Useful Use Case Tips & Techniques

Developed for PDS2011

Brenda Boon

The Concept

- Use cases is a deceptively simple tool that examines a computer system by expressing the interactions between the system and its environment: what goes in and what comes out.

- Use cases in the requirements activity see the system from the angle of what the system can do for the user; they boil down the requirements to the barest essentials.

The Concept – When to Use

- Use cases emerged from the object-oriented development world, but *can be used on projects that follow any development approach* because the user doesn't care how the software is built.

- Use cases shift the perspective of requirements development to discussing what *users* need to accomplish, in contrast to the traditional elicitation approach of asking users what they want the *system* to do.

The Concept - Why Use

- The power of the use-case approach comes from its *task-centric* and *user-centric* perspective.
  - The users have clearer expectations about what the new system will let them do, than if you use the function-centric approach.
  - Use cases also help the developer understand better the user's business.
- Each one encapsulates a set of requirements. which lets you easily manage and track the use cases individually and provides a better alternative to prose requirements

The Concept – Why Use

- There is more to effectively using use cases than just capturing them and putting them into diagrams.
- As you implement use cases, you need to validate them, determine their size, and establish a plan for implementation.
- Then, you need to incorporate the use cases into your system design and turn them into code and documentation.
- Throughout this process, you must also be aware of the status of each use case.

The Concept – Why Use

- Use cases are the backbone of testing and documentation.
- If the use cases are clearly stated and testable, they form the basic system test plan.
- They are also well suited for acceptance testing, in which users test all use cases on the system and approve the system's performance of each one.
- Because use cases represent the user's perspective, they can form the initial user manual, online documentation, or help file.

The Concept – Why Use

- The use-case approach helps with requirements prioritization; a use case has high priority when:
  - It describes the core business processes that the system enables
  - Many users will use it frequently
  - A major user class requested it
  - It provides a capability required for regulatory compliance
  - Other system functions depend on its presence

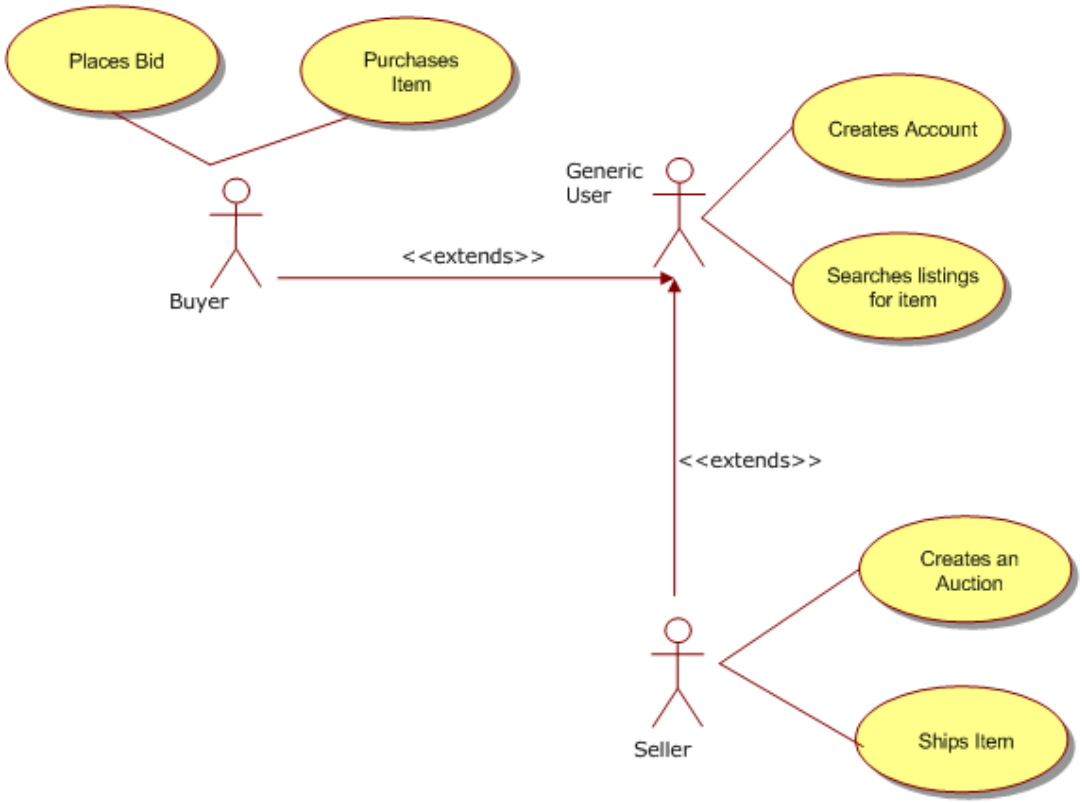
The Basics – Some Definitions

- **Use Case**
  - *Use cases* are text documents. They are actually a page or two of text representing each oval in the use case diagram. Use cases describe a sequence of interactions between a system and an external actor.

- **Actor**
  - Actors are the user of the use case. A person, another software system or a hardware device that interacts with the system to achieve a useful goal; can be something more abstract e.g. a specific date and time; usually depicted as a “stickman” in the use case diagram.

The Basics – Some Definitions

- Use Case Diagram
  - Shows the relationships between the actors and the use cases.



The Basics – Some Definitions

- **Scenario**
  - A specific instance of a use case; scenarios are small narrative descriptions of someone working through the use case, ie. It is one possible path through a use case.
  - Scenarios can be used in two different places in the development life cycle:
    - During requirements to provide immediate feedback to the users and analysts as to whether the use cases are accurate reflections of the users' needs;
    - During testing, to test whether the computer system reflects the requirements.

The Basics

- *Use cases* never initiate actions on their own; the initiator of all interactions is the *actor.*
- *Use cases* are black-box representations and do not include any implementation-specific language such as:
    - People's names (instead of roles)
    - Specific departments in the organization
    - User interface details (buttons, menu navigation)
    - IF-THEN-ELSE statements (developer language)

The Basics

- *Use case diagrams* show the relationships between use cases and use cases and actors.

- *Interactions* between actors and use cases are shown with a straight, unbroken line with arrows. It is not customary to label these lines. The description of what is behind these lines is what constitutes the Basic Course of Events in the use case text.

