XP and Modelling

Peter Dolog
dolog [at] cs [dot] aau [dot] dk
E2-201
Information Systems
February 15, 2007

Software Modelling in Analysis and Design

Business Process Engineering

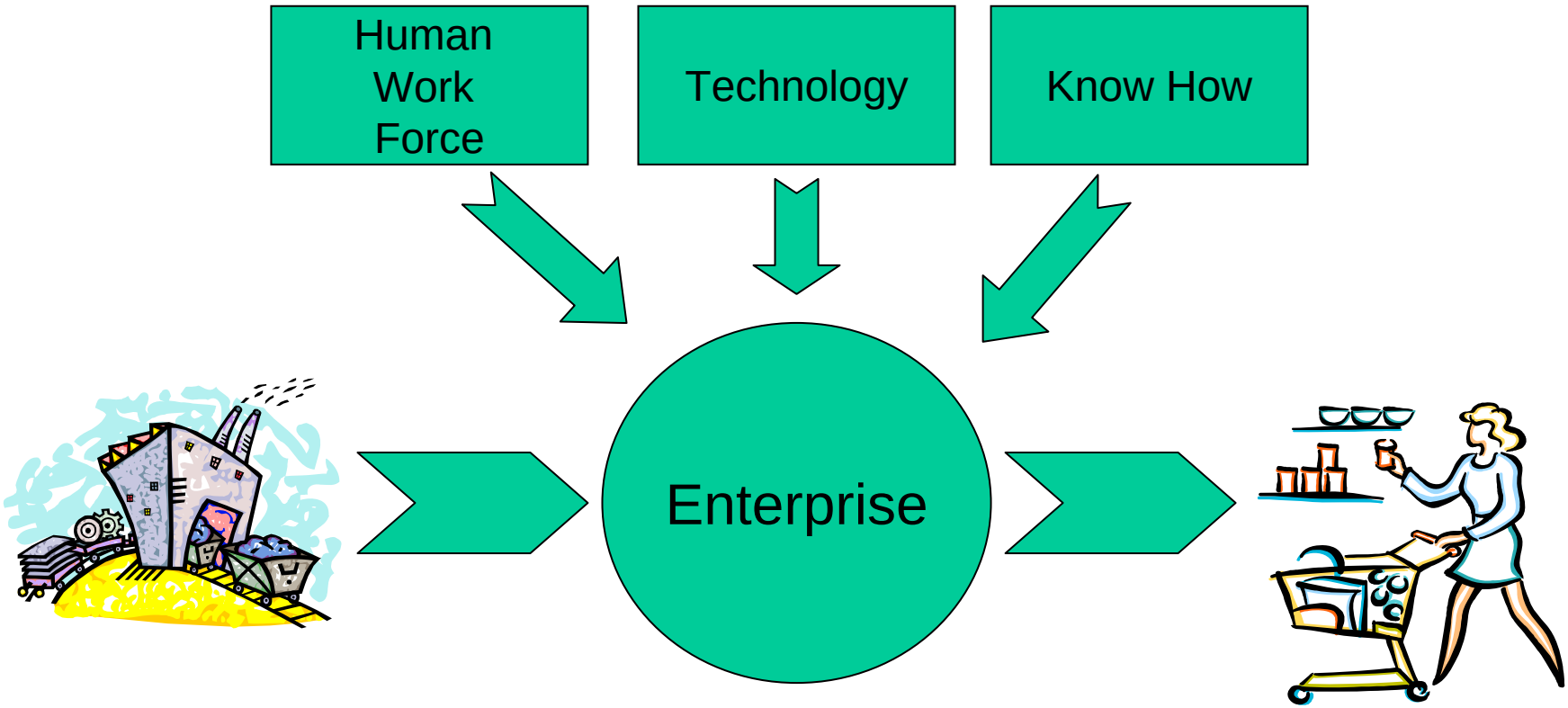
Uses an integrated set of procedures, methods, and tools to identify how information systems can best meet the strategic goals of an enterprise

Focuses first on the enterprise and then on the business area

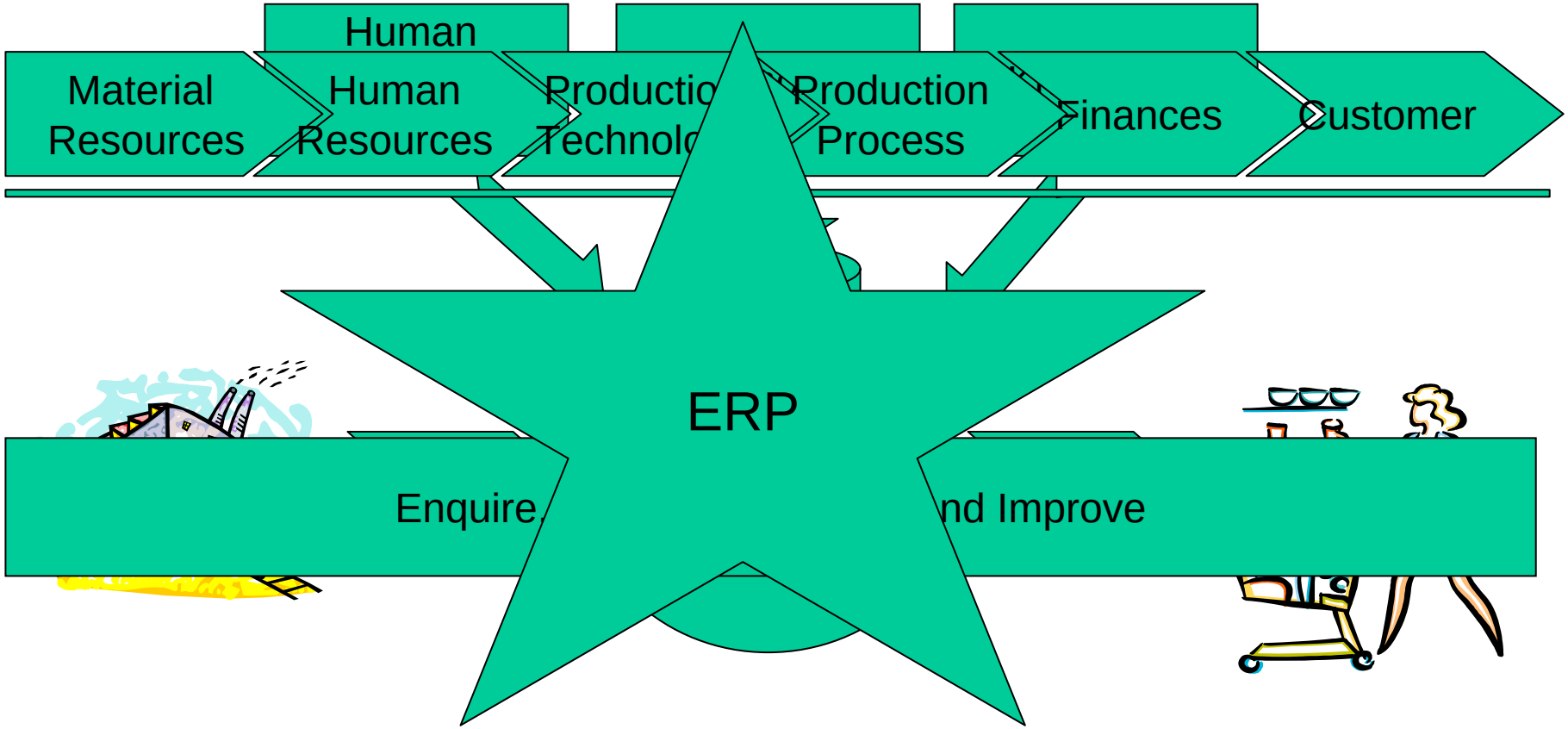
Creates enterprise models, data models and process models

Creates a framework for better information management distribution, and control

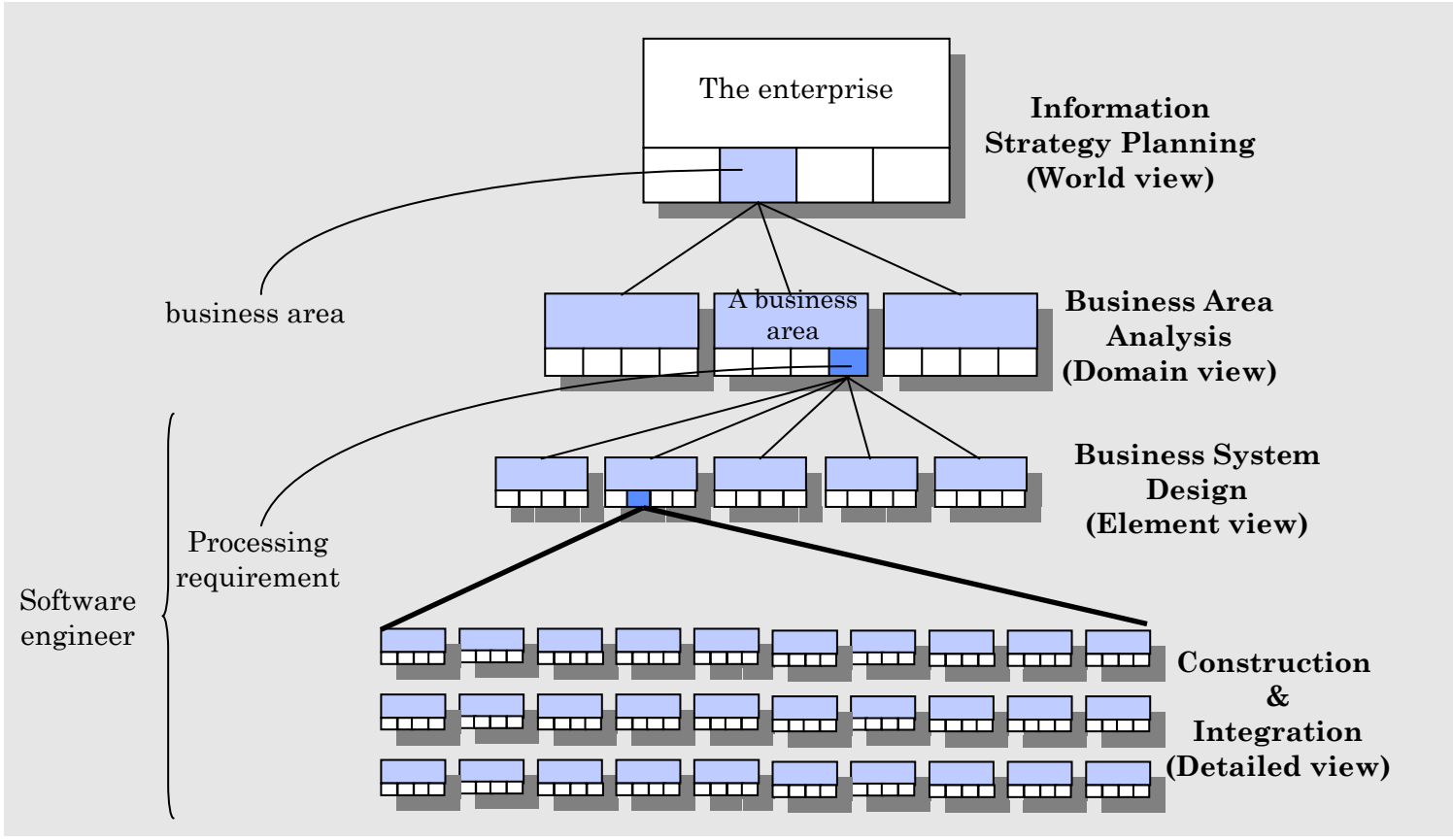
A Company



A Company


Material Resources
Human Resources
Human
Production Technolo
Production Process
Finances
Customer
ERP
Enquire,
nd Improve

