Software Development Methodologies

Lecturer: **Raman Ramsin**

Lecture 7

Integrated Object-Oriented Methodologies:

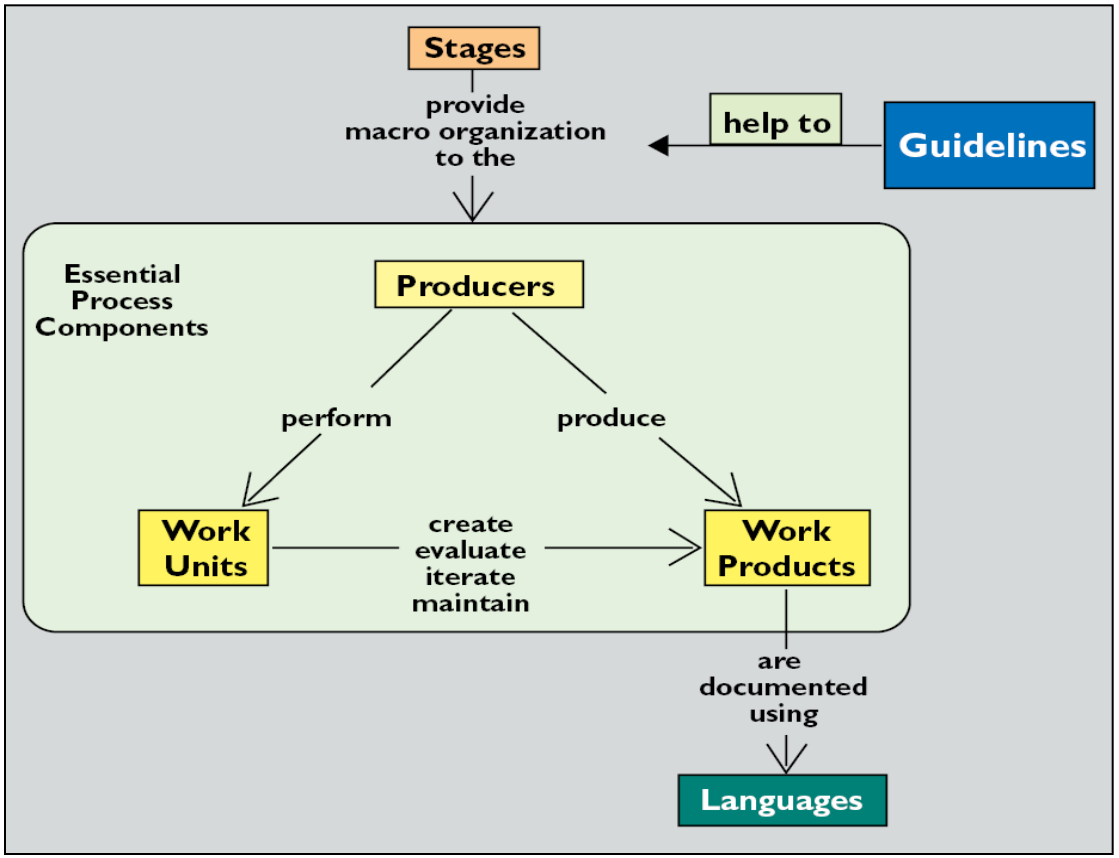
OPEN and FOOM

Object-oriented Process, Environment and Notation (OPEN)

- First introduced in 1996 as the result of the integration of four methodologies: MOSES, SOMA, Synthesis and Firesmith; later deeply influenced by BON and OOram
- Has its own modeling language, OML (OPEN Modeling Language), yet also supports UML
- Presented as a framework called OPF (OPEN Process Framework)
- Contains a component library from which individual process-component instances can be selected and put together to create a specific process instance

OPEN Process Framework (OPF)

- A process metamodel defining five classes of components and guidelines for constructing customized OPEN processes



For each element (represented by box), OPEN permits the user to select how many and which instances will be used. The OPF documentation provides a comprehensive list of suggestions on the best selections together with guidelines on their best organization.

[Firesmith and Henderson-Sellers 2001]

OPF: Component Classes

- *Work Products:* any significant thing of value (document, diagram, model, class, application) developed during the project.
- *Languages:* the media used to document work products, such as natural languages, modeling languages such as UML or OML, and implementation languages such as Java, SQL, or CORBA-IDL.
- *Producers:* active entities (human or nonhuman) that develop the work products.
- *Work Units:* operations that are performed by producers when developing work products. One or more *producers* develop a *work product* during the execution of one or more *work units*.
- *Stages:* durations or points in time that provide a high-level organization to the work units.

