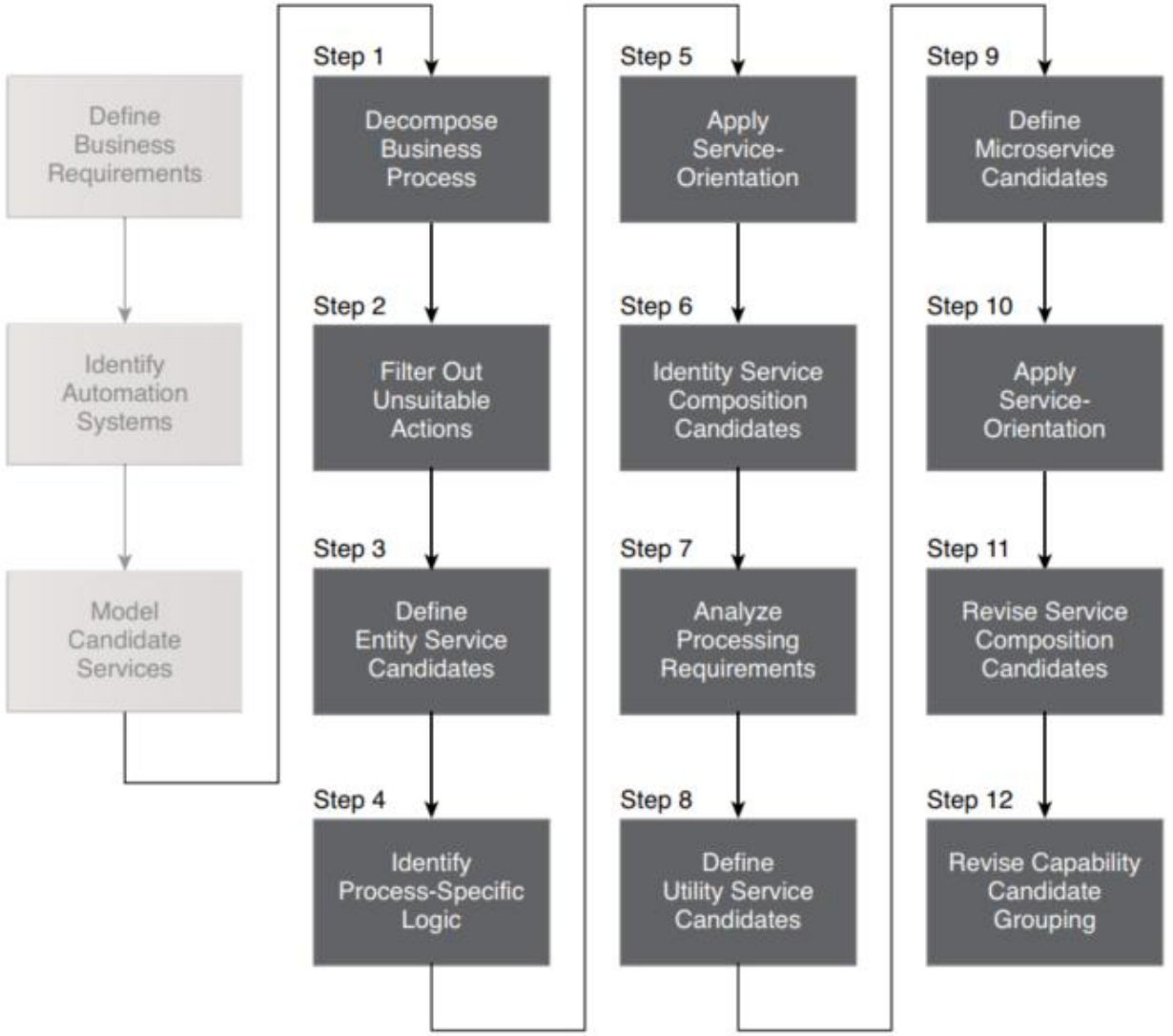
Analysis and Modeling with Web Services and Microservices