The Basics

- Appropriate level of detail:
  - As you proceed through requirements gathering, use cases go from being *general* to being *detailed*; it's important to start at the general level before jumping to details.

How To – Iterations
Iteration 1

Objective: Create placeholders for each major interaction you expect the users to have with the proposed application.

- This initial use case contains only the minimum information that is needed as a placeholder; include names and short descriptions of each interaction.
  1. Identify and document the problem statement
  2. Identify the users, the stakeholders
  3. Interview all stakeholders to get minimum input
  4. Find the actors
  5. Create the initial use cases
  6. Start the business rules catalogue
  7. Create a risk list

How To – Iterations
Iteration 2

Objective: Create a comprehensive set of use cases and business rules that describe the application.

- During this iteration you find the interactions and analyse the processes, asking, Is this right? Can we simplify this?, ec.
  1. Breakout detailed use cases, eg one initial use case may be broken down in two
  2. Create the second iteration of use cases, ie add details to initial set by adding the basic course of events
  3. Document the non-functional requirements
  4. Add business rules
  5. Validate, eg team walkthrough of all use cases and requirements

How To – Iterations
Iteration 3

Objective: To have clear project requirements and to define the system.

- During this iteration you decide what stays and what is cut out of the system requirements and design; you make sure that the solution does not solve unnecessary problems.

How To – Iterations
Iteration 4

Objective: To integrate the non-functional requirements with the use cases, add the user interface requirements and package the documentation for the design effort

How To – Various Starting Points

1. Identify the actors first then the business processes in which each participates.
2. Identify the external events to which the system must respond, then relate these events to participating actors and specific use cases.
3. Express the business processes in terms of specific scenarios, generalize the scenarios into use cases, and identify the actors involved in each use case.
4. Derive likely use cases from existing functional requirements. If some requirements don't trace to any use case, consider whether you really need them.

How To – Essential Elements

- A unique identifier
- A name that states the user task in the form of a verb + object (e.g. Place an order)
- A short textual description written in natural language
  - Describe what the use case is accomplishing. What will the user be doing while "withdrawing funds" or "examining a passbook," for example. Go into detail, but don't describe how the user might use a computer program.
- A list of preconditions that must be satisfied before the use case can begin;
  - What conditions must exist before this use case can complete successfully? The conditions can be internal, or external. For example: The account balance must be greater than the withdrawal Post conditions that describe the state of the system after the use case is successfully completed

How To – Essential Elements

- A numbered list of steps that shows the sequence of interactions between the actor and the system that leads form the preconditions to the post conditions
- Post condition is something that you can check after the use case completes in order to determine the success or failure of the use case. An example post condition: the new balance is the old balance, less the amount withdrawn.

Tools, Hints & Tips: Use Case Do's

- Ask users to review your use cases
- Keep the number of use cases small (not the number of scenarios, which can be numerous)
- Use a common template for use cases in a project.
    - Standardization of a template is more important than what the template itself looks like.

Tools, Hints & Tips: Use Case Do's

- Specify *Business rules* in a separate document and then reference the individual relevant rules in the "Business rules" section of the use case.
    - Business rules are policies that the business establishes that might effect the outcome of the use case. For example, "You may not withdraw more than $20 dollars within a seven-day period." It's not a good idea to clutter up the use-case description with these rules, but they have to be specified somehow to make the document accurate enough to be usable.

Tools, Hints & Tips: Use Case Do's

Inspection Checklist for Use Case Documents

- ❑ Is the use case a standalone, discrete task?
- ❑ Is the goal, or measurable value, of the use case clear?
- ❑ Is it clear which actor(s) benefit from the use case?
- ❑ Is the use case written at the essential level, rather than as a specific scenario?
- ❑ Is the use case free of design and implementation details?
- ❑ Are all anticipated alternative courses documented?
- ❑ Are all known exception conditions documented?
- ❑ Are there any common action sequences that could be split into separate use cases?
- ❑ Is the dialog sequence for each course clearly written, unambiguous, and complete?
- ❑ Is every actor and step in the use case pertinent to performing that task?
- ❑ Is each course defined in the use case feasible?
- ❑ Is each course defined in the use case verifiable?
- ❑ Do the pre- and post-conditions properly frame the use case?

Tools, Hints & Tips: Use Case Don'ts

- **Too many use cases**
  - You may not be writing them at the appropriate level of abstraction; most systems should have 20-50 uses cases
- **Highly complex use cases**
  - Keep the use case simple by addressing the essence of the actor and system behaviours instead of specifying every action in great detail
- **Including user interface design (GUI) in the use cases**
  - Use cases should focus on what the users need to accomplish with the system, not on what the screens will look like
- **Including data definitions in the use case**
  - Create a separate data dictionary
- **Use cases that users don't understand**
  - Write use cases from the user's perspective, not the system's.

Tools, Hints & Tips: Tool Selection

- *Simple Tools*
  - Microsoft Word
  - Microsoft PowerPoint
  - A paper napkin …
- *Complex (and expensive) Tools*
  - Rational Analyst Suite (Requisite Pro, Rational Rose)
  - Together J
  - Doors
  - Etc …

Use Case VS User Story

- **What is a User Story?** Simply put, written from the context of the user as a simple statement about their feature need. They should generally have this format. "As a -role-, I want -goal/desire- so that -benefit-"

- **How is a User Story different than a Use Case?** While a use case is highly structured and tells a story, the User Story sets the stage by stating the need. A User Story is the prelude to the use case by stating the need before the use case tells the story.

- **How does the User Story fit into the process?** User Stories are great as an activity in collecting and prioritizing the high level features. Getting this initial feedback from the customer is a simple way of trying to get all of their needs identified and prioritized. The User Stories will then morph themselves into the business requirements and use cases.

Questions

Selecting an appropriate BPM strategy for your business environment.

BPM STRATEGY

Overview

- Understanding the purpose of BPM
- Measuring BPM success
- Planning for Process Improvement

BPM: What is it?

"Business Process Management (BPM) is a nebulous term fraught with vague descriptions, blurry technology definitions..." *InfoTech (January 2011)*

Business Process Management (BPM) is a collection of methods and tools that document, improve, and monitor business processes.

