Software Characteristics

- Software is a logical system element (instead of a physical element)
  - It is not restricted by any physical laws
  - It lives in unlimited dimensions
  - Can only be described via man-made models
- Modern day software systems are huge and complex
  - Abstraction is the key to combat complexity
  - Need models that highlights the essential properties while omitting inessential details
- Software behavior can only be visualized through execution/simulation within the operational context
  - Need model that captures the operational context
The Unified Modeling Language (UML)

- A modeling language
- Graphical notations for specifying, visualizing and constructing the artifacts of software systems

Multiple View of Software

- **Structural View**
  - Class diagrams
  - Object diagrams
  - Package diagrams
  - Composite Structure diagrams

- **Behavioral View**
  - Sequence diagrams
  - Communication diagrams
  - Statechart diagrams
  - Activity diagrams

- **User View** (operational context)
  - Use Case diagrams

- **Implementation View**
  - Component diagrams
  - Deployment diagrams

- **Environment View**
Class Diagrams

- Provide an overview of the problem domain or target system by describing the objects and classes inside the system and the relationships between them.
A Software Design Model

- Annotate visibilities in Design Class Diagrams via Associations and Dependency relationship

**Design Object Classes**

<table>
<thead>
<tr>
<th>Store</th>
<th>Register</th>
<th>ProductCatalog</th>
<th>ProductSpec</th>
<th>SalesLineItem</th>
<th>TaxCalculator</th>
<th>Payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>address, name</td>
<td>create(pc: ProductCatalog, tc: TaxCalculator)</td>
<td>getDescription()</td>
<td>getPrice()</td>
<td>getDesc()</td>
<td>getTax(s: Money)</td>
<td></td>
</tr>
<tr>
<td>addSale(s: Sale)</td>
<td>makeNewSale()</td>
<td>getSpec(id: ItemID)</td>
<td>getID()</td>
<td>getTotal()</td>
<td>amount: Money</td>
<td></td>
</tr>
<tr>
<td></td>
<td>enterItem(itemID: ItemID, quantity: natural)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>endSale() : Money</td>
<td>loadSpec()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>makeCashPayment(amount: Money)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sale**

- date
- time
- isCompleted
- create(pc: ProductCatalog, tc: TaxCalculator)
- makeNewSale()
- enterItem(itemID: ItemID, quantity: natural)
- endSale(tc: TaxCalculator)
- makeCashPayment(amount: Money)
Another Example

Object Diagram
Package Diagrams

- Shows a hierarchical containment of classes

```
OS
 FileSystem
  FileManager  File  Directory

MemorySystem
  MemoryManager  PageTable  MemoryPage
```

Composite Structure Diagram

- Shows the internal structure (including parts and connectors).

```
Sensor
  : CueingCapsule
  : OrientationCapsule
  : TrackFormingCapsule
  : SFPInterfaceCapsule
```
Multiple View of Software

- **Structural View**
  - Class diagrams
  - Object diagrams
  - Package diagrams
  - Composite Structure diagrams
  - Sequence diagrams
  - Communication diagrams
  - Statechart diagrams
  - Activity diagrams

- **Implementation View**
  - Component diagrams
  - Deployment diagrams

- **User View (operational context)**
  - Use Case diagrams

- **Behavioral View**
  - Component diagrams

- **Environment View**
  - Component diagrams

Component Diagrams

- Component diagrams
  - Captures reusable/replaceable parts of the software with well-defined interfaces

```
interfaces provided

interfaces required
```

components
Multiple View of Software

- Structural View
  - Class diagrams
  - Object diagrams
  - Package diagrams
  - Composite Structure diagrams
  - Sequence diagrams
  - Communication diagrams
  - Statechart diagrams
  - Activity diagrams
- Implementation View
  - Component diagrams
- User View (operational context)
  - Use Case diagrams
  - Deployment diagrams
- Behavioral View
- Environment View

Deployment Diagrams

- Models the run time configuration in a static view and visualizes the distribution of components in an application.
Multiple View of Software

- Class diagrams
- Object diagrams
- Package diagrams
- Composite Structure diagrams
- Sequence diagrams
- Communication diagrams
- Statechart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams

Sequence Diagrams

- Models the interactions between objects based on a time sequence.

```
makeNewSale()

loop [more items]
    enterItem(itemID, quantity)
    description, subtotal

endSale()

................total with taxes
makePayment(amount)
change due, receipt
```
Another Example

\begin{align*}
\text{draw} & : \text{Deck} \rightarrow \text{Hand} \\
\text{loop} \quad [i := 1..n] \\
\text{create} & \rightarrow \text{create} \rightarrow \text{create} \rightarrow \text{create} \\
\text{addCard(c)} & \text{addCard(c)} \\
\text{removeCard(c)} & \text{size} := \text{size} + 1 \\
\text{return(h)} &
\end{align*}