BPE Hierarchy



Information Strategy Planning

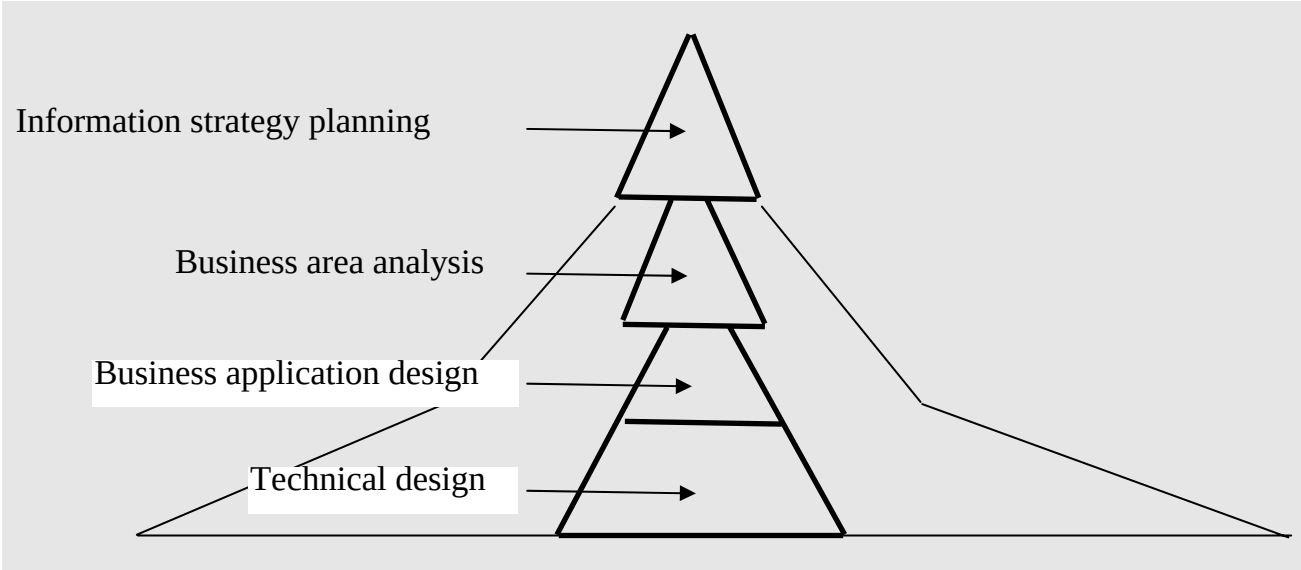
Management issues

- define strategic business goals/objectives
- isolate critical success factors
- conduct analysis of technology impact
- perform analysis of strategic systems

Technical issues

- create a top-level data model
- cluster by business/organizational area
- refine model and clustering

Information Engineering Facility



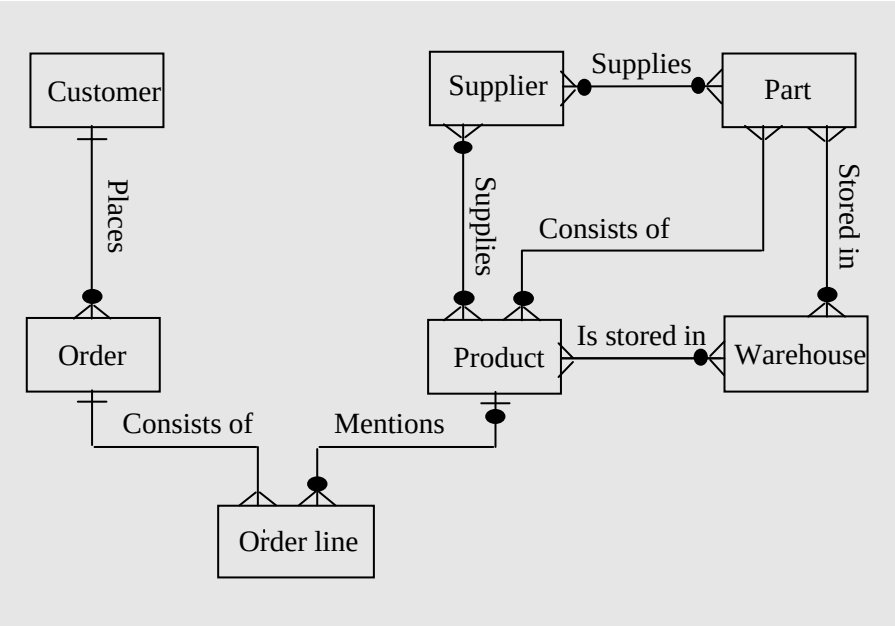
IEF - Information planning

C = Create
R = Read
U = Update
D = Delete

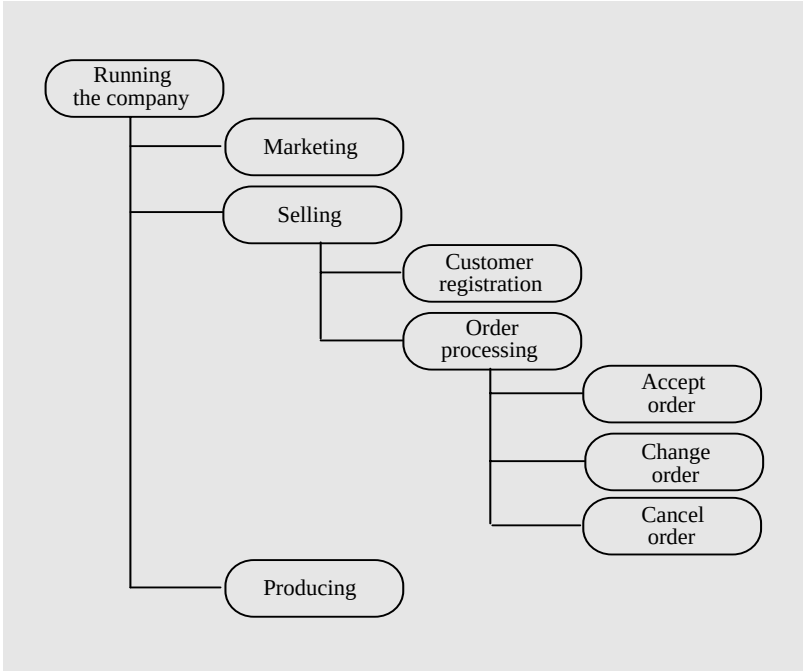
Entity Types	Business Functions Marketing	Customer registration	Accept Order	Change Order	Cancel order	Producing	
Customer				R			
Order				C	U	D	
Order Line					U	D	
Product							
Part							
Supplier							
Warehouse							

Matrices

IEF - Analysis: ERD



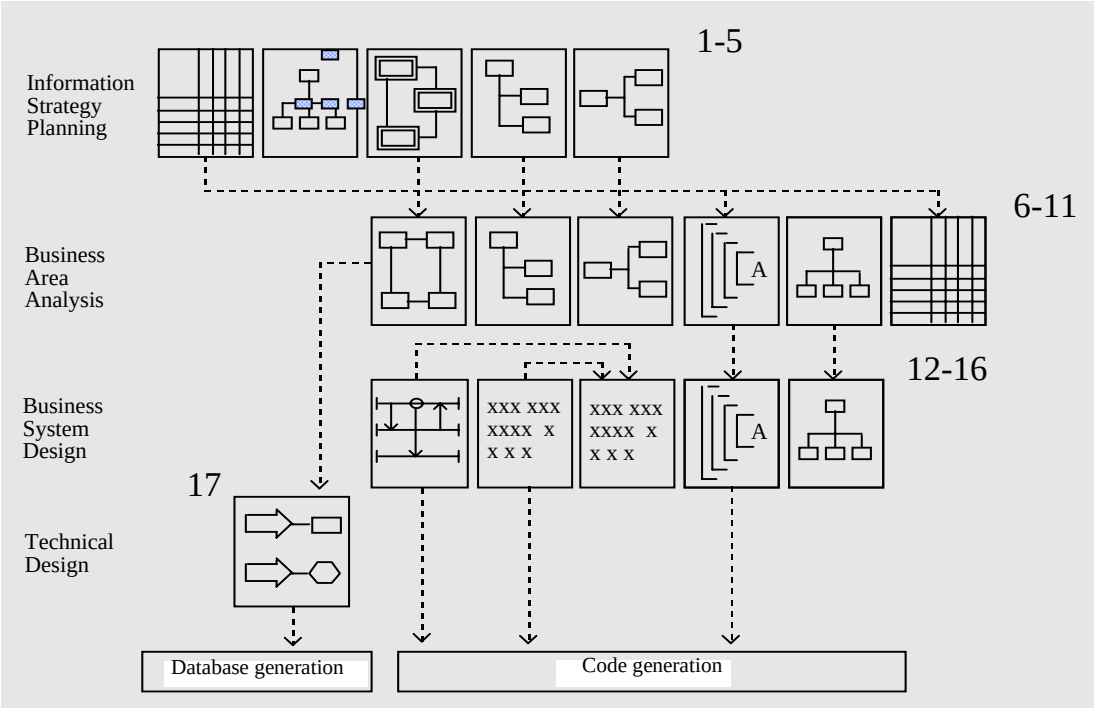
IEF - Analysis: Process Hierarchy



IEF - Analysis: Process Handling

```
Process:    ACCEPT ORDER
        ACCEPT ORDER
            IMPORTS:      Entity View to_be_ordered_product
                          ...........
            EXPORTS:      Entity View confirmed order_line
                          ...........
            ENTITY ACTIONS:       Entity View confirmed order_line
                                  ...........
                    READ to_be_controlled product
                        WITH name EQUAL TO to_be_ordered product name
                    WHEN not found
                    ESCAPE
                        ...........
                        CREATE confirmed order
                        SET date TO "system date"
                        SET number TO "next free value"
                        ASSOCIATE WITH to_be_controlled customer WHICH places IT
                        ASSOCIATE WITH confirmed order_line WHICH details IT
                        WHEN already exists
                              .........

                        MOVE confirmed order TO accepted order,
```

IEF: From Analysis to Code



IEF: From Analysis to Code

Information Strategy Planning:
1 Matrix Processor
2 Organizational Hierarchy Diagram
3 Subject Area Diagram
4 Function Hierarchy Diagram
5 Function Dependency Diagram

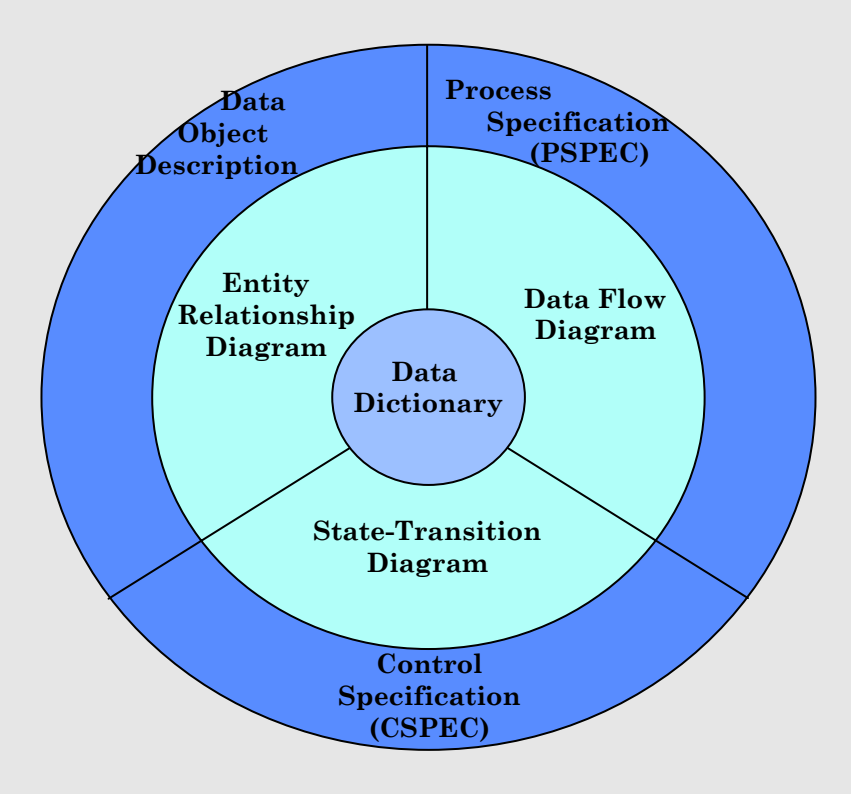
Business System Design:
12 Dialog Flow Diagram
13 Screen Design
14 Prototyping
15 Procedure Action Diagram
16 Structure Chart

Business Area Analysis:
6 Entity Relationship Diagram
7 Process Hierarchy Diagram
8 Process Dependency Diagram
9 Process Action Diagram
10 Structure Chart
11 Matrix Processor

