Software Requirements Analysis and Specification

Background

- Problem of scale is a key issue for SE
- For small scale, understand and specifying requirements is easy
- For large problem - very hard; probably the hardest, most problematic and error prone
- <u>Input</u> : user needs in minds of people
- <u>Output</u> : precise statement of what the future system will do

Background..

- Identifying and specifying req necessarily involves people interaction
- Cannot be automated
- Requirement (IEEE)= A condition or capability that must be possessed by a system
- Req. phase ends with a software requirements specification (SRS) document
- SRS specifies what the proposed system should do

Background..

- Requirements understanding is hard
  - Visualizing a future system is difficult
  - Capability of the future system not clear, hence needs not clear
  - Requirements change with time
  - …
- Essential to do a proper analysis and specification of requirements

Need for SRS

- SRS establishes basis of agreement between the user and the supplier.
  - Users needs have to be satisfied, but user may not understand software
  - Developers will develop the system, but may not know about problem domain
  - SRS is the medium to bridge the commn. gap and specify user needs in a manner both can understand

Need for SRS…

- Helps user understand his needs.
  - users do not always know their needs
  - must analyze and understand the potential
  - the goal is not just to automate a manual system, but also to add value through IT
  - The req process helps clarify needs
- SRS provides a reference for validation of the final product
  - Clear understanding about what is expected.
  - Validation - " SW satisfies the SRS "

Need for SRS…

- High quality SRS essential for high Quality SW
  - Requirement errors get manifested in final sw
  - to satisfy the quality objective, must begin with high quality SRS
  - Requirements defects are not few
    - 25% of all defects in one case; 54% of all defects found after UT
    - 80 defects in A7 that resulted in change requests
    - 500 / 250 defects in previously approved SRS.

Need for SRS…

- Good SRS reduces the development cost
  - SRS errors are expensive to fix later
  - Req. changes can cost a lot (up to 40%)
  - Good SRS can minimize changes and errors
  - Substantial savings; extra effort spent during req. saves multiple times that effort
- An Example
  - Cost of fixing errors in req. , design , coding , acceptance testing and operation are 2 , 5 , 15 , 50 , 150 person-months

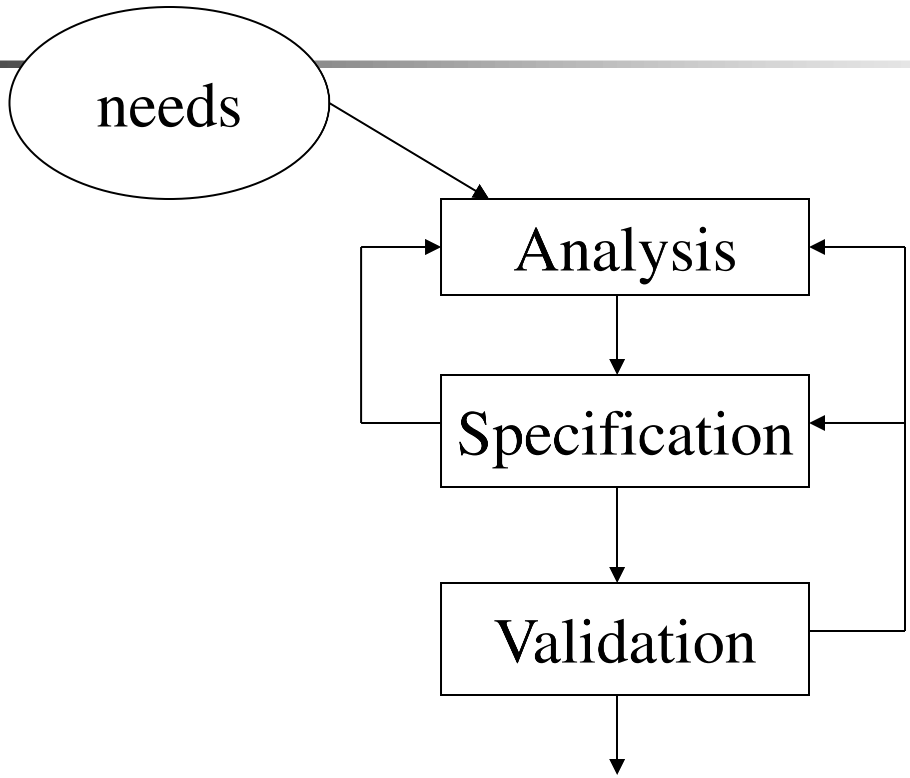
Need for SRS…

- Example …
  - After req. phase 65% req errs detected in design , 2% in coding, 30% in Acceptance testing, 3% during operation
  - If 50 requirement errors are not removed in the req. phase, the total cost
    32.5 *5 + 1*15 + 15*50 + 1.5*150 = 1152 hrs
  - If 100 person-hours invested additionally in req to catch these 50 defects , then development cost could be reduced by 1152 person-hours.
  - Net reduction in cost is 1052 person-hours

Requirements Process

- Sequence of steps that need to be performed to convert user needs into SRS
- Process has to elicit needs and requirements and clearly specifies it
- Basic activities
  - problem or requirement analysis
  - requirement specification
  - validation
- Analysis involves elicitation and is the hardest

Requirements Process..



