Service + Platform (opt.)**	Completes the use case name	• Often specifies implementation technology or platform, but not always. • E.g. Student Complete Registration via Web
3	Actor goal (optional)	What the actor actually wants to achieve	• Usually is self-evident from the use case name, but may need further elaboration • Optional step
4	**Stakeholder interests**	Conditions to be met in the interest of other stakeholders not directly involved in the use case	• Ask "What would make the stakeholder unhappy?", and reverse it • Consider the customer (actor,) *other* customers, owners and managers, regulators, suppliers, … • Uncovers new rules (requirements)
5	**Abstract**	A narrative describing what happens, focusing on the main success case, in 5 +/- 2 sentences	• Provides a "mental image" or "word picture" for the entire user experience • In some cases, this is all the business analyst produces – may be done first • Also useful as a standalone product to convey the essence of the use case (e.g., in a catalog) independently of detailed documentation
6	**Notes**	Additional points or questions	• Includes questions, specific functional requirements, non-functional requirements (e.g. "dirty noisy environment") or anything noteworthy

Initial (concept level) use case description – 2

	Component	Description	Notes
7	Trigger (optional)	• The action that starts the use case	• E.g., "Customer phones Sales Rep with complaint • May be the first step in the use case • Not "essential" – includes "who and how" • Optional step
8	Preconditions (optional)	• What the system can ensure is true at beginning	• E.g., "Service Rep has transaction authority" • Checked once, at the start • Not the same as all the entity state, data value, etc. checking that occurs later • Optional step – really only useful for out-of-context use cases
9	Milestones or steps (optional)	• Main phases or steps in ***main success case***	• Note if sequence is mandatory or optional • Alternate and failure conditions and handling will be documented later • Optional step – service specification covers it
10	Postconditions (optional)	• Important states or data values on completion	• E.g., "Complaint is recorded, and escalated if necessary" • Optional step – service specification covers it

Use Case Abstract sample – Customer Transfer Funds via ATM

May, a busy young mother, needs to quickly transfer SGD 200 from her savings account to her chequing account in order to cover a childcare payment.

She is dismayed there are several people in front of her, but the ATM deals with them efficiently and the line soon clears. Reaching the front of the line (and anxious because there is now a line behind her) she inserts her bank card to begin authorisation, then quickly specifies the "from" and "to" accounts, the amount, and confirms it is an "immediate" transfer.

Declining a receipt, she is done within 30 seconds, and carries on with her busy day, undertaking construction projects at home.

Stakeholder interests

Stakeholder	*Would be unhappy if…*	*So we will…*
Regulator	Money laundering	Establish limits
Joint Account Holder	Drain the account	Establish limits
Bank	Transfer unavailable funds	Confirm "available" balance

Develop initial use case dialogues

1 Develop or propose dialog, always in "when – then" format

Recap the Use Case Abstract for participants, then create dialog:

Trigger – Customer decides to contact Service Centre:

1. When Customer places service call
   Then Service Rep accepts call, greets Customer, and requests Customer Name or ID
2. When Customer provides Name
   Then Service Rep uses Customer Search, System displays search results list, and Service Rep requests confirmation
3. When Customer provides confirmation
   Then Service Rep requests details of the issue or problem…

- ✓ Each clause (step) goes from the "driving" actor having control to that actor being given control again
- ✓ Ultimately, the dialogue must:
  - be satisfying for the actor
  - protect stakeholder interests
  - be valid against concept model dependency and rules
  - enforce service rules and provide the data it needs
- ✓ KEY POINT – it's just fine to give a brief indication of "behind the scenes" system activity – it provides context!

