BTC-210

AGILE USE CASE WORKSHOP FOR BUSINESS ANALYSTS

Course outline

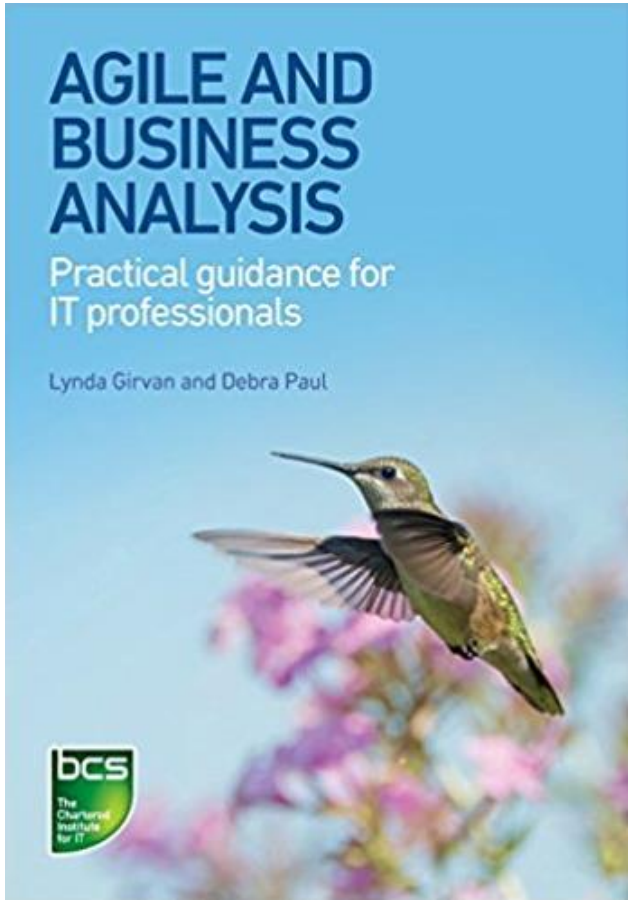
- Two days

Related courses

- Business Analysis
- Software Testing
- Scrum certification
- Project management certification (PMI)

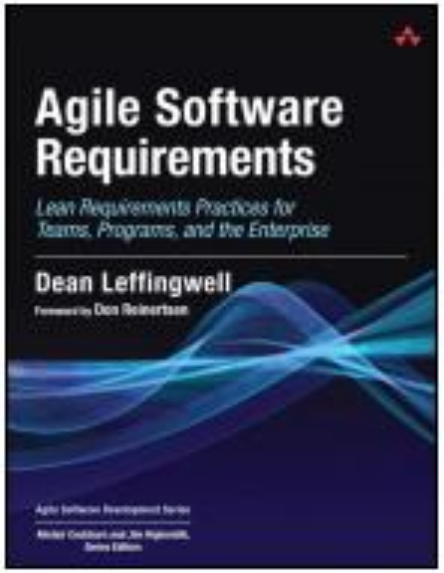
Book

- **Agile and Business Analysis: Practical guidance for IT professionals**synda Girvan and Debra Paul, BCS Learning and Development Ltd., 2017



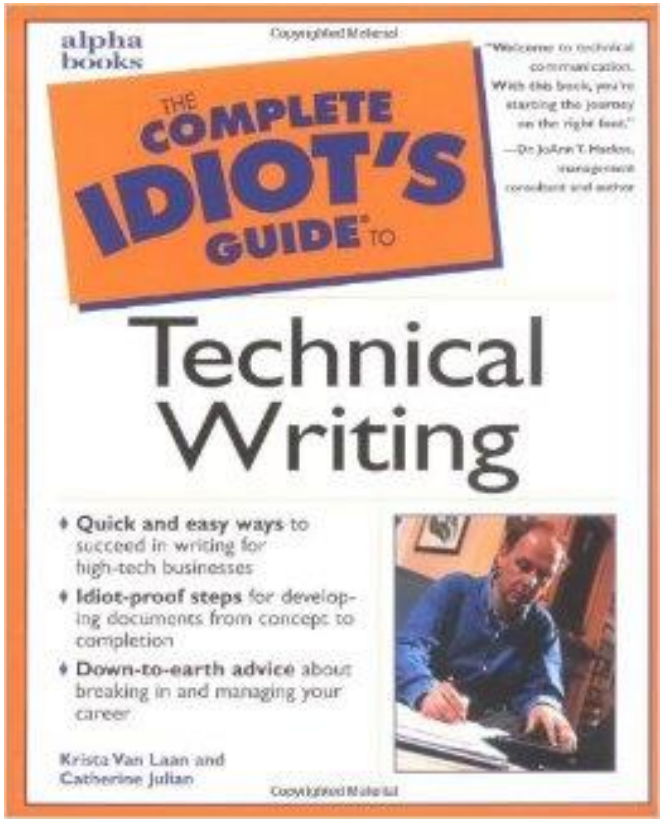
Books, recommended

- **Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise,** Dean Leffingwell, 2010



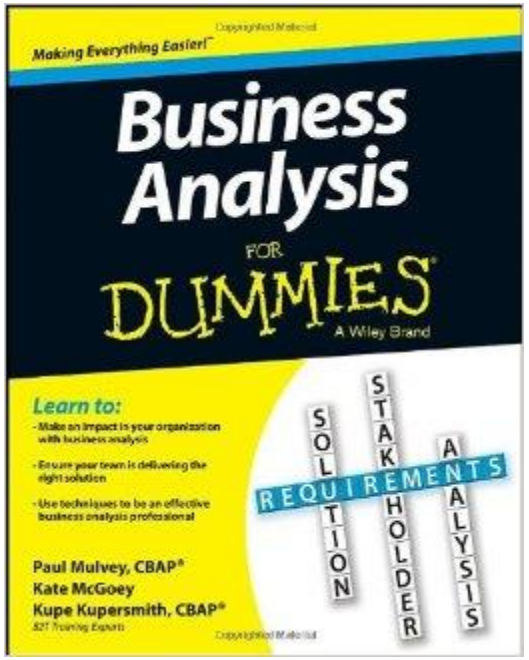
Books, recommended

- **Complete Idiot's Guide to Technical Writing**, Krista Van Laan and Catherine Julian, Alpha Books, 2001



Books, recommended

- **Business Analysis For Dummies** by Kupe Kupersmith, Paul Mulvey, Kate McGoey. July 2013 – recommended
  - $13 used Amazon



Resources

- https://github.com/doughoff/BTC-210
- **Modern Analyst**
  - http://modernanalyst.com/

Business analysis overview

Business analysis values

- Efficiency
  - reuse of documentation
  - reuse of business processes and metrics
  - elimination of rework
  - prevention of bugs
- Effectiveness
  - making the customer/stakeholder satisfied with the result
- Productivity
  - maximizing the velocity of the project team

Business analysis vision

- Decrease the risk of losing value by
  - Understanding the domain of the business problem
  - Communicating domain to team members and stakeholders
- NOT
  - the design vision
    - create a solution plan to the business problem that best addresses the domain and the business constraints

Business analysis strategy goals

- Increase the level of domain understanding
- Produce adequate documentation
  - to manage project
  - to support the next stages of the project
- Assure quality of project

Business analysis tactical goals

- Strategy – enterprise analysis
- Analysis
  - Identify and define – apply business values
  - Gather – understand the domain
  - Structuring – organize the data
  - Verification – get feedback
  - Document and present – make the data useful
- Design – merge data with constraints
- Development – technical analysis, BDD, TDD
- Testing – scenario and system test design

Business analysis operational goals

- Strategy – business plan. CSFs
- Analysis
  - Identify and define – glossary, stakeholder info
  - Gather – interviews, surveys, workshops, observation, user stories
  - **Structure** – KPIs, sequencing, data dictionary, use cases, priorities
  - **Verify** – trends, artefact verification
  - Document and present – requirements docs
- Design – prototypes, interfaces, screen shot scenarios
- Development – state, activity/flowchart models
- Testing – data dictionary validations

Agile overview

Agile overview

- **Principles** not a process
  - 12 principles based on 4 values
- **Quality,** simplicity (not just simple)
- Originally about software development
  - now mostly driven by project management

Agile manifesto and values

- The Manifesto for Agile Software Development
  - http://www.agilemanifesto.org/
- We value
  - **individuals and interactions** over processes and tools
  - **customer collaboration** over contract negotiation
  - **working software** over comprehensive documentation
  - **responding to change** over following a plan

Agile principles

- Process
  - **Work on artefacts early and continuously**
  - Measure progress with reviewed and approved models
  - Manage changing requirements in models and documents
- Quality
  - **Satisfy the customer**
  - Create high quality artefacts
  - Keep artefacts resilient to change but stay simple
  - Have teams do self reviews and change by results
- Teamwork
  - **Create and review artefacts with customers**
  - Work with customers daily
  - Use self-organizing teams
  - Meet with other analysts face-to-face
  - Let the engaged analysts run with the project

Agile terms

- **Backlog:** a prioritized list of requirements or work items that is frequently updated
- **Definition of done/definition of ready:** setting acceptance criteria for a requirement
- **Personas:** a way of identifying and describing users of the system
- **User stories:** a way of capturing requirements
- **Story mapping**
- **Story splitting:** breaking down stories that are too big

Scrum


WE'RE MOVING TO AN AGILE METHODOLOGY FOR SOFTWARE DEVELOPMENT.
I DON'T KNOW ALL OF THE DETAILS, BUT I THINK ONE OF YOU HAS TO BE DESIGNATED THE SCRUMBAG.
DOES THAT SOUND RIGHT?
IT'S BETTER THAN I EXPECTED.
Dilbert.com @ScottAdamsSays
© 2017 Scott Adams, Inc./Dist. by Andrews McMeel
2-6-17

