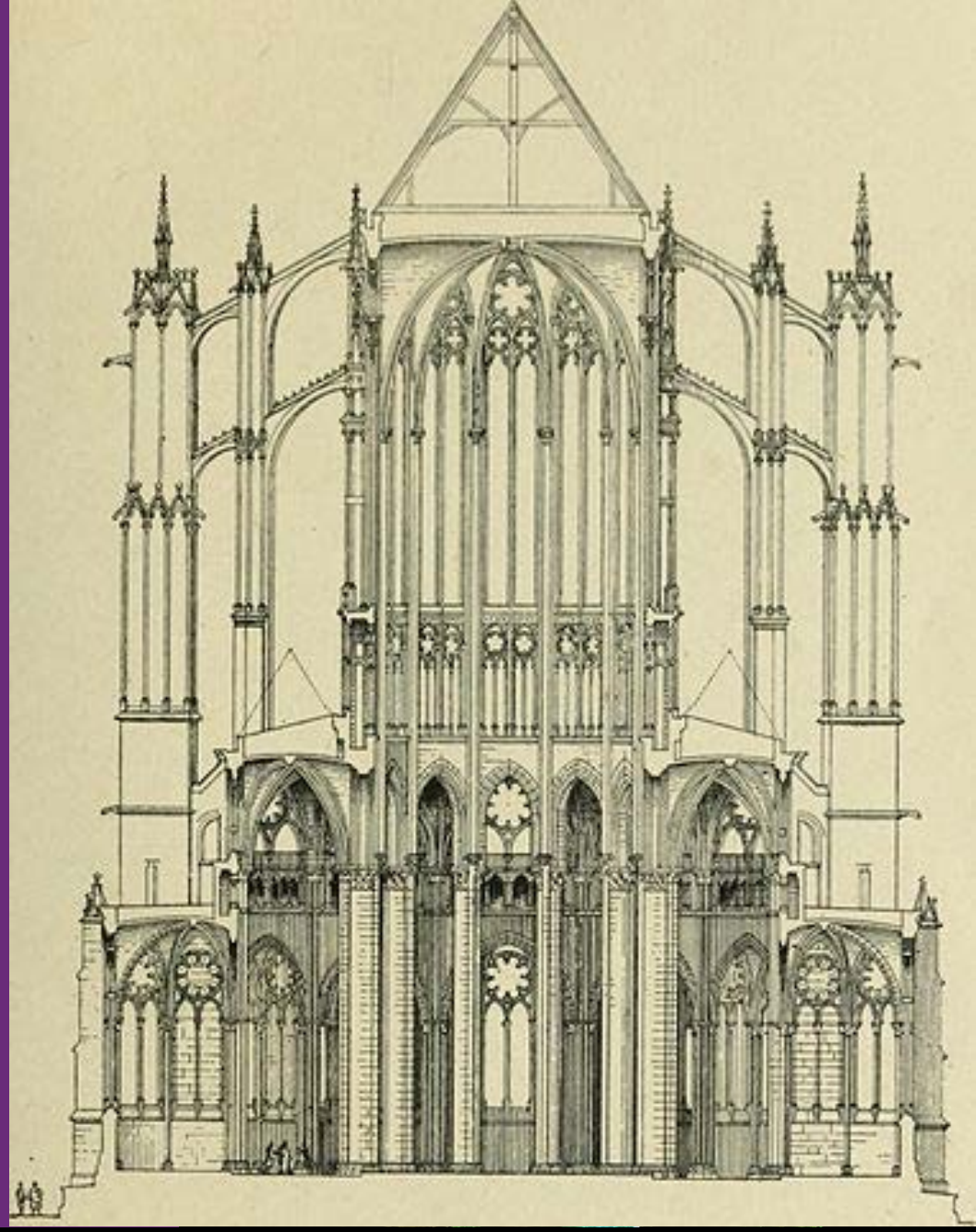
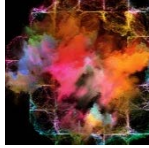
SATURN 2017

13th Software Engineering Institute Architecture Technology User Network Conference