Initial (concept level) use case dialogues

Dialogue – primary case		
#	**Clause**	**Notes & Questions**
	Each clause (step) goes from the actor having control to that actor being given control again. Ultimately, must be a satisfying, functional dialogue.	• Newly discovered requirements • Non-functional requirements • Questions and assumptions
1.	When… Then…	
2.	When… Then…	
3.	When… Then…	
4.	When… Then…	
…	…	

Discussion: Initial dialogue for Customer Transfers Funds via ATM

Dialogue – primary case		
#	Clause	Notes & Questions
1.	When Customer inserts card Then ATM validates card and prompts for PIN	Might have to consider biometric identification, e.g. retina scan
2.	When Customer enters PIN Then ATM validates and displays options	What other validation would be performed at this point? Customer Status? Individual Accounts?
3.	When Customer selects "Transfer" Then ATM prompts for From Account	Can we display available Accounts and available balances? Privacy concerns? From Account = Source Account
4.	When Customer selects From Account Then ATM prompts for To Account	To Account = Destination Account
etc.	etc.	
10.	When Customer selects "Confirm" Then ATM "Transfer Funds" and displays confirmation and prompts "Receipt?"	Should we also display new balances?

For reference: common questions about Use Cases

Remember – the intent is a functional, satisfying dialogue that generates the necessary information for the Service

Slides 85-88 are for reference – skip ahead to 89.

- Should I include all errors, alternatives, and exceptions?

  Not in the initial "main success flow" – add errors etc. later in "alternate sequences"

- Should I show "behind the scenes" activity, e.g. validation

  Yes, but only enough to provide context – not all the step-by-step details

- Can a clause have an "and" in it? Can a clause get "long?"

  Yes, absolutely. A clause goes from, for instance, the system waiting for a response from an actor, through the actor's response, to the next point the system is waiting for response.

For reference: common questions about Use Cases

- Can perspective change?
  E.g., from "When Customer…" to "When ATM…?"

  Usually not – it remains consistent throughout.
  (Note the answer later to the question about which Actor starts the Use Case.)

- Should I show validation with each clause or "save it up?"

  Show validation with each clause. "Saving it up" loses context.

- Should I show specific UI mechanisms?

  No. That should be left to the UI Designer, unless there isn't any choice, e.g. a Bank ATM Card

For reference: common questions about Use Cases

- Should I start with the System or the Actor using the system?

  Generally with the Actor. We commonly assume the system is in an idle state, waiting for the Actor to initiate the Use Case. In another case, Automated Agent calling the Customer, I would start with
  When Customer answers/responds
  Then Agent plays greeting and…

- Should the Use Case include validation performed in the Service?

  Yes, which surprises many people. We need the validation in the Use Case so we can provide appropriate "help and hand-holding."

For reference: common questions about Use Cases

- Should the Service include validation performed in the Use Case?

  Yes. The Service doesn't "trust" the Use Case, especially since the UI for the Use Case might be embedded in an other organisation's application.

- Can I use Use Cases for anything else?
  Yes!
  - *As the basis for creating test cases / scenarios*
  - *Training and implementation*
  - *Embedded Help*
  - *Assessment of purchased or legacy applications*

Refine use cases (detail level)

Now that you've documented the "primary course" (or "main success case" or "happy path"…) it's time to consider what else can happen…

Primary Course:

1. When Service Rep accepts call and greets and asks for Customer ID
   Then Customer provides Customer ID
2. When Service Rep enters Customer ID
   Then System locates Customer and displays Customer profile
3. etc.

Extension Courses:

2.1 Customer ID not on file:

Identify the associated step or steps, and state the condition

The system will present Find Customer, which provides multiple search criteria (name, postal code, etc.) and …

Describe alternative handling as a brief narrative, or in "when – then" format

2.2 Customer's "special handling" flag is set:

Another extension for the same step

When…

Final (detail level) use case extensions

Extension cases		
#	**Condition** State an alternative (error, exception, variation, …) condition that can arise in this clause number. Any number of alternatives could arise for a single clause in the "main success case."	**Handling** • Describe how the condition will be handled as a brief narrative, or in "when – then" format
n.1	Condition:	
n.2	Condition:	
n.3	Condition:	
n+1.	Condition:	
…	…	