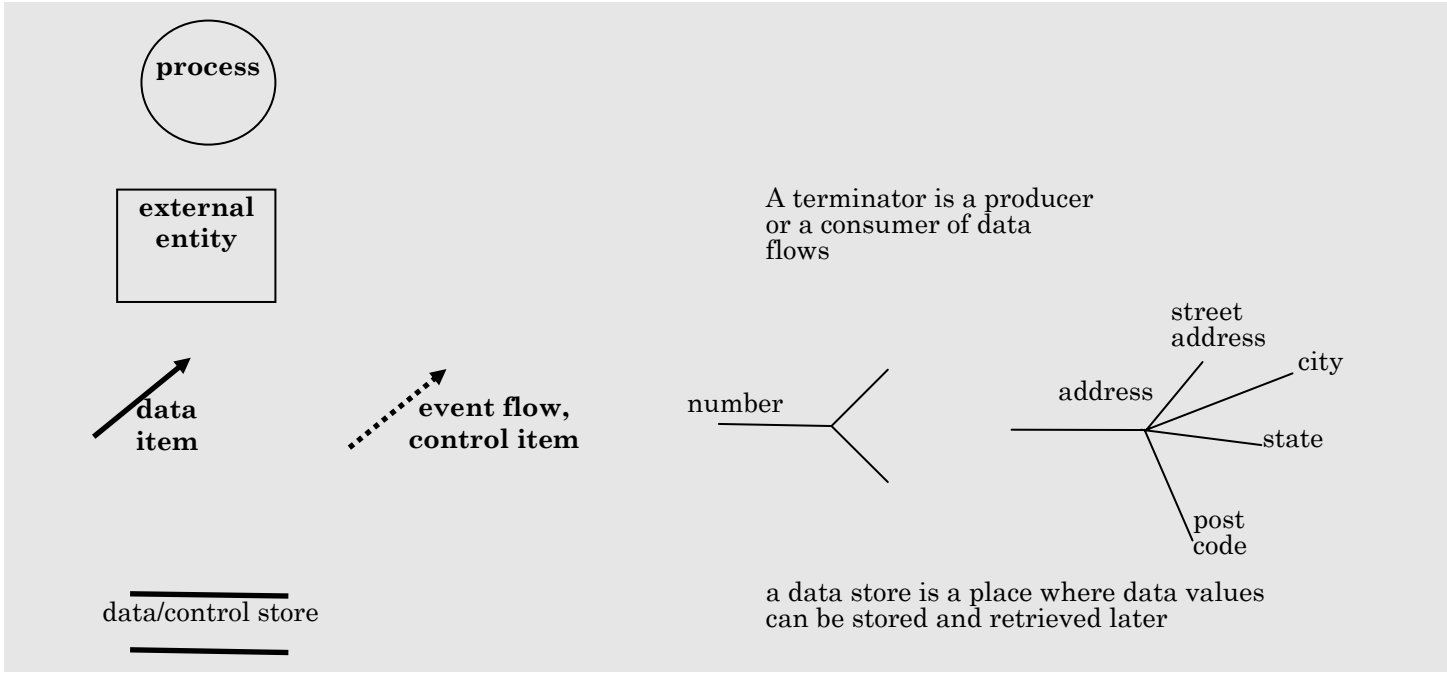
Technical Design:
17 Data Structure Diagram

Structured Analysis & Design

Analysis Model Structure



The Functional Model: The DFD



Structured Analysis

The environmental model

- Statement of purpose
- Context diagram
- Event list

The behavioral model

- Data flow diagrams
- Process Specification

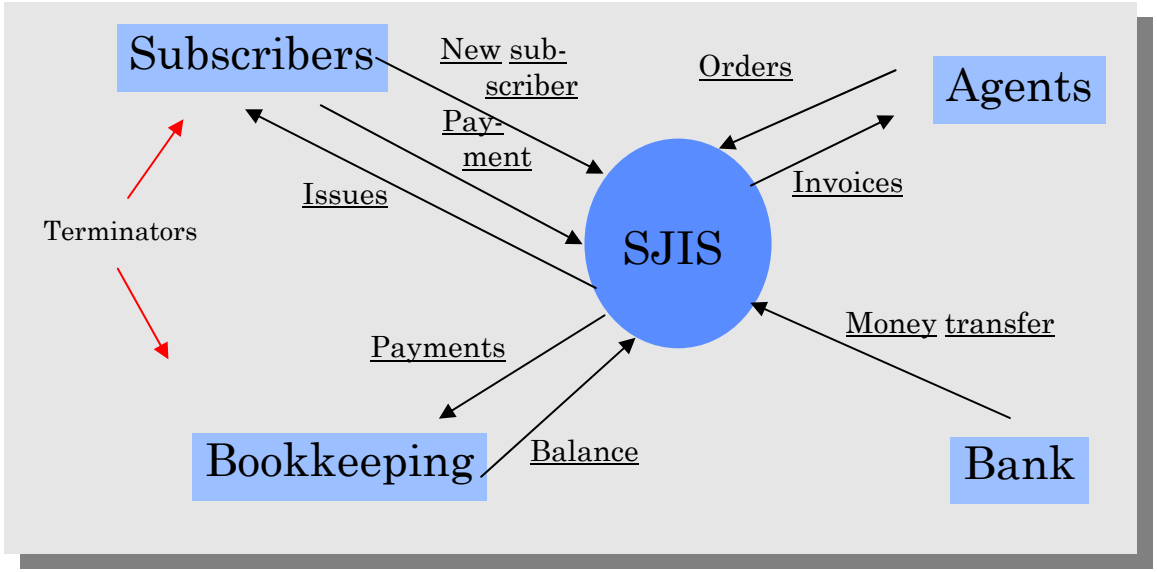
Statement of Purpose

A brief, concise textual statement of the purpose of the system.

Clarifies the boundaries of the system - what will be taken care of by the system, and what will not?

Journal administration:
The computer based system is used for administration of the information necessary to publish the Scandinavian Journal of Information Systems (SJIS). This includes registration of new subscribers, billing, mailing, and registration of subscriber data.

Context Diagram

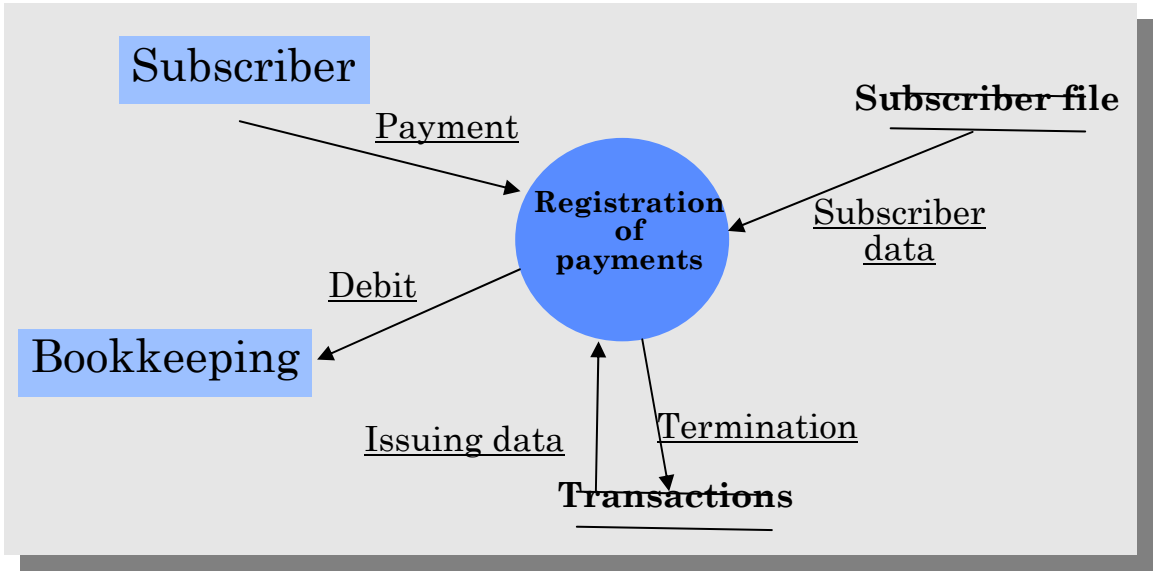


The Event List

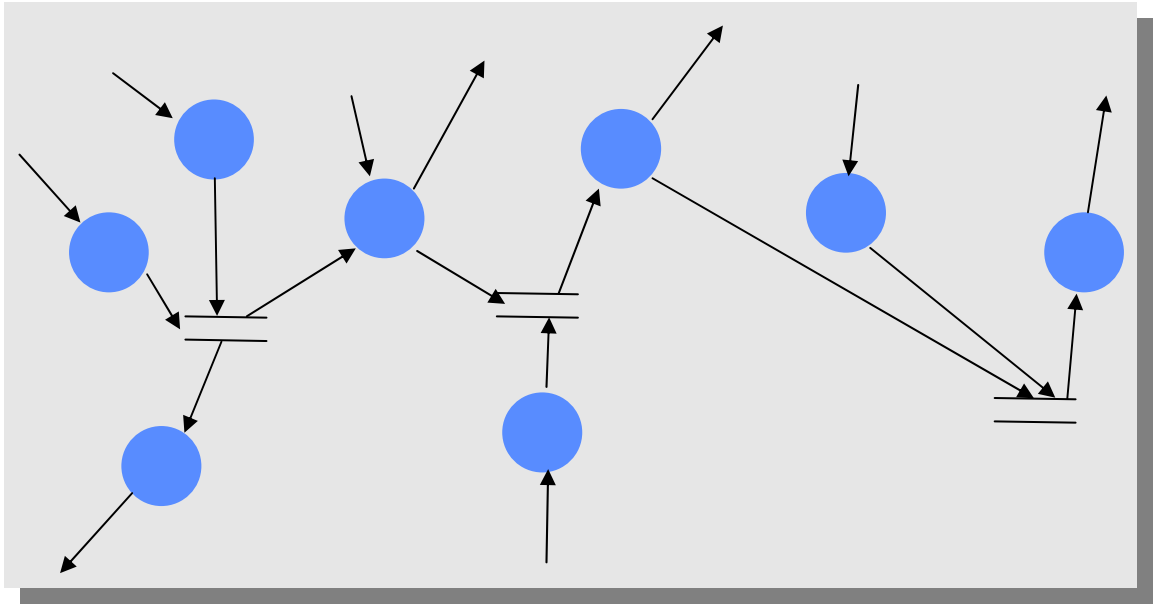
A list of the "stimuli" that occur in the outside world and to which the system must respond.

1. Person or institution enters subscription.
2. Agent enters subscription on behalf of a person or institution.
3. The bank reports a money transfer.
4. Bookkeeping receives details on payments.
5. Agent cancels subscription.
6. Issue is sent to subscriber.
7. Invoice is sent to agent.
8. Invoice is sent to subscriber directly.
9. Subscription is cancelled.
10. Subscriber pays amount due
11. .....

