Structured vs. Object Orient Analysis and Design

SAD vs. OOSAD

Kelvin K. Omieno

June 2016
Outline

- SAD Phases
- OOAD Phases
- SAD vs. OOAD software development
- Adopted Books
- UML in practice
- Conclusions & Recommendations
<table>
<thead>
<tr>
<th>Textbooks</th>
<th>Object Oriented Systems Analysis and Design</th>
<th>Software Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeffrey Hoffer</td>
<td>2nd edition</td>
<td>Lan Summerville</td>
</tr>
<tr>
<td>Joey George</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Joseph Valacich</td>
<td>Joey George</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dinesh Batra</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Joseph Valacich</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jeffrey Hoffer</td>
<td></td>
</tr>
</tbody>
</table>

Visual Modeling with Rational Rose 2002 and UML, 3/E, by Terry Quatrani

Learning UML 2.0, By Kim Hamilton, Russell Miles, O'Reilly, 2006.
In practice - UML software architecture and design description, IEEE Software, 2006


<table>
<thead>
<tr>
<th></th>
<th>Structured</th>
<th>Object-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methodology</strong></td>
<td>SDLC</td>
<td>Iterative/Incremental</td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td>Processs</td>
<td>Objects</td>
</tr>
<tr>
<td><strong>Risk</strong></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Reuse</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Maturity</strong></td>
<td>Mature and widespread</td>
<td>Emerging (1997)</td>
</tr>
<tr>
<td><strong>Suitable for</strong></td>
<td>Well-defined projects with stable user requirements</td>
<td>Risky large projects with changing user requirements</td>
</tr>
<tr>
<td>Phase</td>
<td>Structured</td>
<td>Object-Oriented</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Analysis</td>
<td>Structuring Requirements</td>
<td>Requirement Engineering</td>
</tr>
<tr>
<td></td>
<td>• DFDs</td>
<td>• Use Case Model (find Uses Cases, Flow of Events, Activity Diagram)</td>
</tr>
<tr>
<td></td>
<td>• Structured English</td>
<td>• Object Model</td>
</tr>
<tr>
<td></td>
<td>• Decision Table / Tree</td>
<td>• Find Classes &amp; class relations</td>
</tr>
<tr>
<td></td>
<td>• ER Analysis</td>
<td>• Object Interaction: Sequence &amp; collaboration Diagram, State Machine Diagram, Object to ER Mapping</td>
</tr>
<tr>
<td>Design</td>
<td>• DB design</td>
<td>• Physical DB design</td>
</tr>
<tr>
<td></td>
<td>• (DB normalization)</td>
<td>• Design elements</td>
</tr>
<tr>
<td></td>
<td>• GUI Design</td>
<td>• Design system Architecture</td>
</tr>
<tr>
<td></td>
<td>• (forms &amp; reports)</td>
<td>• Design classes: Checking The Model, Combine Classes, Splitting Classes, Eliminate Classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GUI design</td>
</tr>
</tbody>
</table>
Definitions

- **Systems Analyst**
  - Responsible for analysis and design of information systems

- **Software**
  - Computer programs and associated documentation such as requirements, design models and user manuals

- **Software Engineering**
  - IEEE standard 610-12 (1999) defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."
SW Project phases

Any project in the world has the following phases:

- **Planning**
- **Analysis**: System requirements are studied and structured
- **Design**: Recommended solution is converted into logical and then physical system specifications
  - **Logical design** – All functional features of the system chosen for development in analysis are described independently of any computer platform
  - **Physical design** – The logical specifications of the system from logical design are transformed into the technology-specific details from which all programming and system construction can be accomplished
- **Implementation**
- **Testing**
- **Maintenance**
Outline

- SAD Phases
- OOAD Phases
- SAD vs. OOAD software development
- Adopted Books
- UML in practice
- Conclusions & Recommendations
Structured Analysis and design (SAD)

A. Analysis Phase
1. Determine system requirements
2. Structuring system process requirements
3. Logical requirements (logical modeling)
4. Structuring system data requirements

B. Design Phase
1. Database design (DB normalization)
2. Forms and report design (GUI design)
Structured Analysis and design (SAD)

A. Analysis Phase

1. Determine system requirements:
   - Interviewing: individuals and/or group

2. Structuring system process requirements
   - Data Flow Diagram (DFD) – logical process modeling
   - DFD levels (process decomposition)
   - Context diagram
   - 4 type of DFD
     - Current physical: Adequate detail only
     - Current logical: Enables analysts to understand current system
     - New logical: Technology independent, Show data flows, structure, and functional requirements of new system
     - New physical: Technology dependent
Comparison of DeMarco and Yourdon and Gane and Sarson DFD symbol sets
Context-level data flow diagram showing project scope for Purchasing Fulfillment System (Pine Valley Furniture)
Context Diagram

Context diagram of Hoosier Burger’s food-ordering system
Developing DFDs (Cont.)

- **Level-0 diagram** is a data flow diagram that represents a system’s major processes, data flows, and data stores at a high level of detail.
  
  - Processes are labeled 1.0, 2.0, etc. These will be decomposed into more primitive (lower-level) DFDs.
Level-0 Diagram

Level-0 DFD of Hoosier Burger’s food-ordering system
**Data Flow Diagramming Rules**

**Process:**
A. No process can have only outputs. It is making data from nothing (a miracle). If an object has only outputs, then it must be a source.
B. No process can have only inputs (a black hole). If an object has only inputs, then it must be a sink.
C. A process has a verb phrase label.

**Data Store:**
D. Data cannot move directly from one data store to another data store. Data must be moved by a process.
E. Data cannot move directly from an outside source to a data store. Data must be moved by a process that receives data from the source and places the data into the data store.
F. Data cannot move directly to an outside sink from a data store. Data must be moved by a process.
G. A data store has a noun phrase label.

