Evolutionary Architecture and Design

Pradyumn Sharma
pradyumn.sharma@pragatisoftware.com
www.twitter.com/PradyumnSharma
What is Software Architecture?

- Structure of a system, comprising software elements, externally visible properties of these elements, and the relationships among them.

(Source: Software Architecture in Practice, Len Bass, et al)
Why Architecture?

• Critical to realization of many qualities of interest in a system.
• These qualities can be designed and evaluated at the architecture level.
Architecture

When? How? By Whom?
Scenarios to Consider

- New project, existing architecture framework from earlier applications
- New project, no existing architecture framework (new technology?)
- On-going project
Who is Responsible for the Architecture?

- The Architect?
  - Sitting on a pedestal
  - Takes all the architecture decisions
  - Does not dirty his hands with coding
  - Building ivory tower architectures

- A Central Architecture Team?
  - Even farther removed from the development team
Sounds Familiar?

• “THE architect built the architecture; handed down to us. It’s a robust architecture. But we don’t understand it.”
• “The architect has left the organization. Now we have to manage it somehow.”
#1. Involve the Entire Team
Involve the Entire Team

• Involve the entire team in:
  ➢ Domain modeling
  ➢ Architecture discussion, evolution, implementation
  ➢ Architecture reviews, technical debt sessions
  ➢ Refactoring

• Greater involvement, greater autonomy =>
  greater pride and motivation

• Create a culture and the desire in the team to get better all the time

• Cross-pollination, knowledge sharing across teams

• Pair programming
#2. Have an “Architecture Owner” Role
The Architecture Owner Role

• Responsible for:
  - Bringing the team together for all discussions regarding architecture envisioning and modeling
  - Facilitating architecture modeling and evolution
  - Helping in building a shared understanding
  - Helping the team members enhance their capabilities in understanding architectural principles and tradeoffs involved

• One of the most experienced developers can play this role.

• But involve the entire team. Strive to bring everyone to the same level of competency.
Architecture Owner Should Not...

• Dictate the architecture, preventing others from having their say.
• Guard the architecture as their personal property.
#3. Understand Your Product
Understand Your Product
Understand Your Product
How?

CIS: Product Vision

CIS (Corporate Information System) will be an integrated ERP solution, custom-built to the requirements of our organization. This will help manage all the information required and take appropriate and timely decisions for smooth functioning of the various activities and departments.

Broadly, the areas that will be covered by CIS will be:

- Customer relationship management
- Inquiries, proposals, visits, follow-ups, order booking
- Order execution, monitoring
- Procurement
- Financial accounting
- Human resources management, payroll

Using CIS:

- Executives from various functions will be able to create, view and edit information in a manner that is easy to find and navigate, that helps them stay well-informed about the important events and numbers, and take suitable decisions. For example, as sales personnel book or modify training orders, various concerned users will be able to view or decide details such as assignment of trainers, assignment of training centers, allocation of computers, reproduction of reading material, attendance tracking, invoicing, etc.
- Powerful data mining techniques will be incorporated to analyse historical data, discover patterns and trends, and predict future scenarios, in order to take decisions that help the organization maintain its leadership position.

The system will also:
Understand the Perspectives of the Stakeholders

• How to:
  - Participate in product backlog creation
  - Ask appropriate questions to understand underlying reasons

• Benefits:
  - Team understands business needs of the organization
  - Team understands why a functional / non-functional requirement is valuable to them
  - Recommended solutions more likely to be aligned to the business goals
A Traditional Approach…

• Create and document a detailed architecture of the system
• Possibly, load it with architectural and design patterns in anticipation
• Create an application framework
• Review the architecture
• Similarly, create detailed design, document it and review it
• etc.
The Other Extreme…

• No architecture?
• Just start coding and refactor?
Agile Architecture (and Design)

• Evolve iteratively
• through
  ➢ an initial envisioning
  ➢ implementation of stories
  ➢ refactoring and restructuring
• Avoid waterfall-style BDUF.
• But that does not mean no design up front.
• You don’t want to land up in a messy design due to lack of foresight.
#4. Create an Architecture Vision
Architecture Envisioning

• What?
  ➢ Establish a vision for the architecture and design of the system.
• 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

• Model on whiteboards.
• Take pictures.
• Don’t start documenting yet.
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
UI Prototyping

- Use paper prototyping or whiteboard prototyping.
- Assumption: both Web UI and rich client UI required (Windows Form). We begin with rich client UI.
### UI Prototype Example