BPM 101

- BPM is more than automation
- Don’t focus on BPM as a cost-reduction tool
- **Efficiency** is not the best driver of success
- BPM improves **defect rates**, **delay rates**, **cycle time**, and process **customizability**
- IT should not drive a BPM project
- The most important phase of the BPM project comes at the end

Understanding the Purpose of BPM

- BPM delivers value to the organization by:
  - Streamlining business processes
  - Reducing inconsistency in business process
  - Increasing both the visibility of processes and the state of a process instance
  - Highlighting opportunities to improve processes
- BPM is not synonymous with automation
  - Efficiency vs. Effectiveness

Understanding the Purpose of BPM

Efficient

- Heavily automated
- Minimum of moving parts
- Low-cost
- Optimal cycle times for individual process steps
- Bottom Line focus

Effective

- Consistent, predictable output
- High-quality product
- Optimal cycle time for entire end-to-end process by minimizing process steps
- Top Line focus

Success requires effectiveness as much or more than efficiency.

Measuring BPM Success

- BPM is not cheap!
- BPM takes time
- Five Steps:
  - Identify
  - Document
  - Refine
  - Automate (where applicable)
  - Measure, measure, measure!!!

Measuring BPM Success

BPM can improve...	Why?	Average BPM improvement
Defect rates	BPM improves the predictability of processes by helping organizations understand how they really function	10%
Frequency of service or product delays	BPM minimizes the likelihood of surprises that can delay product or service delivery	10%
Cycle time	Process evaluation can reduce the number of steps in processes	15%
Customization and flexibility	By providing a window into organizational processes, BPM facilitates mixing and matching components to produce new products and services	15%

**BPM improvement numbers are based on Info-Tech survey data, N=73.*

Planning for Process Improvement

- Automation is not the answer to BPM
- BUT automation can provide 2 benefits for improvement planning
  - Process predictability
  - Improved measuring

Planning for Process Improvement

- The real success and benefit of BPM is process improvement
- Improvement based upon meaningful measurement
- Measuring what you were trying to solve in the first place!

Planning for Process Improvement

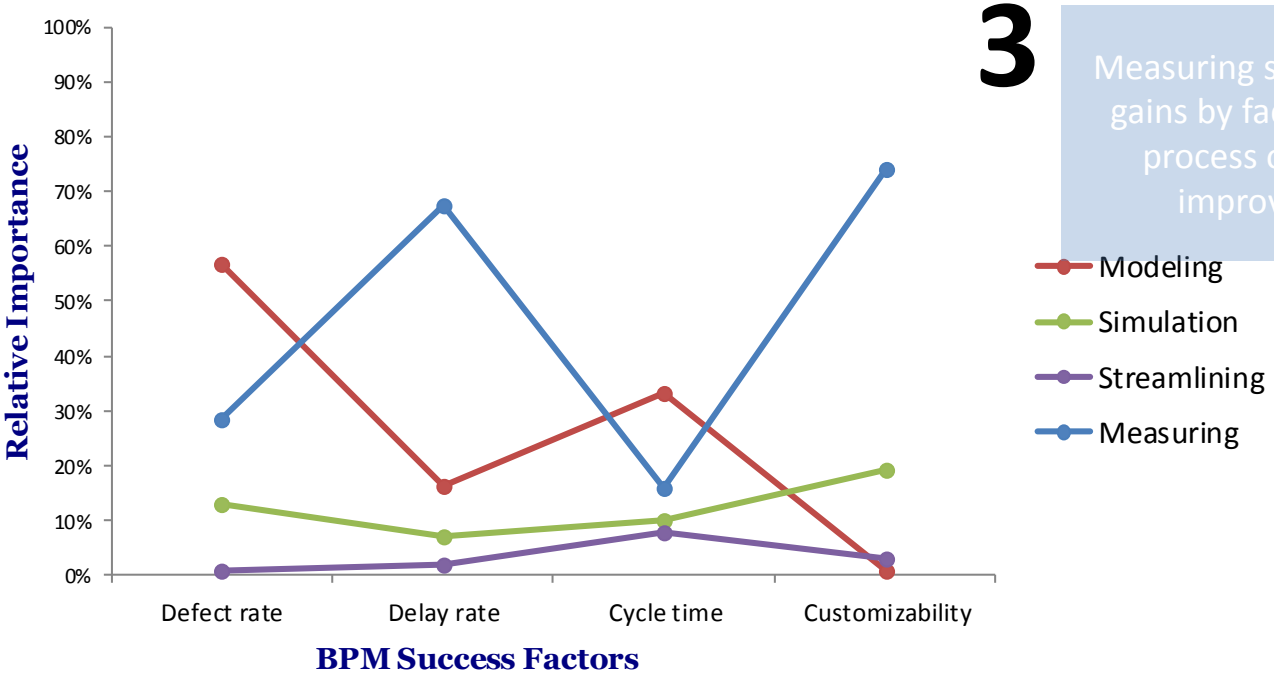
1

Modeling provides the gateway to process improvement but does not make customization any easier on its own.

2

Streamlining process with technology produces only limited benefit on its own.

Correlation of BPM stages to success factors amongst those who have completed a BPM project.



Source: Info-Tech Research Group
N = 73

3

Measuring solidifies BPM gains by facilitating the process of ongoing improvement.

4

Simulation improves customizability by providing the ability to quickly adopt new processes without the need for trial and error.