**Source/Sink:**
H. Data cannot move directly from a source to a sink. It must be moved by a process if the data are of any concern to our system. Otherwise, the data flow is not shown on the DFD.
I. A source/sink has a noun phrase label.
Data Flow Diagramming Rules (Cont.)

Data Flow:

J. A data flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated, however, by two separate arrows because these happen at different times.

K. A fork in a data flow means that exactly the same data goes from a common location to two or more different processes, data stores, or sources/sinks (this usually indicates different copies of the same data going to different locations).

L. A join in a data flow means that exactly the same data come from any of two or more different processes, data stores, or sources/sinks to a common location.

M. A data flow cannot go directly back to the same process it leaves. There must be at least one other process that handles the data flow, produces some other data flow, and returns the original data flow to the beginning process.

N. A data flow to a data store means update (delete or change).

O. A data flow from a data store means retrieve or use.

P. A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

(Source: Adapted from celko, 1987.)
Decomposition of DFDs

- **Functional decomposition** is an iterative process of breaking a system description down into finer and finer detail.

  ◦ Creates a set of charts in which one process on a given chart is explained in greater detail on another chart.
  
  ◦ Continues until no subprocess can logically be broken down any further.
Decomposition of DFDs (Cont.)

- **Primitive DFD** is the lowest level of a DFD.
- **Level-1 diagram** results from decomposition of Level-0 diagram.
- **Level-n diagram** is a DFD diagram that is the result of $n$ nested decompositions from a process on a level-0 diagram.
Level-1 DFD

Level-1 diagram showing the decomposition of Process 4.0 from the level-0 diagram for Hoosier Burger’s food-ordering system.

Processes are labeled 4.1, 4.2, etc. These can be further decomposed in more primitive (lower-level) DFDs if necessary.

Level-1 DFD shows the sub-processes of one of the processes in the Level-0 DFD.

This is a Level-1 DFD for Process 4.0.
Level-\(n\) DFD

Level-2 diagram showing the decomposition of Process 4.3 from the level-1 diagram for Process 4.0 for Hoosier Burger’s food-ordering system.

Processes are labeled 4.3.1, 4.3.2, etc. If this is the lowest level of the hierarchy, it is called a *primitive DFD*.

Level-\(n\) DFD shows the sub-processes of one of the processes in the Level \(n-1\) DFD.

This is a Level-2 DFD for Process 4.3.
Four Different Types of DFDs

- **Current Physical**
  - Process labels identify technology (people or systems) used to process the data.
  - Data flows and data stores identify actual name of the physical media.

- **Current Logical**
  - Physical aspects of system are removed as much as possible.
  - Current system is reduced to data and processes that transform them.
Four Different Types of DFDs (Cont.)

- **New Logical**
  - Includes additional functions.
  - Obsolete functions are removed.
  - Inefficient data flows are reorganized.

- **New Physical**
  - Represents the physical implementation of the new system.
3. Logical requirements (logical modeling)
   - Use structured English to represent DFD because DFD does not show logic
   - Use decision table / tree (logical choice in conditional statement)

4. Structuring system data requirements
   - ER diagram
Modeling Logic with Decision Tables

- **Decision table**: a matrix representation of the logic of a decision which specifies the possible conditions for the decision and the resulting actions.

- Best used for complicated decision logic.
Complete decision table for payroll system example

<table>
<thead>
<tr>
<th>Conditions/Courses of Action</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee type</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Hours worked</td>
<td>&lt;40</td>
</tr>
<tr>
<td></td>
<td>&lt;40</td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>&gt;40</td>
</tr>
<tr>
<td></td>
<td>&gt;40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Stubs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pay base salary</td>
<td>X</td>
</tr>
<tr>
<td>Calculate hourly wage</td>
<td>X</td>
</tr>
<tr>
<td>Calculate overtime</td>
<td></td>
</tr>
<tr>
<td>Produce absence report</td>
<td>X</td>
</tr>
</tbody>
</table>
Modeling Logic with Decision Tables (Cont.)

- **Condition stubs**: that part of a decision table that lists the conditions relevant to the decision

- **Action stubs**: that part of a decision table that lists the actions that result for a given set of conditions
Modeling Logic with Decision Tables (Cont.)

- **Rules**: that part of a decision table that specifies which actions are to be followed for a given set of conditions

- **Indifferent condition**: in a decision table, a condition whose value does not affect which actions are taken for two or more rules
Modeling Logic with Decision Tables (Cont.)

- Procedure for Creating Decision Tables
  - Name the condition and the values that each condition can assume.
  - Name all possible actions that can occur.
  - List all possible rules.
  - Define the actions for each rule.
  - Simplify the table.
### Modeling Logic with Decision Tables

(Cont.)

#### Reduced decision table for payroll system example

<table>
<thead>
<tr>
<th>Conditions/ Courses of Action</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Employee type</td>
<td>S</td>
</tr>
<tr>
<td>Hours worked</td>
<td>−</td>
</tr>
<tr>
<td>Pay base salary</td>
<td>X</td>
</tr>
<tr>
<td>Calculate hourly wage</td>
<td>X</td>
</tr>
<tr>
<td>Calculate overtime</td>
<td></td>
</tr>
<tr>
<td>Produce Absence Report</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>−</td>
<td>&lt;40</td>
<td>40</td>
<td>&gt;40</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B. Design Phase

1. Database design (DB normalization)
2. Forms and report design (GUI design)
**Normalization**: the process of converting complex data structures into simple, stable data structures

The result of normalization is that every nonprimary key attribute depends upon the whole primary key.

**First Normal Form (1NF)**
- Unique rows, no multivalued attributes
- All relations are in 1NF

**Second Normal Form (2NF)**
- Each nonprimary key attribute is identified by the whole key (called full functional dependency)

**Third Normal Form (3NF)**
- Nonprimary key attributes do not depend on each other (i.e. no transitive dependencies)
Object Oriented Analysis and Design

OOAD
Object-Oriented Analysis and Design (OOAD)

- Based on objects rather than data or processes
- **Object**: a structure encapsulating attributes and behaviors of a real-world entity
Object-Oriented Analysis and Design (OOAD)

(Cont.)