OPF: Work Units

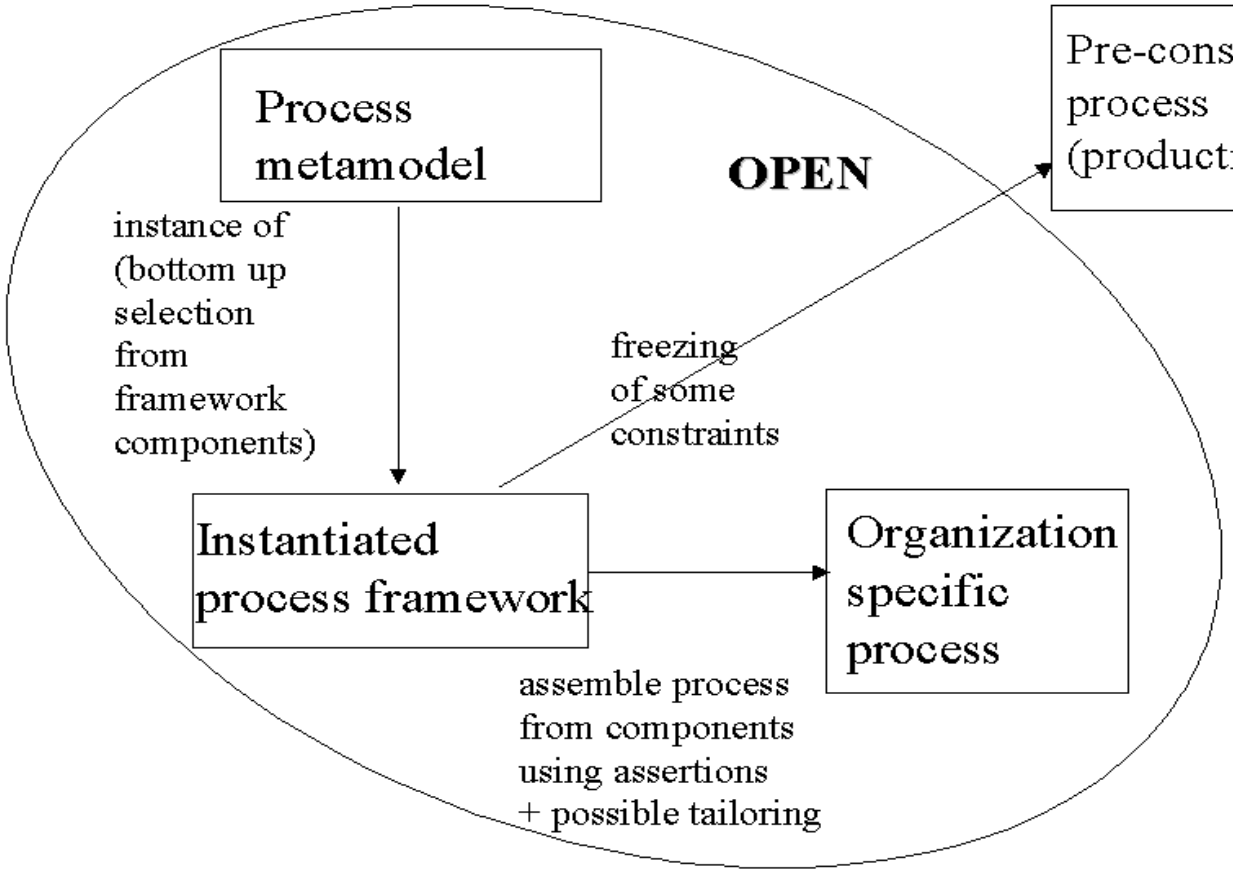
- ***Activity:***
  - a major work unit consisting of a related collection of jobs that produce a set of work products
  - Coarse-grained descriptions of what needs to be done
  - Some important instances defined by OPEN are: Project Initiation, Requirements Engineering, Analysis and Model Refinement, Project Planning, and Build
- ***Task:***
  - Smallest atomic unit of work
  - Small-scale jobs associated with and comprising the activities
  - Resulting in the creation, modification, or evaluation of one or more work products
- ***Technique:***
  - Define how the jobs are to be done
  - Ways of doing the tasks and activities

OPEN: Process Instantiation

- The following tasks are performed (through applying the guidelines proposed by OPF) in order to instantiate, tailor and extend an OPEN process:

1. *Instantiating* the OPEN library of predefined component-classes to produce actual process components
2. *Choosing* the most suitable process components from the set of instantiated components
3. *Adjusting* the fine detail inside the chosen process components
4. *Extending* the existing class library of predefined process components to enhance reusability