#### Companies

- Sales executive
- Classification

| A | B | C | D | E | ..... | X | Y | Z |

- Get data / refresh
- First  Prev  Next  Last  New  Edit  Delete  ...  Exit

<table>
<thead>
<tr>
<th>Company name</th>
<th>Industry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Record: 3 of 100; Total: 1687 | Amen | ... |
UI Prototype Example

New Company

| General | Locations | Persons | Inquiries | Orders |

* Company name: __________________

Industry: __________________

Website: __________________

[OK] [Cancel] [Apply]
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
Identifying Desired Architectural Qualities

Examples

• Portability
• Usability
• Performance
• Security
• Internationalization

Take a few minutes to identify the desired architectural qualities for a system you have to work on.
More Examples of Architecture Qualities

- Modifiability
- Integrity
- Testability
- Conceptual integrity
- Accuracy
- Concurrency
- Customization points
- Internationalization
- Operations
- Maintenance
- Environmental impact
- Reliability
- Regulatory compliance
- Serviceability
- Support
- Dependencies on external systems
Architecture Workshop: Activities

• Domain modeling
• UI prototyping
• Identify desired architecture qualities and concerns
• Identify cross-cutting requirements
Identifying Cross-Cutting Requirements

Examples:

• Audit trail
• Alerts for important events that need attention

Take a few minutes to identify some cross-cutting requirements for a system you have to work on.
More Examples of Cross-Cutting Requirements

- Centralized error logging
- Excel export from all browse windows
Architecture Workshop: Activities

- Domain modeling
- UI prototyping
- Identify desired architecture qualities and concerns
- 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
- 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
Options for Achieving Architectural Qualities

• Portability
  - across database platforms
  - user interfaces (web, Windows, mobile, etc.)

• How to achieve?
  - encapsulate UI, business logic, database access code in separate modules
  - layering / partitioning
  - model-view-controller
Options for Achieving Architectural Qualities

• **Usability**
  - the ease of accomplishing a desired task by a user
  - the kind of support provided by the system to the user

• **Examples of architectural aspects of usability:**
  - ability to save as draft, partial data entered
  - undo operations
  - reuse data entered earlier

• **Examples of non-architectural aspects of usability:**
  - types of UI control used in a form
Usability

• How to achieve?
  - 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
Options for Achieving Architectural Qualities

• Performance
  ➢ generating system response to an event within some time constraint

• Examples of architectural aspects:
  ➢ specifying system resources such as CPU, memory, network bandwidth
  ➢ managing event rate
  ➢ quantum of communication among components, layering
  ➢ database: indexes, partitions, stored procedures

• Examples of non-architectural aspects:
  ➢ choice of algorithms
  ➢ implementation of selected algorithms
  ➢ writing efficient database queries
Examples of Architecture Qualities

• Security
  - preventing unauthorized access to data or services
  - dealing with DoS attacks
  - non-repudiation (a transaction cannot be denied by any party)
  - integrity

• How to achieve?
  - authentication of users
  - authorization of users, limiting access
  - audit trail
  - intrusion detection system
Examples of Architecture Qualities

• Integrity
• How to achieve?
  - 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
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
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
- Prioritize the architecture features. Based on:
  - Business value
  - Cost of implementing early vs cost of implementing late
Prioritizing the Architecture Features

• For the architecture features identified by you for your project:
  ➢ Rank them High / Medium / Low (or an appropriate scale) for business value impact.
  ➢ Assess the cost of implementing them early vs implementing late.
  ➢ Based on the above, create a prioritized Backlog of Architecture Stories.
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
- Prioritize the architecture features. Based on:
  - Business value
  - Cost of implementing early vs cost of implementing late
- Include desired architectural qualities as product backlog items.
Architecture Workshop: Duration

- In most cases, one to three days, depending on:
  - Size and complexity of the project
  - The familiarity and experience with the technologies involved and the domain
  - Availability of existing reusable architecture assets
Benefits

• Clarity on the critical technical issues
• Shared understanding of the architecture and design
• Helps in reducing technical risks
• Helps in better time and cost estimates
• Fosters better communication, sharing of ideas
#5. Build Architecture Through Stories
Acceptance Tests

• Select the first story to implement.
  ➢ Example: Cities master list maintenance

• Write the acceptance tests.

Acceptance Test

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

Form_load:
  fetch data from Cities table
  populate the grid

New_click:
  display blank entry form
  //…on return from the entry form
  re-populate the grid from Cities table
  etc.

OK_click:
  validate data
  INSERT into Cities values …
  exit form
  etc.
Implement the UI and the Story

• Run the acceptance tests.
• Demonstrate to the customer. Get their feedback on the UI and functionality.
• Refactor into a layered architecture.
Layered Architecture

CityUIManager

CityBrowseForm

CityEntryForm

CityController

City

CityDataStore

Acceptance Test

1. 