- **Object class**: a logical grouping of objects sharing the same attributes and behaviors

- **Inheritance**: hierarchical arrangement of classes enable subclasses to inherit properties of superclasses
Outline

- SAD Phases
- OOAD Phases
- SAD vs. OOAD software development
- Adopted Books
- UML in practice
- Conclusions & Recommendations
A. Analysis Phase

- Structuring requirements (Use cases)
- Conceptual data modeling (class diagram)
- Object relationship modeling
  - Class diagram $\rightarrow$ ER diagram

- Analysis classes
  - Class stereotypes
  - Sequence diagram
  - Communication diagram
  - activity diagram
  - State machine diagram
B. Design Phase

- Physical DB design
- Design elements
  - Design classes
  - Design components
  - Design system Architecture
- GUI design
Learning UML textbook

Focus on 4+1 view architecture

Modeling Requirements: Use Cases

Modeling System Workflows: Activity Diagrams

Modeling a System's Logical Structure: Introducing Classes and Class, Sequence State Machine Diagrams

Managing and Reusing Your System's Parts: Component, Package, Deployment Diagrams
Visual modeling with rational rose
text book

- Focus on Rational Unified Process (RUP)
- Talk about 4+1 architectural view Later on the textbook
- Rational Rose Example
OOAD project phases

( my reading and experience )

Analysis

- Requirement gathering, analysis, and modeling (Requirement Engineering)
- Use Case Model find Uses Cases, Flow of Events, Activity Diagram
- Object Model
  - Find Classes & class relations,
  - Object Interaction: Sequence & collaboration Diagram, State Machine Diagram,
- Object to ER Mapping

Design

- Physical DB design
- Design elements
- Design system Architecture
- Design classes: Checking The Model, Combine Classes, Splitting Classes, Eliminate Classes
- Design components
- GUI design
Use Cases Examples
Use Case Diagram in the ESU Course Registration System

- Student
  - Register For Courses
- Billing system
- Registrar
  - Select Courses to teach
  - Request Course Roster
- Professor
  - Select Courses to teach
  - Request Course Roster
- Maintain Course Information
- Maintain Professor Information
- Maintain Student Information
- Create Course Catalog
Use Case: Clinic System Example

- Patient
- Scheduler
- Doctor
- Clerk

- Clinic
  - Cancel Appointment
  - Make Appointment
  - Check Patient Record
  - Request Medication
  - Defer Payment
  - Pay Bill
    - Extension points
      - More Treatment
    - Bill Insurance
Use Case: Bank System Example

- Update Customer Database
- Bank Computer

- Service ATMs
  - Technician
  - Employees
  - Bank Teller
  - Withdraw Money
  - Deposit Money
  - Withdraw Cash from ATM
  - Deposit Cash at ATM
  - Customer
  - Process a Loan
  - Apply for Loan
Activity diagram Examples
Activity Diagram for registration system

Registrar

Create curriculum

Create catalog

Place catalog in bookstore

Mail catalog to students

Open registration

[ Registration time period expired ]

Close registration

Professor

Select courses to teach

Swimlanes
Activity Diagram for back system
Activity Diagram for shipment system
Finding classes (thinking in objects)  
(Registration System)

**Entity class**

**Student**

**Boundary class (GUI interface)**

**Control class**

**Register For Courses**

**RegistrationManager**

```
addStudent(Course, Student)
```

**Course**

- name
- numberCredits
- open()
- addStudent(Student)
Class relations: Inheritance and Multiplicity
(Registration System)

- **RegistrationForm**
- **RegistrationManager**
  - `addStudent(Course, Student)`
- **Professor**
  - `tenureStatus`
- **Student**
  - `major`
  - `0..4` multiplicities
  - `open()`
  - `addStudent(Student)`
- **Course**
  - `0..*` multiplicities
  - `name`
  - `numberCredits`
  - `open()`
  - `addStudent(Student)`
- **CourseOffering**
  - `0..4` multiplicities
  - `location`
  - `open()`
  - `addStudent(Student)`
- **ScheduleAlgorithm**
Relationships

- Three types of relationships are:
  - Association
  - Aggregation
  - Dependency
public class BlogAccount {

    // Attribute introduced thanks to the association with the BlogEntry class
    private BlogEntry[] entries;

    // ... Other Attributes and Methods declared here ...
}

public class BlogEntry {

    // The blog attribute has been removed as it is not necessary for the
    // BlogEntry to know about the BlogAccount that it belongs to.

    // ... Other Attributes and Methods declared here ...
}
Object-Relational Modeling

• Purposes of Object-Relational Modeling
  – Create entity classes
  – Produce database structures
  – Enhance and finalize the attributes in the data model
Object-Oriented Extensions to Relational Modeling

- Generalization
- Multivalued attributes (OK to violate atomicity requirement of 1NF)
- Aggregation
- Object identifiers
- Pointers
- Behaviors
- Richer set of data types
Translating Conceptual Data Model to Object-Relational Model

- Translate classes
- Translate relationships
- Normalize object relations
- Merge object relations
Supertype/subtype relationships

- EMPLOYEE
  - Employee_Number
  - Employee_Name
  - Address
  - Date_Hired
  - Employee_Type

Employee_Type =

- "H" (HOURLY EMPLOYEE)
  - Hourly_Rate

- "S" (SALARIED EMPLOYEE)
  - Annual_Salary
  - Stock_Option

- "C" (CONSULTANT)
  - Contract_Number
  - Billing_Rate
Mapping Supertype/subtype relationships to relations