Planning for Process Improvement

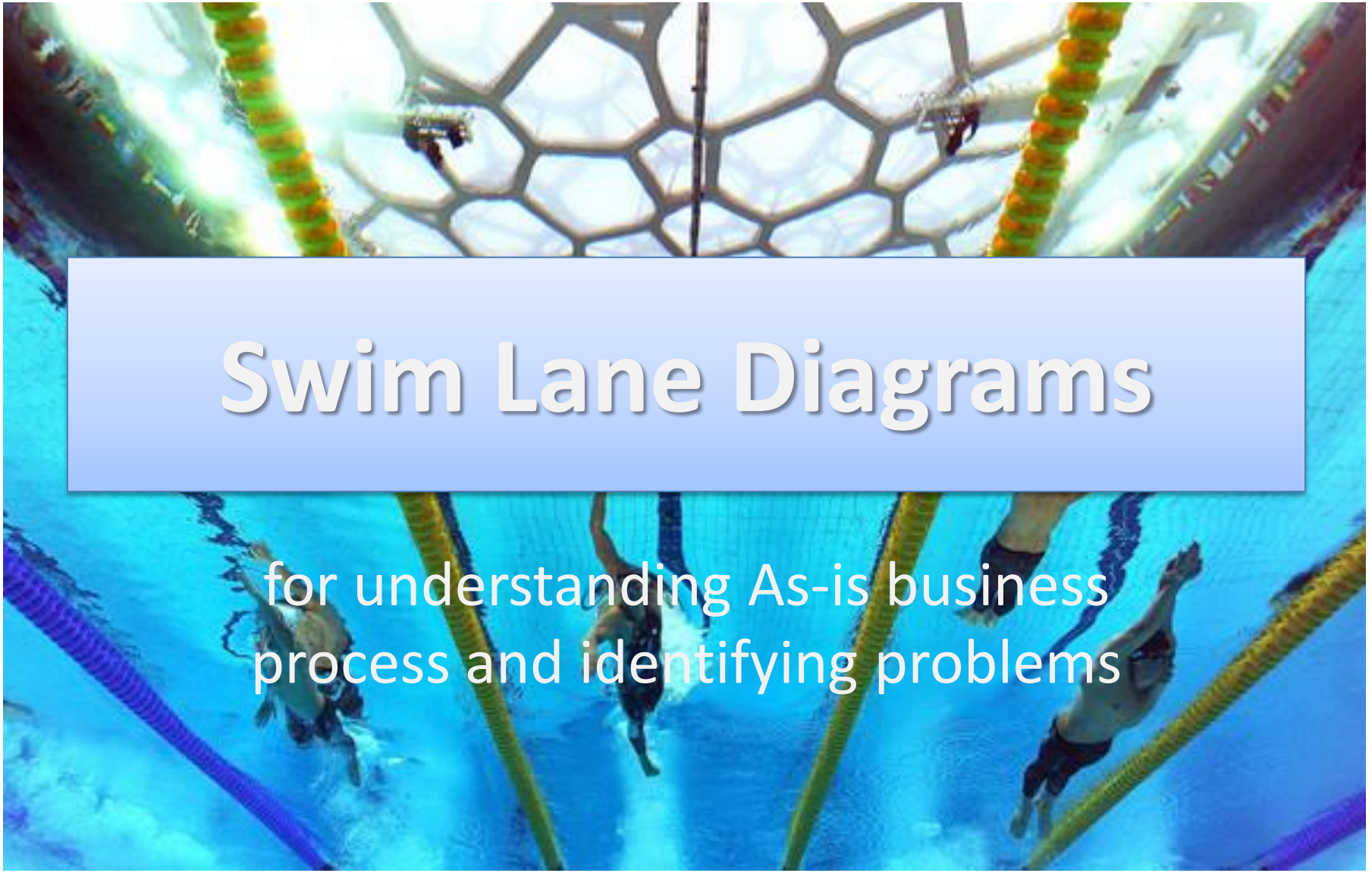
- We need to measure
  - Determines success!
- Deciding what to measure is critical
  - And brings you back to why BPM in the first place
- Measurement resources
  - StaceyBarr.com
  - KPILibrary.com

Summary

- Understanding the purpose of BPM
- Measuring BPM success
- Planning for Process Improvement

Atlantic Canada's Law Firm

New Brunswick Newfoundland & Labrador Nova Scotia Prince Edward Island mcinnescooper.com