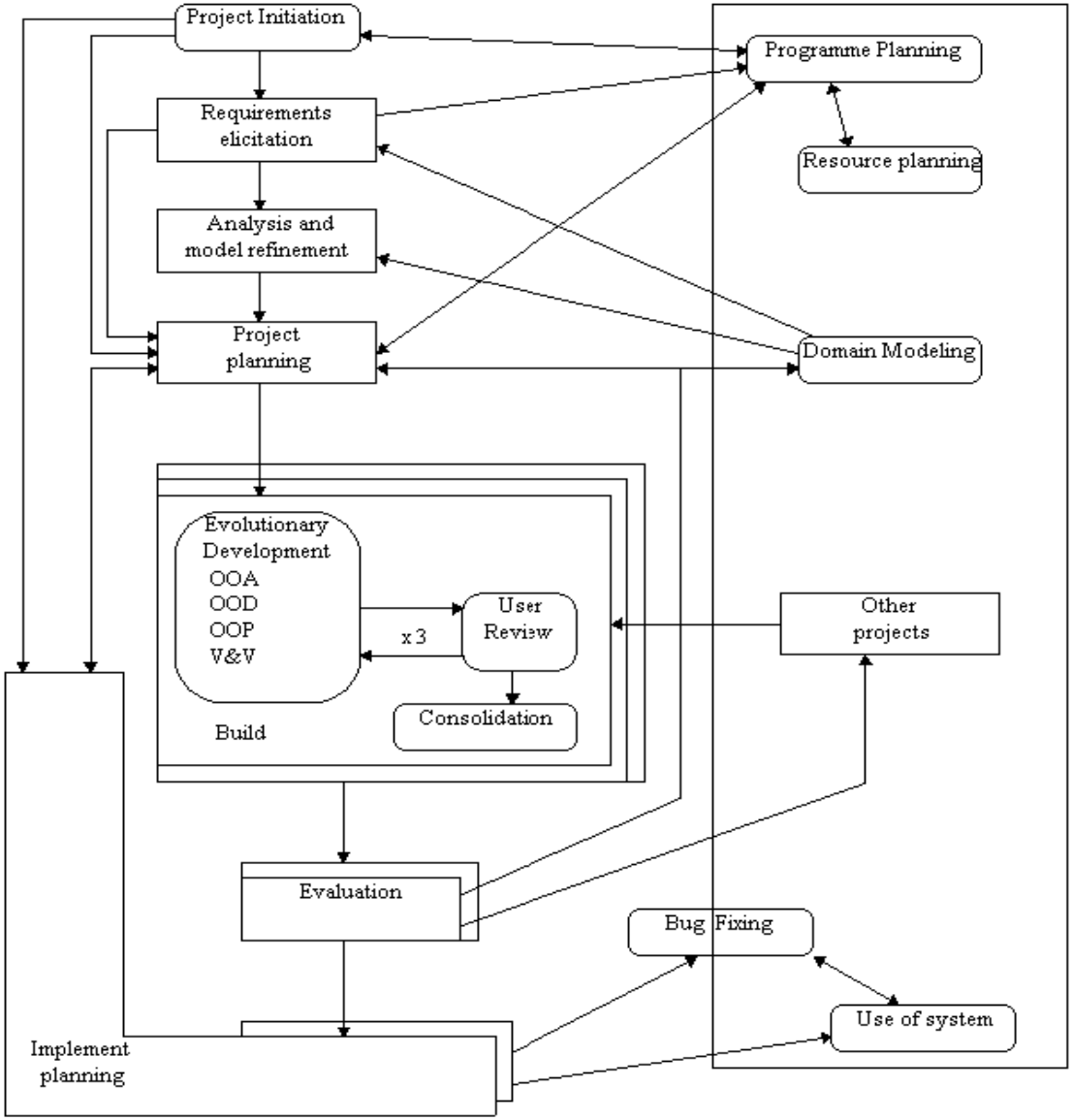
OPEN: Process Instantiation



It is even possible to take sets of constraints to create "pre-tailored" styles of OPEN process

[Firesmith and Henderson-Sellers 2001]

OPEN: Process Instance



[Graham et al. 1997]

OPEN: Strengths and Weaknesses

- ***Strengths***
  - Flexibility and configurability due to the framework definition of the process
  - Well-defined framework (generally and in detail) for instantiating tailored-to-fit processes
  - Accommodates seamless process configurations
  - Accommodates process configurations supporting traceability
  - Accommodates various lifecycles, including iterative-incremental

OPEN: Strengths and Weaknesses

- ***Strengths* (Contd.)**
  - Covers enterprise-level activities and business-process-reengineering
  - Incorporates a rich library of process components
  - Provides guidelines as to how customized processes should be built (especially how stages should be structured and organized)
  - Accommodates comprehensive modeling at all levels (problem domain to objects; logical to physical)
  - Rich modeling-language support (UML and OML)

OPEN: Strengths and Weaknesses

- ***Weaknesses***
  - As a result of merging various methodologies, OPEN is not a specific methodology, but rather a process framework; in trying to remain noncommittal to any single process, it has lost concreteness.
  - OPEN is huge and complex; many developers tend to use typical instances introduced by the authors rather than instantiate their own.
  - The developer is responsible for constructing the methodology; even though OPEN prescribes the framework, components, and guidelines as to how to construct the process, bad instances *can* be built (very much like a Lego game).

Functional and Object-Oriented Methodology (FOOM)

- Introduced in 2001 by Shoval and Kabeli; a refined and more complete version was introduced by Shoval in 2007.
- An object-oriented variant of Shoval's ADISSA methodology of 1988 (ADISSA: Architectural Design of Information Systems based on Structured Analysis).
- Strives to combine the classical process-oriented approach of ADISSA with the object-oriented paradigm.
- Based on and driven by *transactions* – a notion very similar to the use case.
- Mainly targeted at data-intensive information systems.

FOOM: Process

- **Analysis:** concerned with requirements elicitation and problem-domain modeling; two activities performed in parallel or iteratively:
  - *Data Modeling:* modeling the class structure of the problem domain.
  - *Functional Modeling:* identifying/modeling functional requirements.
  - *Data Elaboration:* detailing the data elements in a Data Dictionary.