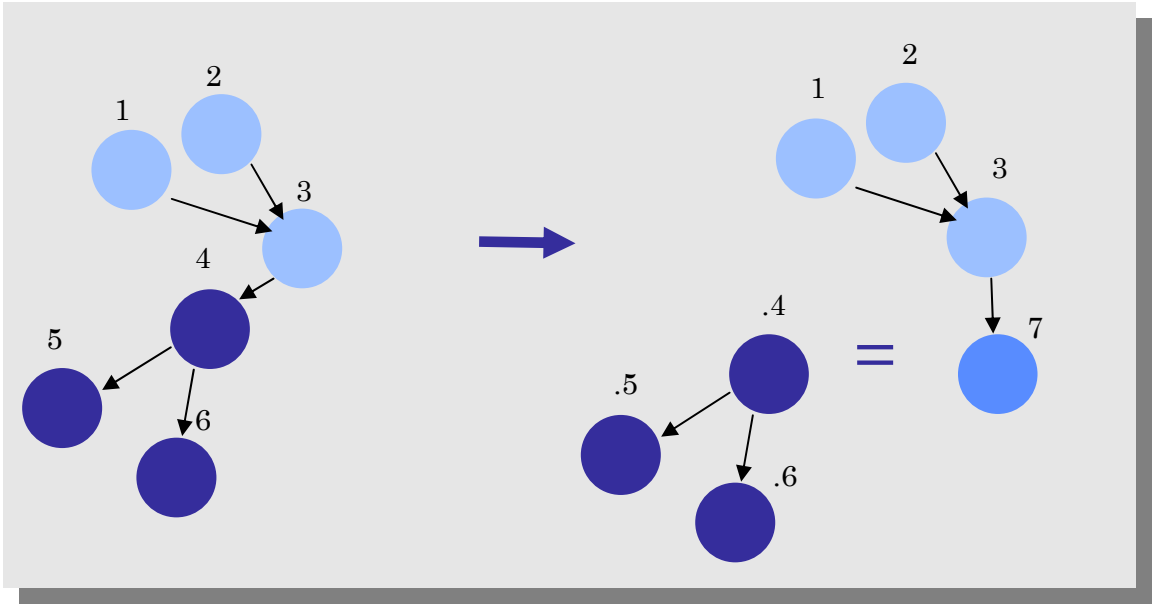
Subscriber pays amount due



Responses In One Diagram

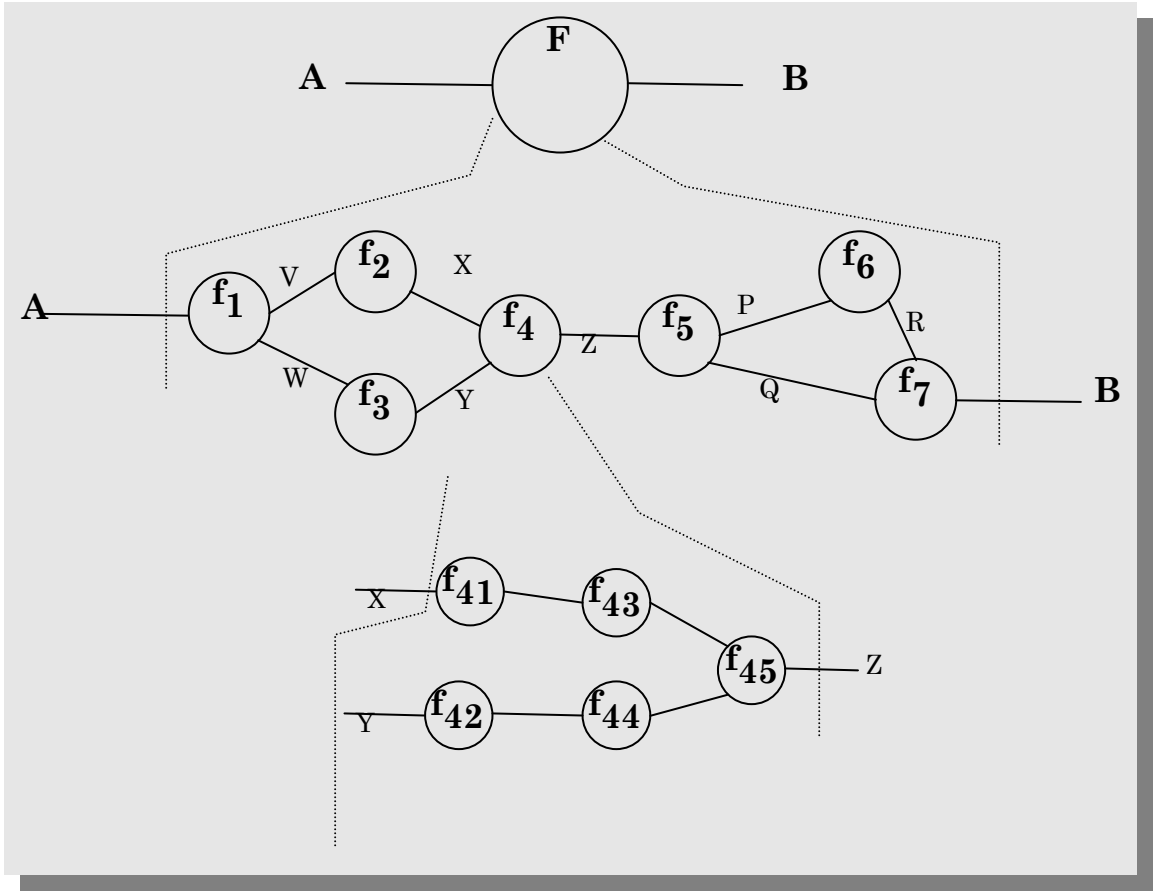


Upward Leveling of Diagrams



Processes 4, 5, and 6 are united into a new process 7.
Processes 4, 5, and 6 are pushed one level down.

Data Flow Hierarchy



Process Specification

```
FIND Subscriber in Subscribers based on
Subscriber#.
IF not found: ERROR(Subscriber unknown)
ELSE
    FIND invoicedata for Subscriber#.
    IF no pending invoice:
        ERROR(No pending invoice)
    ELSE mark invoice as paid.
    WRITE debitinfo to bookkeeping.
```

At the lowest level a process is described in structured English, flowcharts or similar.

Data Dictionary

Name: telephone number

Aliases: phone number, number

Where used: read-phone-number (input)

display-phone-number (output)

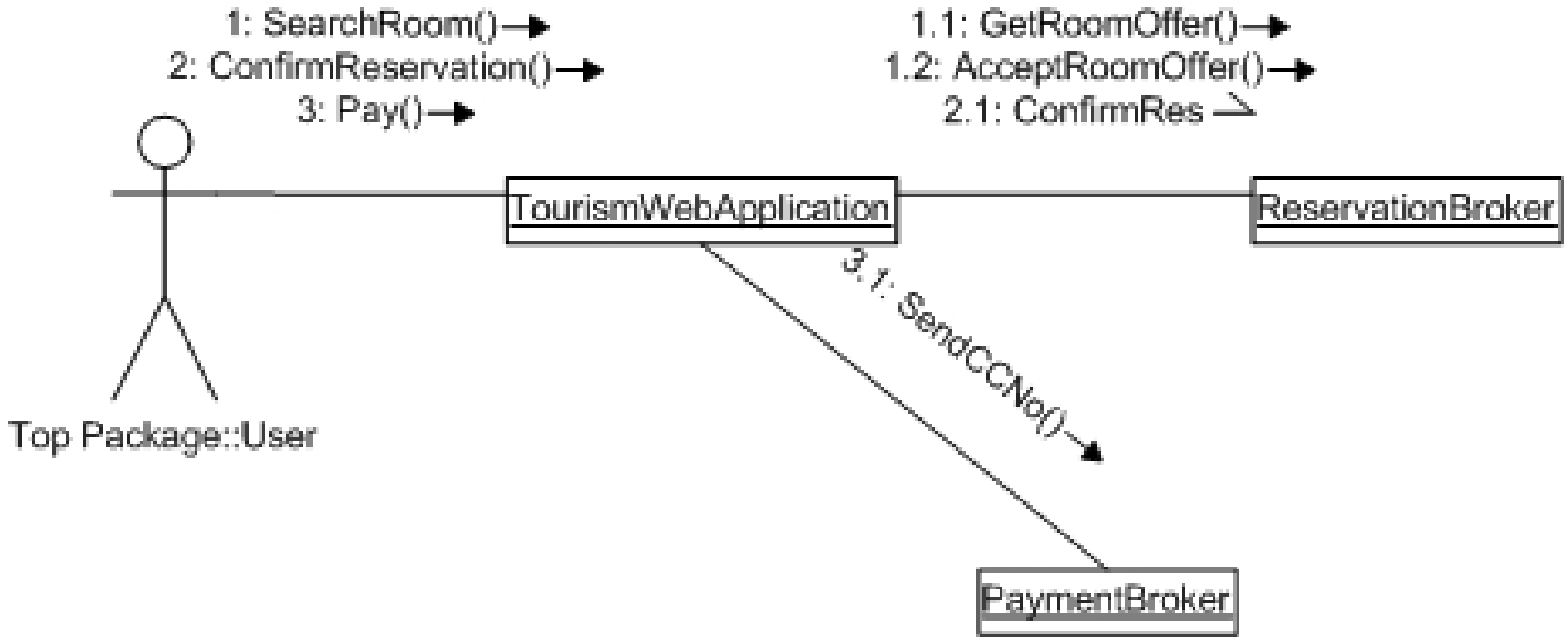
analyze-long-distance-calls (input)

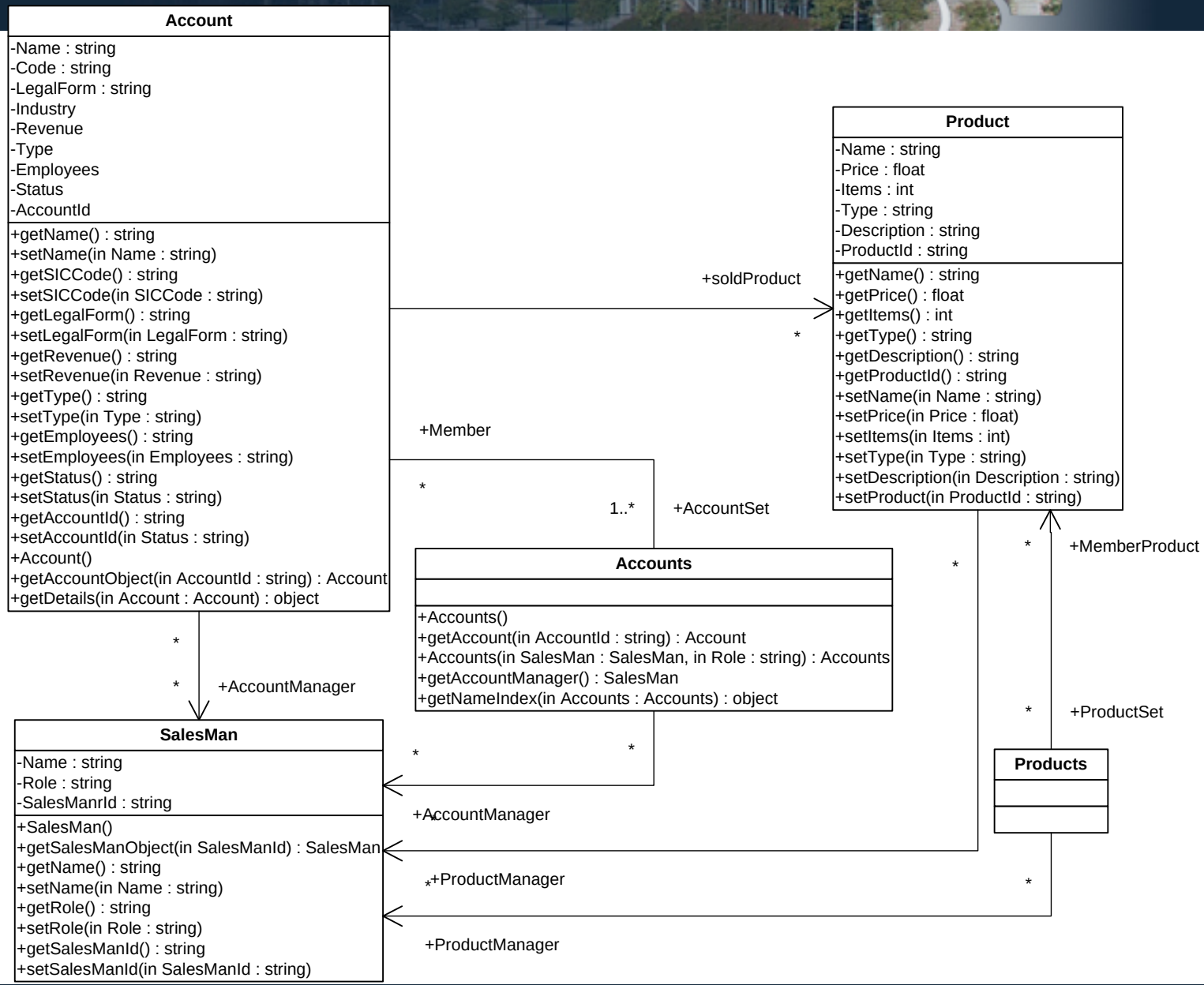
Description:

```
telephone no. = [ local extension | outside
  no.  | 0 ]

outside no. = 9 + [ service code | domestic
  no. ]

service code = [ 211 | 411 | 611 | 911 ]

domestic no. = ( (0) + area code ) + local
```