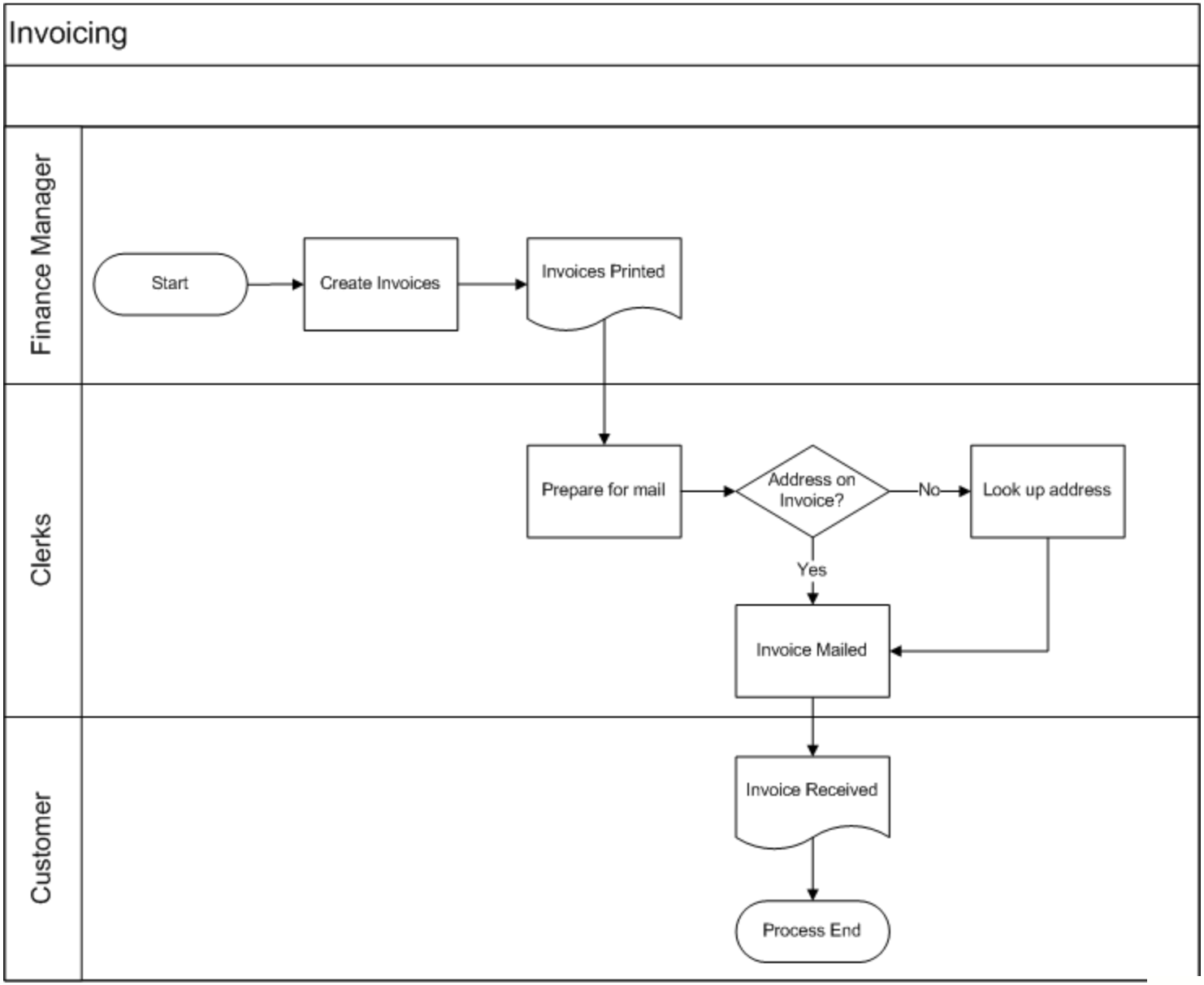


Swim Lane Diagrams

for understanding As-is business process and identifying problems

ANDREW MITCHELL
Senior Business Analyst
http://www.linkedin.com/in/andrewmitchellpmp

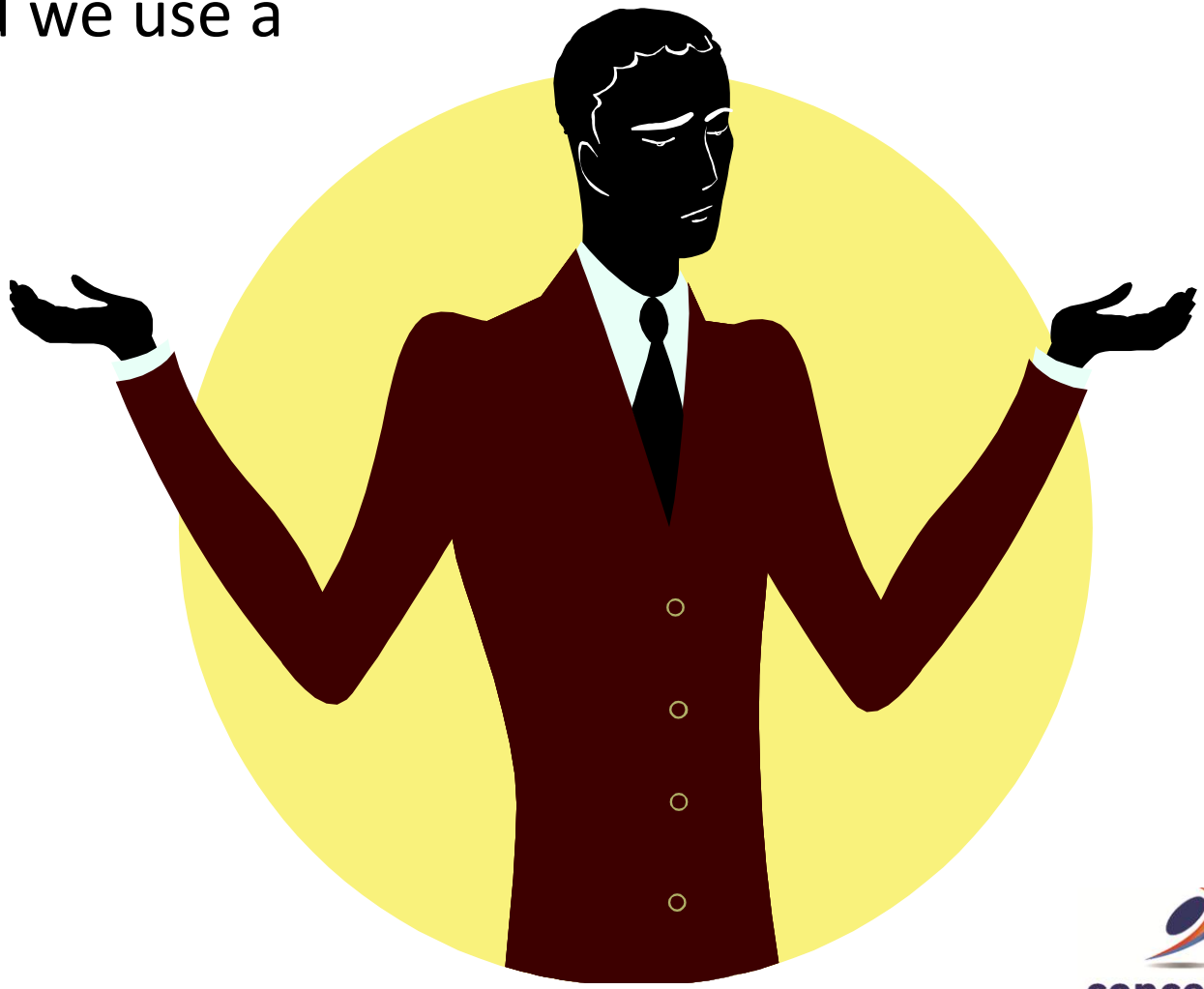
concertia®


Invoicing
Finance Manager
Start
Create Invoices
Invoices Printed
Clerks
Prepare for mail
Address on Invoice?
No
Look up address
Yes
Invoice Mailed
Customer
Invoice Received
Process End

concertia®

Swim Lane Diagrams

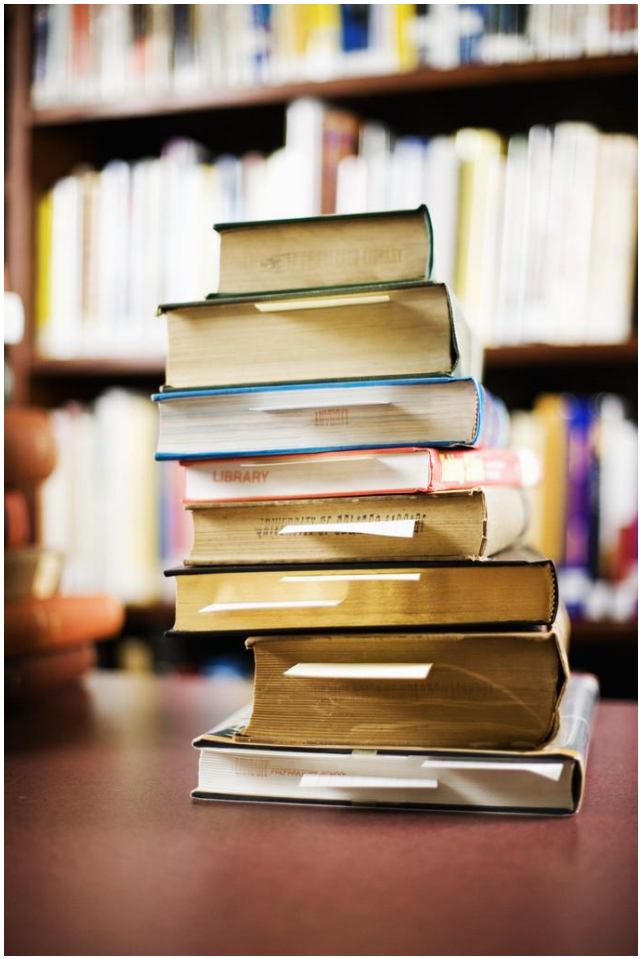
- Why would we use a Swim Lane Diagram?



