Agile Architecture and Design

Pradyumn Sharma
pradyumn.sharma@pragatisoftware.com
www.twitter.com/PradyumnSharma
Presenter: Pradyumn Sharma

- CEO, Pragati Software, Mumbai, India (www.pragatisoftware.com).
- About 30 years in the IT industry.
- Training and consulting: Agile methodologies, Solutions Architecture, Enterprise Architecture, NoSQL.
Software Architecture: When?
Agile Architecture and Design

- Evolve iteratively
- through
  - an initial envisioning
  - implementation of stories
  - refactoring and restructuring
Agile Architecture and Design: Overview

• Basics:
  1. Involve the entire team
  2. Have an “Architecture Owner” role
  3. Understand your product

• The key stuff:
  4. Create and architecture vision
  5. Build architecture through stories
  6. Model and implement incrementally
#1. Involve the Entire Team
Involve the Entire Team in

- Product Backlog Grooming
- Domain modeling
- Architecture discussion, evolution, implementation
- Architecture reviews
- Technical debt sessions
- Refactoring and restructuring
#2. Have an “Architecture Owner” Role
Responsibilities

- **DOs:**
  - Bring the team together for all discussions regarding architecture envisioning and modeling
  - Facilitate architecture modeling and evolution
  - Help in building a shared understanding
  - Help the team members enhance their capabilities in understanding architectural principles and tradeoffs involved

- **DON'Ts:**
  - Dictate the architecture, preventing others from having their say.
  - Guard the architecture as personal property.
#3. Understand Your Product
Understand Your Product
Understand Your Product
#4. Create an Architecture Vision
Create an Architecture Vision

• When?
  – Sprint zero.

• How?
  – Architecture / technical workshop.

• Who?
  – Team as well as the Product Owner (architecture must be based on requirements).
Architecture Workshop: Activities

- Domain modeling
Domain Modeling

![Diagram of Domain Modeling]

- **Owner**
  - 1
- **Subject Group**
  - 1
  - 1..*
- **Subject Matter Expert**
  - 1..*
- **Subject**
  - 1..*
- **Question**
  - question : int
  - rightChoice : int
  - otherChoices : int
  - difficultyLevel : int
  - proposed : int
Domain Modeling
Domain Modeling
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
**UI Prototyping**

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit</th>
<th>Rate</th>
<th>Stock</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfa Black Toner Cartridge (AB001)</td>
<td>No.</td>
<td>$56.00</td>
<td></td>
<td>Edit</td>
</tr>
<tr>
<td>Ameripaper A4 size 500 sheets pack (AM001)</td>
<td></td>
<td>$3.99</td>
<td></td>
<td>Edit</td>
</tr>
</tbody>
</table>
UI Prototyping

Items > New Item

- General
- Specifications
- Vendors

Item name

Code

Description

Unit

Save & Exit  Save & Continue  Cancel
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
Architectural Qualities: Examples

- Portability
- Modifiability
- Performance
- Security
- Testability
- Usability
- Availability

- Conceptual integrity
- Accuracy
- Concurrency
- Customization points
- Internationalization
- Operations
- Maintenance
- Environmental impact
- Reliability
- Regulatory compliance
- Serviceability
- Support
- Dependencies on external systems
Architectural Qualities: Examples

- Portability
- Modifiability
- Performance
- Security
- Testability
- Usability
- Availability

- Conceptual integrity
- Accuracy
- Concurrency
- Customization points
- Internationalization
- Operations
- Maintenance
- Environmental impact
- Reliability
- Regulatory compliance
- Serviceability
- Support
- Dependencies on external systems

Search Engine:
- Performance
- Availability
- Portability

Stock Trading System:
- Security
- Dependencies on external systems
- Performance

Online Travel Agency:
- Dependencies on external systems
- Usability
- Internationalization
Portability Requirements: Examples

• System should run on all popular web browsers as well as smartphones.
• It should be easy to migrate data from one database platform to another.
• Customers should be able to choose the database platform for their respective deployments.
Security Requirements: Examples

- Preventing unauthorized access to data or services
- Dealing with DoS attacks
- Non-repudiation (a transaction cannot be denied by any party)
- Secure transmission of sensitive data
Usability Requirements: Examples