Discussion: Alternate sequences

Extension cases		
#	Condition	Handling
1.1	Unreadable card:	Display a message and request the Customer re-try
1.2	Out of network:	Advise Customer an additional fee will apply and ask for confirmation to proceed
1.3	Card is flagged stolen	Put ATM in "honeypot" mode. Notify Security / Police Put Video Camera in high-res mode Emit a LOUD sound etc.
2.1	Time out waiting for PIN:	Display warning that PIN must be provided with X seconds or card will be taken. If PIN is not provided within the time limit, ingest card.
2.2	Condition:	Etc.
…	…	

A use case description

Use Case Name
Departmental Advisor completes enrollment (of Student in Class(es))
Description
When the student has submitted their enrollment form (by mail, fax, e-mail, etc.) their advisor will complete the enrollment process for that student, including enrolling them in alternative classes or even other courses as necessary. (Also see postconditions.)
Actor(s)
Departmental Advisor or management staff in the Registrar's Department (Students have separate use cases for self-service enrollment)
Distinctions from the essential use case (optional)
Departmental Advisor has capability to override certain rules such as prerequisites.
Preconditions (optional)
• Advisor is logged on, and has a menu of functions available • Assumption: The student has completed the appropriate enrollment form, and the advisor has it.
Primary Course (Normal success steps)
1. When Advisor enters Student Number and selects "Enroll Student" Then system verifies Student and Account status and displays basic student data 2. When Advisor selects semester/session they are enrolling for Then system verifies that semester/session is available 3. When Advisor selects course code (e.g., "Math") for next course to enroll in Then system returns list of all courses for that course code available in selected semester/session that student has prerequisites for, with list of avail. classes 4. When Advisor selects Class…etc.
Postconditions (optional)
• The student will have a verified or waitlisted enrollment in each selected class. • Available space in classes will be updated. • The student's account will be updated either by withdrawing funds electronically, or indicating an account due for handling by the A/R system. • A confirmation report will be produced for the student.
Extension Courses (Alternate and failure steps)
1. Student number not provided: • The advisor can enter any number of characters of the last name. The system will produce a list of matching students to select from. 4. Desired Class full: • If the desired class is full, "waitlist" can be selected for any number of classes (of the same course) in addition to enrolling in one.
Comments, issues, and design notes
• For student search, may need a mechanism for restricting the number of entries in the list. • For student search, need a way to indicate whether the whole name or a subset is being entered.

For reference – develop use case scenarios

Use Case

- ✓ A generalised description of the main steps
- ✓ A single logically complete unit of functionality
- ✓ Defined actor by actor, possibly also by technology
- ✓ Finite number

Key Point

- Uses cases and use case scenarios are frequently confused

Material on scenarios included for reference. Skip to the end – slide 97.