Requirement process..

- Process is not linear, it is iterative and parallel
- Overlap between phases - some parts may be analyzed and specified
- Specification itself may help analysis
- Validation can show gaps that can lead to further analysis and spec

Requirements Process…

- Focus of analysis is on **understanding** the desired systems and it's requirements
- Divide and conquer is the basic strategy
  - decompose into small parts, understand each part and relation between parts
- Large volumes of information is generated
  - organizing them is a key
- Techniques like data flow diagrams, object diagrams etc. used in the analysis

Requirements Process..

- Transition from analysis to specs is hard
  - in specs, **external behavior** specified
  - during analysis, structure and domain are understood
  - analysis structures helps in specification, but the transition is not final
  - methods of analysis are similar to that of design, but objective and scope different
  - analysis deals with the problem domain, whereas design deals with solution domain

Problem Analysis

- Aim: to gain an understanding of the needs, requirements, and constraints on the software
- Analysis involves
  - interviewing client and users
  - reading manuals
  - studying current systems
  - helping client/users understand new possibilities
  - Like becoming a consultant
- Must understand the working of the organization , client and users

Problem Analysis…

- Some issues
  - Obtaining the necessary information
  - Brainstorming: interacting with clients to establish desired properties
  - Information organization, as large amount of info. gets collected
  - Ensuring completeness
  - Ensuring consistency
  - Avoiding internal design

Problem Analysis…

- Interpersonal issues are important
- Communication skills are very important
- Basic principle: problem partition
- Partition w.r.t what?
  - Object - OO analysis
  - Function - structural analysis
  - Events in the system – event partitioning
- Projection - get different views
- Will discuss few different analysis techniques

Informal Approach to Analysis

- No defined methodology; info obtained through analysis, observation, interaction, discussions,…
- No formal model of the system built
- Obtained info organized in the SRS; SRS reviewed with clients
- Relies on analyst experience and feedback from clients in reviews
- Useful in many contexts

Data Flow Modeling

- Widely used; focuses on functions performed in the system
- Views a system as a network of data transforms through which the data flows
- Uses data flow diagrams (DFDs) and functional decomposition in modeling
- The SSAD methodology uses DFD to organize information, and guide analysis