• Ability to save incomplete data as draft and resume later
• Appropriate feedback to the users about state of completeness of a business process / long-running transaction / multi-step data entry, etc.
• Ability to bookmark, roll back, etc.
Availability Requirements: Examples

- System uptime should be at least 99.9%
- When a system service fails, an alert to the Infrastructure team should be raised.
- Even when one service of the system becomes unavailable the other services should continue to run as far as possible.
- It should be possible to deploy newer versions of the components without shutting down the system.
Integrity Requirements: Examples

- System should detect any issues with data consistency or integrity that creep in.
- Likelihood of concurrent access to data: very high / medium / low. => Need for concurrent edits without compromising data integrity.
Architecture Workshop: Activities

• Domain modeling
• UI prototyping
• Identify desired architecture qualities and concerns
• Discuss how to achieve various desired qualities, but don’t commit
Portability: Strategies

• Separation of UI logic, business logic, data access logic in separate components
• Layering, partitioning
Modifiability: Strategies

- Loose coupling between components
- Proper encapsulation
- Layering, partitioning
- Model-View-Controller
- Application framework providing common features and services, extension points for providing story-specific behaviour
- OO principles and design patterns
- Anticipating changes, providing hooks to facilitate changing behaviour
- Non-architectural aspects, such as coding conventions and techniques
Concurrency: Strategies

- Use of Transaction Managers
- Optimistic locking / Pessimistic locking.
- Fine-grained locking / Coarse-grained locking
Usability: Strategies

- Separation of the UI from the rest of the application
- Giving feedback about what the system is doing
- Letting the user issue commands such as Save as Draft, Cancel, Undo, show multiple views
- Design patterns: Command, Memento
- Maintaining a model of the task, or the system, or the user
Performance: Strategies

- Infrastructure planning: server capacity, clustering, failover, virtualization, network bandwidth
- Managing event rate
- Quantum of communication among components, layering
- Database: indexes, partitions, stored procedures
- Non-architectural aspects:
  - choice of algorithms
  - implementation of selected algorithms
  - writing efficient database queries
Security: Strategies

- Authentication of users
- Single sign-on
- Authorization of users, limiting access
- Audit trail
- Intrusion detection system
Integrity: Strategies

• Periodic run of batch programs to check integrity of derived data against raw data
• Discover patterns, fix integrity problems automatically, maintain history of such changes
Strategies for Achieving Architecture Qualities

- Discuss and note down how to achieve the various desired architecture qualities, but don’t commit to them yet.
Architecture Workshop: Activities

• Domain modeling
• UI prototyping
• Identify desired architecture qualities and concerns
• Discuss how to achieve various desired qualities, but don’t commit
• Identify cross-cutting requirements
Cross-Cutting Requirements: Examples

• Audit trail
• Alerts for important events that need attention
• Centralized error logging
• Excel export from all browse windows
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- Discuss how to achieve various desired qualities, but don’t commit
- Identify cross-cutting requirements
- Identify other considerations for architecture
Other Considerations: Examples

- Projected lifetime of the system
- Cost and benefit
- Time to market
- Rollout schedule
- Target market
- Correctness and completeness
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- Discuss how to achieve various desired qualities, but don’t commit
- Identify cross-cutting requirements
- Identify other considerations for architecture
- At every stage, validate with the product owner / customer / end users
Architecture Workshop: Activities

• Domain modeling
• UI prototyping
• Identify desired architecture qualities and concerns
• Identify cross-cutting requirements
• Identify other considerations for architecture
• At every stage, validate with the product owner / customer / end users
• Discuss how to achieve various desired qualities, but don’t commit
• Identify potential technical risks
Potential Technical Risks: Examples

- Technology being used is not mature
- Product is very complex to implement
- Integration requirements are very complex and diverse
- Requirements are unclear, unstable

Killing the project minimizes risk, but also eliminates reward.