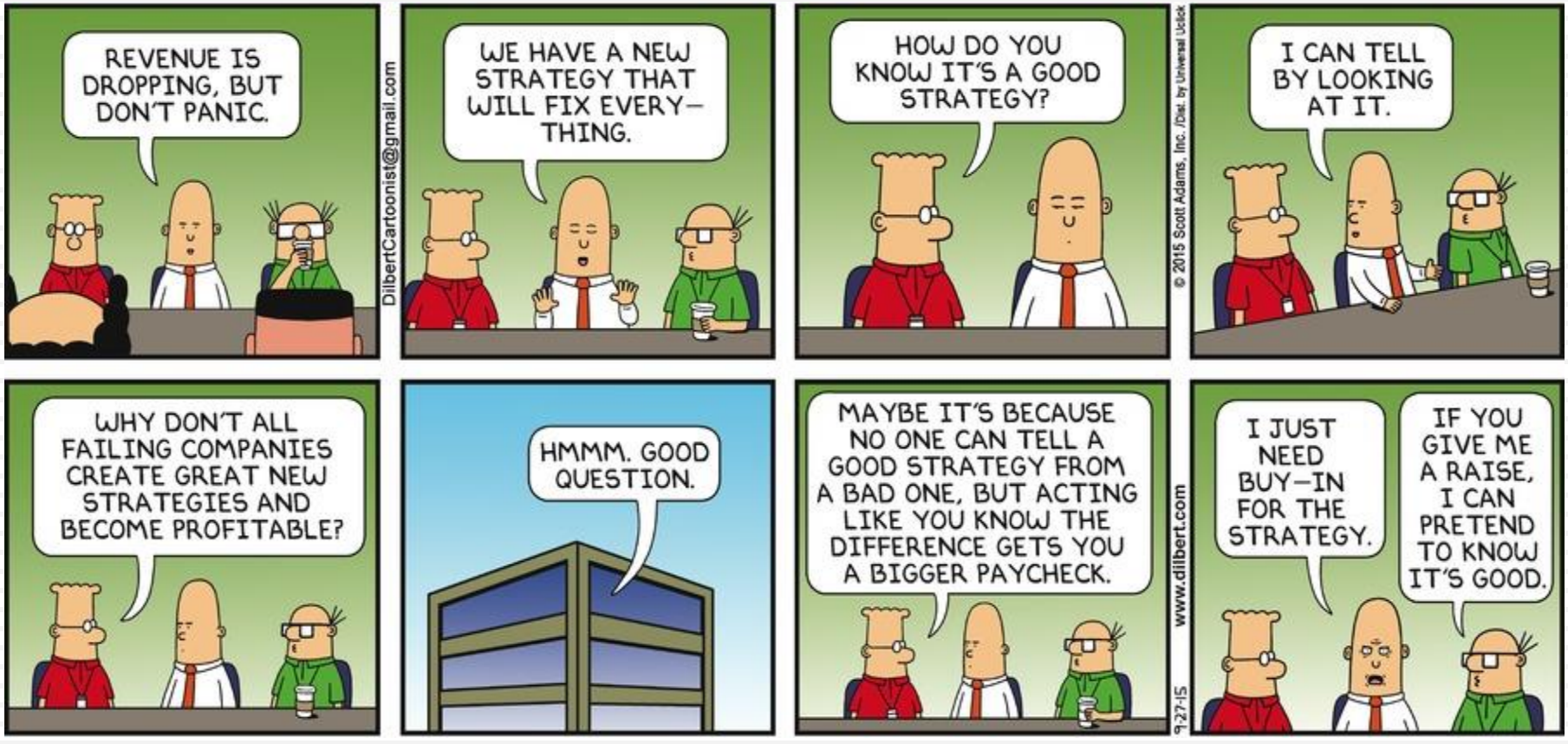
Scrum analysis tasks

- User stories
  - Create and put into product backlog
  - Prioritize on a wall
  - Roll up stories into features into themes
  - Work out details continuously with stakeholders
  - Re-prioritize as necessary
  - Complete requirement just before development
- Use change control
- Confirm software usability, relevance and business value throughout entire process with business users.

Scrum roles

- Product Owner (BA tasks)
  - represents the needs of the business, documents and prioritizes system requirements for backlogs
- Scrum Team
  - a cross-disciplinary team charged with undertaking the agreed work in each sprint
- Scrum Master
  - facilitates the team's work, removing project impediments and ensuring that appropriate Scrum practices are being followed by the team.

Strategy phase



Enterprise planning

- Strategic goals and long-term architecture
- Executive level planning
  - supported by SMEs acting as BAs
- Important to document
  - business goals tied to mission
  - business plan for project tied to business goals

Artefact: The business plan

- Creates the domain where value is found
  - project ties to business goals
  - use cases tie to project
- Important to document:
  - clear problem statement
  - assets needed
  - stakeholders
    - systems
    - internal
    - external
  - value estimate
  - constraints

Artefact - CSFs

- Rolling up selected metrics to help
  - direct activity,
  - correct and teach,
  - validate decisions,
  - justify plans
- two to five per project
- goals/objectives of value achievement
- Often called high-level or business requirements

Optional artefacts

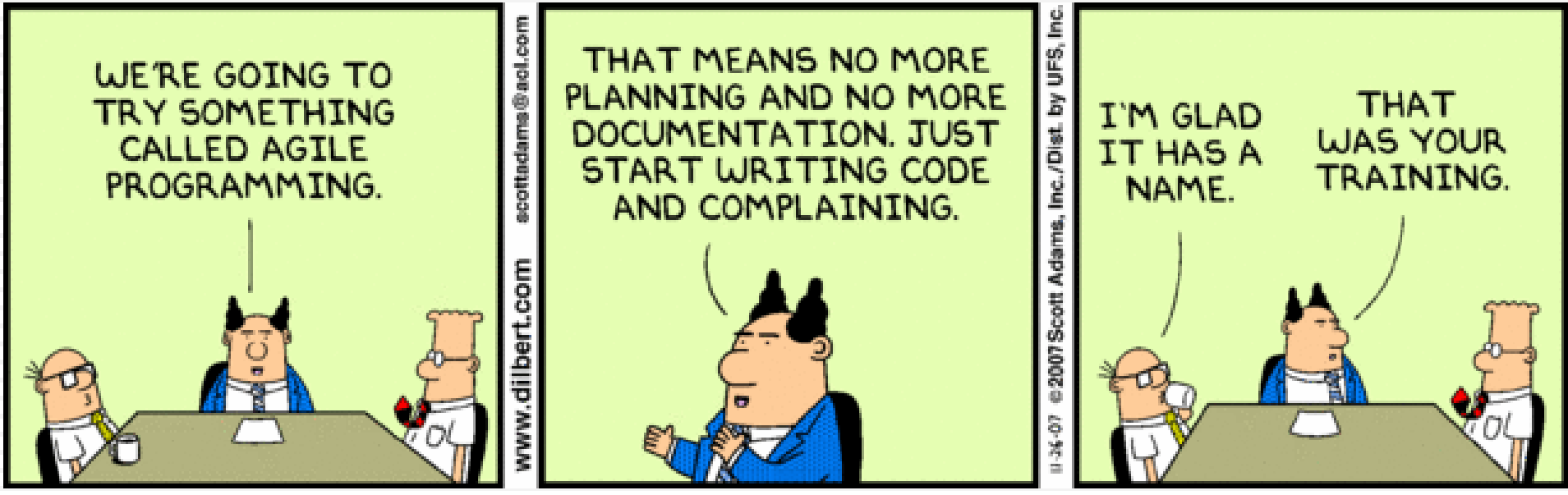
- feasibility study
- top-level architecture
- project strategy plan
- operations concept document

Agile application

- Process
  - **Work on artefacts early and continuously**
  - Measure progress with reviewed and approved models
  - Manage changing requirements in models and documents
- Quality
  - Create high quality artefacts
  - Keep artefacts resilient to change but stay simple
- Teamwork
  - Meet with other analysts face-to-face

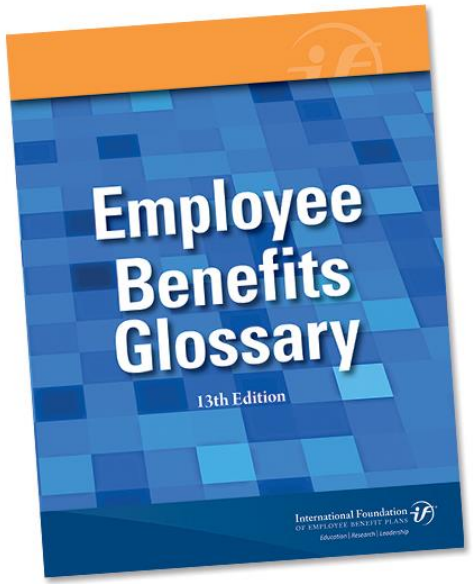
Analysis phase - identify and define

glossary, CSFs



Artefact - Glossary

- Dictionary of common business terms relevant to project
  - use these terms consistently in use cases
  - follow the mapmaker's rule



Artefact: Stakeholder profiles

- Use the list of people from the business case to start. Add others as needed.
- Create table of information with:
  - role, responsibilities,
  - interests,
  - success criteria,
  - concerns,
  - technical proficiency,
  - work environment
- Used for
  - elicitation sessions to create workflow for use cases
  - selecting SMEs for verification of use cases

