A Commercially Robust Process for the Development of OO Software Systems

presented at

9th Annual Borland Conference
Denver, CO

August 11, 1998

by

Timothy D. Korson, Ph. D.
Senior Partner, Software Architects
and
Dean, School of Computing
Southern Adventist University

(423) 238-3288
korson@software-architects.com
www.software-architects.com
Abstract

The successful use of object technology requires far more than simply the adoption of UML, Java, CORBA or ActiveX. What is crucial, is knowing how to use these technologies to build commercially robust software systems. In this session the speaker draws on his experience at NASA, NORTEL, AT&T, IBM, NOVA GASS, and other leading companies to illustrate the pitfalls and best practices of OO software development.

Attendees simulate the problems encountered when applying existing methods by walking through the complete development process from use case development to pattern application and coding. Participants gain a clear understanding of the distinction between domain models, application models, and architectures. The course goes beyond notation and cookbook methods to address the key principles of Object-Oriented Software Development. A percentage of the time is spent discussing the rationale underlying each step of the process so that attendees will leave with an understanding of the fundamental software engineering concepts behind the specific techniques presented. Several case studies are woven through the presentation.
Restricted Use

Software Architects retains ownership of the material contained in these course notes. This copyrighted material is provided to attendees of Software Architects courses under a restricted licensing agreement exclusively for the personal use of attendees of courses taught by Software Architects. This material **may not** be copied or reused without written permission of Software Architects.

Software Architects is prepared to grant rights to reuse all or a portion of this material based upon your request and our explicit permission; all other rights are reserved.

Please respect these rights.
Overview: Methods, Processes, & Life Cycles

OT Promises to:

• Support RAD through reuse of class components, design patterns, architectural frameworks, and domain models

• Improve product quality

• Facilitate business process reengineering

• Solve problems of greater complexity

• Improve maintainability and extensibility

• Improve communication with customers

• Improve communication between developers

These are potential benefits. They do not magically happen. They require the systematic application of rigorous methods
UML

- UML stands for the Unified Modeling Language
- UML is a notation, not a method.
- Rational/Booch’s original intention was to create a unified method\(^1\) which included a unified modeling language, but this was quickly seen to be infeasible, and the focus narrowed to the UML.

The Three Amigos

---

1. see future slide on the Rational Objectory Process
Configuring Methods

• Methods often address only a portion of a complete development process

• The unified method, as an industry generic standard, was abandoned in favor of the proprietary Rational Objectory Process.

• Organizations need to customize and integrate OT methods into the culture, infrastructure, and existing methods of the organization

Every real commercial project we have worked with has tailored and configured an OT method to meet their specific needs.

Once you get past special pilot projects, going “by the book” breaks down.

Customizing a method takes experience!
A Software Development Process Meta-Model

1. Emphasis
   ■ Domain-driven
   ■ Reuse-oriented
   ■ Quality engineered

2. Activities
   ■ Domain Analysis
   ■ Application Analysis
   ■ Application Design
   ■ Class Design / Development
   ■ Incremental Integration and System Testing

3. Notation

4. UMLScope
   ■ Comprehensive

5. Process Model
   ■ Iterative Process
   ■ Incremental Application Development
   ■ Prototype-supported
- This is the way development teams work
- Most managers plan and track as if the development team was using a waterfall model
- When using an iterative model the team schedules will show that iteration is planned, scheduled and tracked.
Incremental Model

- Build a system in increments that represent increasing levels of functionality:
  - Analyze a little...
  - Design a little...
  - Code a little...
  - Test a little...
  - Learn a lot...

- This is different from waterfall until implementation, then incremental development.
Prototyping Model

• Some advocate a process with little formal analysis and design, evolving the prototype into the product.

• This is good for small in-house projects that will forever be supported by the same person(s).

• It is a temptation to skip Domain Analysis and Architecture when using a powerful IDE such as JBuilder from Borland

Note: We will use the term prototype to designate a disposable product.
An OO Development Process

• Basic process is incremental

  Iterate within increments, with prototype support as necessary. Often this will involve reworking one piece of the system several times before an increment is finished. Previous increments are only revisited to fix errors or serious flaws.