- **Design:** concerned with designing implementation-specific classes and adding structural and behavioural detail to the models:
  - *Discovery and Top-level Design of Transactions: transactions* are chains of processes performed in response to external stimuli.
  - *UI Design:* designing a menu-based user interface for the system, and defining the relevant classes.
  - *Input/Output Design:* designing the input forms/screens and the output reports/screens, and defining the relevant classes.
  - *Design of System Behaviour:* providing detailed specifications for the transactions, detailing object interactions and class operations.

FOOM: Process

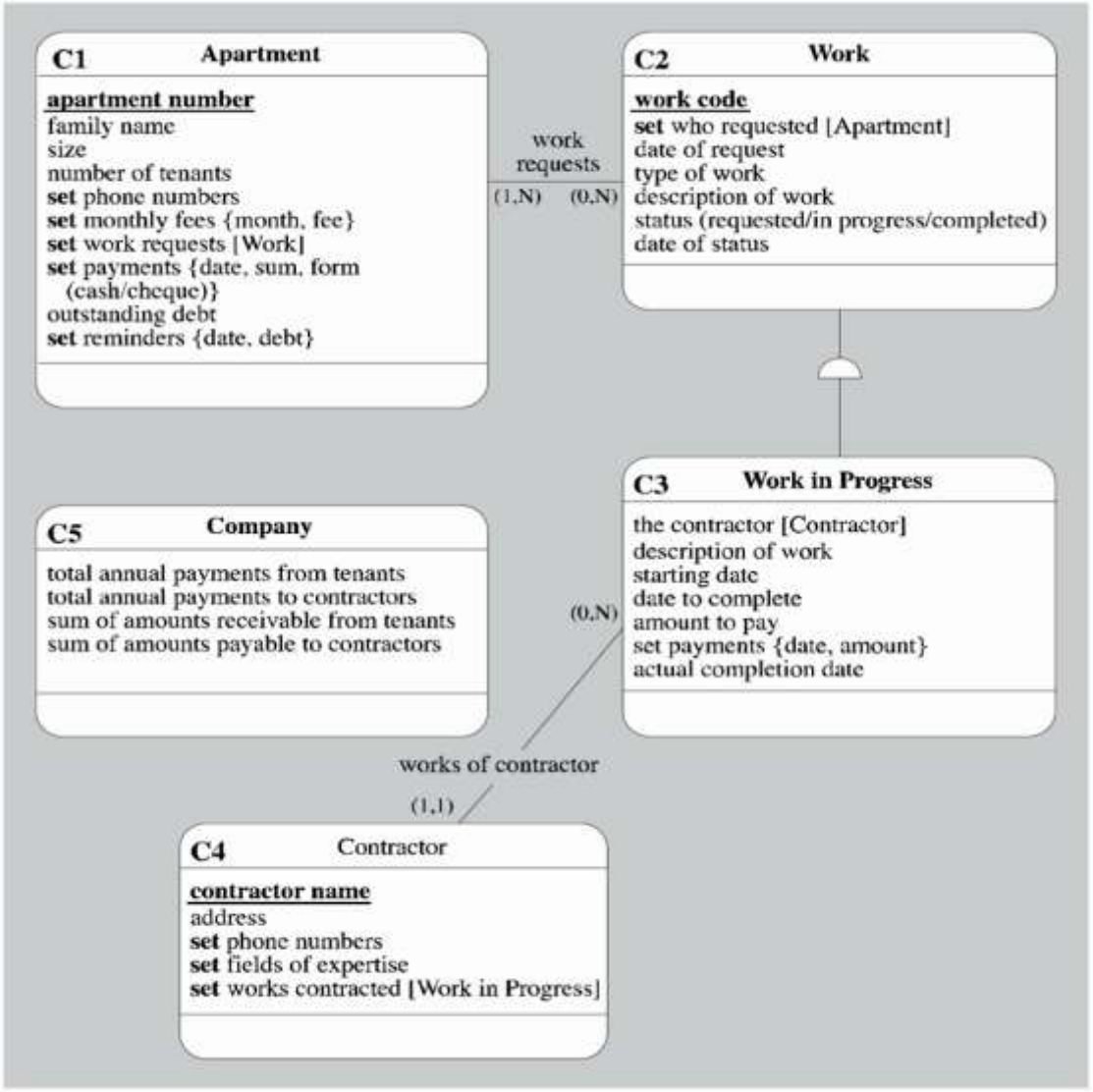
1. Analysis phase	
a) Data modeling	Initial class diagram
b) Functional modeling	Hierarchical OO-DFDs
c) Data dictionary	
2. Design phase	
d) Discovering transactions	Top-level descriptions of transactions
e) Designing the user interface	Menus class and objects
f) Designing the inputs and outputs	- Forms class and objects - Reports classes and objects
g) Creating detailed descriptions of transactions and their decomposing into methods	- Transactions class - Transaction methods and class methods—in pseudo code or message charts

[Shoval 2007]

FOOM: Analysis – *Data Modeling*

- **Data Modeling**
  - Problem-domain classes are identified along with their attributes and relationships.
  - Results are modeled in a *Class Diagram.*
  - Initial class diagram does not include the operations of the classes; methods are added during design.