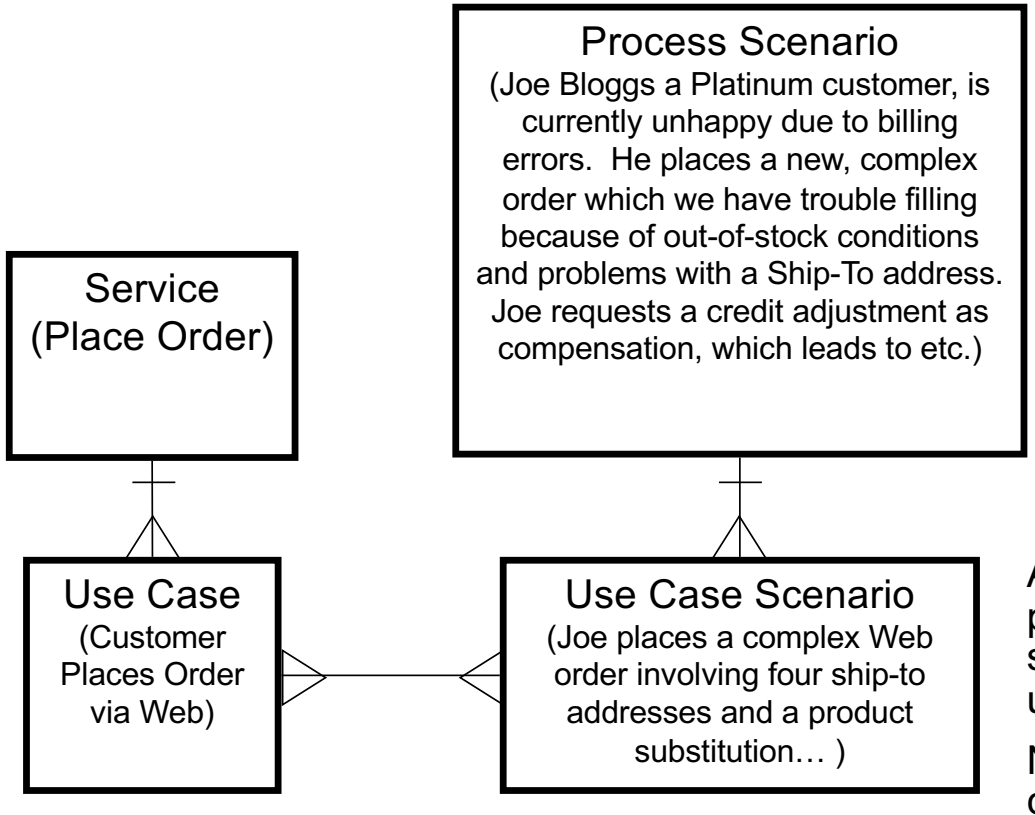
Use Case Scenario

- ✓ A dialogue tracing a single path through a "real world" scenario for one or more use cases with *named actors*, *specific conditions*, pre-determined decisions, and *actual data values*
- ✓ May be part of a process scenario
- ✓ One use case scenario may include many uses cases
- ✓ May take multiple scenarios to fully explore one use case
- ✓ *Caution - potentially infinite*

Key Point

- Covers a single case with *no branching*; decisions and outcomes are already decided

Process and use case scenarios



A single path through an entire end-to-end business process.

Actors, preconditions, decisions decided in advance

A dialogue tracing a single path through a "real world" scenario for one or more use cases.

Named actors, specific conditions, and actual data values

Corresponds to a use case "slice"

A use case scenario description

Use Case(s)	Scenario description/purpose
Dept. Advisor enrolls student in classes Dept. Advisor searches for student by name	Departmental Adviser "Ed Visor" enrolls Student "Stu Dent" in his selected classes. Normal flow with a few minor problems.
Precondition	**Impact**
1. Stu Dent is a second year student 2. Stu is in good standing, with no outstanding debts to the university 3. Ed is already signed on to RegMan app.	1. Already registered at the university with all student data in place 2. None 3. No sign-on for this scenario
Condition	**Expected Handling**
1. Stu failed to provide Student Number 2. Prerequisite to Math 210 is missing 3. Section 3 of Chem 205 is full 4. Successful enrollment in Phys 220, Span 200, and Math 221	1. Use student name search 2. Prompt for waiver which Ed grants 3. Suggest alternate classes; one will be selected, but waitlist Class 3 4. Successful enrollment in Phys 220, Span 200, and Math 221