• Getting started

  Do enough domain and application analysis to form a context, then concentrate on driving an increment through to completion (running, tested code).
Basic Development Process

<table>
<thead>
<tr>
<th>Activity</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Analysis</td>
<td>$I_n$</td>
</tr>
<tr>
<td>Application Analysis</td>
<td>$I_{n+1}$</td>
</tr>
<tr>
<td>Application Design</td>
<td>$I_{n+2}$</td>
</tr>
<tr>
<td>Class Development</td>
<td>$I_{n+3}$</td>
</tr>
</tbody>
</table>

**light activity**  **heavy activity**

Programmer-analysts teams work best
### Getting Started

#### Activity

<table>
<thead>
<tr>
<th>Activity</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Analysis</td>
<td>I&lt;sub&gt;0a&lt;/sub&gt;</td>
</tr>
<tr>
<td>Application Analysis</td>
<td></td>
</tr>
<tr>
<td>Application Design</td>
<td></td>
</tr>
<tr>
<td>Class Development</td>
<td></td>
</tr>
<tr>
<td>Application Assembly</td>
<td></td>
</tr>
</tbody>
</table>

#### Activity Increment

- **I<sub>0a</sub>**: Light activity
- **I<sub>0b</sub>**: Light activity
- **I<sub>0c</sub>**: Heavy activity
- **I<sub>1</sub>**: Heavy activity

**Establishing a basic context**
Stepping through the process: Getting Started

<table>
<thead>
<tr>
<th>Phase</th>
<th>DA</th>
<th>AA</th>
<th>AD</th>
</tr>
</thead>
</table>

----- Project Scope ------

Phase
Stepping through the process: Increment 1

Phase I

DA
AA
AD
CD
IT

----- Project Scope -----
### Stepping through the process: Increment 2

<table>
<thead>
<tr>
<th>DA</th>
<th>AA</th>
<th>AD</th>
<th>CD</th>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>I2</td>
<td>I1</td>
<td>I2</td>
<td>I2</td>
</tr>
</tbody>
</table>

When performing the Ith increment, bring the entire analysis and design up to date.
Choosing Increments
Internal vs. External

■ A use-case driven (external) increment implements some subset of the user requirements (e.g. traffic controller works for non-rush hour straight traffic. No turn traffic, pedestrians or emergency vehicles are yet considered).

■ An architecturally based (internal) increment implements some subset of system functionality (e.g. database access mechanism, inter-process communication).

■ How does one balance scheduling external increments with internal increments?
Incremental Process Implies Incremental Documentation

Other important documents, such as the user manual and test plan, are also developed, and verified incrementally.
Milestones

• Customer milestones can be defined in terms of functionality completed.

• Project tracking and estimation takes two forms
  ■ Macro tracking
    Related to the spiral
  ■ Micro tracking
    At the class level, track convergence and number of new classes. Be careful to track within the same level of abstraction.

One management iteration should not exceed 2 months; 2-6 weeks is preferable

Even in the largest projects, one increment should not exceed 4 months.
Phases in the Application Life Cycle

There are five main phases fundamental to any OO software development process:

■ Domain Analysis
  
  Domain Analysis is the process of coming to an understanding of the domain of the application. The domain is modeled, clarified, and documented, and fundamental objects and their interrelationships are sorted out.

■ Application Analysis
  
  Application Analysis is the process of determining exactly what will be built for the requested application. The requirements specifications are clearly spelled out in this phase.

■ Application Design
  
  In Application Design the mechanisms for implementing the actual application are determined. System architecture, data structures, efficiency considerations, etc., are all dealt with here.

■ Class Development
  
  Classes are developed and tested in the language of implementation.

■ Incremental Integration & System Testing
  
  Classes are integrated into cluster and subsystem modules and finally into a complete application. Integration testing verifies that these modules work correctly, effectively, and meet requirements.
Domain Analysis

high-level software requirements statement, domain knowledge, existing domain models

Domain Analysis

domain specifications, domain models
Traditional Systems Analysis

- **Goal:** Understanding the problem:
  
  a. Document specific requirements for the application.
  
  b. Focus primarily on functional requirements.

- **Scope:** Limited to a specific application such as the Clemson Alumni Donor Tracking System or en route air traffic control.
Domain Analysis

• Goal: Understanding the problem space:
  a. Identify the fundamental concepts of interest in the target domain.
  b. Document their relevant, intrinsic properties and responsibilities.
  c. Recognize and record the relationships among these concepts.