hungdn@ptit.edu.vn

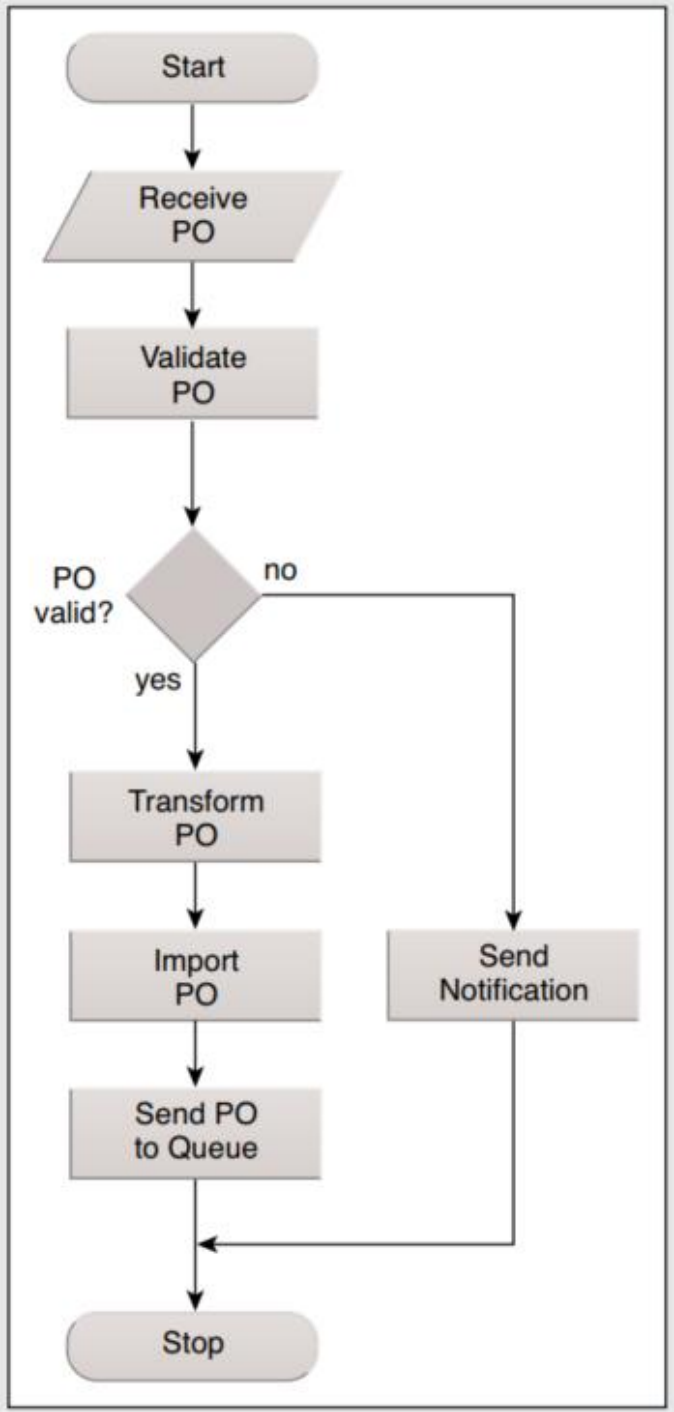
Web Service Modeling Process

- **Figure 6.1 outlines a generic process for Web service modeling**
  - This process can be **customized** for specific use cases
- **Service modeling organizes information from Steps 1 & 2 of service-oriented analysis**



Case study

- **TLS outsources employees for specialized maintenance jobs.**
- **Problem:**
  - Discrepancies in employee timesheets vs. billed customer invoices.
- **Current Process:**
  - A/R clerks manually enter hours from appointment schedules
- **Solution**
  - Integrate a third-party time tracking system with the **distributed accounting solution**
  - **Automated Verification Process** for timesheets



Step 1: Decompose the Business Process (into Granular Actions)

- **Break down the business process into granular actions**

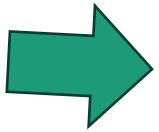
CASE STUDY EXAMPLE

Here is a breakdown of the current business process steps:

1. Receive Timesheet
2. Verify Timesheet
3. If Timesheet is Verified, Accept Timesheet Submission and End Process
4. Reject Timesheet Submission