Communication Diagrams

- Models the interactions between objects
- Time sequencing expressed via message sequence

\begin{align*}
\text{draw} & : \text{Hand} \\
\text{loop} \quad [i := 1..n] \\
\text{create} & \rightarrow \text{create} \rightarrow \text{create} \rightarrow \text{create} \\
\text{addCard(c)} & \text{addCard(c)} \\
\text{removeCard(c)} & \text{size} := \text{size} + 1 \\
\text{return(h)} &
\end{align*}

Note:
Steps 3, 4, and 5 are inside the body.
<table>
<thead>
<tr>
<th>Type</th>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence diagram</td>
<td>Clearly shows sequence or time ordering of messages</td>
<td>Forced to extend to the right when adding new objects; consumes horizontal space</td>
</tr>
<tr>
<td>Communication diagram</td>
<td>Space economical – flexibility to add new objects in two dimension</td>
<td>More difficult to see sequence of messages, fewer notation options</td>
</tr>
</tbody>
</table>

Statecharts

- Models dynamic behavior of an entity that is not only a direct consequence of its inputs, but it also depends on its preceding state.
More Example

Waiting for Data

Processing Data

Logging Data

Data Process

Waiting to Accept alarmRaised

Waiting to Accept alarmRaised

Idle

User Monitoring

Alarm Process

token multiplicity
Activity Diagrams

- An activity diagram shows control flow within a system

- An activity diagram is a special case of a state chart diagram in which states are activities ("functions")

Activity Diagrams (cont’d)

- Two types of states:
  - *Action state*:
    - Cannot be decomposed any further
    - Happens “instantaneously” with respect to the level of abstraction used in the model
  - *Activity state*:
    - Can be decomposed further
    - The activity is modeled by another activity diagram
Statechart Diagram vs. Activity Diagram

Statechart Diagram for Incident
(State: Attribute or Collection of Attributes of object of type Incident)

Activity Diagram for Incident
(State: Operation or Collection of Operations)

Activity Diagram: Modeling Decisions
Activity Diagrams: Modeling Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads

Activity Diagrams: Swimlanes

- Actions may be grouped into swimlanes to denote the object or subsystem that implements the actions.
Multiple View of Software

Structural View
- Class diagrams
- Object diagrams
- Package diagrams
- Composite Structure diagrams

Implementation View
- Component diagrams
- Deployment diagrams

Behavioral View
- Sequence diagrams
- Communication diagrams
- Statechart diagrams
- Activity diagrams

User View (operational context)
- Use Case diagrams

Use Cases

A use case describes a sequence of actions a system performs that yields a result of value to a particular actor.
Grady Booch

- User-centric/Goal Oriented
- Sketch these out as you elicit information
- Identify all the actors
  - Keep an actor list
- What does system do for the user
- Same use case, many scenarios
  - What does that mean?
Use Cases For Requirements

- A communication tool
  - for interaction with stakeholder
  - Language is simple
- Describe
  - existing system behavior
  - expected behavior of new system
- Express functional requirements
  - not quite as useful for non-functional requirements
- Let stakeholder participate in creation

Use Case - Overview

- Actors
- System
- Stimulus – Response
- Information
  - A set of events
  - System responses
  - Observable result or value
  - Some particular actor
  - Variants (i.e. alternate flow of events)
Scenario

- One use case may have many scenarios
- Each scenario consists of a sequence of atomic actions that results in some observable results
- Consider the notion of a complete dialogue

*Use cases provide an effective way to develop and manage scenarios*

Use Case Diagrams

- A graphical representation of the relation between actors and use cases
  - Show the intend use of the system (system requirements) without considering how the system implements anything
  - Provide a bird’s eye view of the scenario set
- Presents a clear view of the system boundary
When Point-of-Sales System is the Boundary

- POS
  - UC-1. Process Sales
  - UC-2. Handle Returns
  - UC-3. Log in

Cashier → POS → Customer

When Entire Store is Boundary

- Store
  - UC-1. Process Sales
  - UC-2. Handle Returns

Customer → Store
Use Case Format

- Many formats are possible
- Fundamental
  - Use Case Name (and unique identifier)
  - Actors
  - Brief description
  - Flow of events (Stimulus – Response)
  - Boundary (Pre-/Post-conditions)
- Optional
  - Other stakeholders
  - Non-functional requirements/constraints

Use Case Types

- Essential Use Case
  - terse one paragraph summary
- Casual Use Case
  - multiple paragraphs that covers various scenarios
- Expanded Use Case
  - all steps and variations are written in detail, with supporting sections
  - mainly used during elaboration phase to firm up requirements

Note: Defer all details to as late as possible.
A Point-of-Sales System Example

A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store.

It includes hardware components such as a computer and bar code scanner, and software to run the system.

It interfaces to various services applications, such as a third-party tax calculator and inventory control.