• Scope: Limited by the boundaries of an area of interest such as interactive graphics or accounting systems.
Domain Analysis Process

Data Gathering
  Identification of concepts

Modeling
  Hypothesis theories

Synthesis
  Development of taxonomies

Abstraction
  Creation of standard abstractions

Refinement

There is no cookbook!
## Domain Analysis Phase - Steps & Deliverables

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain initial understanding of application requirements</td>
<td>Initial problem statement</td>
</tr>
<tr>
<td>Determine domain limits</td>
<td>Domain dimensions</td>
</tr>
<tr>
<td>Identify domain actors and their basic interactions with domain</td>
<td>Use Case Diagram</td>
</tr>
<tr>
<td>Determine the activities of interest within the domain</td>
<td>Domain activities (Domain-level use-cases)</td>
</tr>
<tr>
<td>Identify key objects in domain</td>
<td>List of objects</td>
</tr>
<tr>
<td>Clarify meaning of domain objects</td>
<td>Class description cards</td>
</tr>
<tr>
<td>Determine static relationships between all key domain objects</td>
<td>Domain class diagram</td>
</tr>
<tr>
<td>Record standardized dynamic behavior</td>
<td>State transition diagrams</td>
</tr>
<tr>
<td>Record domain algorithms</td>
<td>Sequence diagrams</td>
</tr>
<tr>
<td>Summarize knowledge of domain</td>
<td>Domain digest</td>
</tr>
</tbody>
</table>
Traffic Intersection

Sensors

controller

Traffic Signal

1 2

1 1 2

1 1

1 2

1 1 1

N
Deciding Domain Dimensions and Boundaries

• This is a business decision. The question is: “How general should we get?” “What are the cost/benefit trade-offs?”

• Four questions to ask:
  
a. What are the strategic directions that are being taken in the domain?
  
b. What are the customer’s views of the first few versions of the application?
  
c. What developments and directions is related technology taking?
  
d. Business decisions: What is my market? What type of volume will I have?
Use Case Diagram

Gradebook System

- Maintain Grade Policy
- Statistics Generation
- Roster Maintenance
- Grade Maintenance
- Historical Archive
- Report Generation

Instructor
Grader
Registrar
Student
Guardian
Use Cases

• A use case is a description of a typical use of an application by a user.

• The perspective is from the user of the system.

• The use case:
  a. Identifies the actor, whose perspective is being described.
  b. Provides a description of the interaction between the actor and the system.
  c. Gives reasons why this is a legitimate and needed use of the system.
  d. Estimates how frequently this use would occur.
  e. Gives a reference where more information can be located.
Class Diagram

- The UML class diagram captures the static structural relationships among the classes. Note that there is no reference to time or sequence in the class diagram.
Build the Domain Class Diagram

User

Permitted Access

Archive

Report

Gradebook

Section Gradebook

Instructor

Grader

Term Schedule

Term Calendar

Course

Term

University Policy

GPA Map

Actual Student Work

Annotation

Guardian

Student Work

Student

Grade

Assignment

Category

Final Grade

Grade Scale

Section Gradebook Policy

Roster

Registration Record

Catalog / Bulletin

rev 1994 Mar 17
A state diagram is a bipartite graph of states and transitions connected by a physical containment and tiling. A state diagram shows:

- The sequence of states that an object goes through during its life in response to received stimuli (events).
- Its actions in response to these stimuli.

A state is a condition during the life of an object during which it satisfies some condition, performs some action, or waits for some event. An object remains in a specific state for some finite period of time.

An event is an occurrence that may trigger the transition from one state to another. Events are usually an explicit signal from one object to another. In some cases, events can be a condition coming true or the passage of a specified passage of a specified period of time.
Domain-level Grade State Diagram
(With Guards)

NOTE: The guards on the transitions are redundant in this diagram. They are included here simply to enhance the clarity of the diagram.
Domain-level Grade State Diagram
(Without Guards)

Grade

Excused

Assigned

Missing

Excuse Submitted

Assignment Submitted & Graded

Assignment Becomes Due

Excused Submitted

Start

Terminal State
Create Domain-level Object Interaction Diagrams

• A sequence diagram\(^1\) is a diagram that formally describes a scenario.