1a. Receive Physical Timesheet Document

1b. Initiate Timesheet Submission



2a. Compare Hours Recorded on Timesheet to Hours Billed to Clients

2b. Confirm That Authorization Was Given for Any Recorded Overtime Hours

2c. Confirm That Hours Recorded for Any Particular Project Do Not Exceed a Pre-Defined Limit for That Project

2d. Confirm That Total Hours Recorded for One Week Do Not Exceed a Pre-Defined Maximum for That Worker

4a. Update the Worker's Profile Record to Keep Track of Rejected Timesheets

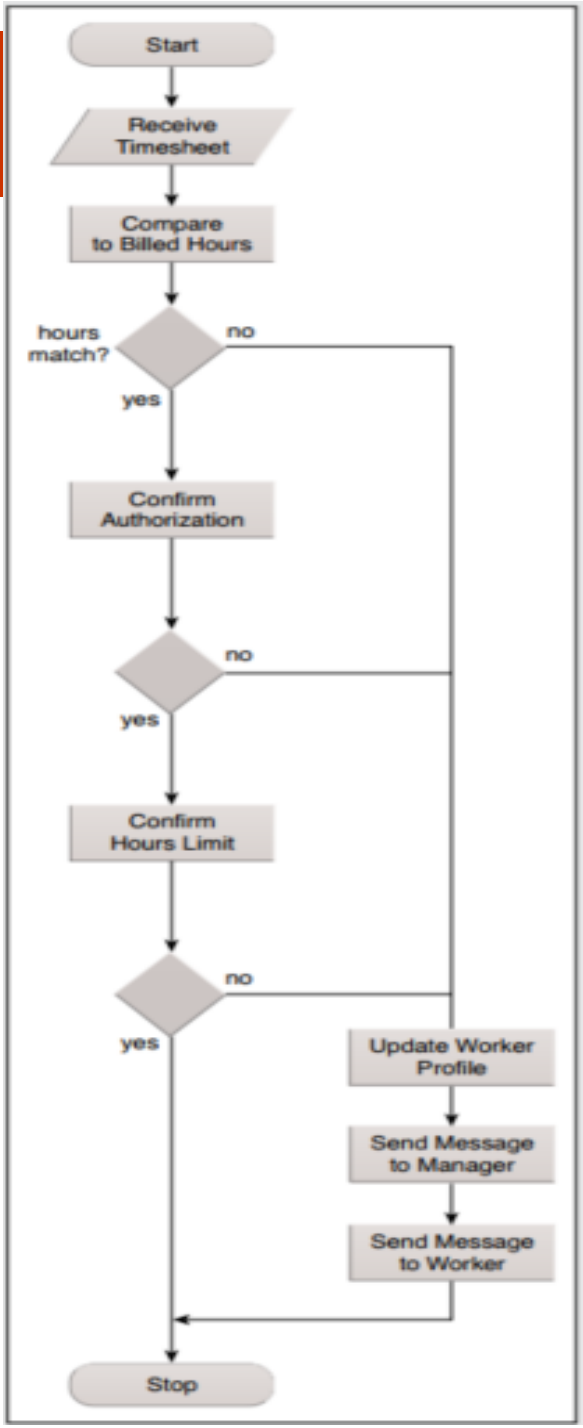
4b. Issue a Timesheet Rejection Notification Message to the Worker

4c. Issue a Notification to the Worker's Manager

Step 1: Decompose the Business Process (2)

Finally, TLS further simplifies the business process logic into the following set of granular actions:

- Receive Timesheet
- Initiate Timesheet Submission
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- Compare Recorded Hours with Billed Hours
- If Hours Do Not Match, Reject Timesheet Submission
- Get Overtime Hours for Date Range
- Get Authorization
- Confirm Authorization
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Get Weekly Hours Limit
- Compare Weekly Hours Limit with Recorded Hours
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- Update Employee History
- Send Message to Employee
- Send Message to Manager
- If Timesheet Is Verified, Accept Timesheet Submission and End Process