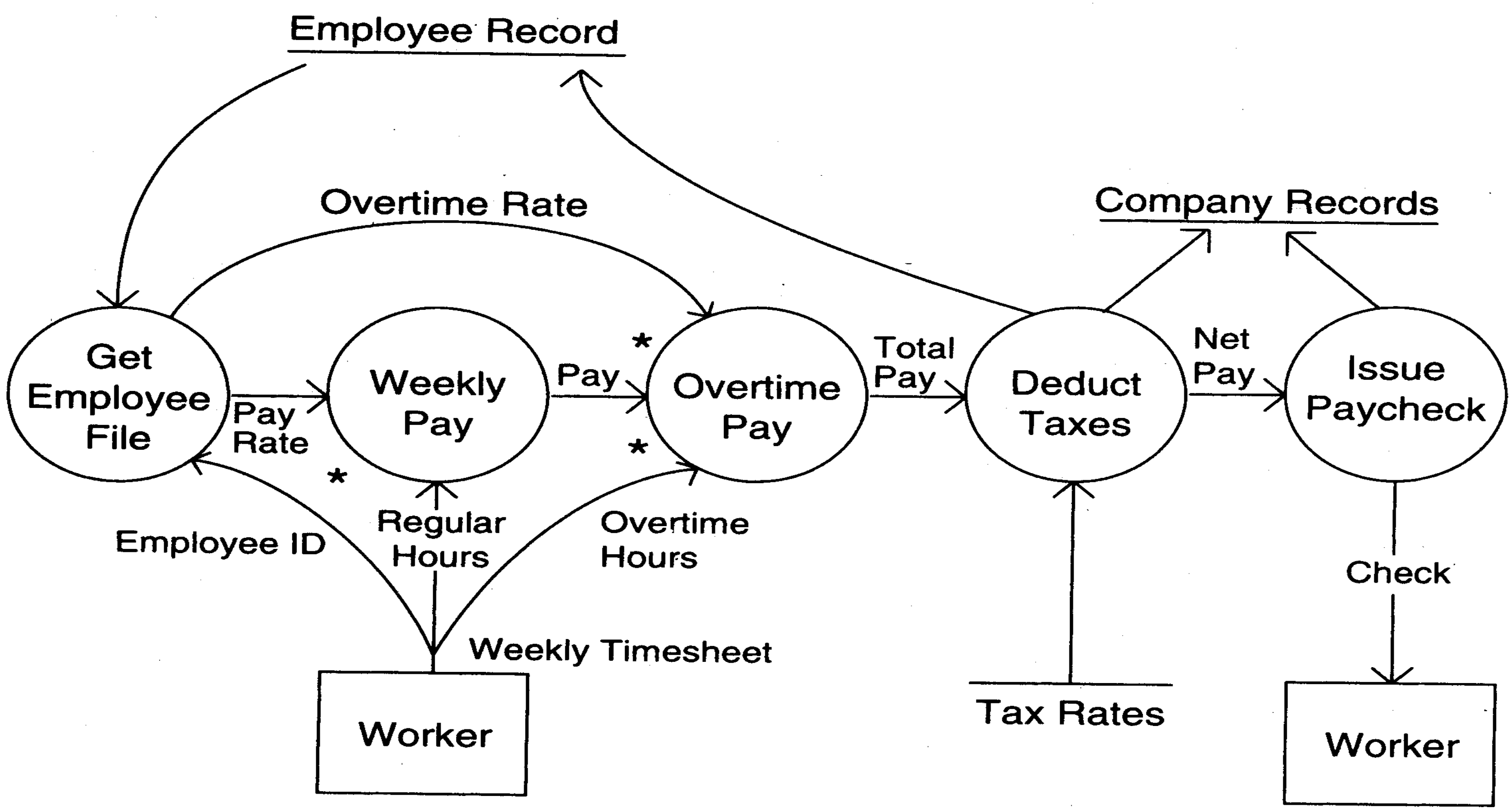
Data flow diagrams

- A DFD shows flow of data through the system
  - Views system as transforming inputs to outputs
  - Transformation done through transforms
  - DFD captures how transformation occurs from input to output as data moves through the transforms
  - Not limited to software

Data flow diagrams…

- DFD
  - Transforms represented by named circles/bubbles
  - Bubbles connected by arrows on which named data travels
  - A rectangle represents a source or sink and is originator/consumer of data (often outside the system)

DFD Example



Employee Record
Overtime Rate
Company Records
Get Employee File
Pay Rate
Weekly Pay
Pay
*
Overtime Pay
Total Pay
Deduct Taxes
Net Pay
Issue Paycheck
*
*
Employee ID
Regular Hours
Overtime Hours
Check
Weekly Timesheet
Worker
Tax Rates
Worker

DFD Conventions

- External files shown as labeled straight lines
- Need for multiple data flows by a process represented by * (means and)
- OR relationship represented by +
- All processes and arrows should be named
- Processes should represent transforms, arrows should represent some data

Data flow diagrams…

- Focus on what transforms happen , how they are done is not important
- Usually major inputs/outputs shown, minor are ignored in this modeling
- No loops , conditional thinking , …
- DFD is NOT a control chart, no algorithmic design/thinking
- Sink/Source , external files

Drawing a DFD

- If get stuck , reverse direction
- If control logic comes in , stop and restart
- Label each arrows and bubbles
- Make use of + & *
- Try drawing alternate DFDs

<u>Leveled DFDs :</u>

- DFD of a system may be very large
- Can organize it hierarchically
- Start with a top level DFD with a few bubbles
- then draw DFD for each bubble
- Preserve I/O when " exploding"

Drawing a DFD for a system

- Identify inputs, outputs, sources, sinks for the system
- Work your way consistently from inputs to outputs, and identify a few high-level transforms to capture full transformation
- If get stuck, reverse direction
- When high-level transforms defined, then refine each transform with more detailed transformations

Drawing a DFD for a system..

- Never show control logic; if thinking in terms of loops/decisions, stop & restart
- Label each arrows and bubbles; carefully identify inputs and outputs of each transform
- Make use of + & *
- Try drawing alternate DFDs

Leveled DFDs

- DFD of a system may be very large
- Can organize it hierarchically
- Start with a top level DFD with a few bubbles
- then draw DFD for each bubble
- Preserve I/O when “ exploding” a bubble so consistency preserved
- Makes drawing the leveled DFD a top-down refinement process, and allows modeling of large and complex systems

Data Dictionary

- In a DFD arrows are labeled with data items
- Data dictionary defines data flows in a DFD
- Shows structure of data; structure becomes more visible when exploding
- Can use regular expressions to express the structure of data

Data Dictionary Example

- For the timesheet DFD

Weekly_timesheet – employee_name + id + [regular_hrs + overtime_hrs]*

Pay_rate = [hourly | daily | weekly] + dollar_amt

Employee_name = last + first + middle

Id = digit + digit + digit + digit

DFD drawing – common errors

- Unlabeled data flows
- Missing data flows
- Extraneous data flows
- Consistency not maintained during refinement
- Missing processes
- Too detailed or too abstract
- Contains some control information

Structured Analysis Method

- Structured system analysis and design (SSAD) – we will focus only on analysis
- Was used a lot when automating existing manual systems
- Main steps
  - Draw a context diagram
  - Draw DFD of the existing system
  - Draw DFD of the proposed system and identify the man-machine boundary