• Each class is shown as a vertical line; each event as a horizontal arrow from the sender class to the receiver class; time flows from the top to the bottom of the diagram.

---

1. Event trace or object interaction diagram

Copyright © 1997, 1998 Timothy Korson. All rights reserved.
Borland Conference, Denver, August 1998
Domain Digest

- The domain digest is a compilation of all that has been learned and documented about a domain and the required application. It captures in one place the needs, understanding, and current assumptions of a given domain and application requirements. This includes the object, dynamic, and functional models, and is summarized and described by additional text.
Who Verifies the “Correctness” of Deliverables?

Once deliverables are being created, they need to be verified for appropriateness, correctness, and relevance. There are certain people or groups who can do this verification.

- Customer
- Domain Experts
- Peer Developers
- Testing Group
- Documentation Group
Domain Analysis Summary

- The minimal set of UML deliverables includes use cases, class diagrams, and sequence diagrams. Of these, the class diagram is core.

- The sequence diagrams show how the use cases are played out in the class diagram.
Prerequisites for Quality

- Definition of project goals
- Right skills
- Right attitude
- Iteration and willingness to throw away and redo
- Peer review and debate
- Proper metrics
- Some management and QA understanding of the technologies involved.
Application Analysis

domain specifications,
domain models,
business requirements,
customer wants,
constraints,
available components

Application Analysis

application model,
detailed application requirements specification
Benefits of Deferring Application Requirements Analysis till After Domain Analysis

After Domain Analysis you are:

• More likely to get the right requirements.
  a. Closer in time to delivery
  b. Customer has more insight

• More likely to get the requirements right.
  a. Analysis team has more knowledge
  b. Starting from a higher level of abstraction
## Application Analysis Phase - Steps & Deliverables

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify system requirements</td>
<td>Application (level 1, 2,...) use cases</td>
</tr>
<tr>
<td></td>
<td>Detailed system requirements</td>
</tr>
<tr>
<td>Design user interface</td>
<td>User manual</td>
</tr>
<tr>
<td></td>
<td>GUI prototypes</td>
</tr>
<tr>
<td></td>
<td>paper prototypes</td>
</tr>
<tr>
<td>Prune and blend domain object models</td>
<td>Application class diagram</td>
</tr>
</tbody>
</table>
Application Use Cases

• At the application level Use Cases provide a means of documenting all the necessary functionality of the system, from the user’s perspective.

• Application Use Cases are more detailed and specific than Domain-level Use Cases.
Requirements Fundamentals
(Don’t Misuse *Use Cases*)

- Keep business requirements separate from interface specifications
- Requirements should be hierarchically organized
- Hierarchical classification of use cases should not be confused with functional decomposition
- Do not directly derive your architecture from your use cases
Specify Customer Requirements

- Use cases capture most of the functional requirements

- User interface requirements may require GUI prototypes and a user’s manual

- Technical constraints
  - Efficiency constraints
  - Hardware requirements
  - Reliability & recovery

- Business constraints
  - Policy for late papers, etc.
The domain model is focused for the current application.

Additional related and supporting domains are identified.
Domains Relevant to the Grading Application

Grading Domain Model
(Major Domain)

Registration Domain

Spreadsheet Domain

Reporting Domain

Grading Application Model
Application Analysis
Summary

■ It is nearly impossible to get good clear requirements, that reflect the client’s real needs, without a robust business domain model that is well understood by both the clients and the development team.

■ Organize use cases hierarchically, but don’t use them for functional decomposition

■ Identify related and supporting domains and start to plan their integration with the core domain.
Application Design

domain models, application model, detailed application requirements specification, existing frameworks and components

class specifications including design mechanisms, sequence diagrams, application architecture
# Application Design Phase - Steps & Deliverables

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
</table>
| ■ Determine application architecture | ■ Static, dynamic, and functional models of the architecture  
■ Pattern Language |  
| ■ Determine software responsibilities for existing classes | ■ Add CRC information to class description cards (CDC’s) |  
| ■ Determine the design mechanisms for implementing associations, resolving multiplicities, lookup methods, inter-process communication, etc. | ■ New responsibilities for existing class  
■ New classes |  
| ■ Design the classes known at this level | ■ Header Files:  
■ Methods with Pre and postconditions  
■ Class Invariant  
■ State Diagrams  
■ Assign risk level |  
| ■ Complete the details of the user interface that are left to the developer’s discretion | ■ Updated user manual |  
| ■ Validate the design against use cases | ■ Sequence Diagrams |
## Application Design Levels