Step 2: Filter Out Unsuitable Actions

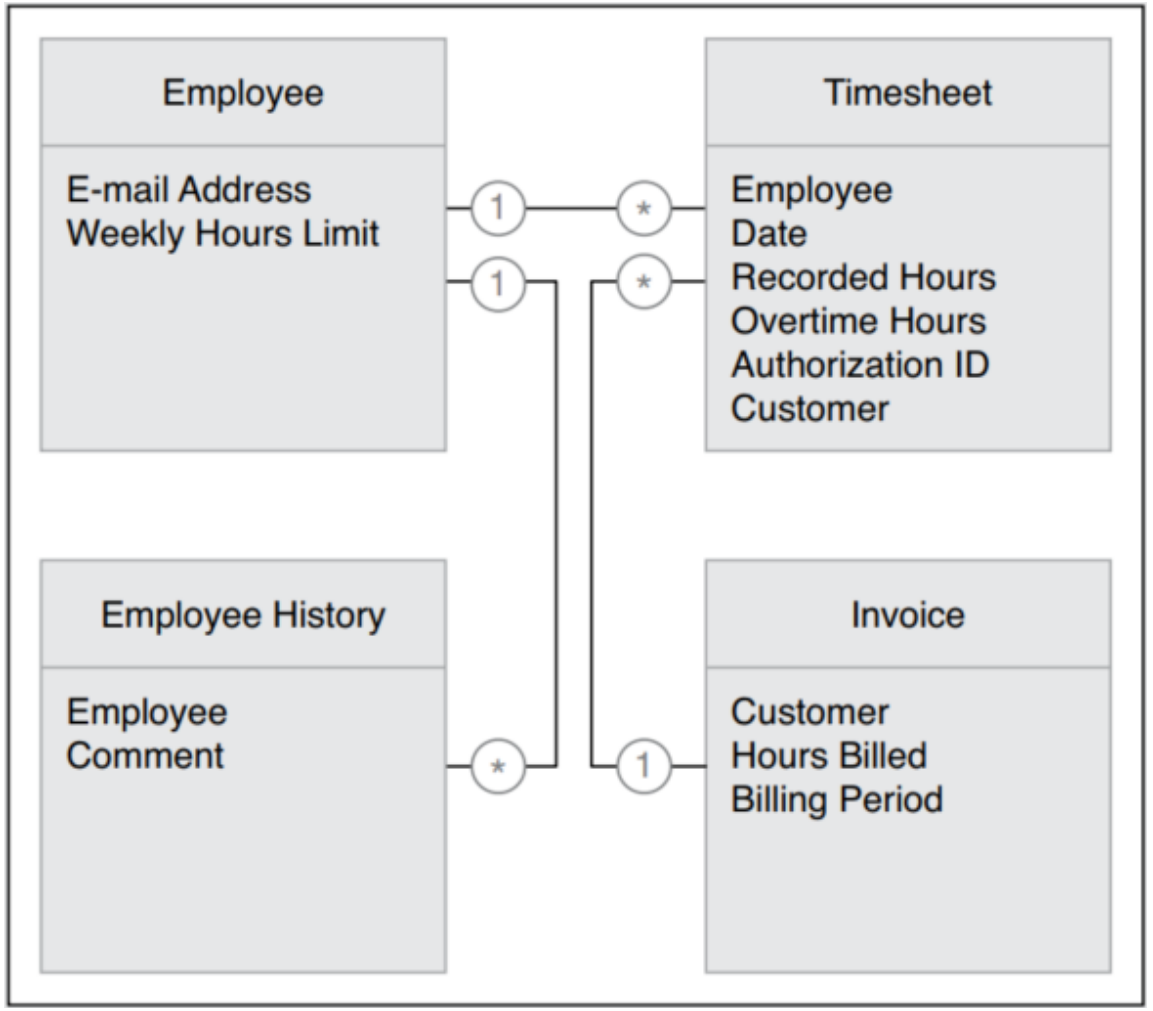
- **Remove actions**
  - Are manual and cannot be automated
  - Belong to existing legacy systems
  - Do not fit into the service-oriented model
- **Filtered Actions**
  - Retained actions → Suitable for automation and service encapsulation
  - A refined set of **service capability candidates**

- ~~Receive Timesheet~~
- Initiate Timesheet Submission
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- Compare Recorded Hours with Billed Hours
- If Hours Do Not Match, Reject Timesheet Submission
- Get Overtime Hours for Date Range
- Get Authorization
- Confirm Authorization
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Get Weekly Hours Limit
- Compare Weekly Hours Limit with Recorded Hours
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- Update Employee History
- Send Message to Employee
- Send Message to Manager
- If Timesheet Is Verified, Accept Timesheet Submission and End Process

Each of the remaining actions is considered a service capability candidate.