Agile terms

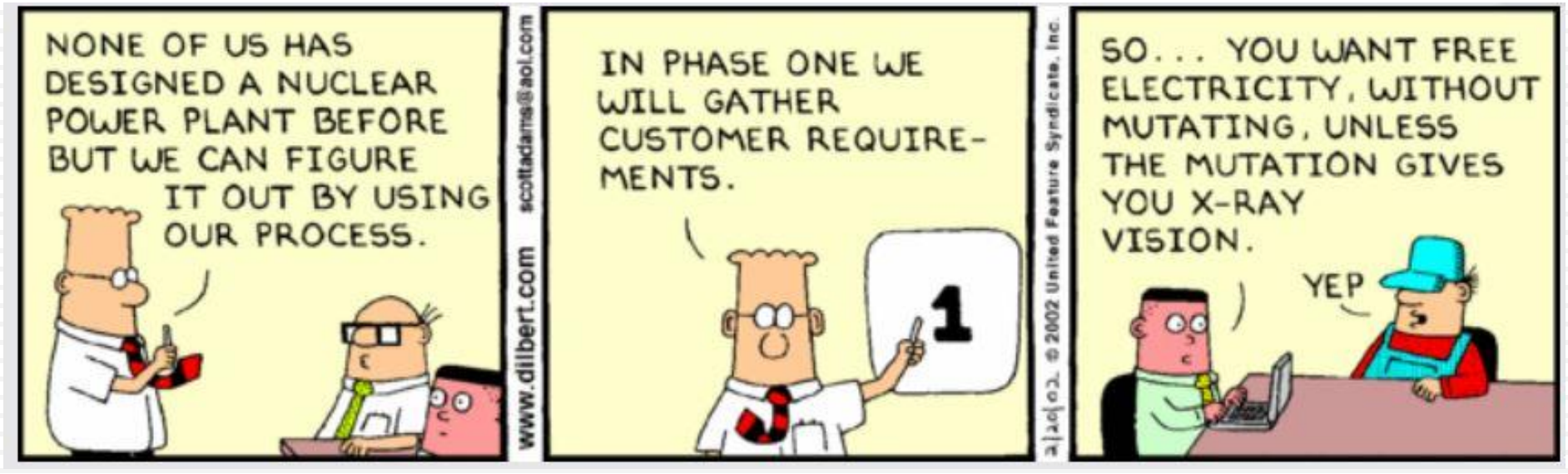
- Definition of Done
  - A Scrum term for CSFs usually applied to user stories
- Minimum Viable Product
  - the goals that provide enough scope for users to see value in what has been decided to be a final product
  - project management decision
  - release management

Agile application

- Process
  - **Work on artefacts early and continuously**
- Quality
  - **Satisfy the customer**
  - Create high quality artefacts
  - Keep artefacts resilient to change but stay simple
- Teamwork
  - **Create and review artefacts with customers**
  - Work with customers daily
  - Meet with other analysts face-to-face

Analysis phase - gathering

interviews, surveys, workshops, observation, user stories, document review



Gathering tools/techniques

- interviews, surveys, workshops, observation, user stories, document review
  - talking with stakeholders
  - output is usually unstructured text
- Design tools/techniques are very useful here also
  - people don't always have words
  - advise stakeholders that this is not UX/GUI
- Creates the raw data for use cases

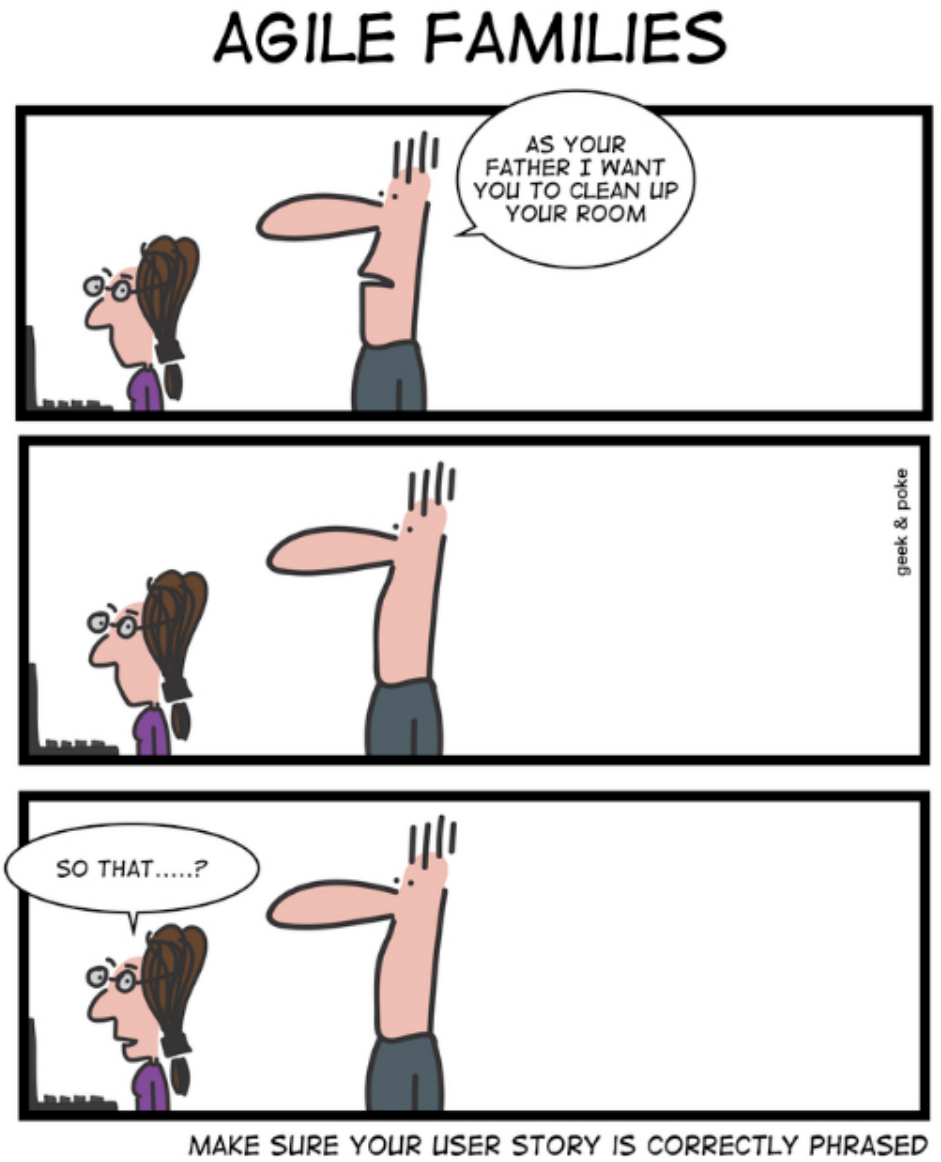
User stories

The user story technique

- As a <some user role>,
  I need the system to <high-level functional requirement> so that I can <get some value>.
- Often mistaken as a requirements analysis tool
- A project management tool
  - to create a tokens for a future conversation – **main intent**
  - to track scope for progress reports
  - to assign deadlines
  - gathered from stakeholders – not rewritten

User story problems

- Poor at defining granularity
  - Use cases provide goals
  - Epics, themes, features?
- Business value is usually not defined
  - Goals = business value
- Not enough detail to implement
  - Use cases give detail
- Informal use cases are better



Agile application

- Process
  - **Work on artefacts early and continuously**
  - Measure progress with reviewed and approved models
  - Manage changing requirements in models and documents
- Quality
  - **Satisfy the customer**
  - Create high quality artefacts
  - Keep artefacts resilient to change but stay simple
  - Have teams do self reviews and change by results
- Teamwork
  - **Create and review artefacts with customers**
  - Work with customers daily
  - Use self-organizing teams
  - Meet with other analysts face-to-face
  - Let the engaged analysts run with the project

Analysis phase - structuring

KPIs, sequence, data dictionary, use cases

Structuring

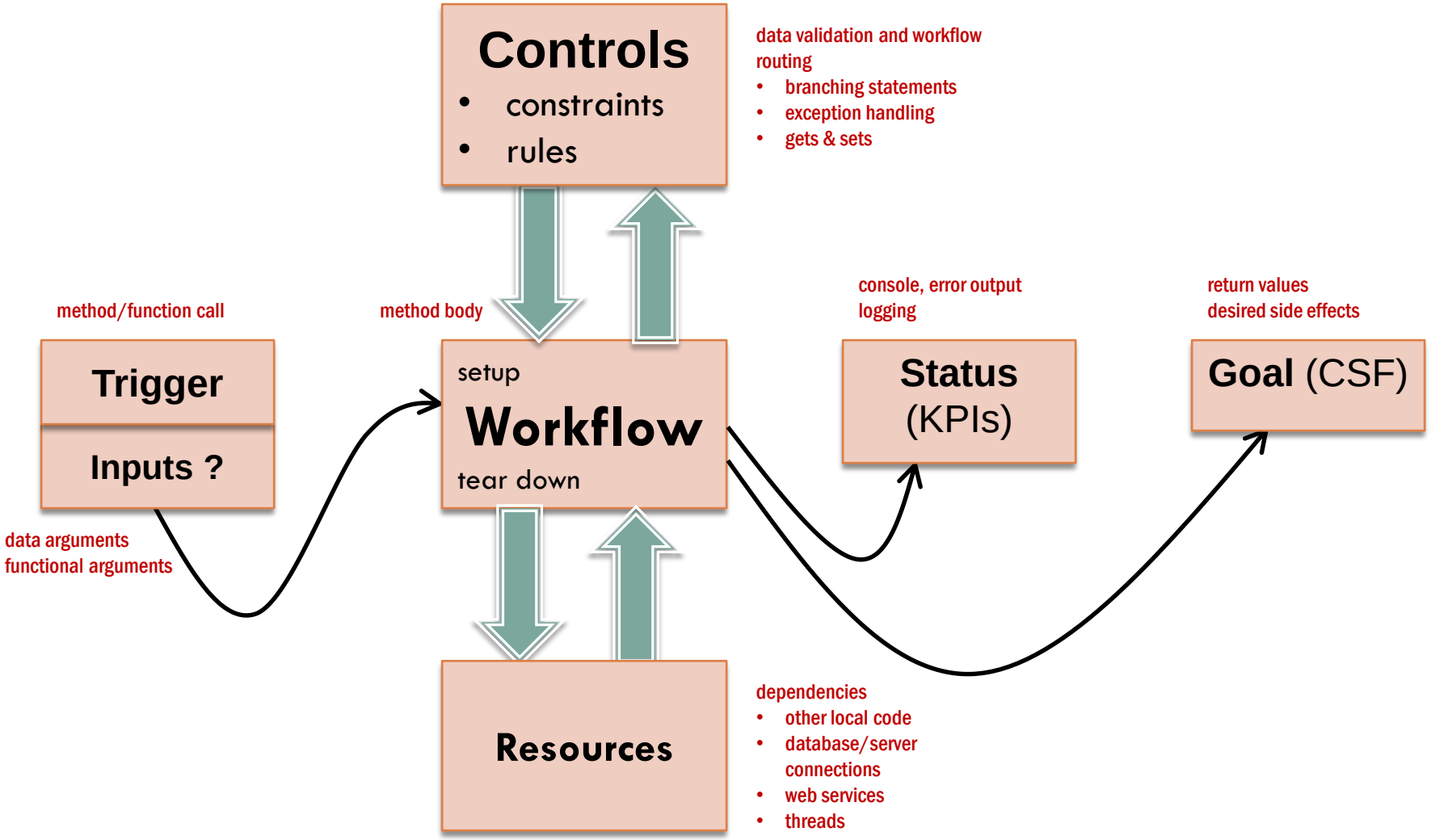
- Converting the data into models for each audience
  - business users
  - programmers and DBAs
  - network administrators
  - management
- Business language for best communication