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architectural</strong></td>
<td><strong>Static, dynamic, and functional models of the architecture</strong>&lt;br&gt;<strong>Pattern Language</strong>&lt;br&gt;<strong>Add CRC information to class description cards (CDC’s)</strong>&lt;br&gt;<strong>New responsibilities for existing class</strong>&lt;br&gt;<strong>New classes</strong>&lt;br&gt;<strong>“Header Files:”</strong>&lt;br&gt;  - Methods with Pre and postconditions&lt;br&gt;  - Class Invariant&lt;br&gt;  - State Diagrams&lt;br&gt;  - Assign risk level&lt;br&gt;<strong>Updated user manual</strong>&lt;br&gt;<strong>Sequence Diagrams</strong></td>
</tr>
<tr>
<td><strong>Cluster</strong></td>
<td><strong>Determine application architecture</strong>&lt;br&gt;<strong>Determine software responsibilities for existing classes</strong>&lt;br&gt;<strong>Determine the design mechanisms for implementing associations, resolving multiplicities, lookup methods, inter-process communication, etc.</strong>&lt;br&gt;<strong>Specify the class syntax and create the class contract</strong>&lt;br&gt;<strong>Complete the details of the user interface that are left to the developer’s discretion</strong>&lt;br&gt;<strong>Validate the design against use cases</strong></td>
</tr>
<tr>
<td><strong>Detail</strong></td>
<td><strong>Complete the details of the user interface that are left to the developer’s discretion</strong>&lt;br&gt;<strong>Updated user manual</strong>&lt;br&gt;<strong>Sequence Diagrams</strong></td>
</tr>
</tbody>
</table>
Overview of Design Activities

• Standard architectures, components, patterns and frameworks are selected and customized
• Associations in the model are designed and multiplicities are resolved
• Attributes may become classes
• Responsibilities are mapped to methods including parameter and return types, pre- and postconditions, and exception handling
• Performance, memory constraints, garbage collection, persistence, etc. are considered
• New classes are created that blend the related domains

This often results in an order of magnitude increase in the number of classes.
Architecture

• An architecture for a system is the pattern of connections among the basic components of the system; the pattern embodies the relations and constraints among the constituent pieces of the system.

• Our preferred method of documenting an OO architecture uses:
  static, dynamic, and functional models of the architecture

A pattern language that describes the interrelationships and constraints of the architecture, and governs instantiation of the architecture.
Documenting Frameworks

- **RAPPeL: A Requirements Analysis Process Pattern Language.**
  http://www.bell-labs.com/people/cope/Patterns/Process/RAPPeL/rapel.html

- **Jim Coplien’s "Development Process Generative Pattern Language"**
  http://www.bell-labs.com/people/cope/Patterns/Process

- **Ralph Johnson’s "Documenting Frameworks Using Patterns"**
  ftp://st.cs.uiuc.edu/pub/patterns/papers/documenting-frameworks.ps

- **"G++: A Pattern Language for the Object Oriented Design of Concurrent and Distributed Information Systems, with Applications to Computer Integrated Manufacturing" by Amund Aarsten, Gabriele Elia, Giuseppe Menga:**
  http://www.polito.it/~cim/Articles/plop94.ps This paper *does* have a directed solution graph! This may be an example of a truly useful semi-generic language consisting of design patterns. Worthy of closer inspection at any rate.
1. stores data to and retrieves gradebook data from
2. displays and executes worksheet menus from
3. displays and executes worksheet interface from
4. performs input/output worksheets from
5. creates and initializes tables from
6. computes and updates grades from
7. performs operations (open, create, save) from
8. displays and executes gradebook menus from
9. performs input/output for menus from
10. retrieves information for reports from
11. creates, opens, closes, writes, reads a workfile from
12. creates, opens, closes, writes a report file from
13. displays & executes report menus & toggles from
14. sets selected report toggles from

The architecture is more than this!
The GradeSheet Architecture is a Composite of Frameworks:

- MVC variant at the system level
  
  ![Diagram of MVC architecture]

- a custom spreadsheet framework at the view level

  See Design Document, section 5.1.5

- the section gradebook architecture is derived from our grading domain model

  Each of these frameworks (as well as the composition) requires substantial documentation
From Domain Analysis to Architecture

- The architecture for each separate domain describes the way the classes within the domain interact and is based on domain relationships.
- The architecture for the application describes the way the different domains interact.