These are implemented as one-to-one relationships
A sequence diagram displays object interactions arranged in a time sequence:

1. Student fills in info
2. Student submits form
3. Registration form adds student to math 101
4. Math 101 adds student
5. Are you open?
6. Math 101 section 1 adds student
public class MessageReceiver {
    public void foo() {
        // Do something inside foo.
    }
}

public class MessageCaller {
    private MessageReceiver messageReceiver;

    // Other Methods and Attributes of the class are declared here
    // The messageReceiver attribute is initialized elsewhere in
    // the class.

    public doSomething(String[] args) {
        // The MessageCaller invokes the foo() method
        this.messageReceiver.foo(); // then waits for the method to return
        // before carrying on here with the rest of its work
    }
}
A collaboration diagram displays object interactions organized around objects and their links to one another.
Sequence Diagram for Bank System

public float getCashOnHand()

public void validateAccountInfo()

public void verifyCardWithBank(int stringCardStrip)

public char getConnected()
State Machine Diagram

여기서 수해

stdNum = roomSize

stdNum < roomSize

stdNum = roomSize

stdNum < 7 &&

انتهى فترة التسجيل

ملغية

انتهى فترة التسجيل
حالة الطالب

 المستوى 1

 مستوى 2

 مستوى 3

 مستوى 4

 خريج

 منقطع

 منتظم

 خريج

 إنهى 36 ساعة معتددة

 إنهى 72 ساعة معتددة

 إنهى 108 ساعة معتددة

 إنهى 142 ساعة معتددة

 State Machine Diagram

67
Showing Components Working Together
Focusing on the key components and interfaces in your system.

Focusing on component dependencies and the manifesting artifacts is useful when you are trying to control the configuration or deployment of your system.
Assembly connectors show components working together through interfaces
System Architecture
## Key Differences Between Structured and Object-Oriented Analysis and Design

<table>
<thead>
<tr>
<th></th>
<th>Structured</th>
<th>Object-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methodology</strong></td>
<td>SDLC</td>
<td>Iterative/Incremental</td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td>Processs</td>
<td>Objects</td>
</tr>
<tr>
<td><strong>Risk</strong></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Reuse</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Maturity</strong></td>
<td>Mature and widespread</td>
<td>Emerging</td>
</tr>
<tr>
<td><strong>Suitable for</strong></td>
<td>Well-defined projects with stable user requirements</td>
<td>Risky large projects with changing user requirements</td>
</tr>
</tbody>
</table>
Software Engineering textbook

Main topics

INTRODUCTION
  ◦ Socio-technical System
  ◦ Emergent property

REQUIREMENTS ENGINEERING
  ◦ Systems Models

DESIGN
  ◦ Architectural Design
  ◦ Application Architectures
  ◦ Object-oriented Design
  ◦ Real-time Systems
  ◦ User Interface Design

SOFTWARE DEVELOPMENT
  ◦ Iterative SW Development
  ◦ SW Reuse
  ◦ CBSE
  ◦ Critical Systems Development
  ◦ Software Evolution

VALIDATION
  ◦ Verification and Validation
  ◦ Software Testing
  ◦ Critical Systems Validation

MANAGEMENT
  ◦ Software Cost Estimation
  ◦ Quality Management
EMERGING TECHNOLOGIES

- Security Engineering
- Service-oriented Software Engineering
- Aspect-oriented Software Development
System categories

- **Technical computer-based systems**
  - Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system.
  - The system is not self-aware.

- **Socio-technical systems**
  - Systems that include technical systems but also operational processes and people who use and interact with the technical system.
  - Socio-technical systems are governed by organisational policies and rules.
Emergent properties

- Properties of the system of a whole that depend on the system components and their relationships.

Non-deterministic

- They do not always produce the same output when presented with the same input because the system’s behaviour is partially dependent on human operators.

Complex relationships with organisational objectives

- The extent to which the system supports organisational objectives does not just depend on the system itself.
Types of emergent property

- **Functional properties**
  - These appear when all the parts of a system work together to achieve some objective.
  - For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.

- **Non-functional emergent properties**
  - Examples are reliability, performance, safety, and security. These relate to the behaviour of the system in its operational environment.
  - They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.
Software Engineering textbook

The systems engineering process

Requirement definitions

System design

Subsystem development

System evaluation

System installation

System decomposition

System integration
Software Engineering textbook
Inter-disciplinary involvement

- Mechanical engineering
- Electrical Engineering
- Architecture
- Software engineering
- Civil Engineering
- Structural engineering
- User Interface design

ATC System Engineering
System Models

• Context models
  ◦ Blocks (box diagram of subsystems)
  ◦ DFDs can be used

• Behavioural models
  ◦ Blocks (box diagram of subsystems)
  ◦ Two types of behavioral model are:
    • Data processing models that show how data is processed as it moves through the system (DFD)
    • State machine models that show the systems response to events.

• Data models DFDs

• Object models
  ◦ Inheritance models
  ◦ Aggregation models
  ◦ Interaction models
<table>
<thead>
<tr>
<th>Phase</th>
<th>Structured</th>
<th>Object-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Structuring Requirements</td>
<td>Requirement Engineering</td>
</tr>
<tr>
<td></td>
<td>• DFDs</td>
<td>• <strong>Use Case Model</strong> (find Uses Cases, Flow of Events, Activity Diagram)</td>
</tr>
<tr>
<td></td>
<td>• Structured English</td>
<td>• <strong>Object Model</strong></td>
</tr>
<tr>
<td></td>
<td>• Decision Table / Tree</td>
<td>• Find Classes &amp; class relations</td>
</tr>
<tr>
<td></td>
<td>• ER Analysis</td>
<td>• Object Interaction: Sequence &amp; collaboration Diagram, State Machine Diagram,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Object to ER Mapping</td>
</tr>
<tr>
<td>Design</td>
<td>• DB design</td>
<td>• <strong>Physical DB design</strong></td>
</tr>
<tr>
<td></td>
<td>• (DB normalization)</td>
<td>• <strong>Design elements</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>GUI Design</strong></td>
<td>• Design system Architecture</td>
</tr>
<tr>
<td></td>
<td>• (forms &amp; reports)</td>
<td>• Design classes: Checking The Model, Combine Classes, Splitting Classes, Eliminate Classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Design components</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GUI design</td>
</tr>
</tbody>
</table>
In practice - UML software architecture and design description, IEEE Software, 2006


In practice - UML software architecture and design description, IEEE Software, 2006
Practitioner reflections on UML use

- 2+ month period, 80 architects participated. Major responsibilities among respondents:
  - analysis (66 %),
  - design (66 %),
  - specification (61 %), and
  - programming (52 %).