concertia®

Swim Lane Diagrams – Why?

- Tells a story that is easily understood
  - Better then meeting minutes
  - Easy to create quickly



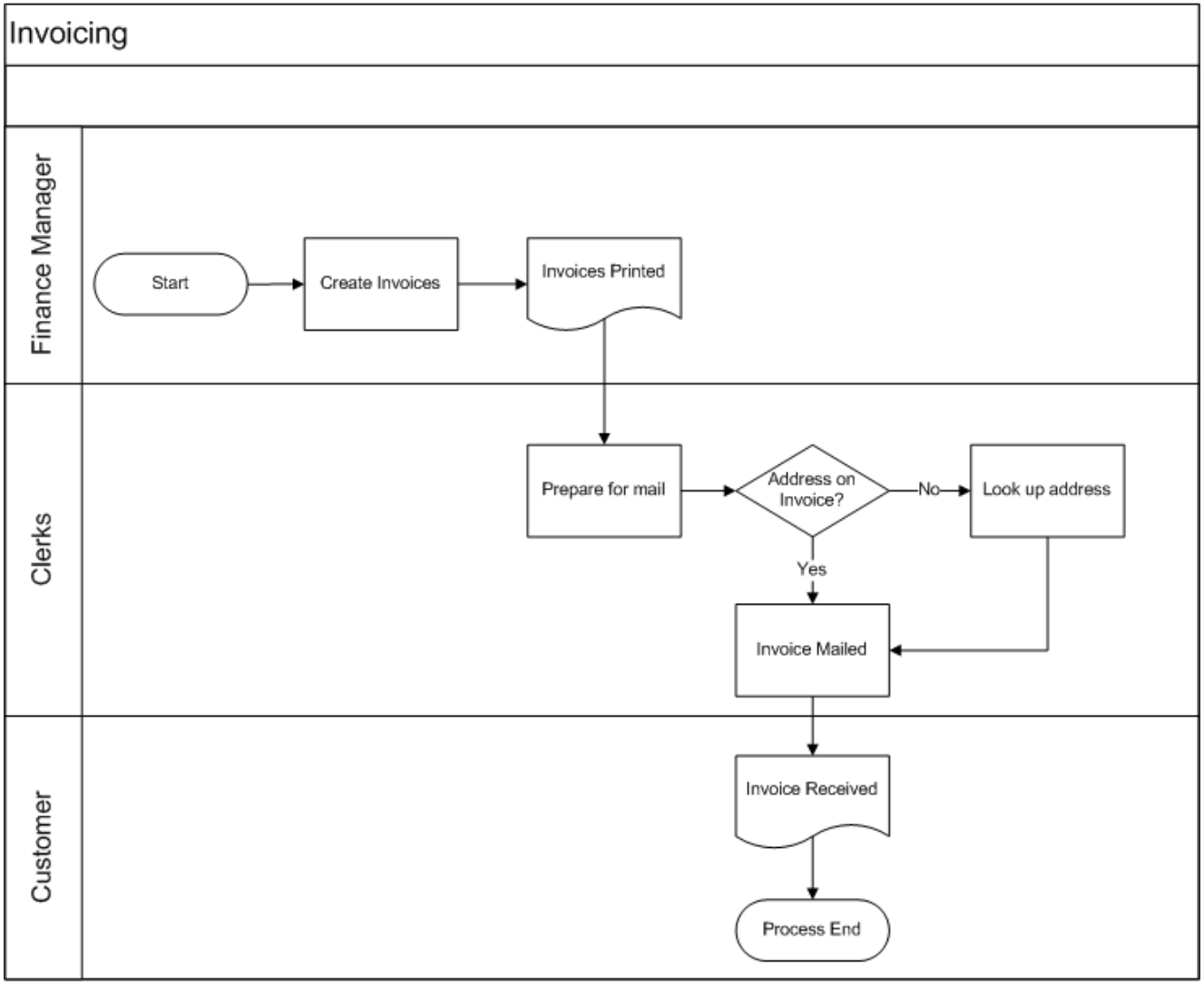
concertia®

Swim Lane Diagrams – Why?