FOOM: Analysis – Data Modeling – *Class Diagram*

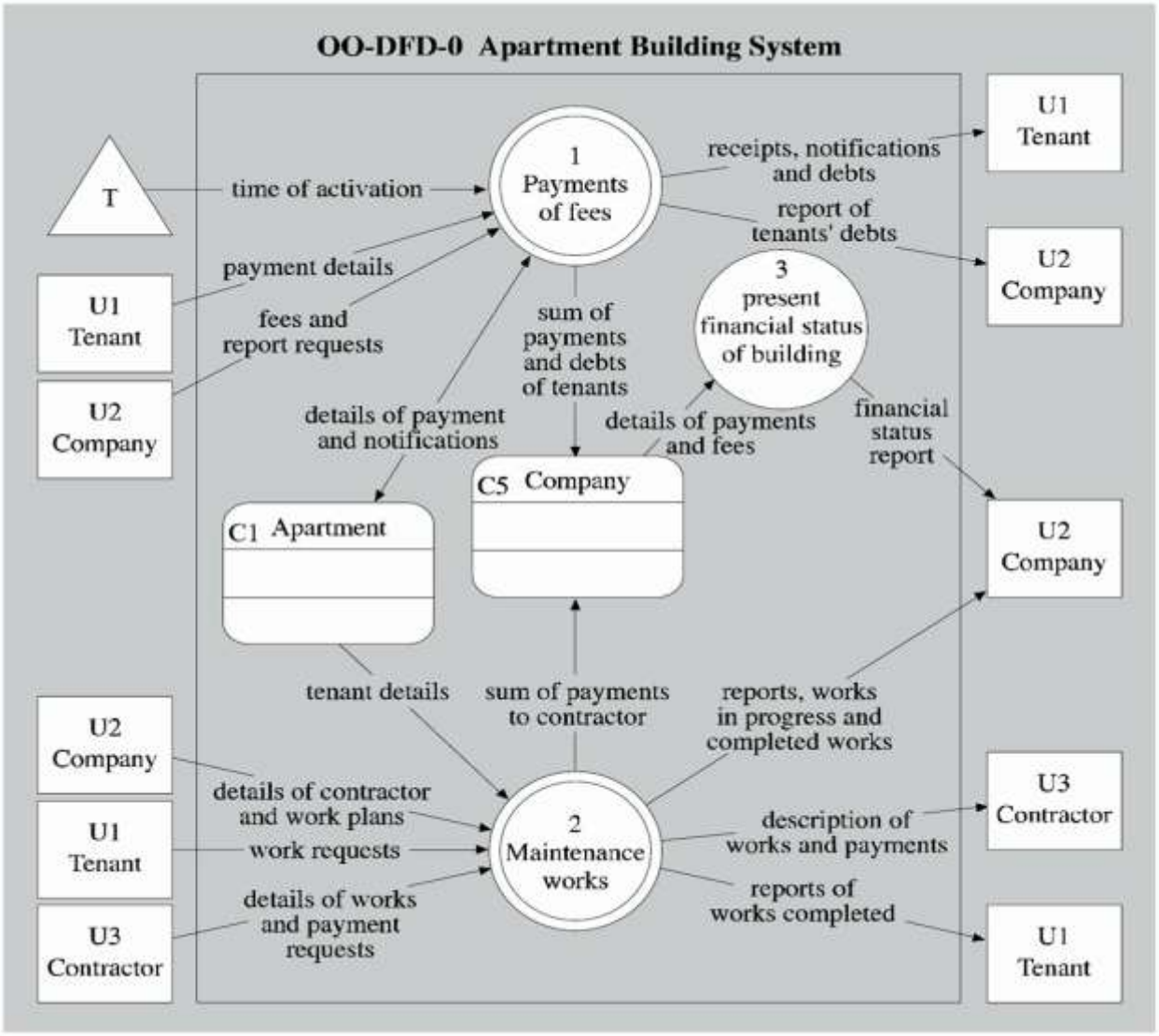


[Shoval 2007]

FOOM: Analysis – *Functional Analysis*

- **Functional Analysis**
  - Functional requirements of the system are elicited and modeled in a hierarchy of *Object-Oriented Data Flow Diagrams* (OO-DFDs).
  - *Classes* replace traditional *data stores.*
  - *External entities* have been expanded to include:
    - *User entities:* interact with human actors.
    - *Time entities:* act as modeling proxies for clocks.
    - *Real-time entities:* act as generators of asynchronous sensor events from the system environment.
    - *Communication entities:* represent other systems interacting with our system via communication channels.

FOOM: Analysis – Functional Analysis – *OODFD*



[Shoval 2007]

FOOM: Analysis – *Data Elaboration*

- **Data Elaboration:** A Data Dictionary is created during the analysis phase:
  - A Data Dictionary is a database or repository of data containing details about the components of the object oriented data flow diagram (OODFD):
    - Processes
    - External entities
    - Classes
    - Data elements carried by data flows.
  - It will be updated and extended throughout the design phase to include details about design products.