A Brief Use Case

Use case: UC-1. Process Sale
Actors: Customer, Cashier (primary)
Description: A customer arrives at the checkout with items to purchase. The cashier uses the POS system to record the purchase items. The system presents a running total and line-item details. The cashier collects payment from the customer and enters the information into the POS system. On completion, the customer receives a receipt from the system and then leaves with the purchased items.
A Casual Use Case

Use case: UC-2. Handle Returns
Actors: Customer, Cashier (primary)

Description:

Main Scenario:
A customer arrives at the checkout with items to return. The cashier uses the POS system to record each return item ...

Alternate Scenario:
If the item identifier is not found in the system, notify the Cashier and suggest manual entry of identifier code. If they pay by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.

An Expanded Use Case

Use case: …
Primary Actor: …
Stakeholders and Interest: …
Entry conditions (a.k.a. Pre-condition): …
Trigger condition: …
Exit conditions (Minimum and Success guarantees): …
Flow of events:
1.…
2.…
…

Exceptions:
2a. …

Special Requirements: …

Technology and Data Variation List: …

Open Issues: …
An Expanded Use Case (cont’d)

**Use case:** UC-1. Process Sales

**Primary Actor:** Cashier

**Stakeholders and Interest:**
- Cashier: wants accurate, fast entry ...
- Customer: wants accurate, fast service ...
- Salesperson: wants sales commission updated.
- Company: wants accurate record of transactions, automatic updates of inventory, ...

**Entry conditions:**
- Cashier is identified and authenticated

**Trigger condition:**
- Customer arrives at POS checkout ...

---

An Expanded Use Case (cont’d)

**Exit conditions:**

*Minimum guarantee:*
- Sales is recorded. Tax is correctly calculated.

*Success guarantee:*
- Sales is recorded. Tax is correctly calculated.
- Accounting and inventory updated. ...

**Flow of events:**
1. Cashier starts a new sale.
2. Cashier enters item identifier.
3. System records sale line item and ...

*Cashier repeats steps 2-3 until indicated done.*
An Expanded Use Case (cont’d)

4. System presents total with taxes calculated
5. Cashier informs Customer the total, asks for payment.
6. Customer pays and Cashier enters payment information into POS.
7. System logs completed sale, sends payment information to the external accounting systems (for accounting and commission) and inventory system (to update inventory).
8. System presents receipt.
9. Customer leaves with receipts and goods (if any).

Exceptions:
*a. When system fails:
   1. To support recovery and correct accounting, ...
3a. Invalid Identifier:
   1. System signal error and reject entry.
3b. There are multiple of same item category:
   1. Cashier enters quantity and then item identifier.
3-6a. Customer asks Cashier to remove an item from the purchase:
   1. Cashier enters item identifier for removal from sale.
   2. System displays running total.
...
An Expanded Use Case (cont’d)

Special Requirements:
... quality attributes like performance, reliability, etc.

Technology and Data Variation List:
... technical constraints on implementation,
  detailed description of data attributes

Open Issues:
...

Mapping use case scenarios to activity diagrams
**Use case:** UC-1. Record Message  
**Primary Actor:** Phone Line  
**Stakeholders and Interest:**  
- User: wants reliable message recording when not available  
- Phone Company: wants phone line hangs up properly when no in use  
**Entry conditions:**  
Answering machine is on. An incoming call is detected.  
**Trigger condition:**  
Some one calls the Phone/Answering Machine  
**Minimum/Success guarantee:**  
Phone line hangs up properly after use. Message recorded to completion.  

### Main Scenario:

1. The phone line rings five times.  
2. The answering machine answers the call.  
3. The phone line stops ringing.  
4. The answering machine plays the announcement.  
5. When the announcement is completed, the machine beeps for five seconds, then records the incoming message.  
6. When the phone line signals the end of message, the answering machine stops recording, sends a hang-up signal to phone line and returns to idle state.
Alternate Scenarios:

1a. The phone line rings three times. The phone line then stops ringing and notifies the answering machine that the call has been answered.

1. The machine remains at the idle state. (I.e., the answering machine only answers calls after the phone line rings five times.)

Use case: UC-2. Retrieve Message
Primary Actor: Phone Line
Stakeholders and Interest:
- User: wants ability to retrieve message anywhere, anytime
- Phone Company: wants phone line hangs up properly when no in use

Entry conditions:
- Answering machine is on. An incoming call is detected.

Trigger condition:
- Some one calls the Phone/Answering Machine

Minimum/Success guarantee:
- Phone line hangs up properly after use. Message successfully played to completion.
Main Scenario:
1. The phone line rings five times.
2. The answering machine answers the call. The phone line stops ringing.
3. The answering machine plays the announcement.
4. When the announcement is completed, the machine beeps the phone line.
5. The machine receives a "*" tone from the phone line within the five-second beeping period.
6. The machine stops beeping and plays all recorded messages.
7. It then sends a hang-up signal to phone line and returns to idle state.
Questions?