–James McGroddy, former CTO at IBM
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- Identify cross-cutting requirements
- Identify other considerations for architecture
- At every stage, validate with the product owner / customer / end users
- Discuss how to achieve various desired qualities, but don’t commit
- Identify potential technical risks
- Take stock of existing reusable architectural assets.
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- Identify cross-cutting requirements
- Identify other considerations for architecture
- At every stage, validate with the product owner / customer / end users
- Discuss how to achieve various desired qualities, but don’t commit
- Identify potential technical risks
- Take stock of existing reusable architectural assets.
- Prioritize the architecture features. Based on:
  - Business value
  - Cost of implementing early vs cost of implementing late
Architectural Features’ Prioritization: Example

- Separation of UI, database, business logic, workflow logic into separate, distributed layers. (Benefits: portability, modifiability).
- Creation of an application framework. (Benefits: avoiding duplication of code in various stories; consistency of behavior; ease of adding cross-cutting requirements).
- Pessimistic Locking for concurrency management.
- Usability features in the framework: Save as Draft, Undo, Redo.
- Cross-cutting requirement: Audit trail.
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- Identify cross-cutting requirements
- Identify other considerations for architecture
- At every stage, validate with the product owner / customer / end users
- Discuss how to achieve various desired qualities, but don’t commit
- Identify potential technical risks
- Take stock of existing reusable architectural assets.
- Prioritize the architecture features. Based on:
  - Business value
  - Cost of implementing early vs cost of implementing late
- Include desired architectural qualities / strategies as product backlog items.
### Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
#5. Build Architecture Through Stories
# Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Story 1: Creating List of Items

Acceptance Test

- 1. ........................
- 2. ........................
- 3. ........................
- 4. ........................
- 5. ........................
- 6. ........................
**Implement the UI and the Story**

**Form_load:**
- fetch data from Items table
- populate the grid

**New_click:**
- display blank entry form
Implement the UI and the Story

SaveAndExit_click:
validate data
INSERT into Items values ...
exit form
etc.
Implement the UI and the Story

Form_load:
- fetch data from Items table
- populate the grid

New_click:
- display blank entry form
  //...on return from the entry form
- re-populate the grid from Items table
  etc.
## Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Story 2: Layered Architecture

Acceptance Test

1. 

2. 

3. 

4. 

5. 

6. 

Distributed Architecture

- ItemsUIManager
- ItemsBrowseForm
- ItemsEntryForm
- ItemsController
- Item
- ItemStore

Acceptance Test

1. ..................
2. ..................
3. ..................
4. ..................
5. ..................
6. ..................
Looking Back

• Miniscule architecture evolved, proven with the code
• Architectural stability and state of completeness: very low
# Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Story 3: Creating List of "Account Heads"

- Write acceptance tests.
- Design the UI.
- Start implementing the story.
- Similarity in implementation with the earlier story...
  => take up the next story also
<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Story 4: Application Framework Evolution
## Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Story 5: Test Framework, "Tax Codes" Story

- Implement a third story, by extending the framework.
- Helps in smoothening some rough edges in the framework.
<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo</td>
</tr>
</tbody>
</table>
Story 6: Framework, Handling Concurrency

• Implementing database record locking in the framework, instead of in individual stories.
• Locking strategies:
  – Pessimistic locking
  – Optimistic locking
  – No locking
• Who chooses the locking strategy?
• And how do they choose it?
• How do we help them make a choice?
• By writing scenarios with UI prototyping.
## Product Backlog

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maintaining the list of &quot;Items&quot; for stock and sale</td>
</tr>
<tr>
<td>2</td>
<td>Implementing Layering and Partitioning in &quot;Items&quot; story</td>
</tr>
<tr>
<td>3</td>
<td>Creating the list of &quot;Account Heads&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Refactoring the &quot;Items&quot; and &quot;Account Heads&quot; stories to extract common behaviour into a &quot;framework&quot; for the application</td>
</tr>
<tr>
<td>5</td>
<td>Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.</td>
</tr>
<tr>
<td>6</td>
<td>Implementing concurrency handling in the framework and testing</td>
</tr>
<tr>
<td>7</td>
<td>Account Heads: for &quot;Customer&quot; types, entering additional details</td>
</tr>
<tr>
<td>8</td>
<td>Framework: Including support for Save as Draft, Undo, Redo.</td>
</tr>
</tbody>
</table>
Similarly...