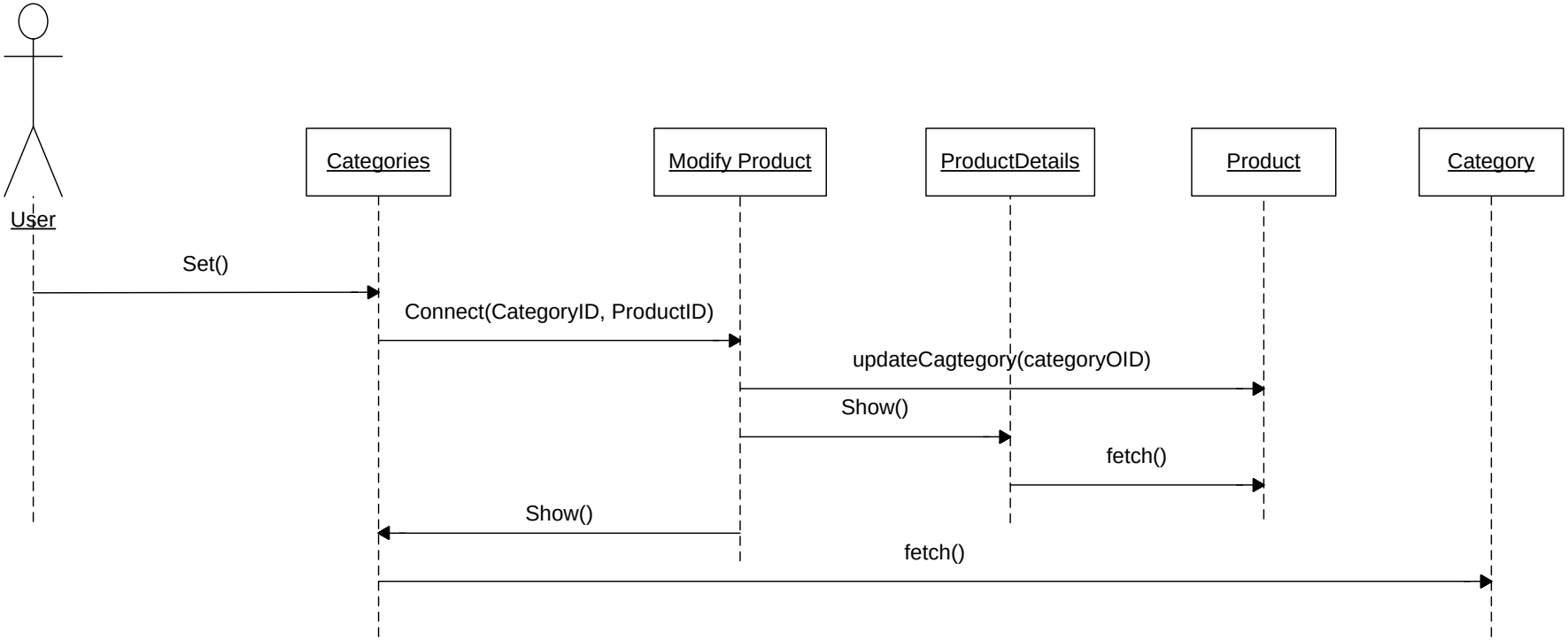
Protocols/Activities

Collabotation/Interaction Diagrams

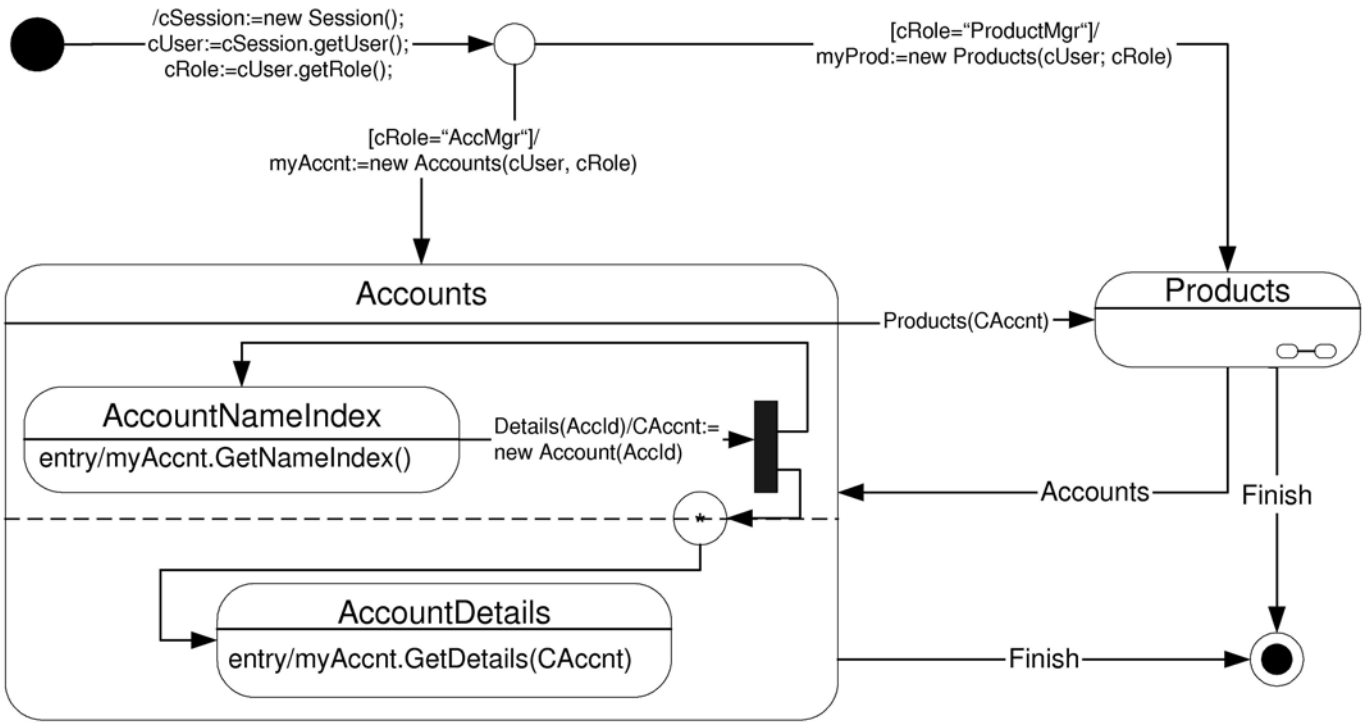



Account
-Name : string
-Code : string
-LegalForm : string
-Industry
-Revenue
-Type
-Employees
-Status
-AccountId
+getName() : string
+setName(in Name : string)
+getSICCode() : string
+setSICCode(in SICCode : string)
+getLegalForm() : string
+setLegalForm(in LegalForm : string)
+getRevenue() : string
+setRevenue(in Revenue : string)
+getType() : string
+setType(in Type : string)
+getEmployees() : string
+setEmployees(in Employees : string)
+getStatus() : string
+setStatus(in Status : string)
+getAccountId() : string
+setAccountId(in Status : string)
+Account()
+getAccountObject(in AccountId : string) : Account
+getDetails(in Account : Account) : object
Product
-Name : string
-Price : float
-Items : int
-Type : string
-Description : string
-ProductId : string
+getName() : string
+getPrice() : float
+getItems() : int
+getType() : string
+getDescription() : string
+getProductId() : string
+setName(in Name : string)
+setPrice(in Price : float)
+setItems(in Items : int)
+setType(in Type : string)
+setDescription(in Description : string)
+setProduct(in ProductId : string)
+soldProduct
*
+Member
*
1..*
+AccountSet
Accounts
+Accounts()
+getAccount(in AccountId : string) : Account
+Accounts(in SalesMan : SalesMan, in Role : string) : Accounts
+getAccountManager() : SalesMan
+getNameIndex(in Accounts : Accounts) : object
*
+MemberProduct
*
*
+AccountManager
SalesMan
-Name : string
-Role : string
-SalesManrId : string
+SalesMan()
+getSalesManObject(in SalesManId) : SalesMan
+getName() : string
+setName(in Name : string)
+getRole() : string
+setRole(in Role : string)
+getSalesManId() : string
+setSalesManId(in SalesManId : string)
*
*
+AccountManager
*
+ProductManager
*
+ProductSet
Products
*
+ProductManager