Grading Domain Model (Major Domain)

Registration Domain

Reporting Domain

Spreadsheet Domain

Grading Application Model
From Architecture to Application

- An application is a specific instantiation of an architecture to satisfy a specific set of requirements.

- The architecture was not chosen or created to meet functional requirements, but because of standard domain relationships and other system requirements such as: portability, architectural standardization, extensibility, performance, distributability, reliability, etc.

- Examples
  - **Switching**
    - architecture is driven by reliability and performance
  - **Ground Control Systems**
    - performance and availability of COTS and GOTS
  - **International Accounting System**
    - portability, extensibility
From Architecture to Application

- Technical system requirements
- Standard frameworks
- Domain relationships

Architecture

Functional Requirements

Application
Watch out for Shortcuts

- RAD tools that tie screens directly to databases give you RAD development, but not RAD modifiability.

- Screens should be tied to the business model.

- The business model should not directly embed SQL, or any other implementation technology, in its methods.

What are the trade-offs?
Associations in the Application Model are Designed and Multiplicities are Resolved

Application Model

Design Model
We want to be able to change sequencing policy without affecting the sensor, signal, lane, intersection, and controller objects.
Applying the Right Pattern

**Controller**

- NewStrategy(FlowStrategy * Policy)
- NextPhase()

**FlowStrategy**

Policy.setController(self)

**Intersection**

**RoundRobin Strategy**

**OnDemand Strategy**

**SelfAdapting Strategy**

**LeftLead Strategy**

**LeftLag Strategy**

NewPhase = Policy.Next_Phase(ThisIntersection.currentState)
ThisIntersection.change_phase(NewPhase)
Design Reviews

- There should be at least three types
  
  Architecture review
  
  Cluster design review
  
  Detail design review
Class Development

class specifications
(including performance constraints, error handling, etc.)

Class Development

classes (completely coded and tested)
## Class Development Phase - Steps & Deliverables

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Implement the classes for this increment</td>
<td>■ Source code plus complete documentation</td>
</tr>
<tr>
<td></td>
<td>■ Supporting utility classes^a</td>
</tr>
<tr>
<td>■ Test the classes for this increment</td>
<td>■ Certified class plus complete regression testing mechanism (PACT)</td>
</tr>
</tbody>
</table>

^a. Sometimes as a part of developing one class the need for a new utility class is found. As a part of the development of the original class, the utility class may be specified and developed.
Incremental Integration & System Testing

classes and frameworks
high-level application design

Incremental Integration & Testing

Partially completed system
Regression testing mechanism
## Incremental Integration and Testing

<table>
<thead>
<tr>
<th>Step</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Assemble and test each cluster</td>
<td>■ Certified clusters plus complete PACT&lt;sup&gt;a&lt;/sup&gt; mechanism</td>
</tr>
<tr>
<td>■ Assemble clusters into system increment and test</td>
<td>■ Certified increment plus complete PACT mechanism</td>
</tr>
<tr>
<td>■ Integrate this increment with existing increments</td>
<td>■ Partially completed system plus complete regression testing mechanism</td>
</tr>
</tbody>
</table>

<sup>a</sup> parallel architecture for component testing
Summary

• Object-oriented tools, languages, middleware solutions, databases, and development environments are finally becoming mature and mainstream.

• Yet many organizations fail to achieve the benefits promised by OT because they focus on the technology and short-change the process.

• Numerous case studies (both successes and failures) exist. We know how to engineer good OO software systems.

• Don’t ignore the fundamental principles of software engineering just because new technology is so much fun to play with and corporate change is so difficult!
For More Information

shot of our home page here

- http://www.software-architects.com
To Further Your Career

- Consider a Masters in Software Engineering that focuses on Object-Oriented Technology