Step 3: Define Entity Service Candidates

- **Identify logical contexts for service grouping**
- **Entity Services**
  - Group steps by business entities
  - *Task services* → Process-specific grouping
- **Note**
  - Focus on defining **contexts**, not the number of steps.



Step 3: Define Entity Service Candidates (2)

- **Identify the service capability candidates considered agnostic**
- **All those classified as non-agnostic are bolded**

- **Initiate Timesheet Submission**
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- **Compare Recorded Hours with Billed Hours**
- **If Hours Do Not Match, Reject Timesheet Submission**
- Get Overtime Hours for Date Range
- Get Authorization
- **Confirm Authorization**
- **If Authorization Confirmation Fails, Reject Timesheet Submission**
- Get Weekly Hours Limit
- **Compare Weekly Hours Limit with Recorded Hours**
- **If Hours Recorded Confirmation Fails, Reject Timesheet Submission**
- Update Employee History
- Send Message to Employee
- Send Message to Manager
- **If Timesheet Is Verified, Accept Timesheet Submission and End Process**

Step 3: Define Entity Service Candidates (3)

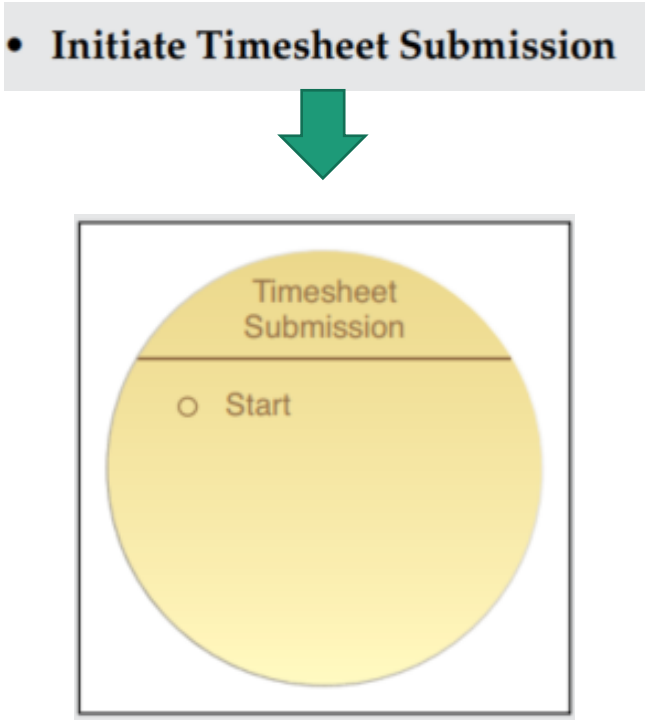
- **Timesheet**
  - Get Recorded Hours for Customer and Date Range
  - Get Overtime Hours for Date Range
  - Get Authorization

- **Invoice**
  - Get Billed Hours for Customer and Date Range

- **Employee**
  - Get Weekly Hours Limit
  - Update Employee History

Step 4: Identify Process-Specific Logic

- **After defining service contexts, remaining logic is classified as non-agnostic (specific to the business process)**
- **Common Non-Agnostic Components:**
  - **Business Rules:** Define policies and constraints.
  - **Conditional Logic:** Determines process flow based on conditions.
  - **Exception Handling:** Manages errors and special cases.
  - **Sequence Logic:** Controls execution order of process steps.



Step 4: Identify Process-Specific Logic (2)

- **Note**
  - Not all non-agnostic actions become **service capability candidates**
  - Some logic remains within service execution rather than becoming standalone services/service capability

CASE STUDY EXAMPLE

The following actions are considered non-agnostic because they are specific to the Timesheet Submission business process:

- Initiate Timesheet Submission
- **Compare Recorded Hours with Billed Hours**
- **If Hours Do Not Match, Reject Timesheet Submission**
- **Confirm Authorization**
- **If Authorization Confirmation Fails, Reject Timesheet Submission**
- **Compare Weekly Hours Limit with Recorded Hours**
- **If Hours Recorded Confirmation Fails, Reject Timesheet Submission**
- **If Timesheet Is Verified, Accept Timesheet Submission and End Process**

Figure 6.8
The Timesheet Submission service candidate with a single service capability that launches the automation of the Timesheet Submission business process.