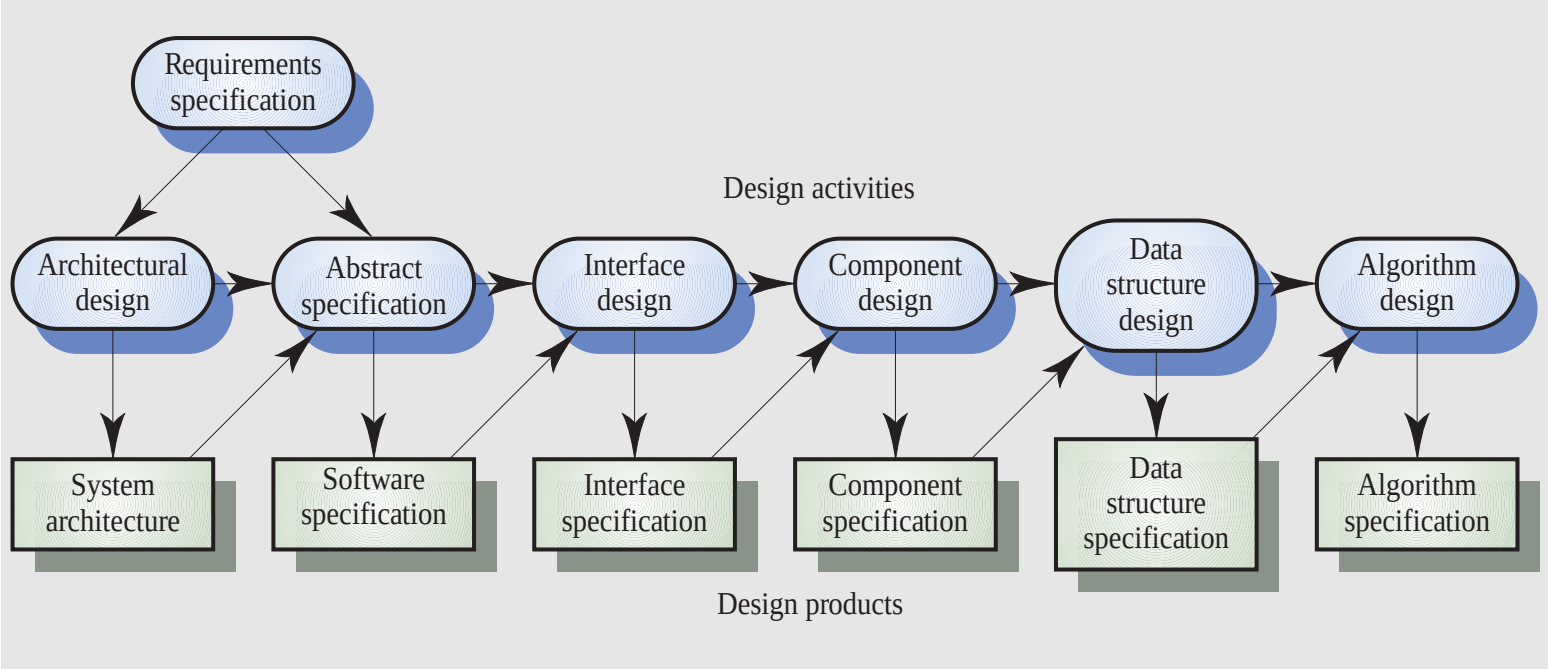
Dialog Sequencing – clasify product



Dialog Sequencing – user interaction



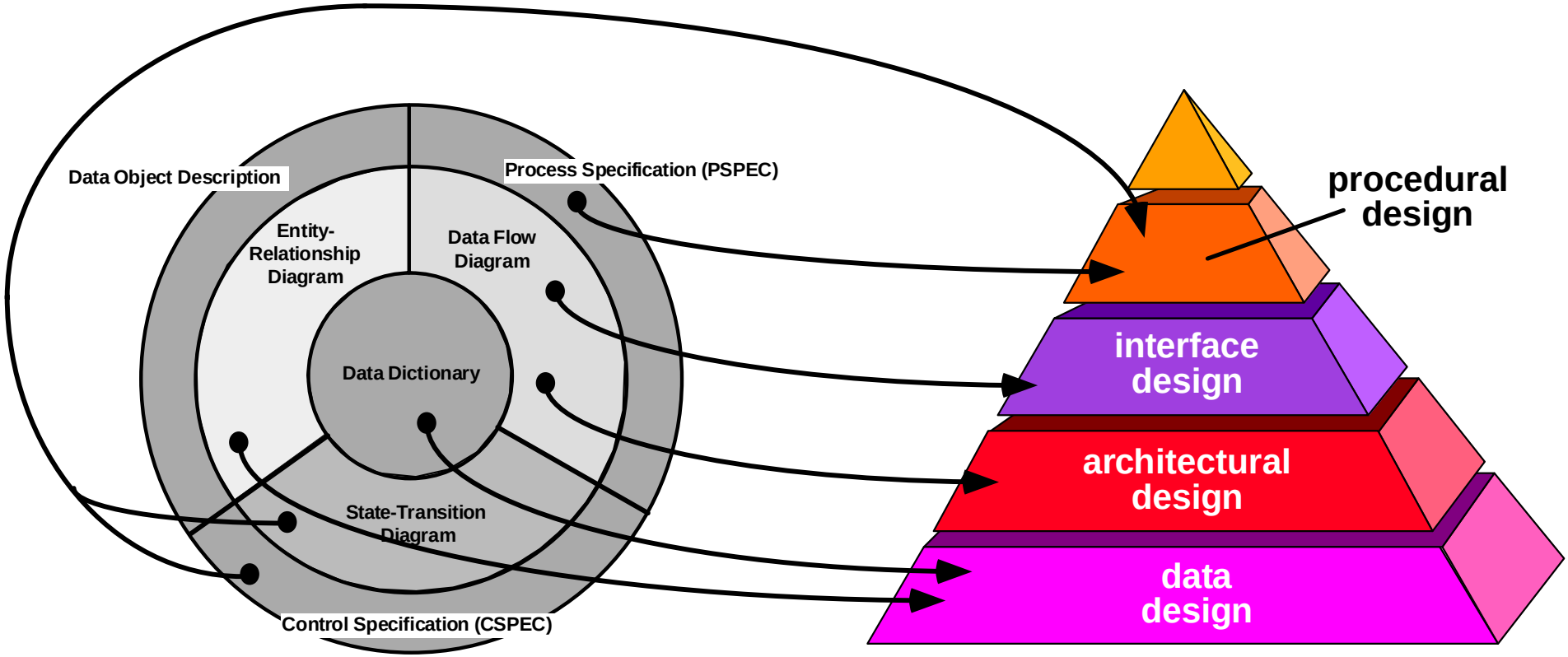
The Software Design Process



Source: Ian Sommerville, Software Engineering

Structured Design

Analysis to Design



THE ANALYSIS MODEL

THE DESIGN MODEL

Structured Design Tools

Structure Chart Shows

Partitioning

each box is a module

Hierarchy

Managing modules are shown above with arrow pointing downward

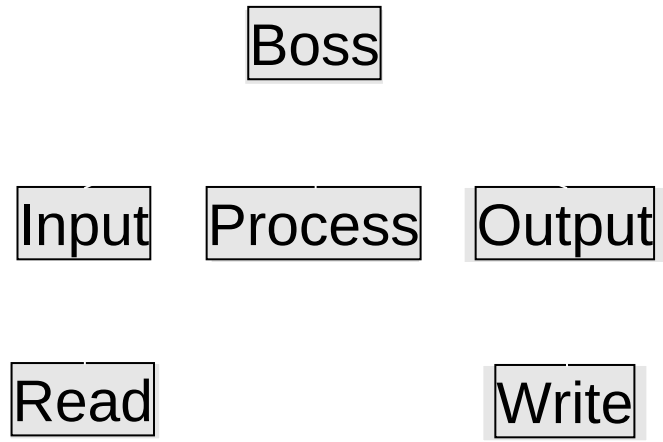
Communication

Small “flags” indicate control, data, and descriptive information as it is passed from one module to another

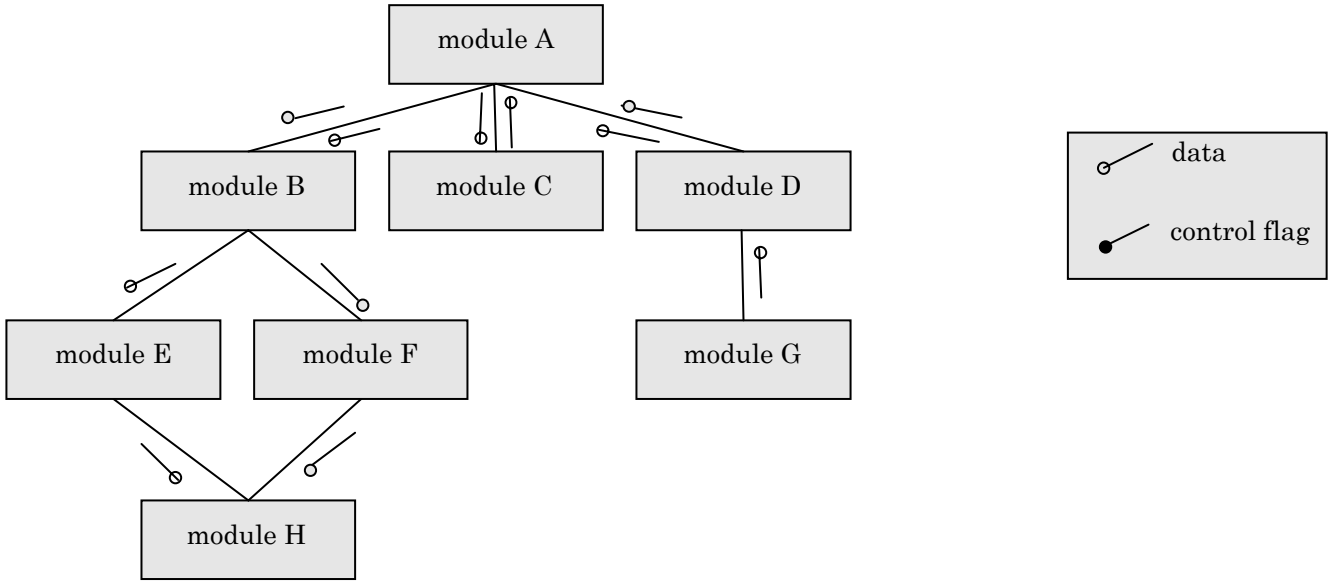
Structure Charts Provides

A semi-formal view of system or program structure

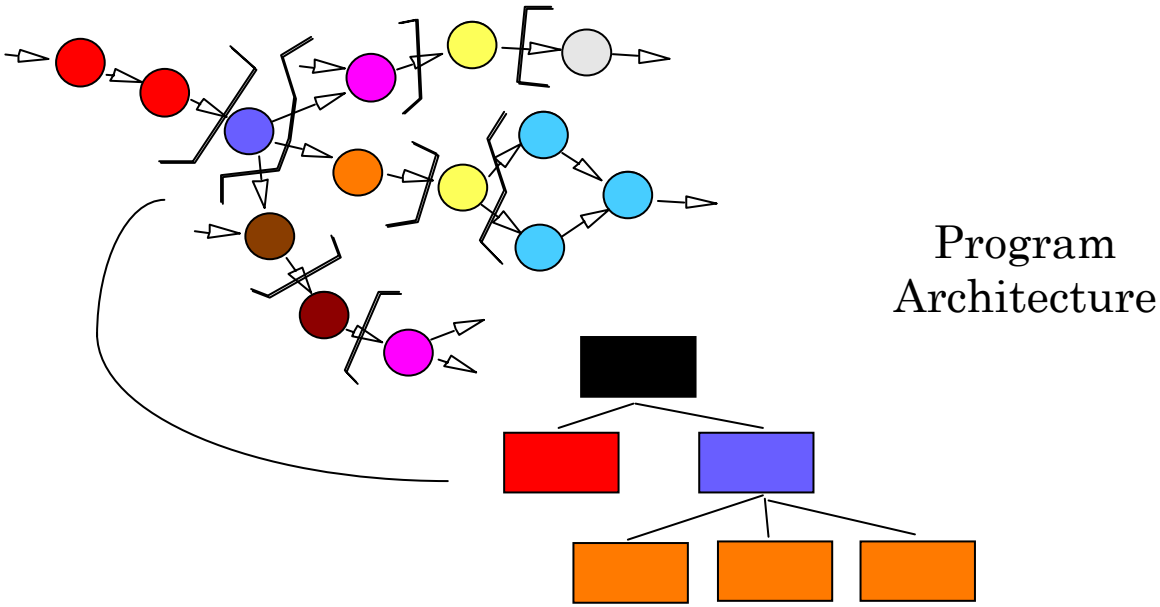
Documentation and blue-prints for programmers and maintainers



Structure Chart



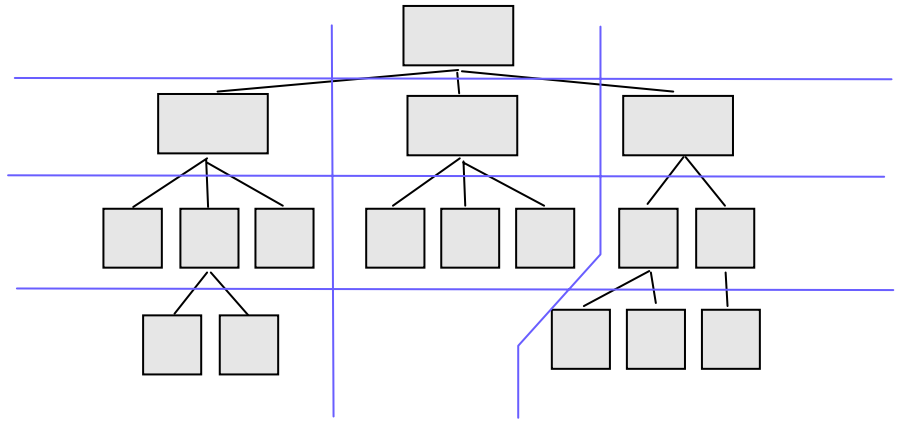
Deriving Program Architecture



Program Architecture

Partitioning the Architecture

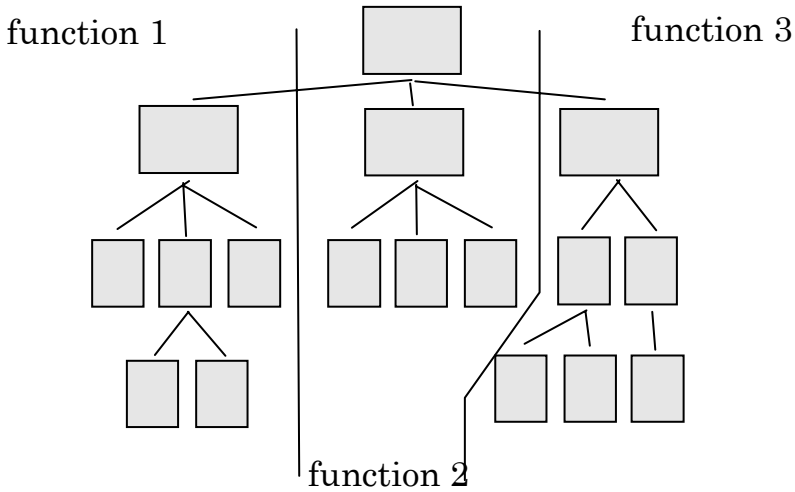
"horizontal" and "vertical" partitioning are required



Horizontal Partitioning

define separate branches of the module hierarchy for each major function

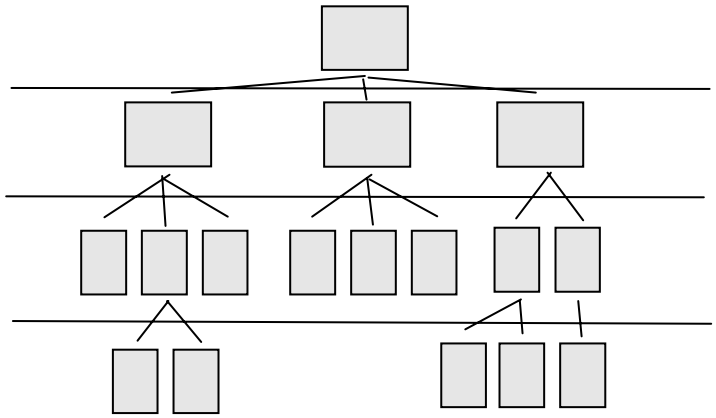
use control modules to coordinate communication between functions