Structuring

- Two main system categories
  - data
  - processes
- Similar to writing recipes
  - ingredients
  - instructions
  - output: inventory list, recipe book, menu, kitchen designer

A generic process/service model

the process parts in computer language



Artefact: KPIs

- Success criteria for processes
- two to five per CSF
- measured with quantity of agreed upon units

Artefact - data dictionary

- Business language data entities required by the system/role to
  - know about
  - track for changes
  - remember to use later
  - report on
- The second most useful document after the use case
  - Use cases use complex data type names
  - Data rules are captured here for reuse

Artefact - data dictionary

- Formal document recording data entities
  - Name, address, city, state, zip, phone, …
- Basic
  - Name/description
  - Parts (attributes/fields)
    - simple data – text, dates, numbers, flags
    - other entities
  - Relationships
    - cardinality
    - dependencies
  - Capacity, security, integrity

Agile

- Affinity diagram (data dictionary)
  - incomplete view of project
  - adds noise from stakeholders
  - useful as gathering tool

Artefact: use case

- Focuses on functional requirements
- Requirements must be of the same granularity
  - start at the goal level
- Detail matches the need
- Non-functional requirements are captured on a case by case basis
  - when important to that specific functional one.

Use case benefits

- Efficiency
  - Creates enough detail
    - for design to eliminate rework
  - Uses sequenced functional requirements
    - to eliminate scope creep
  - Documents testable tasks and scenarios
    - For reusability in technical analysis and testing
- Effectiveness
  - Defines scope at the goal level
    - for better project management
- Productivity
  - Uses business language
    - to maximize understanding of project

Analysis – use case basics

actor table, use case names, priority

Use case definition

- A unit of scope containing
  - a **repeatable**
  - ordered **sequence** of tasks
  - by an initiating party
  - to support a business **goal** (provides value)
- One happy path
  - creates a backbone for errors and extra paths

Use case project types

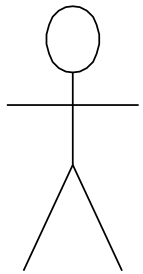
- Types
  - Business - Goals for employee roles
  - System - Goals for a system under development or maintenance
  - Mixed – goes between business and system to get to goal
- Box the use cases by type

Use case process

- **Gather** user expectations
- **Structure** with analysts
  - Find actors
  - Find use case names
  - Write use case details
    - Increase detail when important
    - Invite SMEs when unclear
  - Refactor if complex
- **Verify** with analysts and SMEs
- **Document** use case relationships
- **Present** use cases to users

Actors

- Should have been called roles.
- Actors initiate a use case.
- Actor roles enforce the ability to do processes
  - Actors describe security group names that have **permission** sets.
  - For any two actors, one will have a unique use case that the other doesn't do.



Actor name

Artefact - Actor table

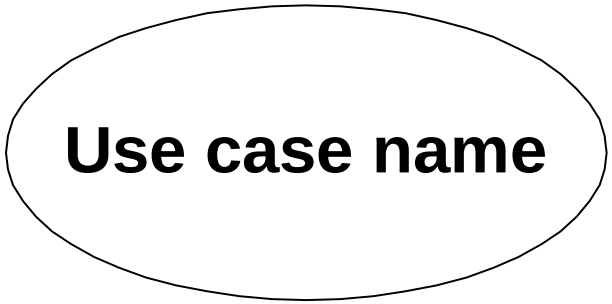
- Identifies and classifies system users by roles and responsibilities
- Includes
  - Names of actual stakeholders (people/systems)
  - Description
  - Related job titles
  - Location
  - Level of expertise
  - Domain expertise
  - Frequency of use

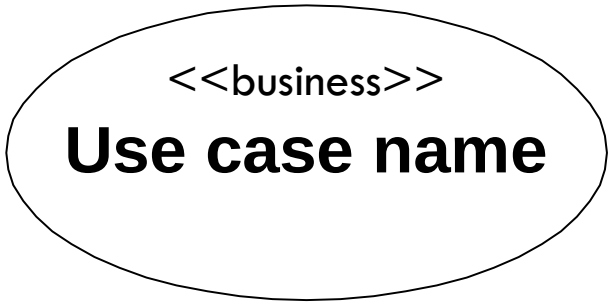
Exercise: actor table

- For each job title or person:
  - customer, manager, executive, visitor, preferred customer, returning customer, customer with a bad credit card, marketing person, guest, unknown user, administrator, programmer, hacker, search engine
- Use an actor name
  - Use **user** and **admin** (easy)
  - Or use all different actor names (hard)
  - Or something in between
- Each actor must have a unique use case that no one else can do.

Use case names

- Use verb-noun clause syntax
- Use domain words
  - The mapmaker's rule
- Don't goldplate
  - Keep notes to ask users later though





Granularity – scope quantity

- **Group of goals**
  - What broad grouping of goals do you want the system or <role> to do?
  - manage, handle, control, do, work with, take care of
  - Higher level manager
- **Goal**
  - What sequence of steps leading to a goal will give value to the business?
  - Scope like PM's WBS: 3 - 10 days of work
  - Lower level manager
  - Target for initial requirements document

Granularity - scope quantity

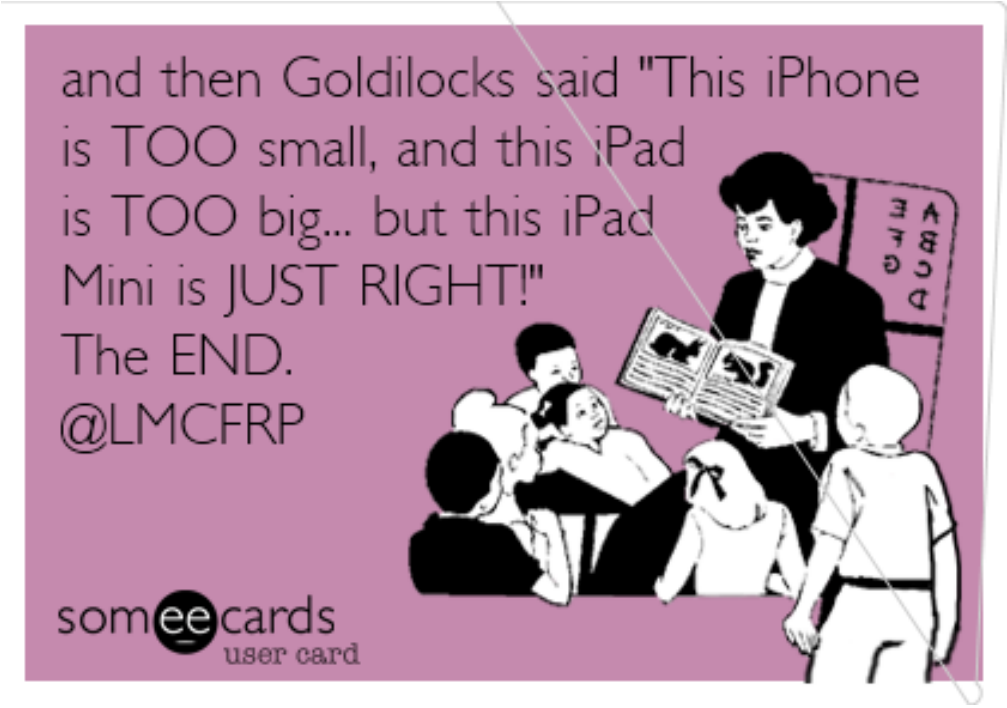
- **Partial goal** / group of tasks
  - What are the individual or named processes in the scenario?
  - no or little business value by itself
  - Staff
- **Task**
  - What are the specific actions that need to happen that are the basic steps of the scenarios?
  - Staff SME
- **Design "requirement"**
  - an idea about how it should be built
  - Record as a design recommendation

Granularity – breaking up groups

- Groups of goals can be broken up into goals by
  - difference in final results
  - use of different business rules
  - separating simple and complex tasks
  - using different data sets
  - difference in middle tasks
  - seeing a CRUD combination

Granularity - scope quantity

- Too small
  - Log in
  - Log out
  - Search
- Too big
  - Manage accounts
- Just right
  - Deliver package
  - Adjust account
  - Edit personal data



Use case levels

- Aim to deliver business value – a goal
- Groups of goals will come out easily
  - Break them into individual value parts
  - Use SMEs to help
- Partial goals will distract you
  - Look for what they are a part of
- Groups can organize partials or goals

Use case validation

- Is it repeatable? Could you do it 1000 times?
  - Walk up, do it, walk away.
  - Does the system return to the **same state** it was in before you started the use case?
    - I can do it, then you can do it.
    - If not, then it's part (task, function) of a bigger use case
- The business gains **value**
  - What was the **goal** that was achieved?
- Strong actionable verb
  - Vague verbs indicate a **group** of use cases

Exercise – actors and names

- Brainstorming
  - Round-robin questions first, then more open discussion
  - No criticism
  - Then apply the rules after you run out of ideas
  - Humorous ideas don't end the session, they sometimes spark new results.
- Actors table – candidates
- Use case summary – candidate and validated
- Actors table – validated

Prioritization

- When to prioritize
  - Early
  - Progressively
- Value to project management
  - Selection of scope based on budget and schedule.
  - Understanding user desire
- How to prioritize value
  - It's not how important to the stakeholder it is, it's about the business
  - Don't ask the stakeholder for "their" priority