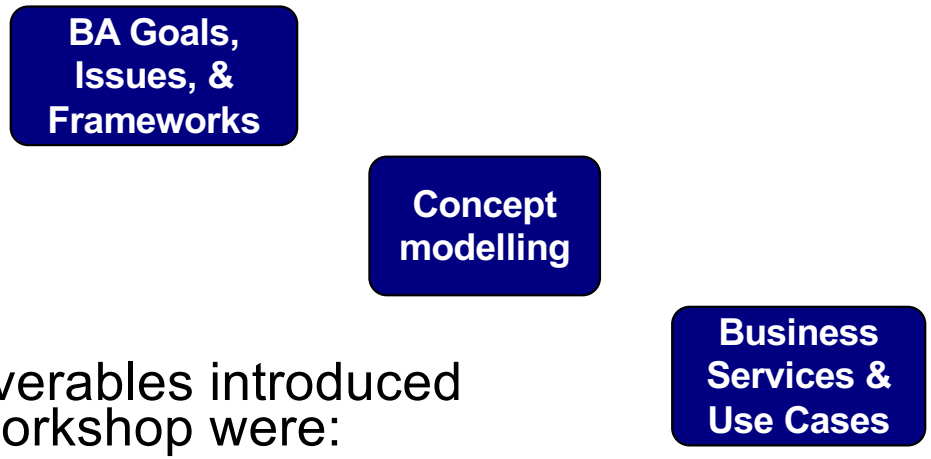
Dialogue	Notes
When Ed selects "Enroll" from "menu" Then system presents a dialog box to select the student (either by number or by name)	
When Ed enters first five characters of last name Then system lists several matching students, showing student's full name, birth date, permanent residence city, and student number	Should have option to display full info about any student in the list. Should it take two steps to get to this point?
When Ed selects Stu from list and selects "enroll" Then system verifies Stu's academic and account status, displays Stu's basic personal/student data (to be determined,) declared major, course history list, and department selection list	
When Ed selects "Phys" from Dept. list and enters "220" Then system checks that Smith has the necessary prerequisites, and lists all classes plus their current status (open, full, pending, …)	Should allow selection of department name either from pulldown list or direct entry.
When advisor selects "Phys 220, Class 3" Then system initiates transaction to enroll Smith in that class and displays confirmation	
When Ed selects "Math" from Dept. list and enters "210" Then system checks that Smith has the necessary prerequisites – he doesn't – so system displays waiver dialog.	If Ed had just entered "Math", the system would list all of the courses that Stu had the prerequisites for—Math 210 would not be on the list.
etc.	

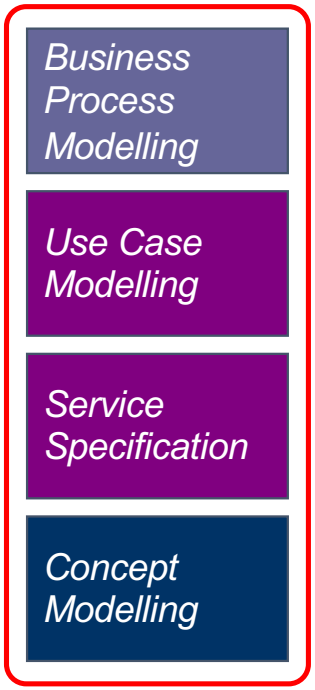
Refine use cases

Possibilities include

- Add newly-discovered use cases
- Update use cases with newly-discovered steps and dialogues
- Update Service Specifications to reflect new discoveries (and the Concept Model and Process Model too!)

Requirements Modelling Wrap-up





The key deliverables introduced during this workshop were:

- Business Process (quick examples only)
- *external* Use Case Model
- *internal* Service Specification
- *fundamental* Concept Model (quick review only)

"Inside-out"

Use Cases & Services – key points to take away

- We didn't cover it, but *Business Process* is a great framework for:
  - Getting business attention
  - Organising other analysis
- Use the *Concept Model – just don't call it that too soon!*
- Separate *Use Cases* and *Business Services* (external vs. internal)
- Remember to separate *what* from *who and how*
- Scope – Concept – Detail
- Simplicity – it's a business, not a system or DB

Use Cases & Services – wrap-up discussion

Looking back over the class, was there a key point (or points) that mattered most to you?
Please be prepared to share this with the class.
Thanks!