Vertical Partitioning - Factoring

design so that decision making and work are stratified

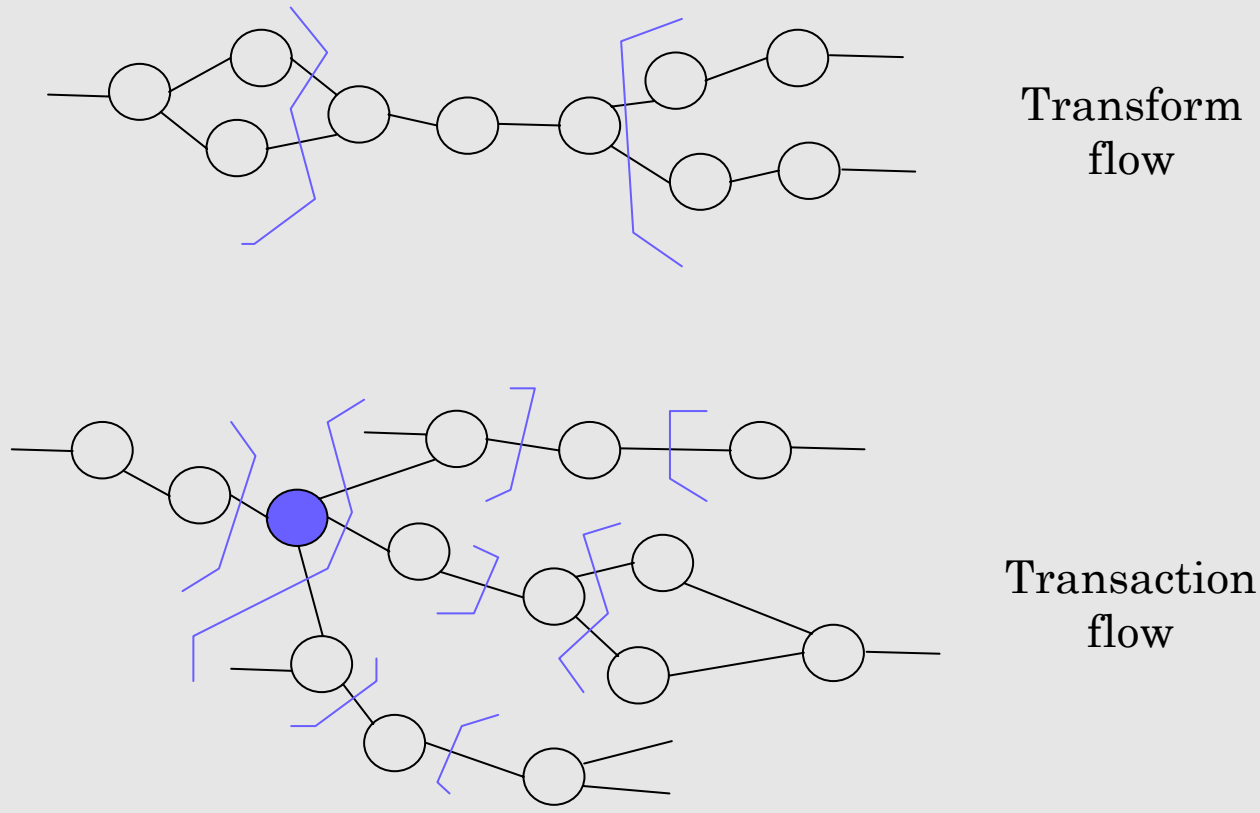
decision making modules should reside at the top of the architecture



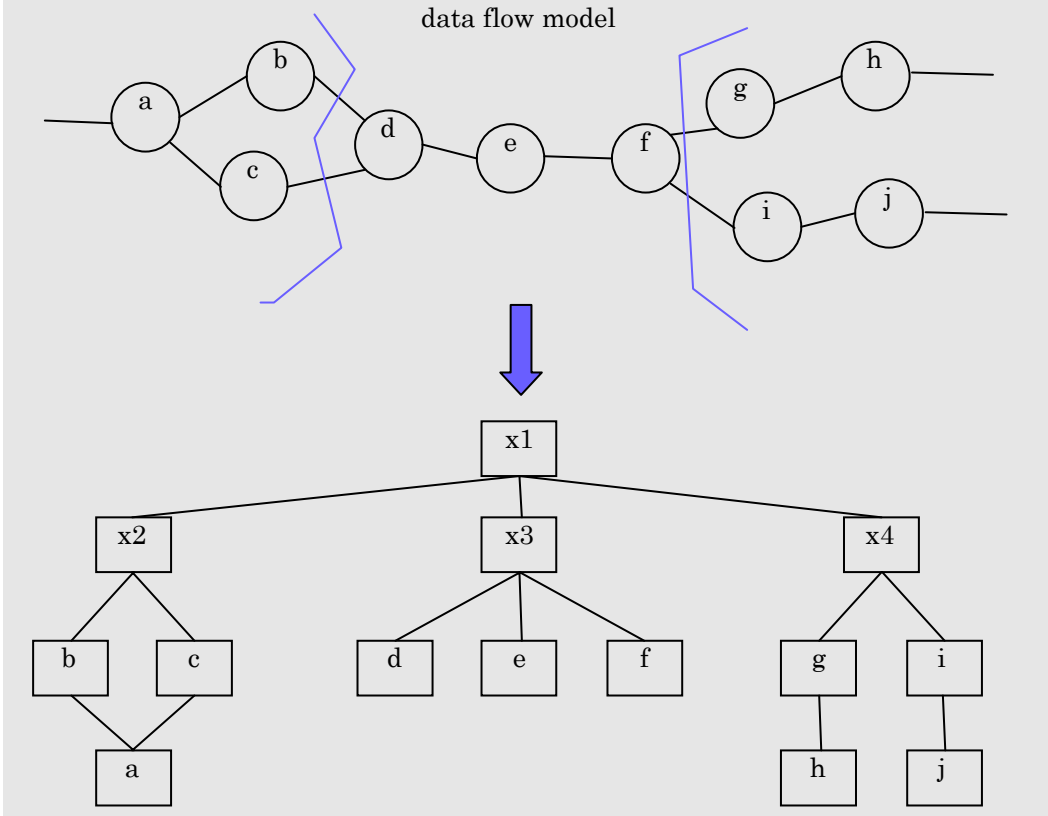
decision-makers

workers

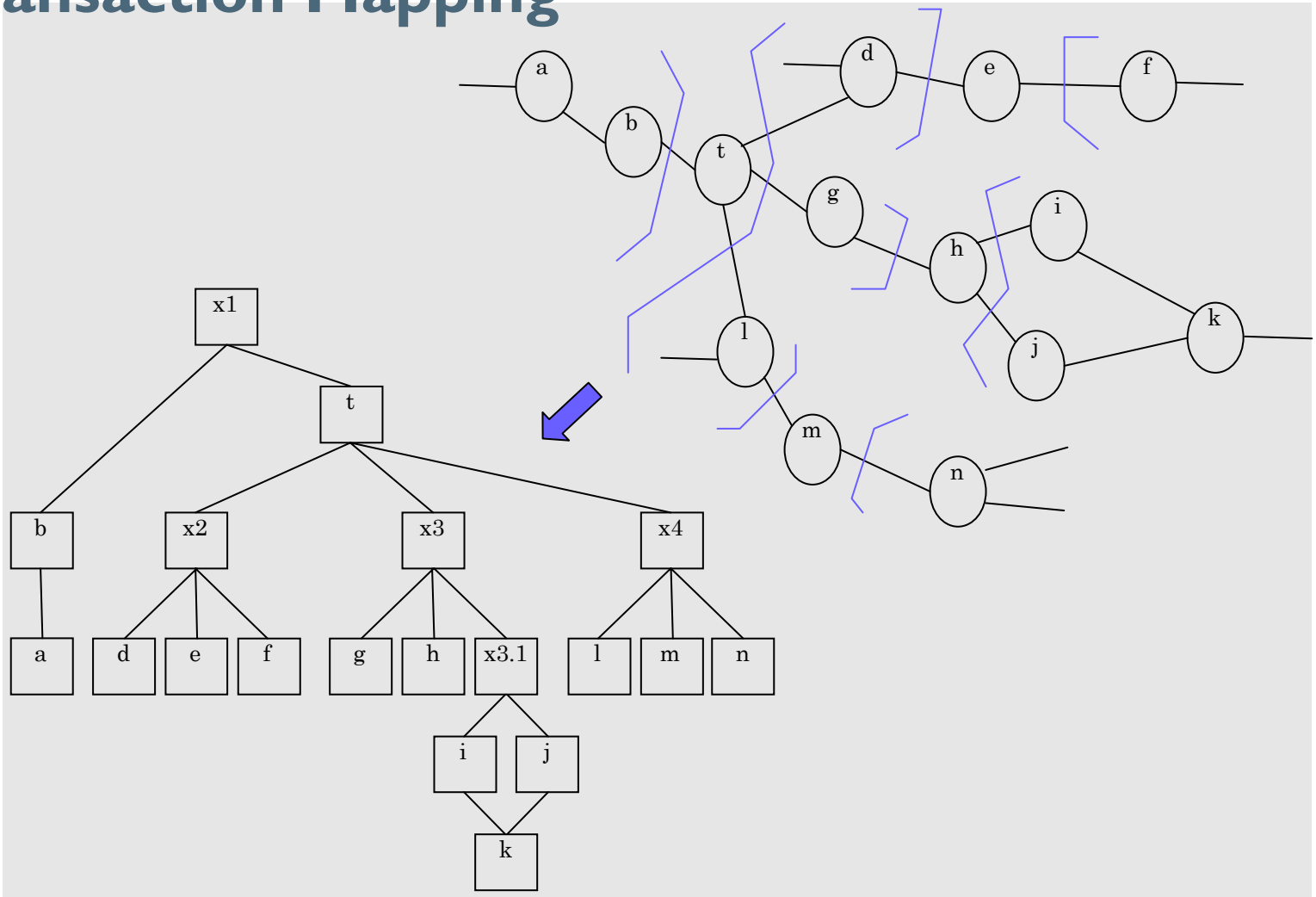
Flow Characteristics



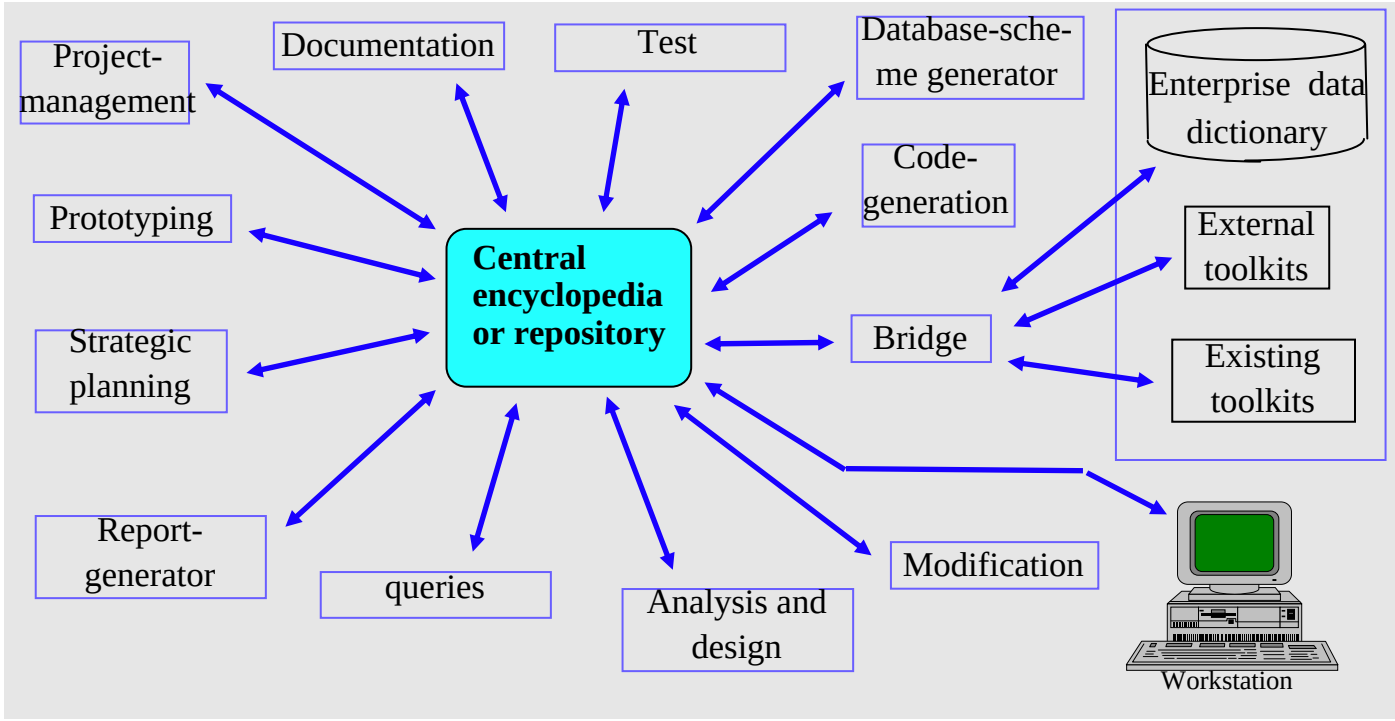
Transform Mapping



Transaction Mapping



Integrated Case — I-CASE



Agile vs. SASD

Agile	SASD
Business value for the customer/user	Business value for the customer/user
Develop as quickly and cheaply as possible	Use models and work products to ensure quality
Involve the user in development	User is involved early on, but not during development
Iteratively elicit requirements (negotiation/emergence)	With req'ts established, sequentially follow process model
Priority on eliminating unneeded req'ts	Priority on traceability and work products

XP

Larman Ch. 8

eXtreme?