- The respondents came from different application domains.
  - Most worked in information systems (61 %)
  - 28 % worked in embedded systems
  - a few worked in tool and operating systems development.

- 60% worked in projects of more than 5 person-years.
Survey respondents’ use of UML for 4+1 architectural views
Survey respondents’ assessment of their adherence to the UML standard.
Deadline vs. completeness as stopping criteria for different project sizes.
Problems encountered due to incomplete models correlated to respondent demographic data regarding project size.
Problems with UML descriptions

- **Scattered information:** Design choices are scattered over multiple views.
  - some dependencies might show up in the logical view, while others appear in the process view.

- **Incompleteness:** The architects focus on what they think is important.

- **Inconsistency.**
  - UML-based software development is inevitably inconsistent.
  - Industrial systems are typically developed by teams.
  - Different teams can have different understandings of the system as well as different modeling styles, and this can lead to inconsistent models.
Other problem classes include the following:

- **Diagram quality.** UML lets architects represent one design in different ways.
  - they can decompose a diagram that contains too many elements into several **smaller diagrams**.
  - they can influence how easy the model is to **understand** and how it gets interpreted.

- **Informal use.**
  - Architects sometimes use UML in a very **sketchy** manner.
  - These diagrams **deviate** from official UML syntax, making their meaning ambiguous.

- **Lack of modeling conventions.**
  - case studies show that engineers use UML according to individual habits.
  - These habits might include layout conventions, commenting, visibility of methods and operations, and consistency between diagrams.
Defects in industrial UML models

Subjective impression obtained via the survey

Objective measurements about the quality of industrial UML models. (14 case studies of different sizes from various organizations and application domains)

<table>
<thead>
<tr>
<th>Case study</th>
<th>No. of classes</th>
<th>No. of person-years spent on modeling</th>
<th>No. of team members</th>
<th>Life stage of the model</th>
<th>Purpose of modeling</th>
<th>CMM level (estimated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>734</td>
<td>15</td>
<td>5</td>
<td>Final</td>
<td>Implementation</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>168</td>
<td>20</td>
<td>20</td>
<td>Unknown</td>
<td>Unknown</td>
<td>2–3</td>
</tr>
<tr>
<td>C</td>
<td>108</td>
<td>20</td>
<td>10</td>
<td>Unknown</td>
<td>Unknown</td>
<td>2–3</td>
</tr>
<tr>
<td>D</td>
<td>716</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>443</td>
<td>10</td>
<td>10</td>
<td>Final</td>
<td>Unknown</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>Final</td>
<td>Unknown</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>75</td>
<td>10</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>478</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>705</td>
<td>12</td>
<td>6</td>
<td>Development</td>
<td>Implementation</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td>51</td>
<td>12</td>
<td>6</td>
<td>Development</td>
<td>Analysis</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>14</td>
<td>1–5</td>
<td>2</td>
<td>Inception</td>
<td>Analysis</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>46</td>
<td>1–5</td>
<td>2</td>
<td>Final</td>
<td>Implementation</td>
<td>2</td>
</tr>
<tr>
<td>M</td>
<td>73</td>
<td>0–5</td>
<td>1</td>
<td>Final</td>
<td>Tutorial, abstraction of real world</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>359</td>
<td>5</td>
<td>5</td>
<td>Semifinal</td>
<td>Implementation</td>
<td>1–2</td>
</tr>
</tbody>
</table>
Defects found in the case studies

- Methods that are not called in sequence diagrams
- Classes not occurring in sequence diagrams
- Objects without names
- Messages not corresponding to methods
- Classes without methods
It is important to investigate the benefits obtained from modeling.

- this paper reports on **controlled experiments**, spanning two locations, that investigate the impact of UML documentation on software **maintenance**.

- Results show that:
  - for complex tasks and past a certain learning curve, the availability of UML documentation **may** result in **significant improvements** in the functional correctness of changes as well as the quality of their design.
  - there does not seem to be any **saving of time**. For **simpler** tasks, the time needed to update the UML documentation **may be substantial** compared with the potential benefits, thus motivating the **need for UML tools** with better support for software maintenance.
## Tested Hypotheses

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time excluding diagram modifications</td>
<td>$H_0: T(UML) \geq T(no\ UML)$</td>
<td>$H_a: T(UML) &lt; T(no\ UML)$</td>
</tr>
<tr>
<td>Time including diagram modifications</td>
<td>$H_0: T'(UML) = T'(no\ UML)$</td>
<td>$H_a: T'(UML) \neq T'(no\ UML)$</td>
</tr>
<tr>
<td>Correctness</td>
<td>$H_0: C(UML) \leq C(no\ UML)$</td>
<td>$H_a: C(UML) &gt; C(no\ UML)$</td>
</tr>
<tr>
<td>Quality</td>
<td>$H_0: Q(UML) \leq Q(no\ UML)$</td>
<td>$H_a: Q(UML) &gt; Q(no\ UML)$</td>
</tr>
</tbody>
</table>