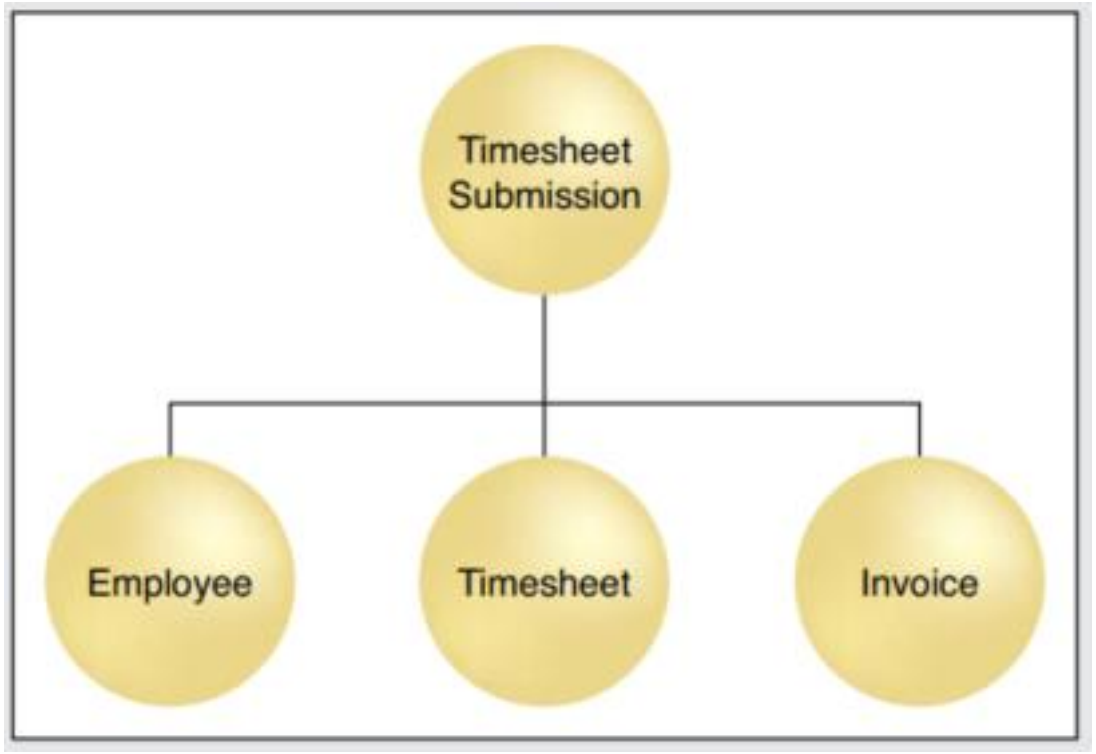
The Initiate Timesheet Submission action forms the basis of a service capability candidate, as explained in the upcoming Timesheet Submission task service candidate description. The remaining actions are bolded to indicate that they represent logic that is carried out within the Timesheet Submission task service, upon execution of the Initiate Timesheet Submission action, which is renamed to the Start service capability candidate (Figure 6.8).

Step 5: Applying Service-Orientation Principles

- Adjust and enhance **service candidates** based on key SOA principles
- Ensure services are well-structured and optimized for **reusability** and **scalability**
- Key SOA Principles to Apply
  - **Service Loose Coupling:** Minimize dependencies between services for flexibility.
  - **Service Abstraction:** Expose only necessary details to maintain security and simplicity.
  - **Service Autonomy:** Ensure services have independent control over their logic and execution.