- Story 7: Account Heads, entering additional details for "Customer" type
- Story 8: Framework, support for Save as Draft, Undo, Redo
- Story 9: Framework, Audit Trail
- Likewise for other cross cutting functional or UI requirements. Examples:
  - Change history
  - Pagination in browse windows
  - Search within a browse window
  - Excel export from all browse windows
#6. Model and Implement Incrementally
Model and Implement Incrementally

• Model throughout the lifecycle, in small increments.
• Split large, complex stories.
• Example: Trainer tracking chart in a Training Monitoring System.
# Example: Trainer Tracking Chart

<table>
<thead>
<tr>
<th>Trainer(s)</th>
<th>9-Sat</th>
<th>10-Sun</th>
<th>11-Mon</th>
<th>12-Tue</th>
<th>13-Wed</th>
<th>14-Thu</th>
<th>15-Fri</th>
<th>16-Sat</th>
<th>17-Sun</th>
<th>18-Mon</th>
<th>19-Tue</th>
<th>20-Wed</th>
<th>21-Thu</th>
<th>22-Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vipul</td>
<td>Citgro</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>Varsha</td>
<td>Relianc</td>
<td>#Flex So</td>
<td>#Flex So</td>
<td>#Flex So</td>
<td>#Flex So</td>
<td>+Reliance</td>
<td>+Reliance</td>
<td>+Reliance</td>
<td>+Reliance</td>
<td>+Reliance</td>
<td>+Lehma</td>
<td>+Lehma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Viraj</td>
<td>Godrej</td>
<td>#Abbott I</td>
<td>#Abbott I</td>
<td>#Accel F</td>
<td>#Bhati</td>
<td>#Transoc</td>
<td>#Clariant</td>
<td>#Clariant</td>
<td>#Transoc</td>
<td>+Dun &amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swati</td>
<td>[2] Reli</td>
<td>+Mahind</td>
<td>Leave</td>
<td>Leave</td>
<td>#Nelco L</td>
<td>#Nelco L</td>
<td>Leave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anuradha</td>
<td>Dewan</td>
<td>#The Gre</td>
<td>#The Gre</td>
<td>Leave.</td>
<td>Attendin</td>
<td>Attendin</td>
<td>#Bank of</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minal</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>Altaf</td>
<td>G Ve</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>+CICI P</td>
</tr>
<tr>
<td>Zubair</td>
<td></td>
<td></td>
<td>Travel</td>
<td>+Godrej</td>
<td>+Godrej</td>
<td>Travel</td>
<td>#Transoc</td>
<td>+Godrej</td>
<td>+Godrej</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rajiv akya</td>
<td>APL Pri</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>Leave</td>
<td>#Transoc</td>
<td>#Transoc</td>
<td>#Transoc</td>
<td>#Transoc</td>
<td>#Transoc</td>
</tr>
<tr>
<td>Bhavna</td>
<td>leave.</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
<td>#Datam</td>
</tr>
</tbody>
</table>

### Work order Number
C/2007/2095 - S

### Day 1 of 3

**Venue**
Client location

**Location**
MIDC , Knowledge Center

**Client**
Datamatics Limited - 15

**Subject**
Visual Basic .NET 2.0

**Timings**
09:30 AM to 05:30 PM

**M E**
Rosy

**Expected Pax**
15

**Actual Pax**
12

---

**Pradyumn**

**Majrul**

**Yatin Naik**

**Chetan**

**Ganesh B**

**Girish Rao**

**Chita**

**Vanishali**

---

Pragati Software Pvt. Ltd., 207, Lok Center, Marol-Maroshi Road, Marol, Andheri (East), Mumbai 400 059. [www.pragatisoftware.com](http://www.pragatisoftware.com)
Model and Implement Incrementally

- Static grid with hard-coded data. Simple navigation keys.
- Dynamic grid, data retrieval from database.
- Cut, Copy, Paste, Excel export options.
- Options to add / edit work orders from the grid.
- Tracking of incremental changes. Change history.
Model and Implement Incrementally

- At the start of each iteration, during the sprint planning meeting, have discussions on incremental modeling, design changes.
- Apply architecture and design patterns as required, gently.
- Don’t worry about getting your architecture right on the first day.
- Test-driven development.
- Refactoring.
- Build automation.
- Continuous integration.
Questions?
Thank You

Pradyummn Sharma
pradyummn.sharma@pragatisoftware.com
Twitter: PradyummnSharma