FOOM: Design – *Transaction Discovery and Design*

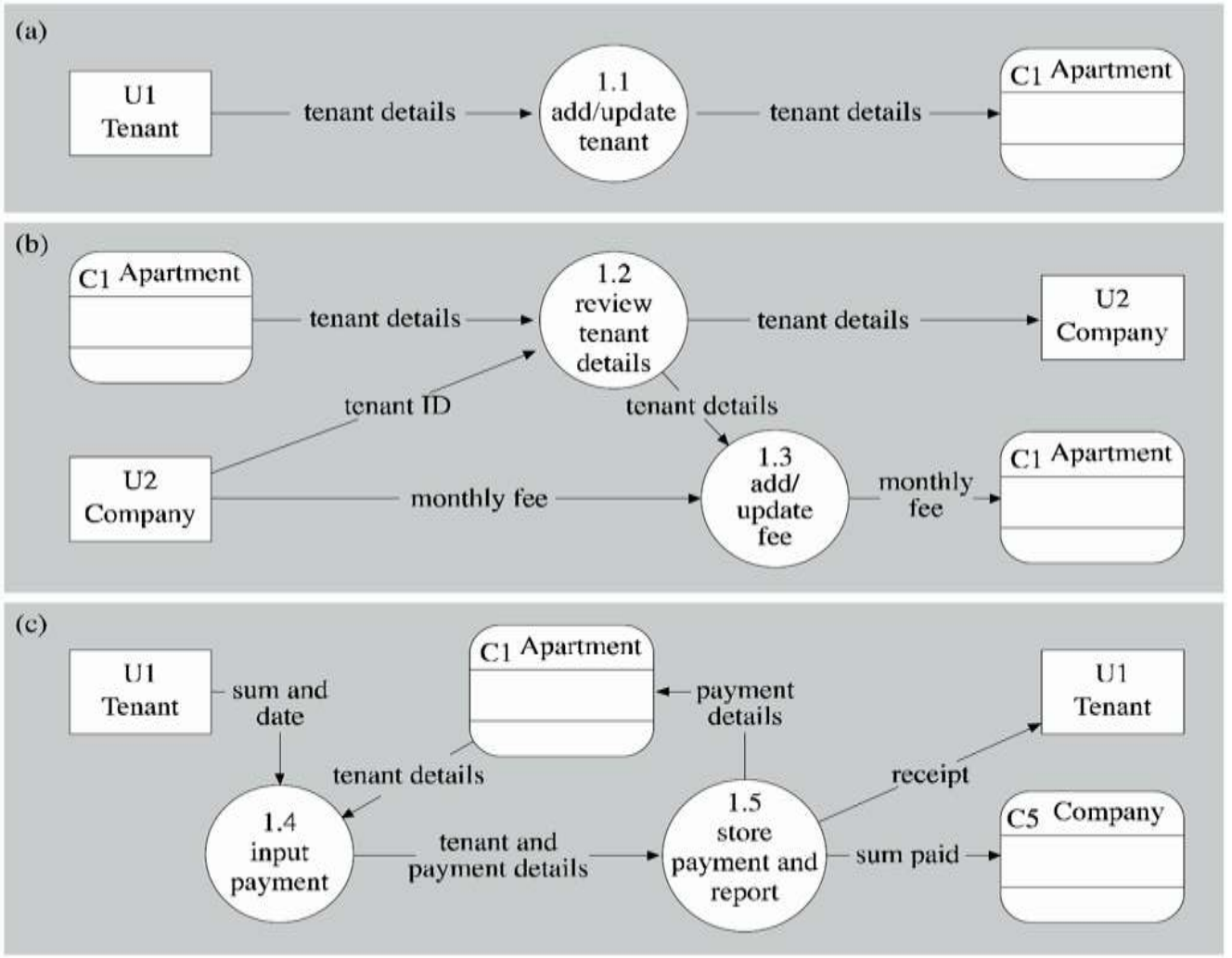
- ***Transaction***
  - Analogous to the modern-day use case.
  - A unit of functionality performed by the system in direct support of an external entity.
  - Triggered (initiated) as a result of an *event,* originating from an external entity.
- **Transaction Discovery and Design:** consists of three activities:
  1. *Identification of transactions:* the transactions of the system are identified from the hierarchy of analysis OO-DFDs.
  2. *Description of transactions:* top-level transaction descriptions are provided in a structured language.
  3. *Definition of the "Transaction" class:* a "Transaction" class is added to the class diagram, acting as a *utility class.*

FOOM: Design – *Transaction Discovery and Design (1)*

- **Identification of transactions**
  - Transactions of the system are identified from the hierarchy of OO-DFDs constructed during analysis.
  - The OO-DFD hierarchy is traversed in order to isolate the transactions:
    - Each transaction consists of one or more chained leaf processes, and the data classes and external entities connected to them.
    - Each transaction has one or more external entities at one end and data classes and/or external entities at the other.

FOOM: Design – *Transaction Discovery and Design (1)*

Identification of transactions



[Shoval 2007]

FOOM: Design – *Transaction Discovery and Design (2)*

- **Description of transactions**
  - A top-level transaction description is provided in a structured language referring to all the components of the transaction:
    - Every data-flow from or to an external entity is translated to an "Input from..." or "Output to..." line.
    - Every data-flow from or to a data class is translated to a "Read from..." or "Write to..." line.
    - Every data-flow between two processes translates to a "Move from... to..." line.
    - every process in the transaction translates into an "Execute function..." line.
    - The process logic of the transaction is expressed by using standard structured programming constructs.
  - The descriptions are extensively used during later stages of design as a basis for designing the application-specific features.