Prioritization – from the stakeholder

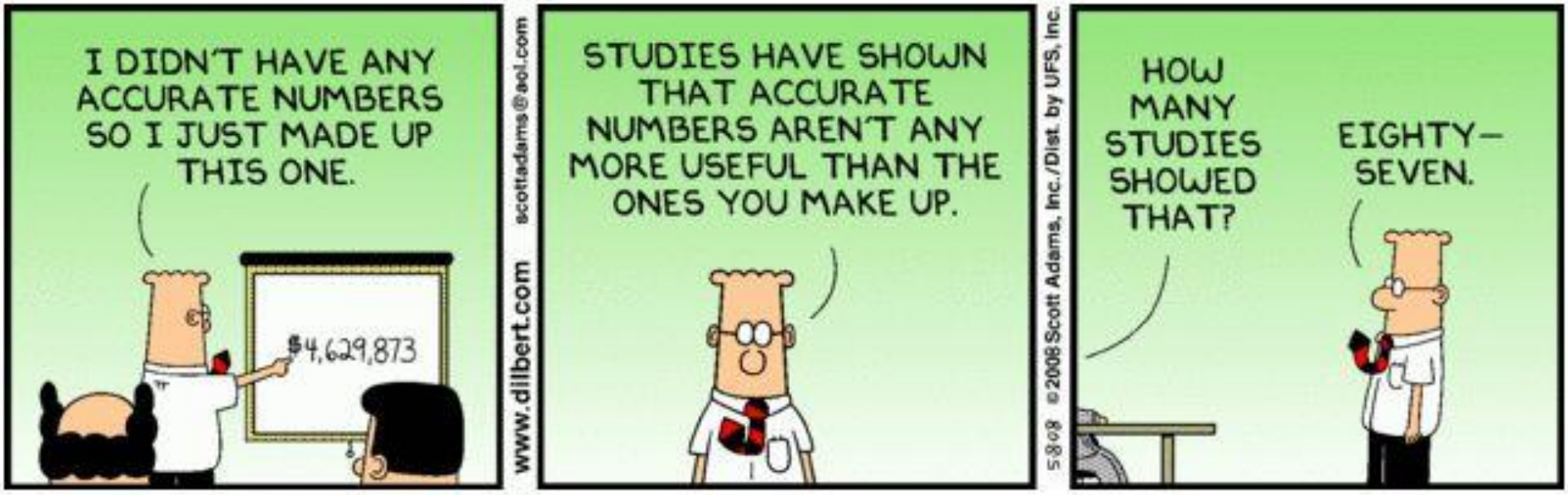
- Not recommended for business value
- Levels – 1,2,3 or mandatory, desirable, nice to have
- Kano – Dissatisfiers, Satisfiers, Delighters
- Analytical Hierarchy Prioritization – compares pairs
- WSJF - User business value, Time criticality, Risk reduction/opportunity enablement
- MoSCoW = must have, should have, could have, want to have but won't have this time

Prioritization – business value

- Essential
  - Scope of use (impact)
    - how much of the business will it improve?
    - how many of the staff will it help?
    - Externally equated to target market
  - Business value (urgency)
    - how much do you wish the business had it now?
    - how bad will the business look if it fails in the future?
    - what level of person is asking for it?
    - Externally equated to price willing to pay
- Optional
  - Anything else important to the business

Prioritization

- Using weighted averages
  - "I'll give it a 9.27"
  - No units = no metrics = no standard
  - Use for understanding but not communications



Prioritization

- Other categories to use
  - ROI
  - Satisfaction
- Negative risk = Impact * Urgency * probability of failure

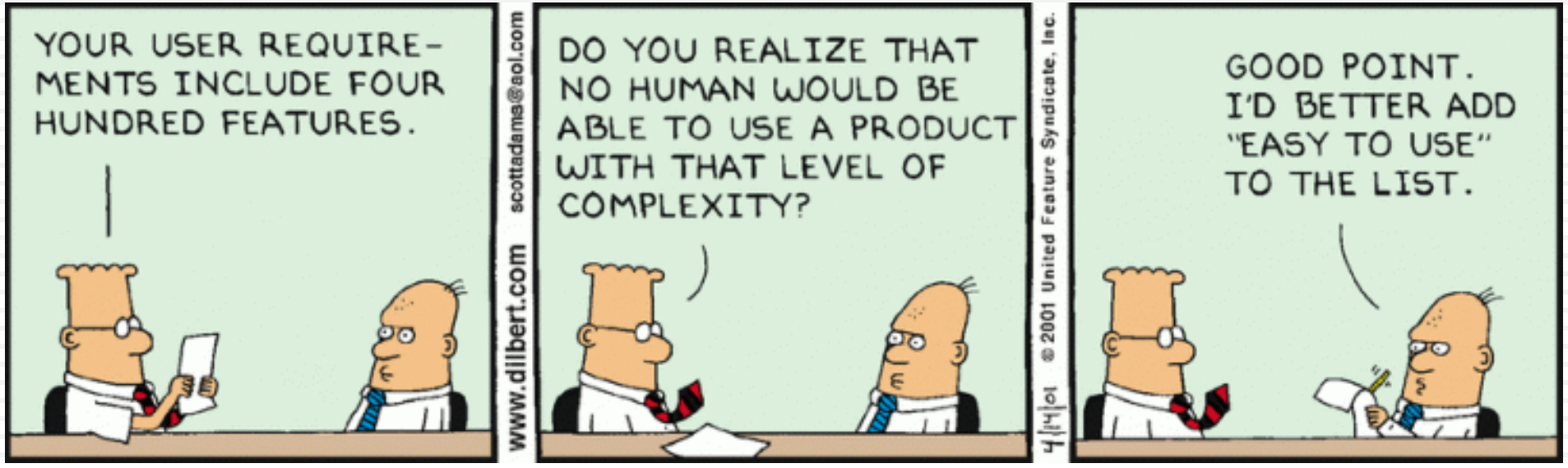
Use case prioritization

- Well-scoped use cases can be used by project managers to estimate project size.
- Priorities can help meet schedule and budget
- **Exercise** - prioritization

Estimation

- User stories – many methods
  - volume (granularity), knowledge, uncertainty, complexity, ideal days, estimation poker, T-shirt sizing, relative velocity
- Use cases – same as work package
  - 3-10 days of effort based on knowledge and complexity.

Use cases – functional detail



Course of events

- Always a “happy path”
  - A success scenario
  - Problems will be captured later
- No conditionals
  - No if-then-else statements
  - Multiple partial sequences (loops) should be expressed as optional parts.
- Detail level
  - as much detail as possible without design
- No design (without constraints)
  - e.g. button click, submit buttons or anything that connects system to hardware, software, tools, or materials

Course of events

- Syntax
  - The use case starts when the actor …
  - Response: The system …
- Possibly multiple actors could initiate the use case
- Numbering
  - Group one or more statements/tasks together
  - Smaller increments are better when you need to start in the middle at a specific spot due to an error.
  - Start with system does… usually.

Functional detail - tasks

- Tasks are sequenced low-level activities that can't be broken down any further
- The task statement contains
  - A responsible party/noun
  - The action/verb to be done
  - A description of the things/direct object which the verb acts on.
- A system functional requirement starts with "the <system> shall …"
- A business functional requirement starts with "the <role> shall …"

Process rules – happy paths

- A rule to two happy paths is 2 use cases
  - not an error path
- The **functional** part of a requirement…
  - The system shall print the Large Sales report …
- …May be modified with a **rule** part
  - …**If**/**when** sales are > 100,000 then using rate chart DF3
  - …Use rate chart DF3 **when** sales are > 100,000
  - …On Thursday
  - …When I say so

Too much in a use case

- Any use of conditional logic for workflow will indicate separate use cases when
  - The outcome is different
  - Steps are skipped
  - Steps are not always included
  - A rule is used to alter workflow to another happy path

Alternative flows

- Done after structuring so numbering is done once.
- Two types
  - **Extension** points return back where you came from after an optional set of steps. <<extends>>
  - **Failure** points stop the use case, return you to a different point, or fix the problem and let you continue.
- Write in your choice of styles (informal, formal)
  - **Bad thing happens** (13, 15) – try to fix and return to 12.
- Include a return point or end the use case.

Extension points

- **Extension points** are optional partial goals
- The point at which the non-error extension set of tasks starts is where it returns when it is finished.
- An path that is extra to the flow
  - a gift wrap
  - additional product offer to accept
  - a search to do before continuing

Use case alternative flows (failure)

- Generally tied to a rule that fails
- Written in informal or formal style

Specifying rules for failures

- Short one-time use rules are better included in use case documentation below the functional statement
  - The system validates the amount.
    - RULE: **Available funds** - Account balance is larger or equal to than amount requested.
    - RULE: **Daily total withdrawal**: Amount requested is less or equal to **R#24 MAXIMUM WITHDRAWAL AMOUNT**
    - RULE: **Increments** – Must be in increments of $20.
- Larger rule tables, decision trees, or reused rules are better in a separate rule document and referred to.
  - **R#24 MAXIMUM WITHDRAWAL AMOUNT**: $500 per day starting at midnight.

Exercise – course of events

- Write out a set of instructions that is your tasks for
  - getting ready for work
  - fixing a lunch
  - washing a dog
  - painting a wall

Analysis phase - verification

Verify granularity

- Strong actionable verb
  - Vague verbs indicate a **group** of use cases
- The business gains **value**
  - What was the **goal** that was achieved?
  - No value at the end **to the business** indicates that this small group of tasks is a part of another use case

Verify analysis language

- Can the **system** functionality be applied to a phone interface?
  - If you talk about a GUI then that's design.
  - Design language is appropriate when there is a project constraint to use a specific design

Verify testability

- Does your complete scenario meet the walk-away test?
  - Walk up, do it, walk away
- Does the system return to the **same state** it was in before you started the use case?
  - I can do it, then you can do it.
  - If not, then it's part (task, function) of a bigger use case

Verify completeness

- Can you define a requirement (at a higher level) that summarizes a group of requirements?
- Can you define a requirement (at a lower level) that is a part of the requirement?
- Are there other higher scenarios that would use this use case?
- Are there other lower scenarios that would use this use case?

Verify granularity

- Any use of conditional logic for workflow will indicate separate use cases when
  - The outcome is different
  - Steps are skipped
  - Steps are not always included
  - A rule is used to alter workflow
- Break use case in to individual use cases and then structure later

Verify lack of design

- Can the **system** functionality be applied to a phone interface?
  - If you talk about a GUI then that's design.
- If you talk about **how** something is done, that's a rule or design.
  - If there are constraints, a design feature like a screen, mouse, or keyboard works.