- Helps you identify problems / areas for concern

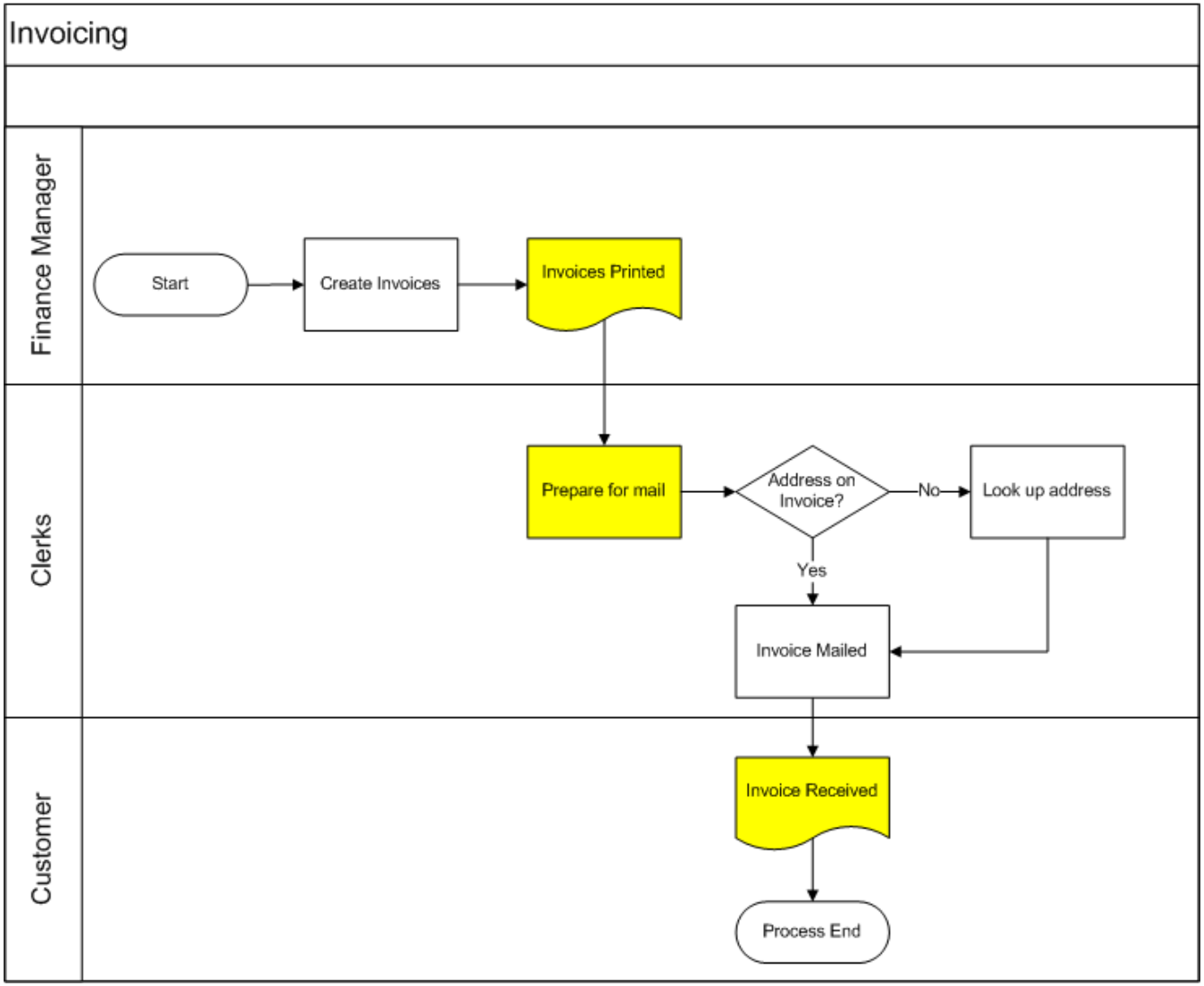
concertia®

Problem Identification



concertia®

Problem Identification

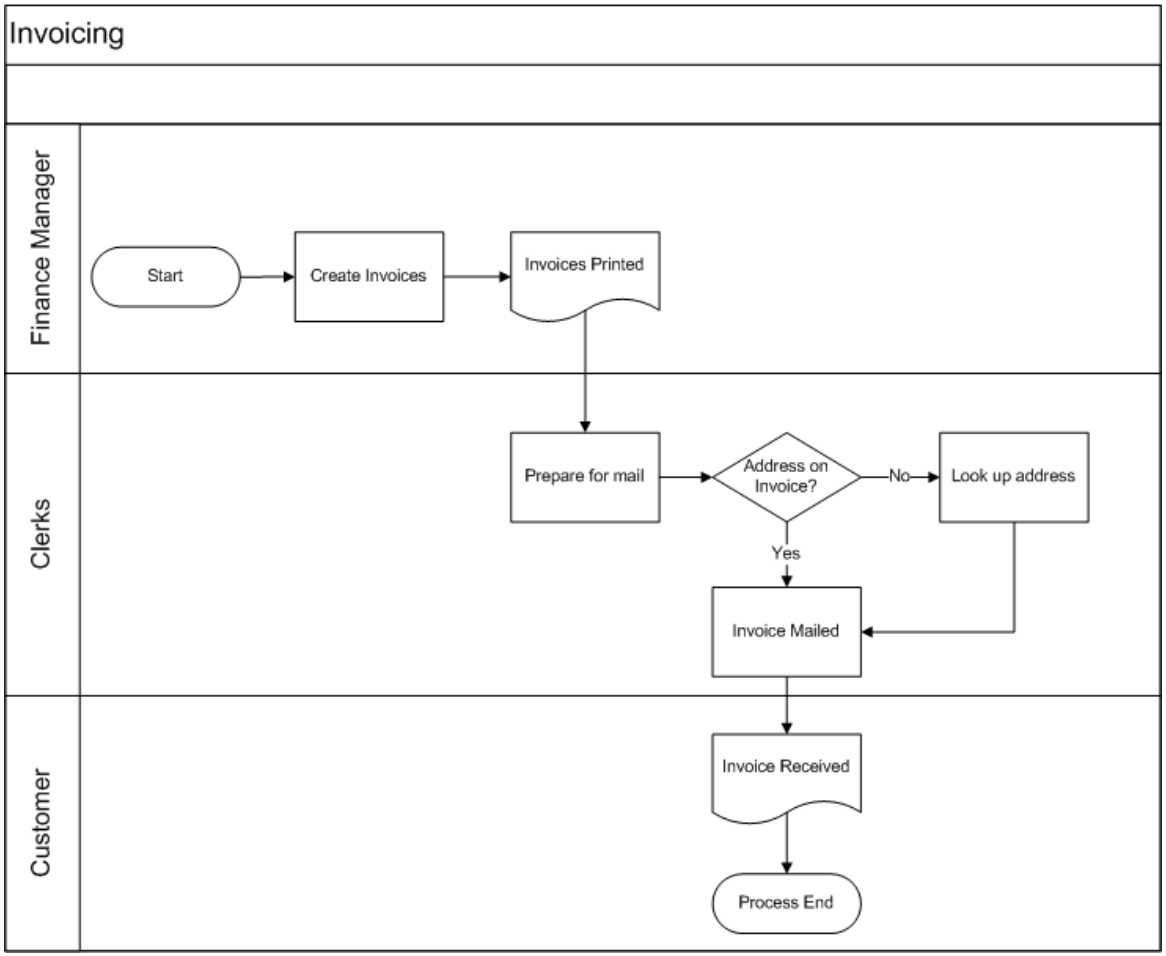


Swim Lane Diagrams – Why?