FOOM: Design – *Transaction Discovery and Design (2)*

Description of transactions

Begin transaction 1.1 *(The ID of the transaction is made of the numbers of its functions.)*

Input from U1 Tenant: type of operation desired *(It is assumed that an input screen is displayed and the user is asked to select "add" or "update" a tenant.)*

If action = "add" then input from U1 Tenant: new tenant details to add

Else *(if action = "update")* then input from U1 Tenant: tenant details to update *(It is assumed that upon selection of "update" the function first asks the user to input an apartment number or tenant family name; based on that it retrieves the details of that tenant and presents them on the input screen/form, enabling the user to key in the changes. Note that the transaction diagram does not show a "read' dataflow from class C1 to function 1.1; it is implied because, as we know, any add or update activity is preceded by a find/read activity.)*

Execute F1.1: Add/update tenant details

Write to C1 Apartments: new or updated tenant details

End transaction.

[Shoval 2007]

FOOM: Design – *Transaction Discovery and Design (3)*

- **Definition of the "Transaction" class**
  - A "Transaction" class is added to the class diagram acting as a *utility class.*
  - The "Transaction" class encapsulates operations for implementing the process logic of complex transactions.
  - Operations of this class will be revised during the last stage of design:
    - Transactions that are not deemed suitable to be assigned to ordinary classes due to their over-complexity remain in this class as operations.

FOOM: Design – *UI Design*

- The OO-DFD hierarchy is traversed in a top-down fashion in order to produce the menu-based interface of the system:
  - A main menu, initially empty, is defined for the system.
  - for each process at the topmost level of the hierarchy that is connected to a user entity, a menu-item is defined and added to the main menu.
  - at any level of the OO-DFD hierarchy, for every non-leaf process connected to a user entity, a corresponding submenu is defined and initialized as empty.
  - for every process (leaf or non-leaf) that is connected to a user entity a menu-item is defined and added to its parent-process's submenu.
  - The menu tree derived is then refined into the user-interface of the system.
- In order to realize this interface, a "Menu" class is defined and added to the class diagram of the system.
  - Instances of this class will be the run-time menus, with their items saved as attribute values.

FOOM: Design – *Input/Output Design*

- The descriptions of the transactions are used for determining input forms/screens and output reports/screens:
  - An input form/screen will be designed for each "Input from" line appearing in the transaction descriptions.
  - An output report/screen will be designed for each "Output to" line.
- Two new classes, the "Form" class for the inputs and the "Report" class for the outputs, are added to the class diagram.
  - The actual screens, forms, and reports are instances of these classes, with the titles and data-fields stored as attribute values.

FOOM: Design – *Design of System Behaviour*

- **Design of System Behaviour**

  1. Top-level transaction descriptions are mapped into detailed descriptions.
  2. Detailed transaction descriptions are decomposed into various methods which are then attached to proper classes.
  3. Methods are described using *pseudo code* or *message charts.*

FOOM: Design – *Design of System Behaviour (Activity 1)*

- **Detailing transaction descriptions**
  - Every "execute function..." command is replaced by a detailed description of the function, according to its internal process logic.
  - Every "input from U..." command is replaced by a reference to a predefined input screen or a different input device, as defined for that input.
  - Every "output to U…" command is replaced by a reference to a predefined output screen or a different output device, as defined for that output.
  - Every "read from C..." command is replaced by a more detailed command that includes the conditions for retrieval (if any).
  - Every "write to C…" command states that data of a certain object or objects need to be updated.
  - Every "move to F…" command indicates the activation of the next function.

FOOM: Design – *Design of System Behaviour (Activity 2)*

- **Method identification and assignment**
  - Each "Input from User…" is translated into a message to the Display method of the Forms class, including the ID of the specific object.
  - Each "Output to User…" is translated into a message to the Display method of the Reports class, including the ID of the specific object.
  - Each "Read from Class…" is translated into a message to the method of that class, whose task is to search and retrieve one object or more.
  - Each "Write to Class…" is translated into a message to the method of that class, whose task is to perform the specific Write command.
  - A detailed description of a transaction may include commands which describe a certain procedure (corresponding to an "Execute function…"):
    - Determine if it can be defined as a specific method that can be detached from the "main" part of the transaction and attached to one of the classes involved.
    - If a procedure cannot be assigned to any class, it remains in the transaction.