If code reviews are good, we'll review code all the time (pair programming).

If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).

If design is good, we'll make it part of everybody's daily business (refactoring).

If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).

Source: Kent Beck XP Explained

eXtreme?

If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).

If integration testing is important, then we'll integrate and test several times a day (continuous integration).

If short iterations are good, we'll make the iterations really, really short - seconds and minutes and hours, not weeks and months and years (the Planning Game).

Source: Kent Beck XP Explained

The Basic Problem: Risk

Schedule slips

The software is not ready at the expected date

Project canceled

After numerous slips the project is canceled

System goes sour

After a couple of years the cost of making changes or the defect rate rises so much that the system must be replaced

Defect rate

The production system is not used since the defect rate is too high

Source: Kent Beck XP Explained

XP's solution

Schedule slips

Short cycles and frequent releases.

Project canceled

The Customer chooses the smallest release that give maximal value. So less could go wrong and the value of the software is greatest.

System goes sour

A comprehensive suite of tests are run after every change. The system is kept in prime condition.

Defect rate

Tests from perspective of both programmers and customer.

Source: Kent Beck XP Explained

The Basic Problem: Risk II

Business misunderstood
: The software solves the wrong problem

Business changes
: The business problem the software is designed to solve is replaced

False feature rich
: The software includes a lot of features which were fun to program, but fail to make the customer much money

Staff turnover
: Programmers start to hate the program and leave

Source: Kent Beck XP Explained

XP's solution II

Business misunderstood

The customer is an integral part of the team
The specifications are continuously refined.

Business changes

Shorter release cycles
During a release the customer could substitute or provide new functionality

False features

Only the highest priority tasks are addressed

Staff turnover

Programmers accept responsibility for estimating and completing their own work
Human contacts are encouraged among the team
Team members are treated as intelligent species

Source: Kent Beck XP Explained

The core values of XP

Communication

Simplicity

Feedback

Courage

Respect

Source: Kent Beck XP Explained

Core principles of XP

Rapid feedback

Assume simplicity

Incremental change

Embracing change

Quality work

Source: Kent Beck XP Explained

Four constraints in priority order

The system (code and tests together) must communicate everything you want to communicate.

The system must contain no duplicate code. (1 and 2 together constitute the Once and Only Once rule).

The system should have the fewest possible classes.

The system should have the fewest possible methods

Source: Kent Beck XP Explained

Identify your next small step - in XP you'll take small steps and run fast

Think of how you will test that you've accomplished this step.

Write the code for one of your tests.

Write just enough code that your test compiles.

Your test should be failing at this point.

Now you write just enough code to make your test pass. You may be tempted to take care of other issues, but you need to stay focused on your current goal. Pass the test.

The Planning Game

Business writes a story describing desired functionality
Stories are written on index cards
Development estimates stories
Velocity determines number of stories per iteration
Business splits and prioritizes stories and determines the composition of releases
Velocity is measured and adjusted every iteration
Customer steers development

XP Planning

Release Planning On-site customer and development team collaborate to:

- Identify software features
- Document the features with "stories"
- Prioritize the features
- Estimate the features
- Schedule features into iterations

Iteration Planning

In 2 week cycles, the development team will:

- Define detailed tasks for each feature
- Sign up for tasks and re-estimate at the task level
- Test, Design, Code, repeat

Iteration scope and resources are flexible, but not time or quality

XP Planning

Release Plan

Release Point 1				Release Point 2				Release Point 3			
1 -----	5 -----	9 -----	13 -----	17 -----	21 -----	25 -----	29 -----	33 -----	37 -----	41 -----	45 -----
2 -----	6 -----	10 -----	14 -----	18 -----	22 -----	26 -----	30 -----	34 -----	38 -----	42 -----	46 -----
3 -----	7 -----	11 -----	15 -----	19 -----	23 -----	27 -----	31 -----	35 -----	39 -----	43 -----	47 -----
4 -----	8 -----	12 -----	16 -----	20 -----	24 -----	28 -----	32 -----	36 -----	40 -----	44 -----	48 -----
1	2	3	4	5	6	7	8	9	10	11	12

Features

Iterations

Iteration Plan

Feature 13
task
task
task
Feature 14
task
task
task
Feature 15
task
task
task
Feature 16
task
task
task

Release Plan contains ...

- **Small Releases** (1-3 months apart)
- Equal size iterations (2 weeks)
- Features (no detail developer tasks)
- Feature effort estimates

Iteration plan contains…

- Features
- Detail Developer tasks (less than 2 days)

XP Planning Jargon

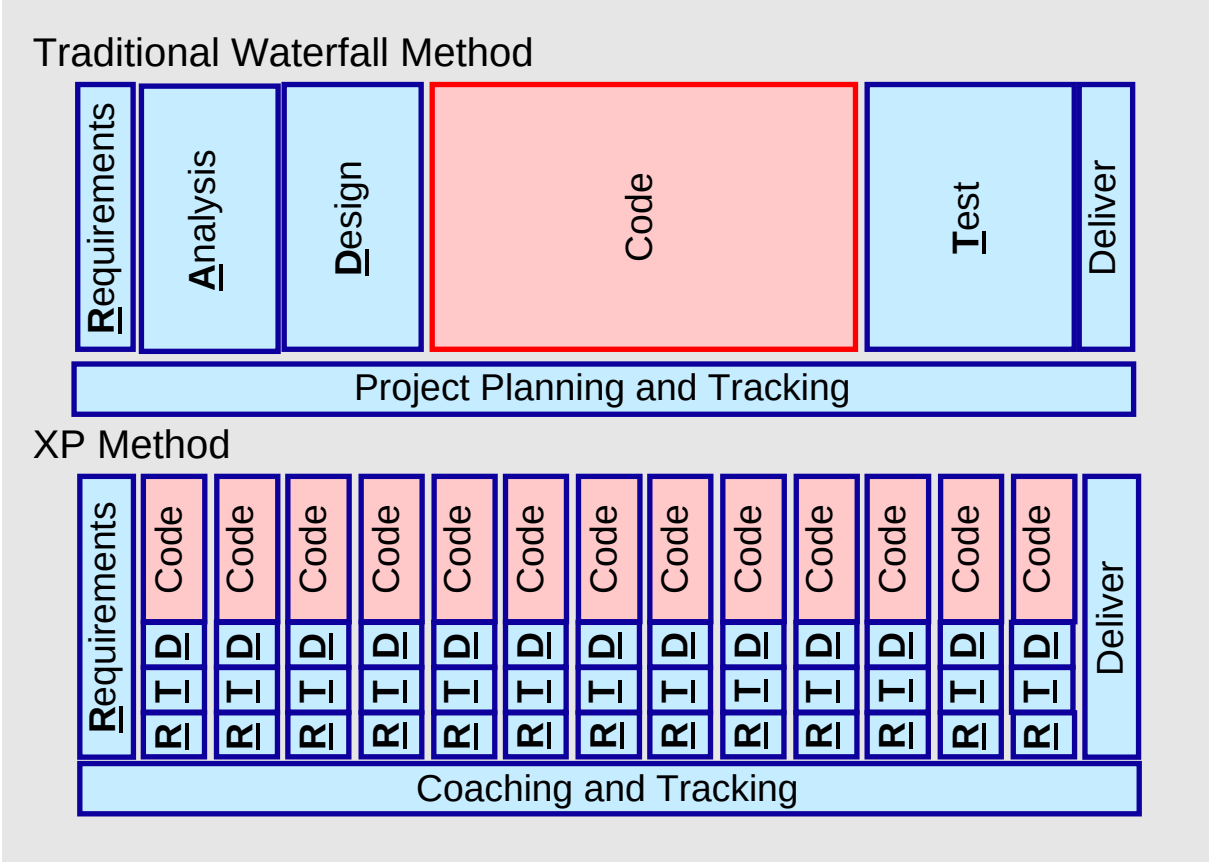
Features are estimated in Work Units. A work unit is pure coding time, not including:

- Design time
- Testing time
- "Spikes" (experiments)
- "Sand" (small bugs)
- Refactoring
- Non Programming tasks

Velocity is estimated in Work Units per Iteration.

Load Factor is Velocity per Available Programmer Days.

Load Factor Less Than 100%



Stand up meetings

Stand up meetings occur every morning at a scheduled time
VERY short meetings (goal 5 minutes or less)
Each team member briefly describes:

- Task status
- Issues
- Lessons learned
- Plan for the day

Task completion is recorded - Successes are celebrated
Software engineering principles and suggestions to improve velocity and methodology are briefly discussed
Iteration metrics are updated and posted near the task board - Project status is clearly visible

Stand up meetings

User Stories

User stories = lightweight use cases
2-3 sentences on a file card that:

- The customer cares about
- Can be reasonably tested
- Can be estimated & prioritized

Promises for Conversation

Stories are made up of two components

- The written card
- The series of conversations

•Between customer and programmers

The conversation will be captured as additional documentation that will be attached to the story

- Design sessions
- Acceptance tests
- Application code

Sample Stories (from XP Installed)

Union dues vary by union, and are taken only in the first pay period of the month. The system computes the deduction automatically. The amount is shown in the attached table.

Splitting a large story

Allow the user to add new service types to the system's initial list. For example, he may wish to add a special entry for getting the car washed at the high school's "free" wash. Include the standard fields amount and date, plus allow the user to add any additional text or numeric fields. Reports should automatically sum any numeric fields. (Programmer note: story needs splitting. Please separate text and numeric fields into two stories, plus one for the summing.)

(Split 1) Allow the user to add new service types, including the standard fields plus any additional text fields desired.

(Split 2) Allow the user to add numeric fields to user defined service types.

(Split 3) In all reports, show totals of all numeric fields, not just the standard gallons and dollar amount fields.

Process Union Dues Refunds
- Create format for transaction
- Add transaction to accepted transactions list
- Create Union Dues Refund Station

An Iteration Planning Practice

Gather around a whiteboard for the iteration plan

The Customer stands at the whiteboard

- Writes the name of one story on the board and describes what the story means

The team asks questions and the group discusses the story

- Until they are sure they get it, but not longer

The team brainstorm the task for the story

- Writing short names for the tasks on the whiteboard right under the story name. It might look like this:

Programmer Signs Up for Work

Programmers sign up for work to allow individuals or pairs to accept the primary responsibilities for completing specific work

Programmers estimate their own work

Programmers feel more commitment to work that is scheduled for completion in time they can believe in

Customer responsibilities

Need
Stories
Resources
Priorities
Acceptance

Developer responsibilities

Time estimates
Design
Code
Quality

Customer needs to know

How long
What's done
How good

Developer needs to know

- What to do
- When to do it
- When done

Promises To Programmers

They will be able to work on things that really matter, every day.

They won't have to face scary situations alone.

They will be able do everything in their power to make their system successful.

They will make decisions that they can make best, and they won't make decisions they aren't best qualified to make.

Promises to customers & managers

They will get the most possible value out of every programming week.

Every few weeks they will be able to see concrete progress on goals they care about.

They will be able to change the direction of the project in the middle of development without incurring exorbitant costs.

System Control Variables

Cost

- Often the most constrained variable
- Throwing more money at a problem does not always solve it

Time

- More time can improve quality and increase scope

Quality

- The most difficult control variable (not as easy to measure)
- External quality and internal quality

Scope

Variables Control

External forces (managers, customers) get to pick the value of 3 of the 4 variables

Development team picks the value of the 4th

Cost?

Time?

Quality?

Scope?

eXtreme Programming Practices

The Planning Game (quickly determine the scope of the next release. Priorities)

Short releases (release new versions in very short cycles)

Metaphor (find a simple metaphor describing how the system works)

Simple design (Make the design as simple as possible)

Testing (test the code continuously. Write the tests before the production code)

Refactoring (restructure the code to remove duplications, improve communication, simplify, or add flexibility)

eXtreme Programming Practices

Pair programming (two programmers at one machine)

Collective ownership (everyone owns and can change any code anywhere in the system)

Continuous integration (integrate and build the system many times a day)

Sustainable pace (don't work more than 40 hours a week)

Whole team (include a real user in the team)

Coding standards (use a coding standard to improve communication)

An Attempt to Compare Paradigms

	Traditional	Agile
Why	Method det.	Product vision
What	Model det.	Code
When	Sequential	Iterative
Who	Roles	Team
Where	Diff. places	Same place
How	Description	Acting
How much	Method det.	Cust. dec.