Verify use case course of events

- Completeness check – role playing game
  - Person = system or role
  - Data known by system listed on one sheet of paper
  - Designs sketched on paper, hold up when active
  - Messages passed on sticky notes, one for request, one for response, data recorded

Verify use case course of events

- Testability check -play out differently
  - Use data range boundary values
  - Substitute more extreme values
  - Repeat more times
  - Make environment worse

Exercise – verify course of events

- Apply tests to your course of events from the previous exercise.

Use cases – more detail

Use case sections

- Metadata
- Course of events / task sequence
- Optional
  - Preconditions
  - Post-conditions
    - Guarantees (minimal & maximum success)
  - Alternative flows – options
  - Alternative flows - errors

Use case metadata

- Recommended
  - Name – verb-noun syntax
  - ID and date
  - Actor(s)
  - Stakeholder originator
  - Priority (goal level and above, business value not personal)

Use case metadata

- Optional
  - Project
  - System / subsystem
  - Date updated
  - Cross-references
    - Business rules, data, prompts & menu text, designs
  - Level
  - Tracing
  - Index

Use case metadata

- Optional
  - Purpose
  - Explanations
  - Examples of ways to meet
  - Stability
  - Complexity
  - Stakeholders' interests

Non-functional requirements

- Placed in use case notes, another category, or other documents
- Use cases can have specific NF requirements
  - security issues
  - capacity needs
  - maintenance needs
- Tests
  - What adjective or adverb describes how all/a subset of the functional parts should behave?
  - What details do not affect what the functionality of the system or <role> does?

Non-functional requirement types

- Notes
- Security
- SLAs (expectations of performance)
- Quality or integration
- Data dictionary – data rules
- Process rules
- Interfaces
- Design recommendations
- Prompts, menus, and messages

Pre- and post-conditions

- Pre-conditions
  - block the use case from doing the first step.
  - validate the state of the software before anything happens specific to that use case.
  - A log on is not a pre-condition in a system use case
- Post-conditions restate the important points connected to the goal.
  - Optional usually

Agile

- User story acceptance criteria
  - same as post-conditions in use case

Special requirements / notes

- Put things like SLAs and location or time needs in a special category.
- Non-functional requirements that are specific to this use case should be documented with the use case.
- Admin people can understand why a requirement should be met.

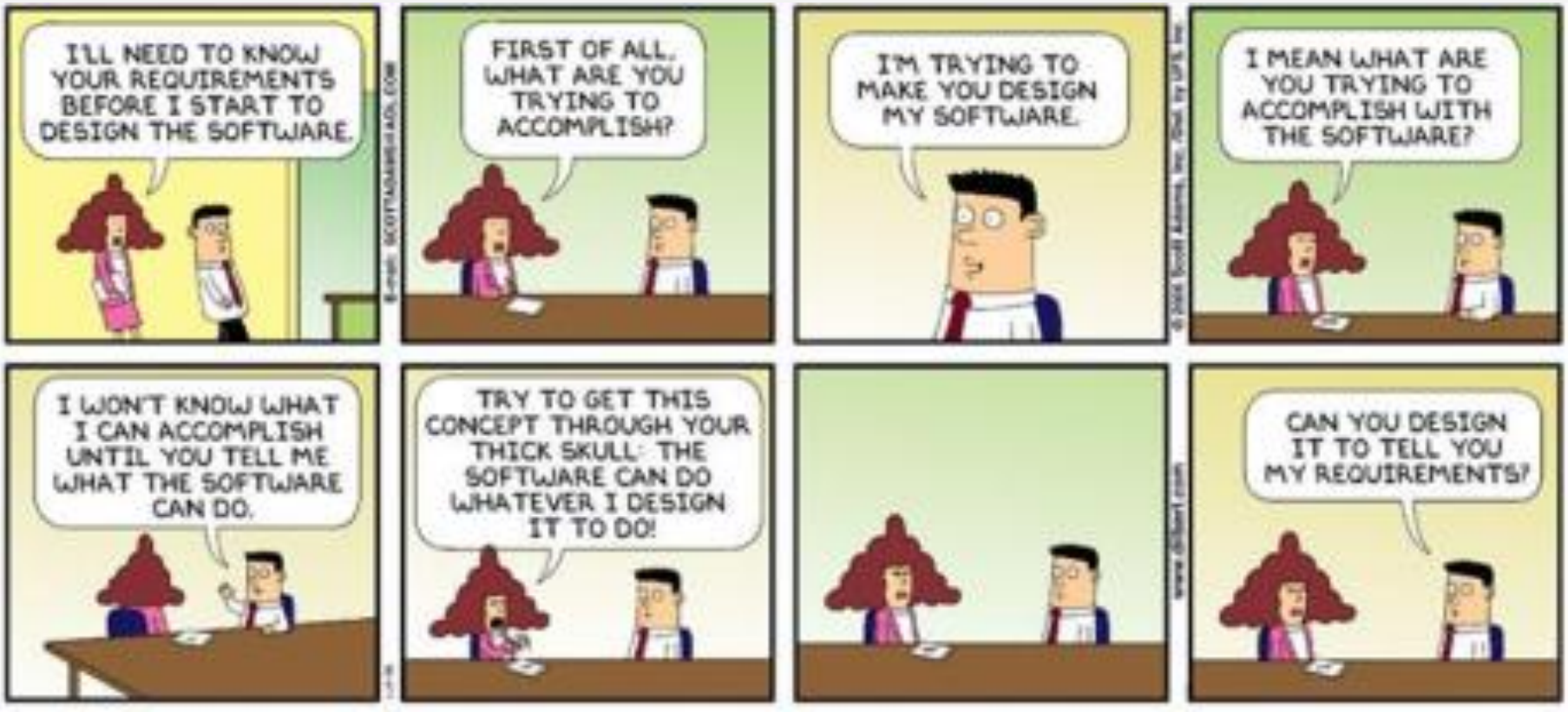
Using references

- Move out the details that are not functional
  - Small detailed parts are OK for clarity.
    - Sub points, mark the type
  - Use the specific document to capture reusable or complex rules, designs, etc.
    - Process rules
    - Data dictionary
    - Designs - menus, screens
    - Externalized text – prompts, error messages
  - Use character style to show rules & data dictionary items

Exercise – use case detail

- Create pre-conditions / post-conditions if any
- Write/use course of events
- Create sections and reference for
  - Data dictionary [DD] or just bold text
  - Business rules [R#] or subpoint – and bold text
  - Designs [D#] – screen prototypes
  - Reports [R#] – report prototypes
- Create error flows
- Create alternative flows

Analysis phase - documentation

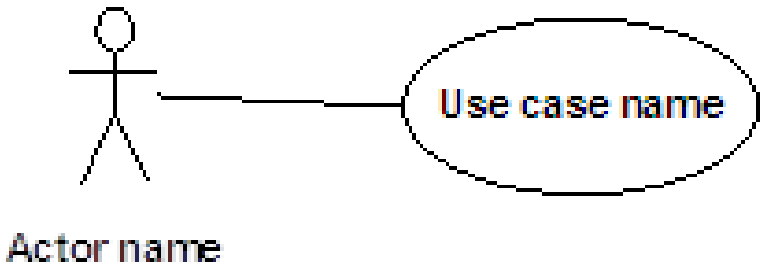


Use case text styles

- Mix ‘n’ match
- Informal – the story
  - An elevator speech
  - Use for a table of contents
  - A descriptive sentence or paragraph
- Formal – all the facts
  - When it’s important to be clear
  - Up to several pages

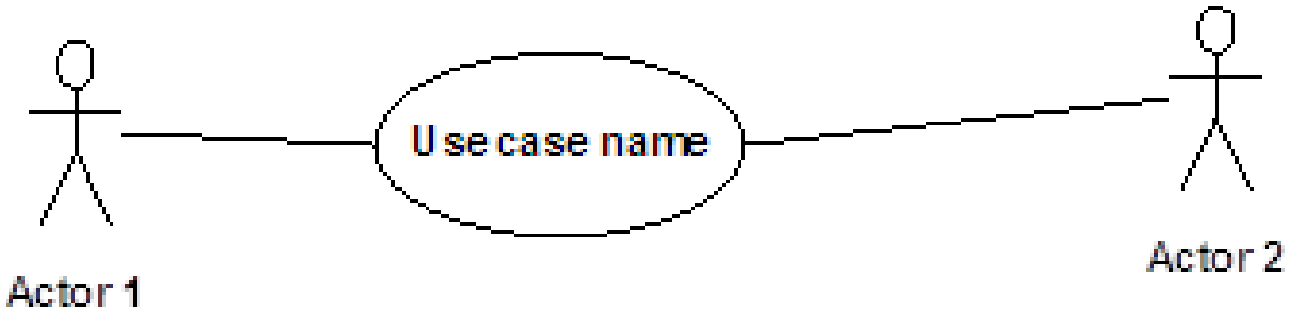
Use cases and actors

- Only to show scope, granularity and triggers!
  - No sequences so no arrows!
- Split diagrams into readable sections.
- Only show actors who initiate use cases on diagrams.
- Keep lines from crossing when possible