Step 6: Identify Service Composition Candidates

- **Goals of Service Composition Analysis**
  - Validate service groupings and interactions
  - Define relationships between task & entity services
  - Identify service compositions for automation
  - Include exception handling scenarios



the service composition candidate hierarchy

Step 7: Analyze Processing Requirements

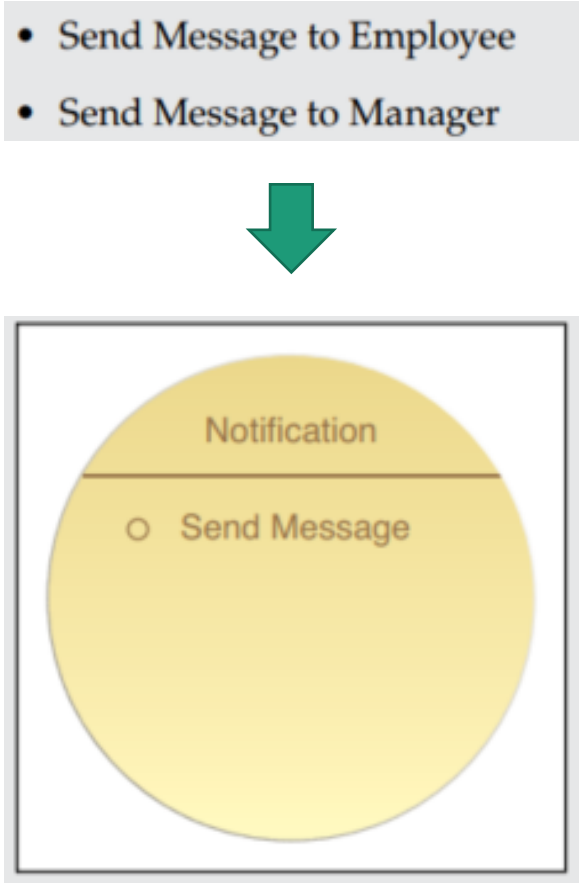
- **Objective**
  - Shift focus from **business logic** to **processing and implementation details**.
  - Identify **technology-centric service logic** that may lead to **microservices** or **utility services**
- **How**
  - External Dependencies
  - Performance & Reliability
  - Implementation & Environmental Constraints

CASE STUDY EXAMPLE

- **Utility Service Identification**
  - The **"Send Message to Employee"** and **"Send Message to Manager"** actions require encapsulation into a **utility service layer**.
- **Refining Task Services**
  - The **"Confirm Authorization"** action interacts with a **external system** that has strict **SLA requirements** for **performance and failover**
  - Keeping it within the **Timesheet Submission task service** may risk **performance issues**
- **Solution**
  - The **"Confirm Authorization"** logic should be separated into a **microservice**

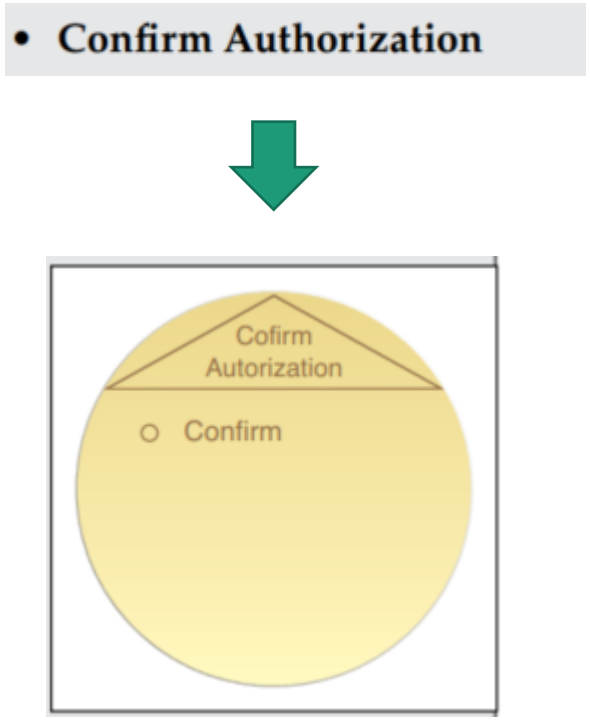
Step 8: Define Utility Service Candidates

- **Break down agnostic processing logic into granular actions**
  - Ensure each action is **explicitly labeled** based on **functionality**, not tied to a specific business process step
  - -> Group these actions into **logical utility service candidates**
- **Grouping Criteria**
  - Legacy system association
  - Solution component relationship
  - Logical function categorization