2. 

3. 

4. 

5. 

6. 
Distributed Architecture

- Refactor into a distributed architecture.
Looking Back

• Architecture is evolved, proven with the code
• Architectural stability and state of completeness: low
Select Another Story

• Select the second story to implement.
  ➢ Example: Companies master maintenance

• Goal: Establish an application framework through refactoring of the existing story and implementation of a new story.
## Companies Master

### List of Companies

<table>
<thead>
<tr>
<th>Companyname</th>
<th>Website</th>
<th>Assigned</th>
<th>Category</th>
<th>Type</th>
<th>Status</th>
<th>IT/NonIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z by 7</td>
<td>(null)</td>
<td>unassigned</td>
<td>Non Client</td>
<td>Non Core</td>
<td>Regular</td>
<td>IT</td>
</tr>
<tr>
<td>Global Services</td>
<td><a href="http://www.3globalservices.com">www.3globalservices.com</a></td>
<td>Roy</td>
<td>Client</td>
<td>Core</td>
<td>Regular</td>
<td>Non IT</td>
</tr>
</tbody>
</table>

### Company Entry

* Company name: 
  - 

* Website: 
  - 

* Assigned to: 
  - 

* Company Status: 
  - 

* Company Category: 
  - Non Client

* Actively Talking: 
  - 

### Additional Details

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>M. Ehlert</td>
<td>Marketing Executive</td>
<td>Marol, Andheri (East)</td>
</tr>
<tr>
<td>R. Wilke</td>
<td>IT Director</td>
<td>Marol, Andheri (East)</td>
</tr>
</tbody>
</table>
Framework Evolution
Framework Evolution

Common behavior in the framework’s classes (such as UIManager), with hook methods for story-specific implementation.
Test Effectiveness of the Framework

- Implement a third story, by extending the framework.
- Helps in smoothening some rough edges in the framework.
Another Example: 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.
Similarly...

- More architectural qualities are implemented through framework refactoring. Examples:
  - Database platform independence
  - Internationalization
- 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
Keep in Mind

• Prove your architecture with code.
• Deploy early and frequently; get the customer’s feedback.
• For any major architecture decision / feature to be implemented, have brainstorming session with the team before and after its implementation.
Don’t Over-Generalize

• Consequence of over-generalization:
  ➢ Increases maintenance burden
  ➢ Does not deliver value to the customer
  ➢ Difficult to know how much to test.

• Defer decisions to generalize.

• But design so that incorporating changes should be easier later.
#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 CIS.
Example: Trainer Tracking Chart

<table>
<thead>
<tr>
<th>Trainer(s)</th>
<th>3-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>Pragati Software Pvt. Ltd., 207, Lok Center, Marol-Maroshi Road, Marol, Andheri (East), Mumbai 400 059. <a href="http://www.pragatisoftware.com">www.pragatisoftware.com</a></td>
<td><img src="image_url" alt="Image of the Trainer Tracking Chart" /></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>
</tbody>
</table>
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.
#7. Review the Architecture and Design
Review Frequently

- Collective architecture and design reviews.
- Technical debt workshops.
Technical Debt

• Increasing harder to add features (functional as well as non-functional)
• Adding a feature with current framework and architecture takes much more effort than without it.
• Things if you don’t do now will hinder future development.
• Debt compounds over time.
• Put all the debt items in the product backlog.
• If defect rate is high, do root-cause analysis. Identify, prioritize and fix major reasons of defects.
• Bugs can cost many times more after deployment to fix.
Review Frequently

- Collective architecture and design reviews.
- Technical debt workshops.
- When required, restructure (perform large-scale refactoring).
- Code reviews.
- Pair programming.
- Add technical debt to the product backlog. Requires courage to communicate this.
- Keep refactoring, keep simplifying things.
#8. Document, When Needed
What to Document?

• Things that aren’t very obvious, such as:
  - reasons behind an important architecture / design decision,
  - operational procedures.

• Key aspects of the architecture, such as:
  - Deployment structure
  - Usage of the application framework, with some class diagrams and sequence diagrams
  - Domain model
Keep in Mind

- Create “lean and sufficient” documentation.
- Don’t over-document. Software is your primary goal.
- Document stable things, not speculative things.
#9. Build Reusable Assets
Build Reusable Assets

- Build reusable arch assets. Not just frameworks, also diagrams.
- But, remember: use before reuse.
Questions?
Thank You

Pradyumn Sharma
pradyumn.sharma@pragatisoftware.com
Twitter: PradyumnSharma