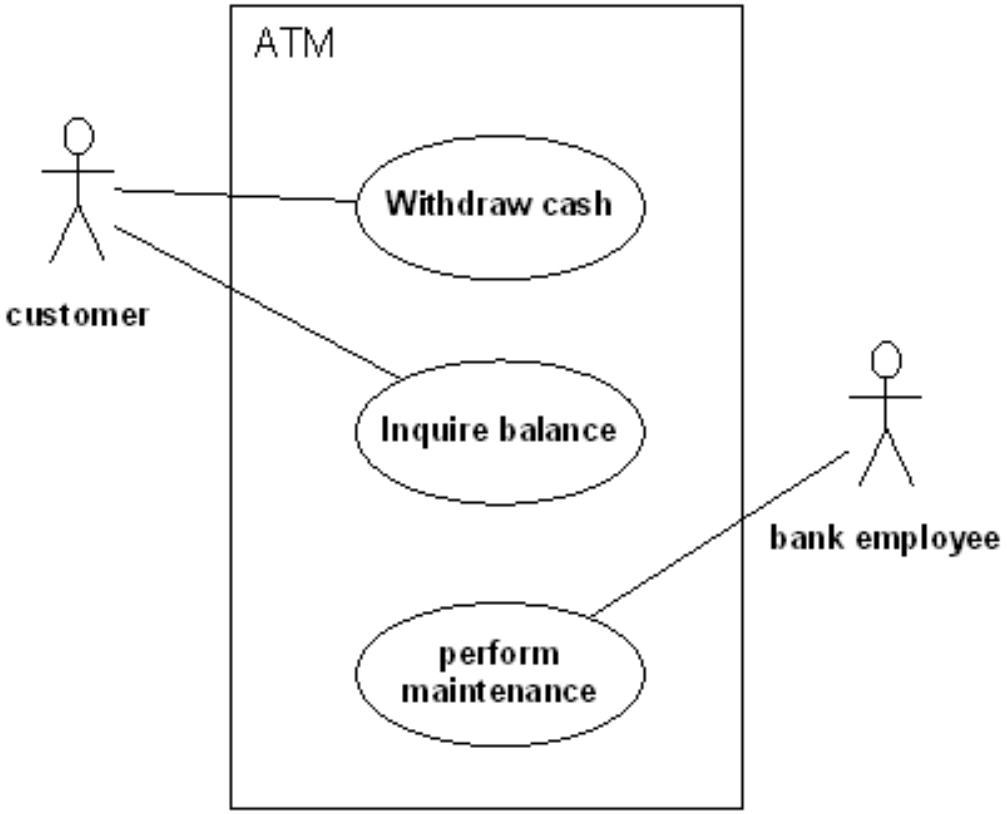
Use cases and actors

- Place actor on either side, but no directionality.
  - Not a flow or sequence diagram, but a scope diagram.
- Keep lines from crossing when possible
- **Exercise** – use case diagram



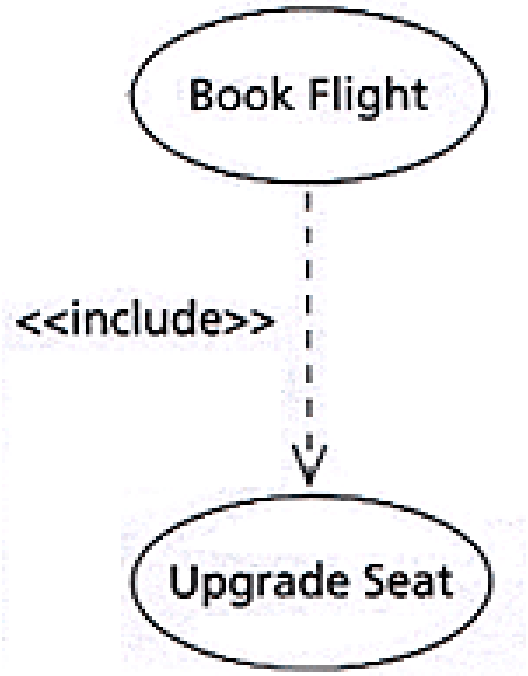
Domain

- The box of a domain surrounds the use cases but not the actors.



Includes

- Use special includes when
  - users expect that name (log in)
  - options are important to see
- Shown as use case ellipse with a dependency arrow and stereotype (category surrounded with <<European quotes>>).
- Include is a partial goal level, not goal level.



Include vs. different use case

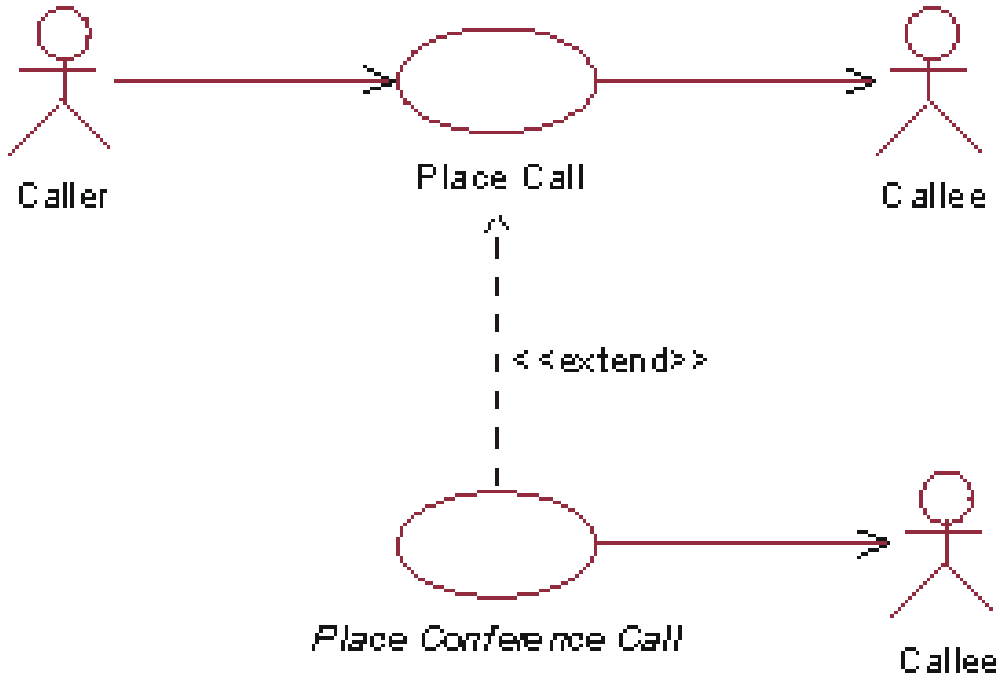
- Write includes into use cases as bolded names in course of events
  - **Do Log On (SF24)** workflow returns here
  - <<includes>> **Do Log On (SF24)**
- Alternate flow (extends) – separate section not in course of events
  - Optional partials
    - **(#3) Print Receipt (SF33)**
    - (#3) <<extend >> **Print Receipt (SF33)**

Including a group of partials

- For a choice of processes to make multiple use cases simpler to document.
- <<include GP1 Log In>>
  - GP1.1 – Log in with credentials
  - GP1.2 – Log in with Google
  - GP1.3 – Log in with Facebook
- Each partial use case may have a different flow but the goal achieved is the same.

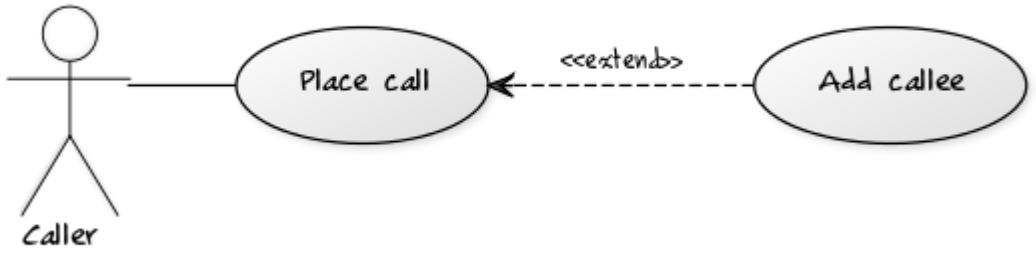
Extend

- Dashed arrow pointing from **optional workflow** to use case – a dependency
  - (diagram incorrectly uses arrows on solid lines and shows non-actor)



Extend

- No indication of availability in the course of events
- Will branch from the beginning of the numbered statement referred to in the extends statement.
- Show the formal steps below the extends statement or informal description after.
- Larger process flows can be named and documented as a partial goal use case.
- **Add callee** (#3) – Actor selects option to add callee, enters number, and confirms callee has connected.