Step 9: Define Microservice Candidates

- **Evaluate non-agnostic processing logic for potential microservice encapsulation**
- **Ensure autonomy, performance, and reliability for critical business functions**
- **Key Considerations for Microservices**
  - Autonomy
  - Performance Requirements
  - Reliability & Failover
  - Versioning & Deployment Needs

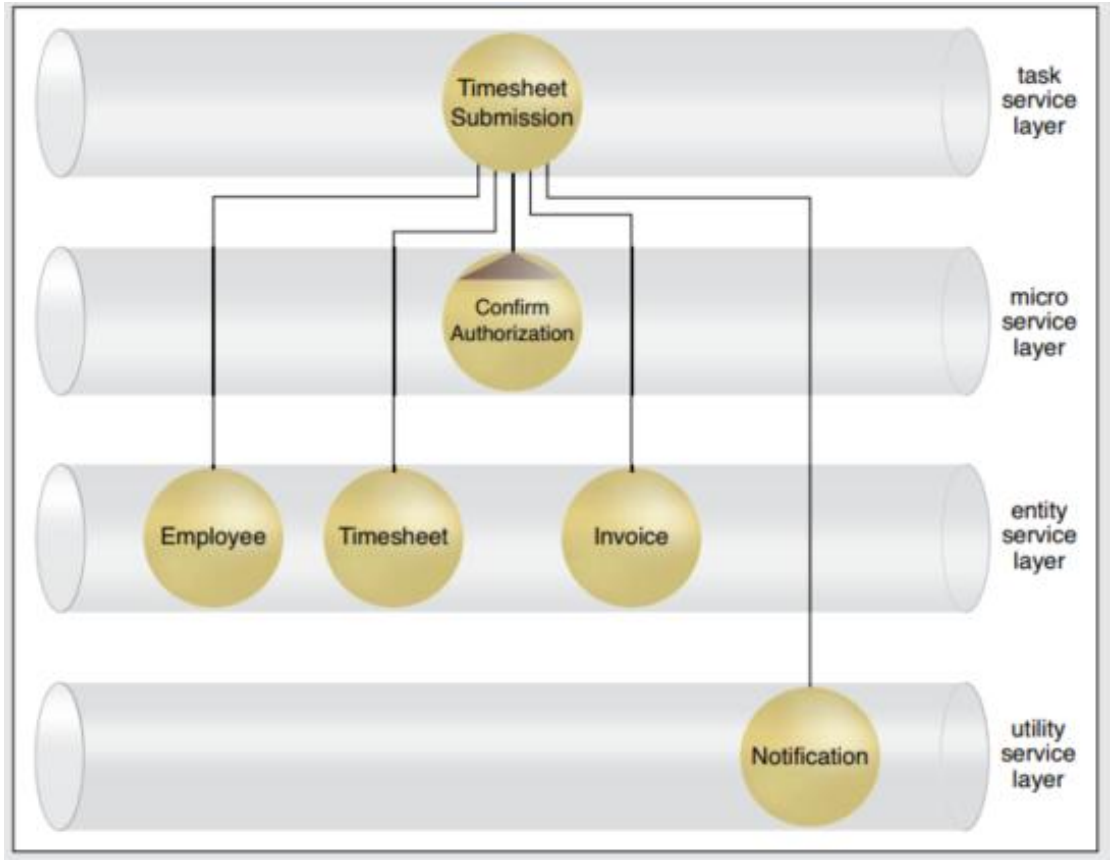


Step 10: Apply Service-Orientation

- This step is a repeat of Step 7, provided here specifically for any new utility service candidates that may have emerged from the completion of Steps 8 and 9

Step 11: Revise Service Composition Candidates

- **Revise the previously identified service compositions (from Step 6)**
  - Integrate newly defined utility services and capabilities into process flows.
  - Expand service compositions by mapping business & utility services



Step 12: Revise Capability Candidate Grouping

- Adjust service capability groupings based on artifacts from Step 11
  - Identify missing processing steps
  - Adjust or introduce new service candidates
- Prepare for next Iterations

This process description assumes that this is the first iteration through the service modeling process

Q & A