## Summary of Competencies

<table>
<thead>
<tr>
<th></th>
<th>Oslo</th>
<th>Ottawa</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO programming courses (lecture + laboratory)</td>
<td>2 courses: 120 hours</td>
<td>2 courses: 120 hours</td>
</tr>
<tr>
<td>Introduction to UML course(s) (lecture + laboratory)</td>
<td>2 software engineering courses introducing UML as one of several topics: Approximately 40 hours of UML-specific training</td>
<td>1 course: 60 hours</td>
</tr>
<tr>
<td>Other topics addressed in previous courses</td>
<td>Operating systems, databases, computer architecture, data communication, software engineering</td>
<td>Operating systems, databases, real-time systems, computer architecture</td>
</tr>
</tbody>
</table>
### Oslo—Descriptive Statistics per Task

<table>
<thead>
<tr>
<th>Dep. Var.</th>
<th>Group</th>
<th>Min</th>
<th>Med</th>
<th>Max</th>
<th>Task 1</th>
<th></th>
<th></th>
<th>Min</th>
<th>Med</th>
<th>Max</th>
<th>Task 3</th>
<th></th>
<th></th>
<th>Min</th>
<th>Med</th>
<th>Max</th>
<th>Task 4</th>
<th></th>
<th></th>
<th>Min</th>
<th>Med</th>
<th>Max</th>
<th>Task 5</th>
<th></th>
<th></th>
<th>Min</th>
<th>Med</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (excl. model modification)</td>
<td>No UML</td>
<td>20</td>
<td>75</td>
<td>240</td>
<td>5</td>
<td>20</td>
<td>60</td>
<td>10</td>
<td>22</td>
<td>55</td>
<td>16</td>
<td>54</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>31</td>
<td>53</td>
<td>95</td>
<td>8</td>
<td>15</td>
<td>23</td>
<td>12</td>
<td>19</td>
<td>35</td>
<td>45</td>
<td>65</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T’ (incl. model modification)</td>
<td>UML</td>
<td>43</td>
<td>70</td>
<td>105</td>
<td>15</td>
<td>27</td>
<td>41</td>
<td>24</td>
<td>36</td>
<td>85</td>
<td>62</td>
<td>101</td>
<td>132</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C (binary)</td>
<td>No UML</td>
<td>46%</td>
<td></td>
<td></td>
<td>91%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>91%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>46%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>56%</td>
<td></td>
<td></td>
<td>89%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>89%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Ottawa—Descriptive Statistics per Task

<table>
<thead>
<tr>
<th>Dep. Var.</th>
<th>Group</th>
<th>Task 1</th>
<th></th>
<th></th>
<th>Task 2</th>
<th></th>
<th></th>
<th>Task 5</th>
<th></th>
<th></th>
<th>Task 6</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T (excl. model modification)</td>
<td>No UML</td>
<td>22</td>
<td>66.5</td>
<td>159</td>
<td>20</td>
<td>75</td>
<td>154</td>
<td>32</td>
<td>89.5</td>
<td>180</td>
<td>74</td>
<td>166.5</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>24</td>
<td>82</td>
<td>240</td>
<td>38</td>
<td>87</td>
<td>162</td>
<td>35</td>
<td>99</td>
<td>196</td>
<td>51</td>
<td>141</td>
<td>184</td>
</tr>
<tr>
<td>T’ (incl. model modification)</td>
<td>UML</td>
<td>40</td>
<td>128</td>
<td>261</td>
<td>74</td>
<td>149.5</td>
<td>180</td>
<td>58</td>
<td>141</td>
<td>223</td>
<td>75</td>
<td>168</td>
<td>184</td>
</tr>
<tr>
<td>C’ (ratio)</td>
<td>No UML</td>
<td>2/8</td>
<td>8/8</td>
<td>8/8</td>
<td>0/8</td>
<td>5/8</td>
<td>8/8</td>
<td>0/8</td>
<td>5/5</td>
<td>5/5</td>
<td>0/12</td>
<td>0/12</td>
<td>12/12</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>2/8</td>
<td>8/8</td>
<td>8/8</td>
<td>2/8</td>
<td>5/8</td>
<td>7/8</td>
<td>1/8</td>
<td>5/5</td>
<td>5/5</td>
<td>0/12</td>
<td>5/12</td>
<td>12/12</td>
</tr>
<tr>
<td>Q (Correct changes)</td>
<td>No UML</td>
<td>1/9</td>
<td>4/9</td>
<td>4/9</td>
<td>0/8</td>
<td>8/8</td>
<td>8/8</td>
<td>1/4</td>
<td>4/4</td>
<td>4/4</td>
<td>0/4</td>
<td>1/4</td>
<td>4/4</td>
</tr>
<tr>
<td>Q’ (Incorrect changes)</td>
<td>No UML</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Experimental results

- The goal was to shed some light on the cost effectiveness of model-driven development with UML.
- focused on whether models help software engineers to make quicker and better changes to existing systems.

- The results of the two experiments are mostly consistent.
  - When considering only the time required to make code changes, using UML documentation does help to save effort overall.
  - On the other hand, when including the time necessary to modify the diagrams, no savings in effort are visible.
  - in terms of the functional correctness of the changes:
    - using UML has a significant, positive impact on the most complex tasks.
    - In the Ottawa experiment, which also investigated the design of the changes, using UML helped to achieve changes with superior design quality, which would then be expected to facilitate future, subsequent changes.
    - the above statements apply only with qualifications.
  - Benefits are not likely to be derived if the tasks to be performed lie below a certain level of complexity or if software engineers have not reached a certain level of skill regarding the use of UML models for analyzing the effects of changes, in addition to having received substantial training in UML modeling.
  - Furthermore, current tools still need substantial improvements in the way they support changes to models and the checking of consistency.
The Unified Modeling Language (UML) is the de facto standard for object-oriented software analysis and design modeling.