Context Diagram

- Views the entire system as a transform and identifies the context
- Is a DFD with one transform (system), with all inputs, outputs, sources, sinks for the system identified

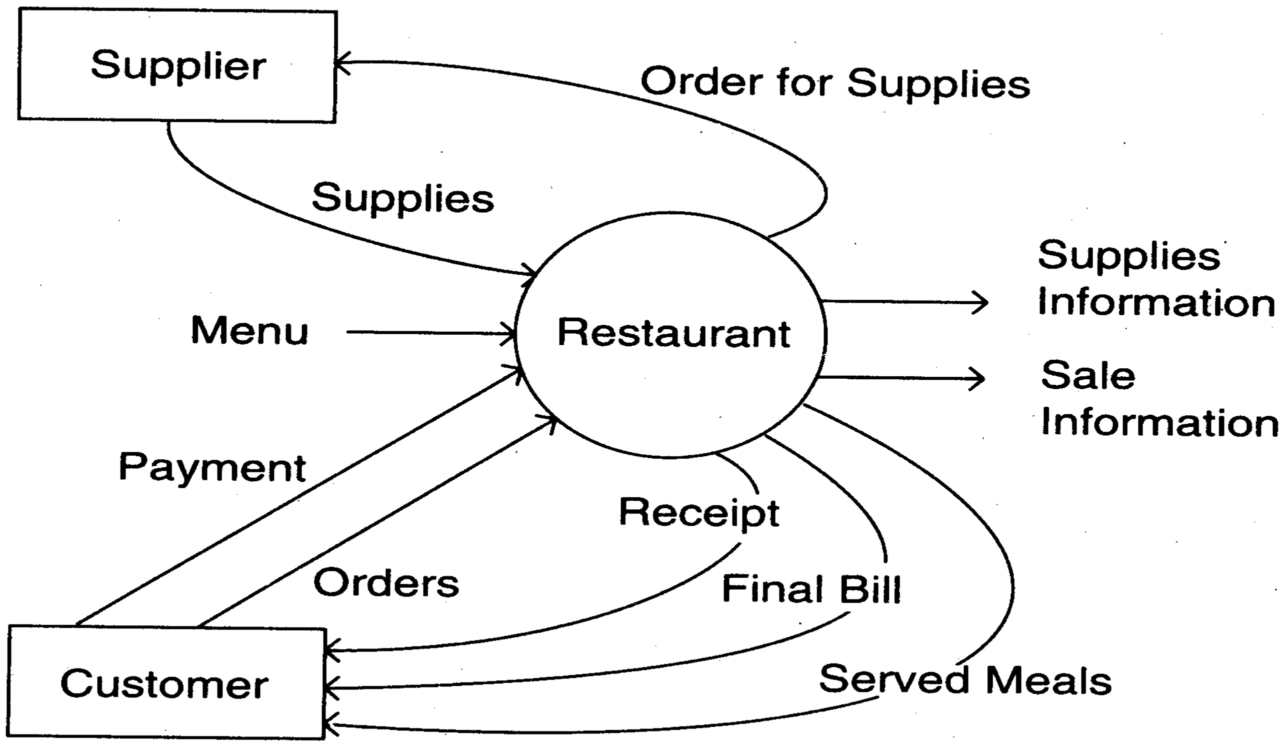
DFD of the current system

- The current system is modeled as-is as a DFD to understand the working
- The context diagram is refined
- Each bubble represents a logical transformation of some data
- Leveled DFD may be used
- Generally obtained after understanding and interaction with users
- Validate the DFD by walking through with users