Other courses for analysts by Alec Sharp

Working With Business Processes – *Process Change in Agile Timeframes* 2 days

Business processes matter, because business processes are how value is delivered. Understanding how to work with business processes is now a core skill for business analysts, process and application architects, functional area managers, and even corporate executives. But too often, material on the topic either floats around in generalities and familiar case studies, or descends rapidly into technical details and incomprehensible models. This workshop is different – in a practical way, it shows how to discover and scope a business process, clarify its context, model its workflow with progressive detail, assess it, and and transition to the design of a new process by determining, verifying, and documenting its essential characteristics. Everything is backed up with real-world examples, and clear, repeatable guidelines.

Business-Oriented Data Modelling – *Useful Models in Agile Timeframes* 2 days

Data modelling was often seen as a technical exercise, but is now known to be essential to other initiatives such as business process change, requirements specification, Agile development, and even big data, analytics, and data lake implementation. Why? – because it ensures a common understanding of the things – the entities or business objects – that processes, applications, and analytics deal with. This workshop introduces concept modelling from a non-technical perspective, provides tips and guidelines for the analyst, and explores entity-relationship modelling at contextual, conceptual, and logical levels using techniques that maximise client involvement.

Working With Business Processes Masterclass – *Aligning Process Work with Strategic, Organisational, and Cultural Factors* 3 days

This 3-day interactive workshop combines the core content from two highly-rated classes by Alec Sharp – "Working With Business Processes" and "Advanced Business Process Techniques." This structure is popular because it gets both new and experienced practitioners to the same baseline on Claritiq's unique, agile, and ultra-practical approach to Business Process Change. First, it shows how to effectively communicate Business Process concepts, discover and scope a business process, assess it and establish goals, and model it with progressive detail. Then, it shifts to advanced topics – specific, repeatable techniques for developing a process architecture, encouraging support for change, and completing a feature-based process design. The emphasis is always on ensuring business process initiatives are aligned with human, social, cultural, and political factors, and enterprise mission, strategy, goals, and objectives.

Business-Oriented Data Modelling Masterclass – *Balancing Engagement, Agility, and Complexity* 3 days

Our most popular workshop! This intensive 3-day workshop combines the core content from two popular offerings by Alec Sharp – "Business Oriented Data Modelling" and "Advanced Data Modelling." First, the workshop gets both new and experienced modellers to the same baseline on terminology, conventions, and Clariteq's unique, business-engaging approach. We ensure a common understanding of what a data model *really* is, and maximising its relevance. Then, we provide intense, hands-on practice with more advanced situations, such as the enforcement of complex business rules, handling recurring patterns, satisfying regulatory requirements to model time and history, capturing complex changes and corrections, and integrating with dimensional modelling. Always, the philosophy is that a data model is a description of a business, not of a database, and the emphasis is on engaging the business and improving communication.

Model-Driven Business Analysis Techniques – *Proven Techniques for Processes, Applications, and Data* 3 days

Simple, list-based techniques are fine as a starting point, but only with more rigorous techniques will a complete set of requirements emerge, and those requirements must then be synthesised into a cohesive view of the desired to-be state. This three-day workshop shows how to accomplish that with an integrated, model-driven framework comprising process workflow models, a unique form of use cases, service specifications, and business-friendly data models. This distinctive approach has succeeded on projects of all types because it is "do-able" by analysts, relevant to business subject matter experts, and useful to developers. It distills the material from Clariteq's three, two-day workshops on process, data, and use cases & services.

**** Note: two-day in-person workshops are delivered virtually as three half-day sessions via Zoom.*
Three-day in-person workshops are delivered virtually as five half-day sessions via Zoom.

Thank you!

Alec Sharp, West Vancouver, BC, Canada

If you have questions or comments…
don't be shy, get in touch!

- e: asharp@clariteq.com
- ig: @alecsharp01
- m: +1 604 418-3352