- **few empirical studies** exist which investigate the **costs** and evaluate the **benefits** of using UML in realistic contexts.

- Such studies are needed so that the software industry can make informed decisions regarding the extent to which they should adopt UML in their development practices.
This is the first controlled experiment that investigates the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real nontrivial system, using professional developers as subjects, working with a state-of-the-art UML tool during an extended period of time.

- **Control Group:** had no UML documentation
- **UML Group:** had UML documentation.
  - had, on average, a practically and statistically significant 54% increase in the functional correctness of changes (p = 0.03)
  - insignificant 7% overall improvement in design quality (p = 0.22)
### Tested Hypotheses

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time excluding diagram modifications</td>
<td>$H_0: T(\text{UML}) \geq T(\text{no-UML})$</td>
<td>$H_a: T(\text{UML}) &lt; T(\text{no-UML})$</td>
</tr>
<tr>
<td>Time including diagram modifications</td>
<td>$H_0: T'(\text{UML}) = T'(\text{no-UML})$</td>
<td>$H_a: T'(\text{UML}) \neq T'(\text{no-UML})$</td>
</tr>
<tr>
<td>Correctness – Num. of submissions of a solution with a fault</td>
<td>$H_0: C(\text{UML}) \geq C(\text{no-UML})$</td>
<td>$H_a: C(\text{UML}) &lt; C(\text{no-UML})$</td>
</tr>
<tr>
<td>Correctness – Introduced a fault breaking existing functionality</td>
<td>$H_0: C'(\text{UML}) \geq C'(\text{no-UML})$</td>
<td>$H_a: C'(\text{UML}) &lt; C'(\text{no-UML})$</td>
</tr>
<tr>
<td>Correctness – Introduced a fault stemming from not taking into account all existing behavior</td>
<td>$H_0: C''(\text{UML}) \geq C''(\text{no-UML})$</td>
<td>$H_a: C''(\text{UML}) &lt; C''(\text{no-UML})$</td>
</tr>
<tr>
<td>Design Quality</td>
<td>$H_0: Q(\text{UML}) \leq Q(\text{no-UML})$</td>
<td>$H_a: Q(\text{UML}) &gt; Q(\text{no-UML})$</td>
</tr>
</tbody>
</table>
Selection of Subjects

- Subjects were recruited via a request for consultants being sent to Norwegian consulting companies. The request specified a flexible range of time, for which the consultants would be needed, along with the required education and expertise. Companies replied with resume of potential candidates and these were then screened to verify that they indeed complied with the requirements. The subjects were required to at least have a bachelor’s degree in informatics (or its equivalent), some familiarity with UML (use case, class, sequence, and state diagrams), and some project experience with the following technologies: Struts, JavaServer Pages (JSP), Java 2, HTML, the Eclipse IDE, and MySQL.

- Note that the recruitment of all subjects could not be completed before the start of the experiment. This was due to several practical reasons:
  1. The market for these skilled professionals is very tight.
  2. We could not give the consulting companies definite start and end dates as to when the consultant would be working.
  3. The consulting companies could not give us an exact start date for consultants
  4. The consulting companies often could not guarantee that the consultant would be available.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Group</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Min</th>
<th>Lower Quartile</th>
<th>Median</th>
<th>Upper Quartile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>No UML</td>
<td>31.7</td>
<td>5.9</td>
<td>25</td>
<td>28</td>
<td>30</td>
<td>37</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>34.5</td>
<td>5.2</td>
<td>28</td>
<td>31</td>
<td>32.5</td>
<td>39</td>
<td>45</td>
</tr>
<tr>
<td>Degree (1=bachelors, 2=masters)</td>
<td>No UML</td>
<td>1.7</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>1.7</td>
<td>0.7</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Years of Study at University</td>
<td>No UML</td>
<td>4.9</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>4.7</td>
<td>1.2</td>
<td>2.5</td>
<td>4.5</td>
<td>5</td>
<td>5.5</td>
<td>6</td>
</tr>
<tr>
<td>Years of Study in Computer Science</td>
<td>No UML</td>
<td>3.7</td>
<td>1.4</td>
<td>2</td>
<td>2.5</td>
<td>3.5</td>
<td>4.5</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>4.0</td>
<td>1.7</td>
<td>1</td>
<td>3</td>
<td>4.75</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Average Grade in Computer Science Courses</td>
<td>No UML</td>
<td>2.1</td>
<td>0.4</td>
<td>1.7</td>
<td>1.8</td>
<td>2</td>
<td>2.5</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>2.2</td>
<td>0.5</td>
<td>1.5</td>
<td>2</td>
<td>2.1</td>
<td>2.5</td>
<td>3</td>
</tr>
<tr>
<td>Years Of Experience With (YOEW) Java</td>
<td>No UML</td>
<td>4.9</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>4.8</td>
<td>2.1</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>YOEW Servlets / JSP</td>
<td>No UML</td>
<td>2.8</td>
<td>1.3</td>
<td>0.5</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>2.9</td>
<td>1.9</td>
<td>0.3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>YOEW Struts</td>
<td>No UML</td>
<td>0.9</td>
<td>0.5</td>
<td>0.3</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>1.3</td>
<td>1.5</td>
<td>0</td>
<td>0.021</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>LOC in Java</td>
<td>No UML</td>
<td>187100.0</td>
<td>320433.2</td>
<td>6000</td>
<td>30000</td>
<td>50000</td>
<td>100000</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>255500.0</td>
<td>397649.0</td>
<td>5000</td>
<td>20000</td>
<td>75000</td>
<td>200000</td>
<td>1000000</td>
</tr>
<tr>
<td>LOC in C++</td>
<td>No UML</td>
<td>3200.0</td>
<td>6663.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>20000</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>7000.0</td>
<td>11595.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20000</td>
<td>30000</td>
</tr>
<tr>
<td>LOC in C</td>
<td>No UML</td>
<td>1300.0</td>
<td>3199.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>1100.0</td>
<td>2079.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>LOC in C#</td>
<td>No UML</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>1300.0</td>
<td>3199.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>Used Borland Together</td>
<td>No UML</td>
<td>40%</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>20%</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Used a UML Tool</td>
<td>No UML</td>
<td>90%</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>90%</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>
### BESTweb System Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Packages</td>
<td>6</td>
</tr>
<tr>
<td>Number of Classes</td>
<td>50</td>
</tr>
<tr>
<td>Lines of Java Code</td>
<td>2921</td>
</tr>
<tr>
<td>Number of JavaServer Pages (JSP)</td>
<td>17</td>
</tr>
<tr>
<td>Number of Attributes</td>
<td>107</td>
</tr>
<tr>
<td>Number of Overridden Methods</td>
<td>33</td>
</tr>
<tr>
<td>Number of Methods</td>
<td>275</td>
</tr>
<tr>
<td>Total Number of Children</td>
<td>13</td>
</tr>
<tr>
<td>Maximum Depth of Inheritance Tree</td>
<td>3</td>
</tr>
<tr>
<td>Number of Libraries Used</td>
<td>20</td>
</tr>
</tbody>
</table>