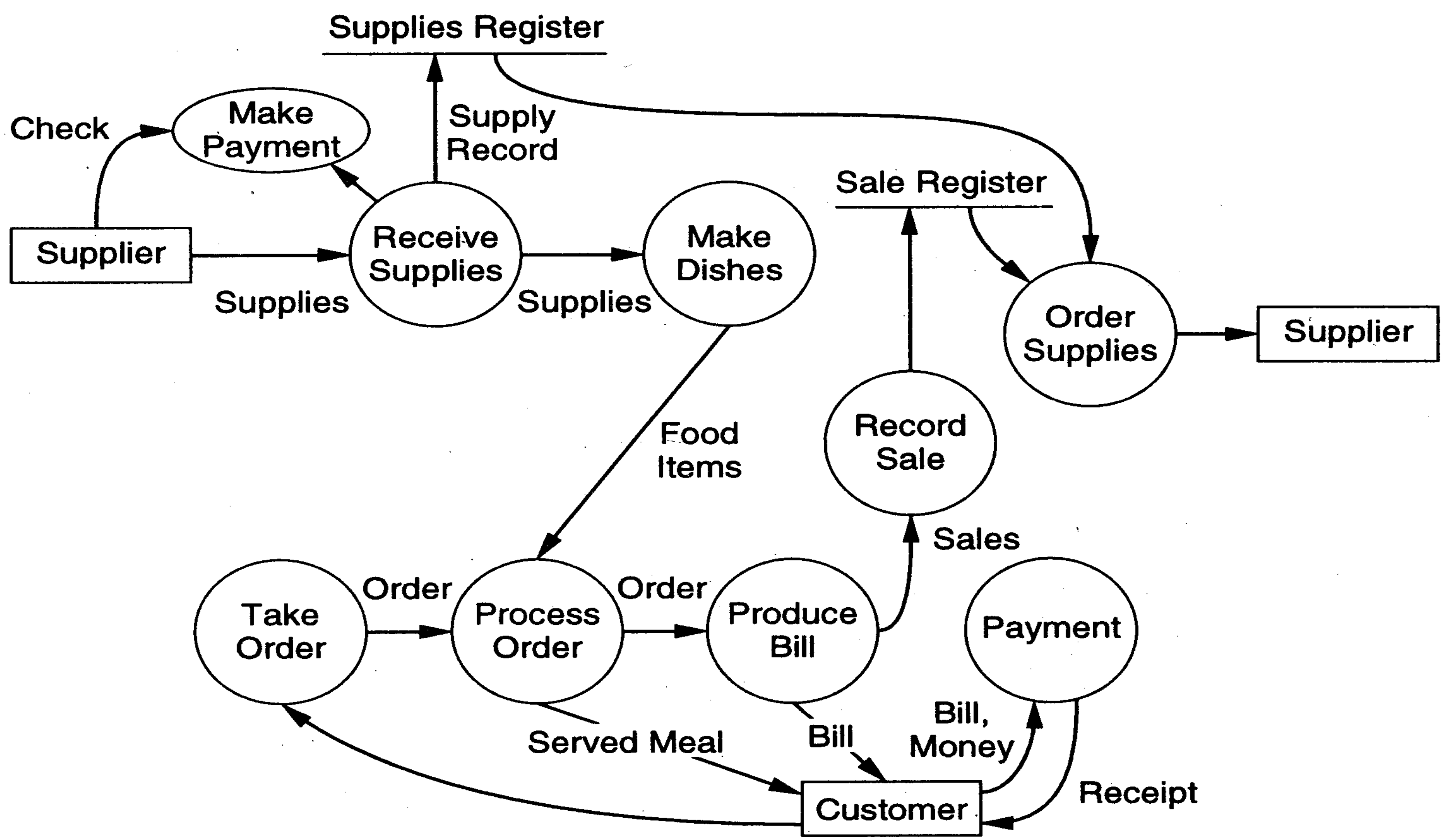
Modeling the Proposed System

- No general rules for drawing the DFD of the future system
- Use existing system understanding
- DFD should model the entire proposed system - process may be automated or manual
- validate with the user
- Then establish man-machine boundary
  - what processes will be automated and which remains manual
- Show clearly interaction between automated and manual processes