- Great communication tool for:
  - Business
  - Managers
  - Executives / Sponsors
  - Project Team / Developers

Communication Tool

- Getting agreement on the process
- Easily to change and review



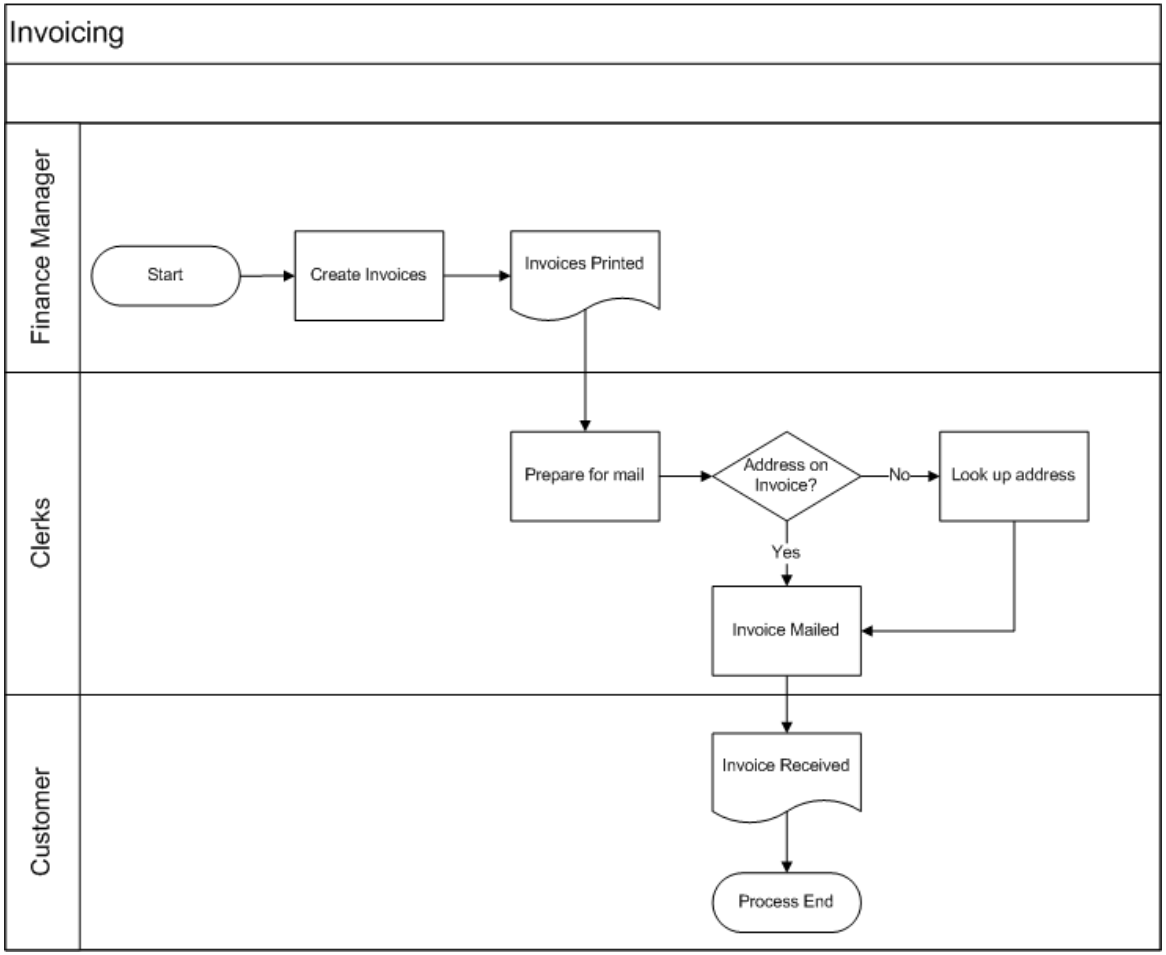
Swim Lane Diagrams – Why?

- Help you identify requirements
  - If you are replacing an automated process with a manual one.
  - If you are looking for a COTS Solution

concertia®

Requirements Identification

- The solution support this process
- Identifies key users and functions that must be supported



concertia®

Swim Lane Diagrams

- When would we use a Swim Lane Diagram?

concertia®

Swim Lane Diagrams – When?

- When you need to understand the process
  - Many players
  - Not already well understood
  - Not already documented or documentation outdated

concertia®

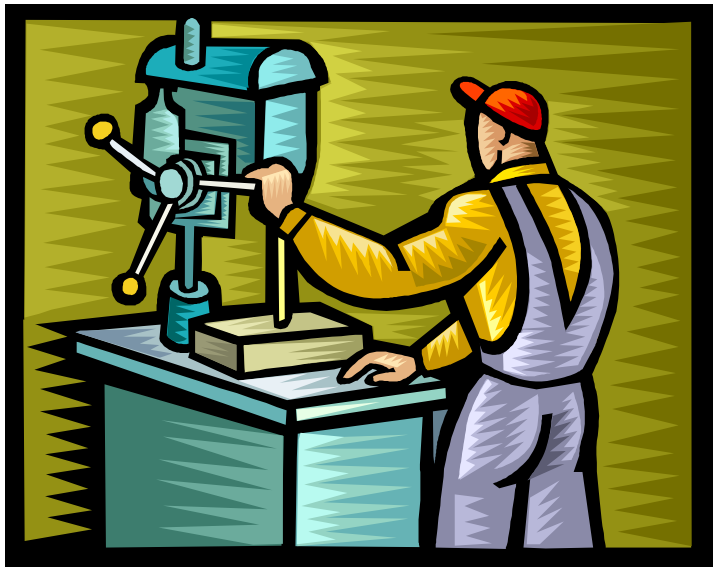
Swim Lane Diagrams – When?

- It is a great tools for use in a workshop

concertia®

Swim Lane Diagrams

- How to use them?



concertia®

Swim Lane Diagrams – How?

- Workshops
- Meetings with individuals
- Presentations to Executives / Sponsors
- Any communications about process and requirements.

concertia®

Swim Lane Diagrams

- Build them in your meetings
- Modify often
- Keep versions
- Get sign off

concertia®