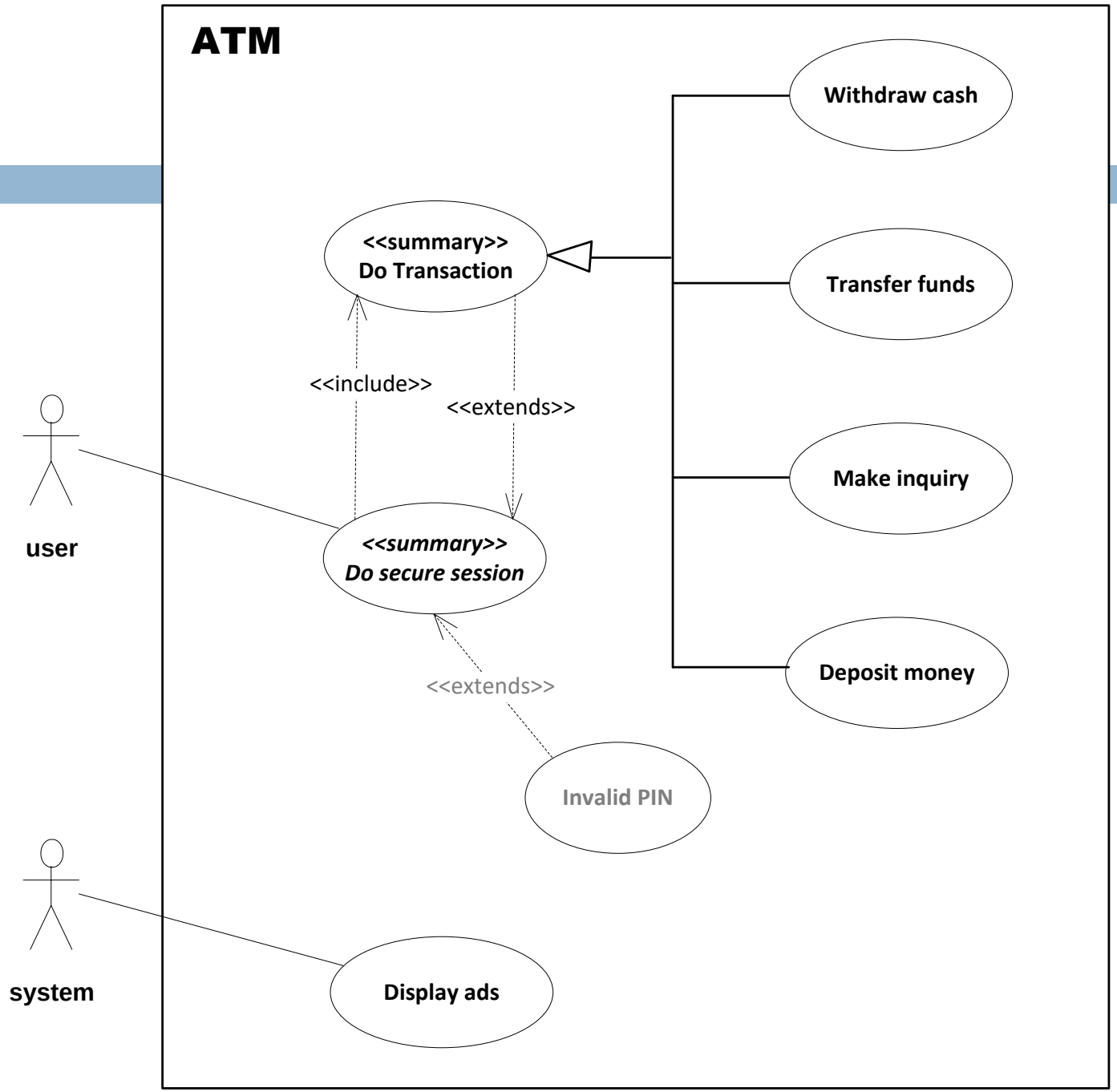


Security pattern

- Security is a use case wrapper around other use cases
  - Start session (authenticate)
  - <<include>> Do secure process
  - End session (clean up and deaccess)
- This is allows for <<extends>> Do another secure process as an option
- Don't do
  - <<include>> Start secure session
  - Process tasks…
  - <<include>> End secure session

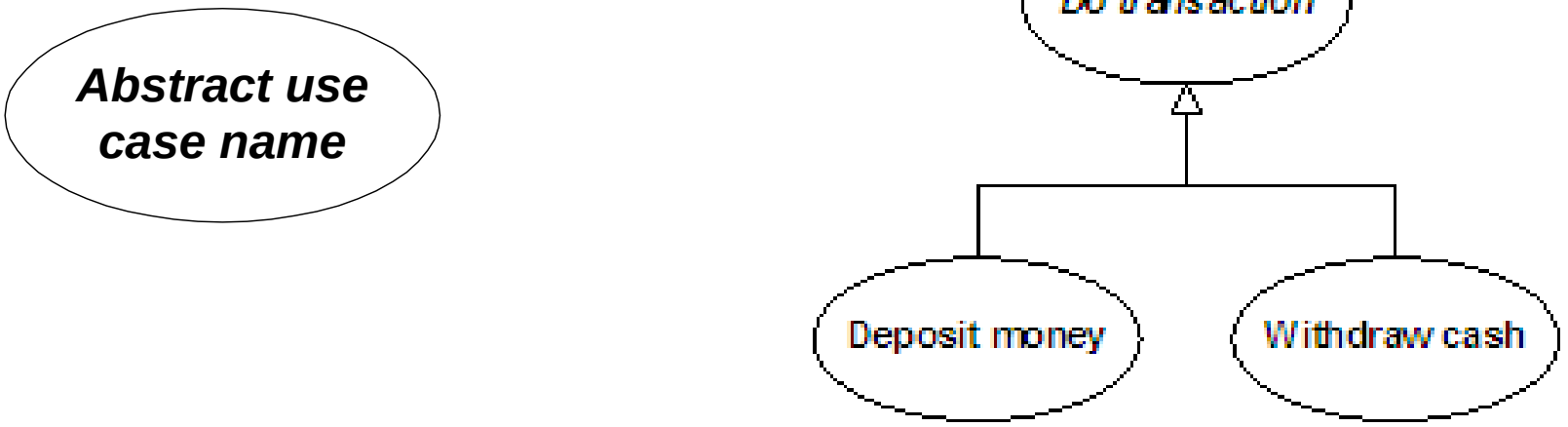
Security

- Summary = group
- Invalid PIN is an error path



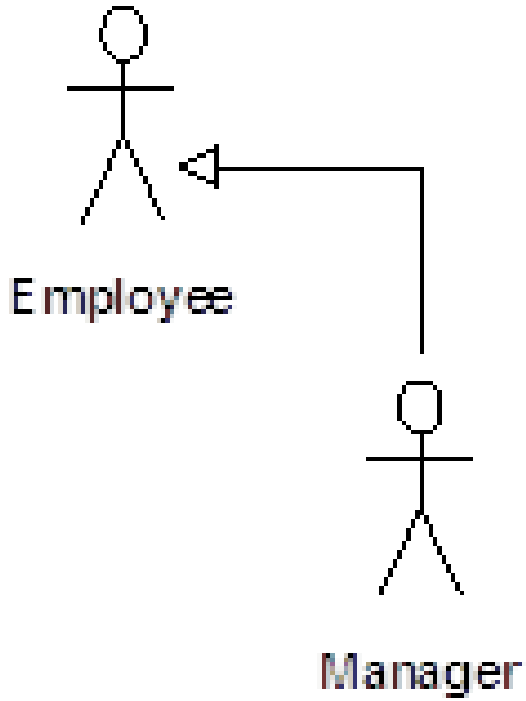
Use case structure - groups

- Use grouping when it cleans up a use case diagram and makes it easier to understand.
- Produce several versions of the functionality



Role generalization

- Actors can share use case initiation
- Show with generalization arrow



Use case diagram tools

- Text driven
  - https://yuml.me/diagram/usecase/draw

Design phase

prototypes, interfaces, screen shot scenarios

System dependencies

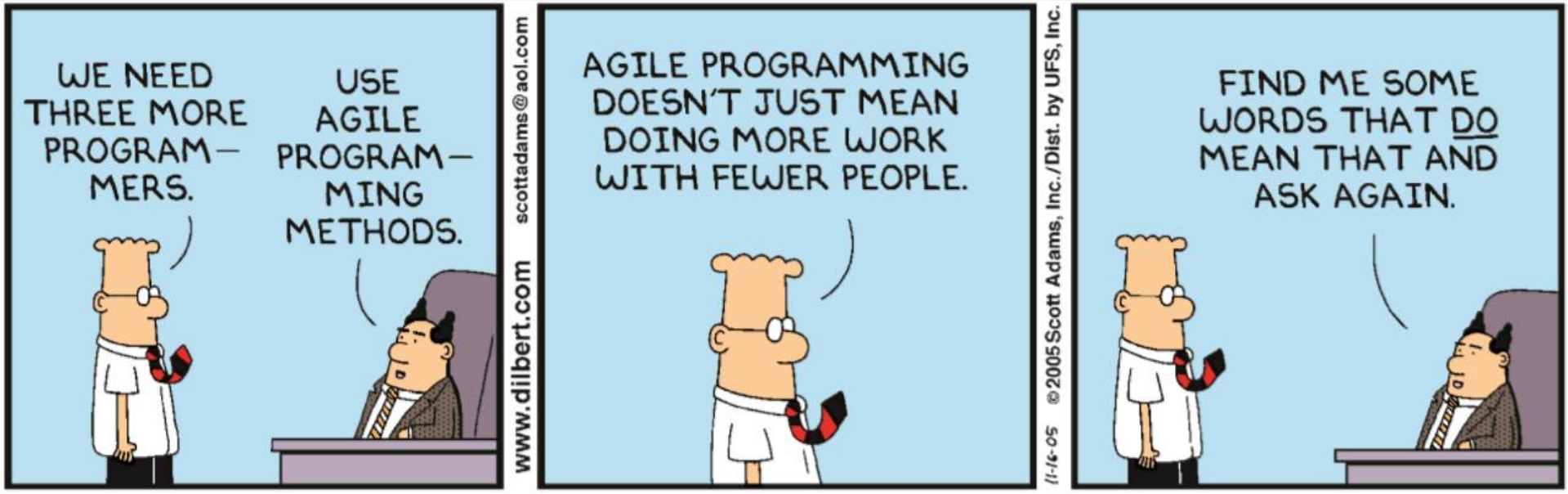
- Fitting the requirements to the constraints
  - moving functions to systems
  - moving functions to subsystems
  - moving functions to libraries/components/DLLs
  - systems talking to other systems (APIs)
  - refactoring systems

Dual purpose tools

- Tools here can be used in analysis when constraints are known.
  - paper reports
  - web page wireframes
  - screen navigation

Development phase

state, activity/flowchart models



Programming models

- State chart
  - use to show change in data over a complete process
  - order status over a transaction
- Activity diagram
  - use to show a time structured flow of process
  - can be divided into swim lanes for functional segmentation

Traceability

- Use words “roll-up” and “drill-down” to talk about relationships between levels with business language.
- All higher level requirements have lower levels
- All lower level requirements have higher level
- Traceability matrix
  - Assigns codes
  - Tracks relationships

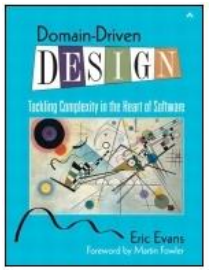
ID	USER REQUIREMENTS	FORWARD TRACEABILITY
U2	Users shall process retirement claims.	S10, S11, S12
U3	Users shall process survivor claims.	S13

Traceability

- Used to show
  - completion of requirements in coding
  - test coverage of code
  - areas affected by bugs

Resources

- **Domain-Driven Design: Tackling Complexity in the Heart of Software** by Eric Evans. Addison-Wesley Professional, Aug 2003



- **Analysis Patterns** by Martin Fowler
- For programmers
  - Larman, Craig 1998. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design.* Prentice Hall PTR. (get the 2nd version not the 3rd)

Testing phase

test case design, data dictionary validations

Test case design

- Use cases provide test case designs
- Scenario clean/positive tests
  - The happy path of the use case course of events
- Scenario dirty/negative tests
  - The course of events with a specific alternative path to produce a failure and how to handle it
- System tests
  - Any notes for non-functional requirements can be tested

Data dictionary validation

- used to provide data sets to test cases for clean and dirty tests
- best when described with equivalence class partitioning
  - tests use boundaries

Data dictionary validation

- Add to each entity entry
- Constraints / rules
  - Validation – bounds, members
  - Dependency
  - Formats for input and output
  - Examples
- How it is used
  - read/write
  - aggregated/processed report