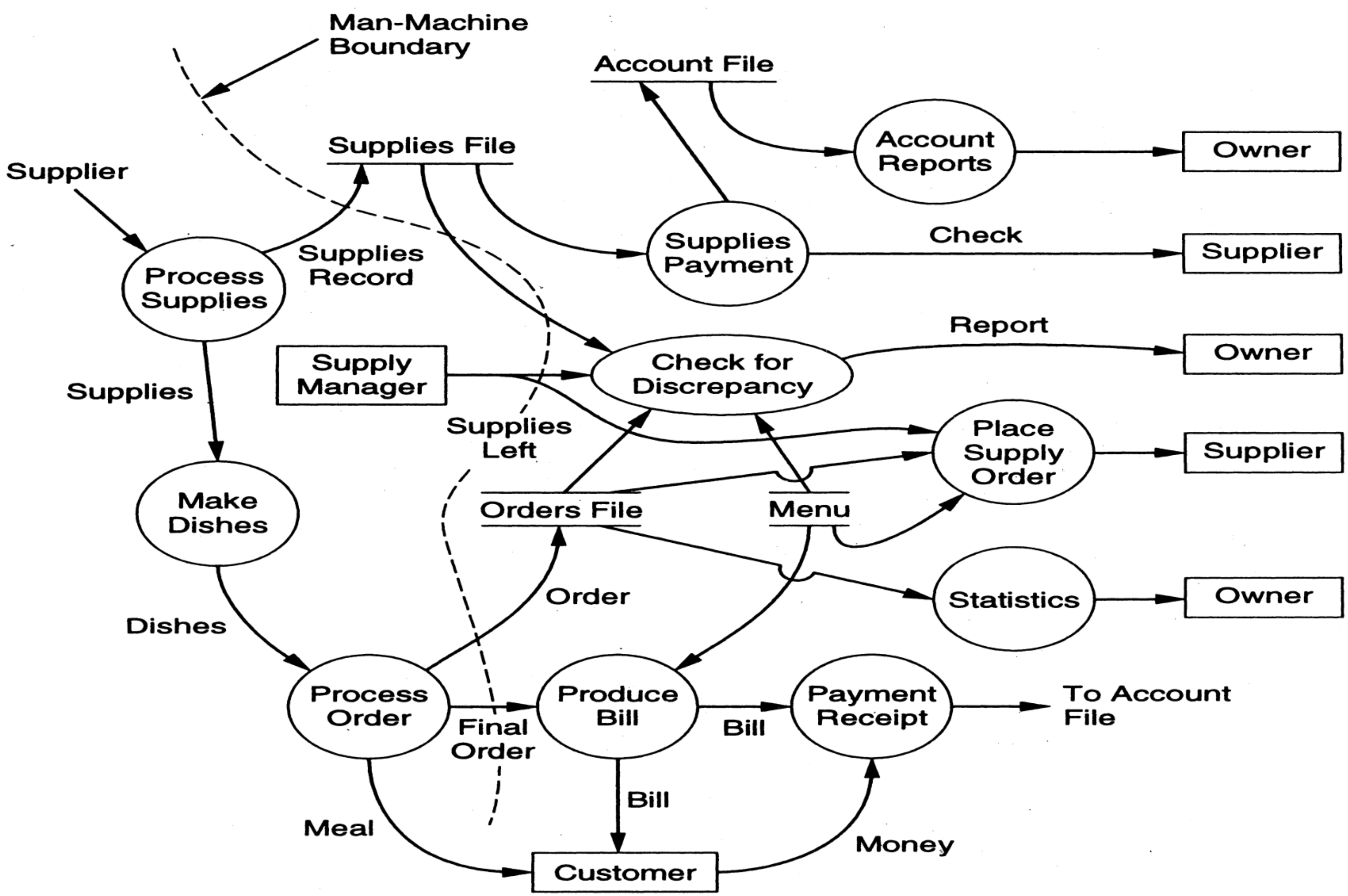
Example – context diagram



Example – DFD of existing sys



Example – DFD of proposed system



Other Approaches to RA

- Prototyping
  - Evolutionary
  - Throw-away
- Object Oriented
  - Classes, attributes, methods
  - Association between classes
  - Class hierarchies
  - The OOD approach is applied, except to the problem domain

Requirements Specification

- Final output of requirements task is the SRS
- Why are DFDs, OO models, etc not SRS ?
  - SRS focuses on external behavior, while modeling focuses on problem structure
  - UI etc. not modeled, but have to be in SRS
  - Error handling, constraints etc. also needed in SRS
- Transition from analysis to specification is not straight forward
- knowledge about the system acquired in analysis used in specification

Characteristics of an SRS

- Correct
- Complete
- Unambiguous
- Consistent
- Verifiable
- Traceable
- Modifiable
- Ranked for importance and/or stability

Characteristics…

- Correctness
  - Each requirement accurately represents some desired feature in the final system
- Completeness
  - All desired features/characteristics specified
  - Hardest to satisfy
  - Completeness and correctness strongly related
- Unambiguous
  - Each req has exactly one meaning
  - Without this errors will creep in
  - Important as natural languages often used

Characteristics…

- Verifiability
  - There must exist a cost effective way of checking if sw satisfies requirements
- Consistent
  - two requirements don’t contradict each other
- Traceable
  - The origin of the req, and how the req relates to software elements can be determined
- Ranked for importance/stability
  - Needed for prioritizing in construction
  - To reduce risks due to changing requirements

Components of an SRS

- What should an SRS contain ?
  - Clarifying this will help ensure completeness
- An SRS must specify requirements on
  - Functionality
  - Performance
  - Design constraints
  - External interfaces

Functional Requirements

- Heart of the SRS document; this forms the bulk of the specs
- Specifies all the functionality that the system should support
- Outputs for the given inputs and the relationship between them
- All operations the system is to do
- Must specify behavior for invalid inputs too

Performance Requirements

- All the performance constraints on the software system
- Generally on response time , throughput etc => dynamic
- Capacity requirements => static
- Must be in measurable terms (verifiability)
  - Eg resp time should be xx 90% of the time

Design Constraints

- Factors in the client environment that restrict the choices
- Some such restrictions
  - Standard compliance and compatibility with other systems
  - Hardware Limitations
  - Reliability, fault tolerance, backup req.
  - Security

External Interface

- All interactions of the software with people, hardware, and sw
- User interface most important
- General requirements of “friendliness” should be avoided
- These should also be verifiable

Specification Language

- Language should support desired char of the SRS
- Formal languages are precise and unambiguous but hard
- Natural languages mostly used, with some structure for the document
- Formal languages used for special features or in highly critical systems

Structure of an SRS

- Introduction
  - Purpose , the basic objective of the system
  - Scope of what the system is to do , not to do
  - Overview
- Overall description
  - Product perspective
  - Product functions
  - User characteristics
  - Assumptions
  - Constraints

Structure of an SRS…

- Specific requirements
  - External interfaces
  - Functional requirements
  - Performance requirements
  - Design constraints
- Acceptable criteria
  - desirable to specify this up front.
- This standardization of the SRS was done by IEEE.