FOOM: Design – *Design of System Behaviour (Activity 3)*

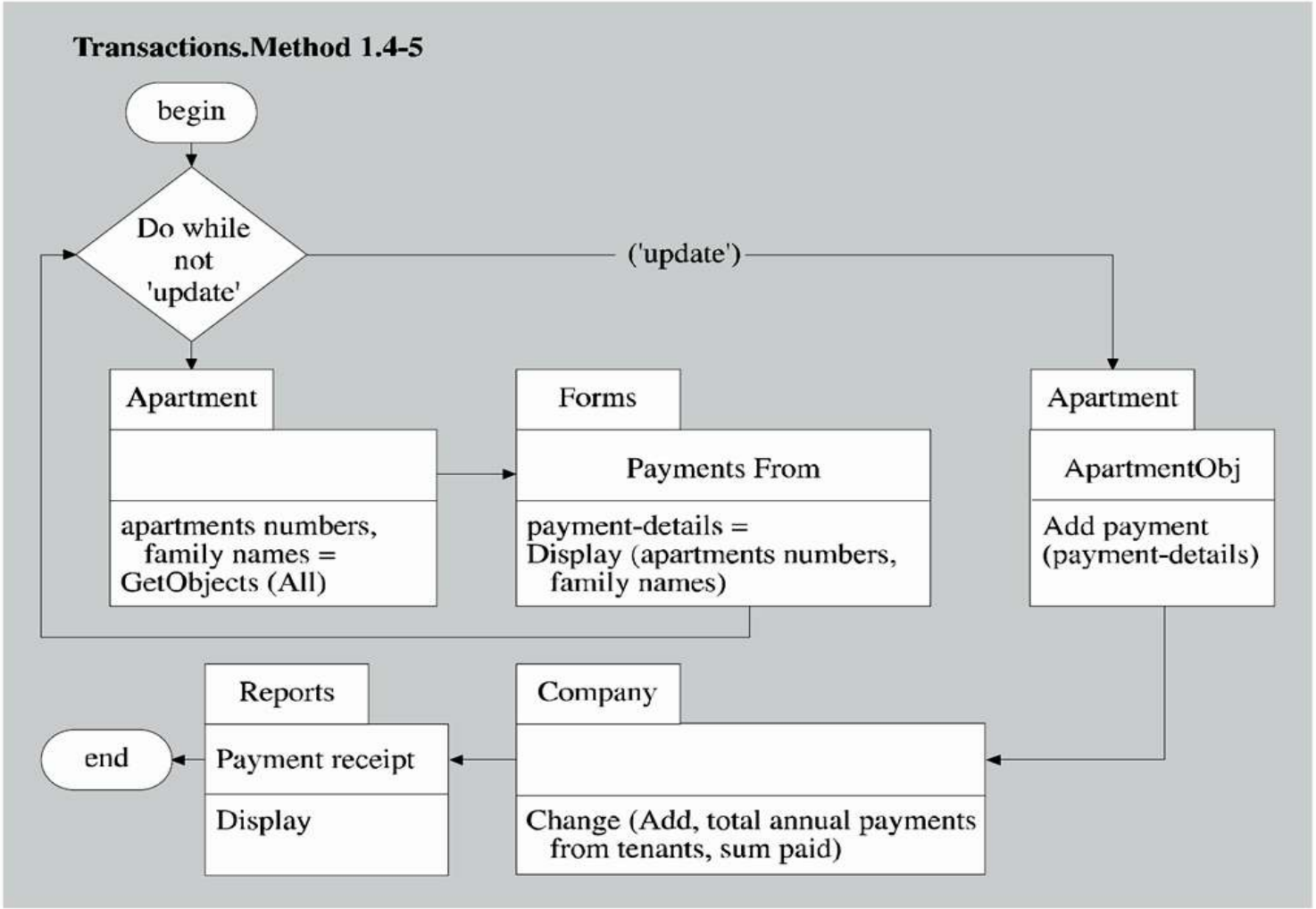
Describing methods – ***Pseudo Code***

```
1.  Begin Transaction.Method 1.4/5
2.  Do while "update" was not chosen:
3.      apart-numbers, family-names = Apartment.GetObjects (All)
4.      payment-details = Forms.payment form.Display (apart-numbers, family-names)
5.  End while
6.  Apartment.Add payment (payment-details)
7.  Company.Change (Add, total annual payments from tenants, sum paid)
8.  Reports.Payment receipt.Display
9.  End.
```

[Shoval 2007]

FOOM: Design – *Design of System Behaviour (Activity 3)*

Describing methods – ***Message Chart***



[Shoval 2007]

FOOM: Strengths and Weaknesses

- ***Strengths***

  - Based on functional and structural modeling of the problem domain and the system
  - Traceability to requirements (via *transactions*)
  - Appealing to domain experts and the SA/SD community (due to the popularity of DFDs)
  - Attention to interface design and I/O design based on the *transactions* identified and the OO-DFDs

FOOM: Strengths and Weaknesses

- ***Weaknesses***
  - No implementation, deployment and maintenance phases
  - Only suitable for data-intensive information systems
  - Seamlessness suffers because OO-DFDs are not exactly object-oriented
  - The process is vague as to how *operations* and *transactions* extracted from the OO-DFDs are assigned to classes
  - No modeling of logical architecture and physical configuration
  - Poor behavioural modeling (performed only in later stages of design at the inter-object level)
  - Lack of formalism
  - The issue of design-level refinements to the *Data Model* (class diagram) is not properly addressed (only "Form", "Menu", "Report", and "Transaction" classes are added)

References

- Graham, I., Henderson-Sellers, B., Younessi, H., *The OPEN Process Specification*. Addison-Wesley, 1997.
- Firesmith, D., Henderson-Sellers, B., *The OPEN Process Framework: An Introduction*. Addison-Wesley, 2001.
- Shoval, P., *Functional and Object Oriented Analysis and Design : An Integrated Methodology.* Idea Group Publishing, 2007.