### Class Diagrams: Metrics

<table>
<thead>
<tr>
<th>Class Diagram</th>
<th>Num. of Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>no.simula.bestweb.web</td>
<td>36</td>
</tr>
<tr>
<td>no.simula.bestweb.web.admin</td>
<td>10</td>
</tr>
<tr>
<td>no.simula.bestweb.pubmdl</td>
<td>6</td>
</tr>
<tr>
<td>no.simula.bestweb.parser</td>
<td>8</td>
</tr>
<tr>
<td>no.simula.bestweb.index</td>
<td>6</td>
</tr>
<tr>
<td>no.simula.bestweb.db</td>
<td>8</td>
</tr>
</tbody>
</table>
# Sequence Diagrams: Metrics

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Num. of Objects</th>
<th>Num. of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>Change Display Settings</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Filter Publications List using Selected BEST-codes</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Show All Publications</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>User’s Search</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Query Search</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>Sort Publications List</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>View Publication Details</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Prepare for showStatistics.jsp</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>View Statistics of Publications Per Year</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>View Statistics of Publications Per BEST-code Category</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Add User</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>Remove User</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>Upload a Library File</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>System Initialization</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Load BEST-codes and Publications</td>
<td><strong>17</strong></td>
<td><strong>30</strong></td>
</tr>
</tbody>
</table>
The experimental process (the documents represent deliverables).
EXPERIMENTAL RESULTS
In terms of time, the UML subjects used more time if the UML was to be updated (though slightly less if it were not).

With the total time $T$ that the subjects spent on the five tasks, we see that

- the UML group completed the tasks slightly faster (1.4%) than the no-UML group.
- This difference is not practically or statistically significant.
- Taking the time that it takes to update the UML docs into account:
  
  - UML group spent 14.5% more time on the five tasks, though this difference is not statistically significant either and may therefore be due to chance.
- On average, the UML subjects spent 14.8% of the total time reading the UML docs and 13.2% updating the docs.
UML was always beneficial in terms of functional correctness (introducing fewer faults into the sw).

The subjects in the UML group had, on average, a practically and statistically significant 54% increase in the functional correctness of changes (p= 0.03)

- UML also helped produce code of better quality when the developers were not yet familiar with the system.

- A significant difference was found for Task 1, where the UML group’s design quality score was 56.2% higher (p= 0.0025) though, across all the tasks, there was an insignificant % improvement in design quality  p = (0.22)
All of the qualitative evidence suggests that the observed impact of UML on change quality and productivity is probably very conservative in this experiment.

The UML subjects were at a disadvantage when it came to Struts experience and familiarity with Java.

We also observed that half of the subjects only used two diagram types, with the use case and sequence diagrams being, by far, the most used.

Four of the subjects did not use the UML to the extent that they could have due to concern that UML would make them less efficient and out of habit (not being used to using UML).

The subjects also experienced severe problems when dealing with the tool and in understanding the large sequence and class diagrams. However, the qualitative evidence also explains the observed benefits of UML.

The no-UML group had more problems in understanding a complex part of the system.
All subjects found the UML to be generally useful: The largest benefits were the traceability of use cases to code and the ability to quickly get an overview of the system.

The results of this experiment, both qualitative and quantitative, can also be used to guide industrial adoption with respect to, at the very least, applications with similar properties (e.g., Web applications).

In the case of developers who are not very experienced in using UML and who will perform maintenance tasks on a system that they are not familiar with, the use case diagram and the sequence diagrams seem to be the most cost-efficient parts of UML. This appears to be the case for two reasons.

- First, developers inexperienced with UML are overwhelmed by too many diagram types and will only use those that are easy to use.
- Next, these two diagrams help them quickly identify the relevant code for the specific functionality needed to perform the maintenance tasks. Given these advantages, these two types of diagrams can also be considered a cost-efficient starting point for introducing UML into the organization.
What is the situation of sw dev many companies?

- No diagrams or models or even ER diagram at all!!

- Why?!!!:
  - It is time consuming !!!!
  - 1 developer performs all tasks (analysis, design, implementation)

- What are artifacts that delivered to developers:
  - Psuedocode (can be consider as structured English)
  - Screens

- Steps for sw developments:
  - Create DB with required information
  - Map tables to forms (GUI) / web pages
Conclusions & Recommendations

System Customers
- I want it yesterday and it’s got to be right!

Marketing
- It’s got to deliver benefits that I can sell!

Developers and Managers
- What kind of tools are you going to give me to create this thing?

Software Architect
- I should have gone into accounting instead!

End Users
- Please make it fast and easy to use!

Operations Personnel
- Make sure it’s maintainable and reliable so I can support it!