Use Cases Approach

- Traditional approach for fn specs – specify each function
- Use cases is a newer technique for specifying behavior (functionality)
- I.e. focuses on functional specs only
- Though primarily for specification, can be used in analysis and elicitation
- Can be used to specify business or org behavior also, though we will focus on sw
- Well suited for interactive systems

Use Cases Basics

- A use case captures a contract between a user and system about behavior
- Basically a textual form; diagrams are mostly to support
- Also useful in requirements elicitation as users like and understand the story telling form and react to it easily

Basics..

- Actor: a person or a system that interacts with the proposed system to achieve a goal
  - Eg. User of an ATM (goal: get money); data entry operator; (goal: Perform transaction)
- Actor is a logical entity, so receiver and sender actors are different (even if the same person)
- Actors can be people or systems
- Primary actor: The main actor who initiates a UC
  - UC is to satisfy his goals
  - The actual execution may be done by a system or another person on behalf of the Primary actor

Basics..

- Scenario: a set of actions performed to achieve a goal under some conditions
  - Actions specified as a sequence of steps
  - A step is a logically complete action performed either by the actor or the system
- Main success scenario – when things go normally and the goal is achieved
- Alternate scenarios: When things go wrong and goals cannot be achieved

Basics..

- A UC is a collection of many such scenarios
- A scenario may employ other use cases in a step
- I.e. a sub-goal of a UC goal may be performed by another UC
- I.e. UCs can be organized hierarchically

Basics…

- UCs specify functionality by describing interactions between actors and system
- Focuses on external behavior
- UCs are primarily textual
  - UC diagrams show UCs, actors, and dependencies
  - They provide an overview
- Story like description easy to understand by both users and analysts
- They do not form the complete SRS, only the functionality part

Example

Use Case 1: Buy stocks

Primary Actor: Purchaser

Goals of Stakeholders:

- Purchaser: wants to buy stocks
- Company: wants full transaction info

Precondition: User already has an account

Example …

- Main Success Scenario
  1. User selects to buy stocks
  2. System gets name of web site from user for trading
  3. Establishes connection
  4. User browses and buys stocks
  5. System intercepts responses from the site and updates user portfolio
  6. System shows user new portfolio stading

Example…

- Alternatives
  - 2a: System gives err msg, asks for new suggestion for site, gives option to cancel
  - 3a: Web failure. 1-Sys reports failure to user, backs up to previous step. 2-User exits or tries again
  - 4a: Computer crashes
  - 4b: web site does not ack purchase
  - 5a: web site does not return needed info

Example 2

- Use Case 2: Buy a product
- Primary actor: buyer/customer
- Goal: purchase some product
- Precondition: Customer is already logged in

Example 2…

- Main Scenario
  1. Customer browses and selects items
  2. Customer goes to checkout
  3. Customer fills shipping options
  4. System presents full pricing info
  5. Customer fills credit card info
  6. System authorizes purchase
  7. System confirms sale
  8. System sends confirming email

Example 2…

- Alternatives
  - 6a: Credit card authorization fails
    - Allows customer to reenter info
  - 3a: Regular customer
    - System displays last 4 digits of credit card no
    - Asks customer to OK it or change it
    - Moves to step 6

Example – An auction site

- ***Use Case1:*** Put an item for auction
- ***Primary Actor:*** Seller
- ***Precondition:*** Seller has logged in
- ***Main Success Scenario:***
  - Seller posts an item (its category, description, picture, etc.) for auction
  - System shows past prices of similar items to seller
  - System specifies the starting bid price and a date when auction will close
  - System accepts the item and posts it
- ***Exception Scenarios:***
  - -- 2 a) There are no past items of this category

    * System tells the seller this situation

Example – auction site..

- ***Use Case2:*** Make a bid
- ***Primary Actor:*** Buyer
- ***Precondition:*** The buyer has logged in
- ***Main Success Scenario:***
  - Buyer searches or <u>browses</u> and <u>selects</u> some item
  - System shows the rating of the seller, the starting bid, the current bids, and the highest bid; asks buyer to make a bid
  - Buyer specifies bid price, max bid price, and increment
  - Systems accepts the bid; <u>Blocks funds in bidders account</u>
  - System updates the bid price of other bidders where needed, and updates the records for the item

- ***Exception Scenarios:***
  - -- 3 a) The bid price is lower than the current highest

    * System informs the bidder and asks to rebid
  - -- 4 a) The bidder does not have enough funds in his account

    * System cancels the bid, asks the user to get more funds

Example –auction site..

- ***Use Case3:*** **Complete auction of an item**
- ***Primary Actor:*** Auction System
- ***Precondition:*** The last date for bidding has been reached
- ***Main Success Scenario:***
  - Select highest bidder; send email to selected bidder and seller informing final bid price; send email to other bidders also
  - Debit bidder's account and credit seller's account
  - Transfer from seller's account commission amount to organization's account
  - Remove item from the site; update records
- ***Exception Scenarios:*** None

Example – summary-level Use Case

- ***Use Case 0*** **: Auction an item**
- ***Primary Actor:*** Auction system
- ***Scope:*** Auction conducting organization
- ***Precondition:*** None
- ***Main Success Scenario:***
  - Seller performs <u>put an item for auction</u>
  - Various bidders <u>make a bid</u>
  - On final date perform <u>Complete the auction of the item</u>
  - Get feed back from seller; get feedback from buyer; update records

Requirements with Use Cases

- UCs specify functional requirements
- Other req identified separately
- A complete SRS will contain the use cases plus the other requirements
- Note – for system requirements it is important to identify UCs for which the system itself may be the actor

Developing Use Cases

- UCs form a good medium for brainstorming and discussions
- Hence can be used in elicitation and problem analysis also
- UCs can be developed in a stepwise refinement manner
  - Many levels possible, but four naturally emerge

Developing…

- Actors and goals
  - Prepare an actor-goal list
  - Provide a brief overview of the UC
  - This defines the scope of the system
  - Completeness can also be evaluated
- Main Success Scenarios
  - For each UC, expand main scenario
  - This will provide the normal behavior of the system
  - Can be reviewed to ensure that interests of all stakeholders and actors is met

Developing…

- Failure conditions
  - List possible failure conditions for UCs
  - For each step, identify how it may fail
  - This step uncovers special situations
- Failure handling
  - Perhaps the hardest part
  - Specify system behavior for the failure conditions
  - New business rules and actors may emerge

Developing..

- The four levels can drive analysis by starting from top and adding details as analysis proceeds
- UCs should be specified at a level of detail that is sufficient
- For writing, use good technical writing rules
  - Use simple grammer
  - Clearly specify all parts of the UC
  - When needed combine steps or split steps

Requirements Validation

- Lot of room for misunderstanding
- Errors possible
- Expensive to fix req defects later
- Must try to remove most errors in SRS
- Most common errors
  - Omission - 30%
  - Inconsistency - 10-30%
  - Incorrect fact - 10-30%
  - Ambiguity - 5 -20%

Requirements Review

- SRS reviewed by a group of people
- Group: author, client, user, dev team rep.
- Must include client and a user
- Process – standard inspection process
- Effectiveness - can catch 40-80% of req. errors

Sizing With Function Points

Sizing

- Effort for project depends on many factors
- Size is the main factor – many experiments and data analysis have validated this
- Size in the start is only an estimate; getting size estimates from requirement is hard
- Need a size unit that can be "computed" from requirements
- Function points attempt to do this

Function Points

- Is a size measure like LOC
- Determined from SRS
- Defines size in terms of " functionality "
- Why "measure" size early ?
  - Needed for estimation and planning
- Five different parameters
  - external input type
  - external output type
  - logical internal file type
  - external interface file type
  - external inquiry type

Function Points…

These five parameters capture the functionality of a system

- within a type , an element may be simple , average or complex
- A weighted sum is taken

External input type :

- each unique input type
- A input type is unique if the format is different from others or if the specifications require different processing.

Function Points…

- Simple : a few data elements
- Complex : many data elements and many internal files needed for processing
- Only files needed by the application are counted. (HW/OS config. Files are are not counted)

External output type :

- each unique output that leave system boundary
- E.g.
  - Reports , messages to user , data to other applications
- Simple : few columns

Function Points…

- Average : many columns
- Complex : references many files for production

Logical internal file type :

- An application maintains information internally for its own processes
- Each logical group of data generated , used and maintained
- Same for simple , average and complex

Function Points…

- External interface file type
  - logical files passed between application
- External inquiry type
  - input , output combination
- Weights

External Input		3	4	6
External Output		4	5	7
Logical int. file	7	10	15	
External int. file		5	7	10
External inquiry		3	4	6

Function Points…

Unadjusted function point :

- Basic function points
- Adjusted for other factors
- 14 such factors
  - performance objectives , transaction rate etc.
- Final FP is adjusted
  - differs at most 35%

Function Points…

- Interest in FP
  - since obtained at requirements => major advantage
- Well correlated with size
  - in some what interchangeable and tables exist
- 1 FP = 70 LOC of C
- Works well for MIS , but not for system type
- Major draw back - subjectivity
  - not repeatable
  - not precisely known ever for a built system
  - not addictive

Summary

- Having a good quality SRS is essential for Q&P
- The req. phase has 3 major sub phases
  - analysis , specification and validation
- Analysis
  - for problem understanding and modeling
  - Methods used: SSAD, OOA , Prototyping
- Key properties of an SRS: correctness, completeness, consistency, traceablity, unambiguousness

Summary..

- Specification
  - must contain functionality , performance , interfaces and design constraints
  - Mostly natural languages used
- Use Cases is a method to specify the functionality; also useful for analysis
- Validation - through reviews
- Function point is a size metric that can be extracted from the SRS