Agile Architecture and Design

Pradyumn Sharma

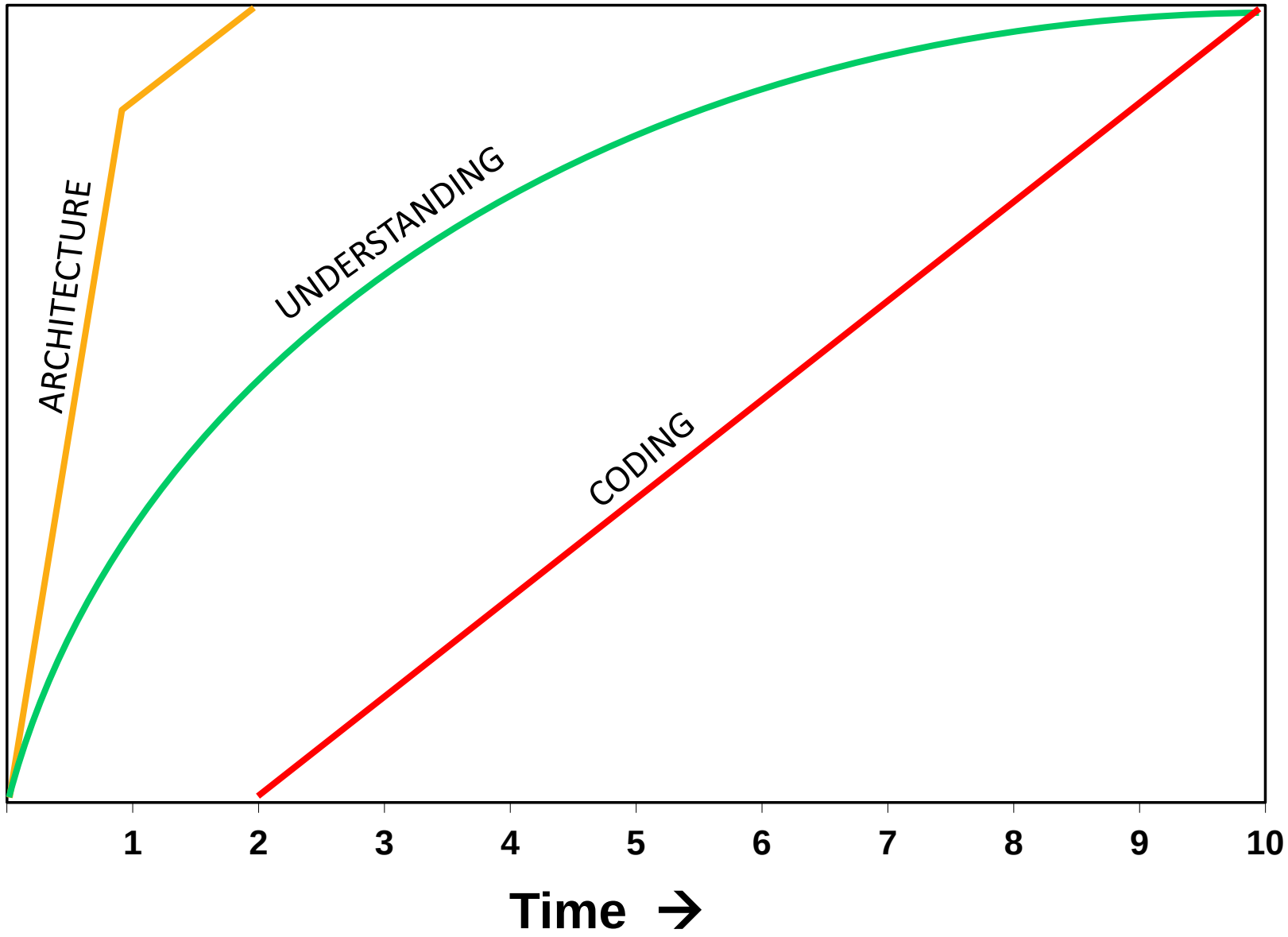




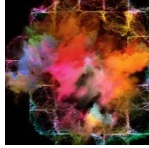
Presenter: Pradyumn Sharma

- CEO, Pragati Software, Mumbai, India (www.pragatisoftware.com).
- About 33 years in the IT industry.
- Training and consulting: Agile methodologies, Solution Architecture, Enterprise Architecture, Big Data and Analytics.

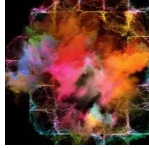
Architecture: Waterfall



Architecture: Agile



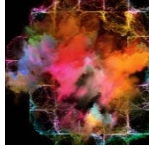
- Evolve iteratively
- Through
  - an initial envisioning
  - implementation of stories
  - refactoring and restructuring



Agile Architecture and Design: Overview

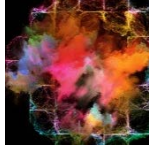
- Basics:
  1. Involve the entire team
  2. Have an “Architecture Owner” role
  3. Understand your product
- The key stuff:
  4. Create an architecture vision
  5. Build architecture through stories
  6. Model and implement incrementally

#1. Involve the Entire Team



Architecture Role: Traditional

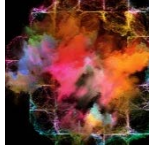
- Characteristics:
  - Specialist
  - Privileged
- Possible consequences (not all may apply):
  - Ivory tower architecture
  - BDUF
  - Analysis paralysis
  - Too abstract
  - Lack of shared understanding among team members



Architecture Role: Agile

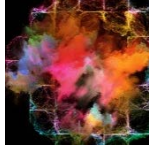
- Involve the entire team in:
  - Architecture discussion, evolution, implementation
  - Architecture reviews
  - Technical debt sessions
  - Refactoring and restructuring

#2. Have an “Architecture Owner” Role



Architecture Owner

- Master builder
- Depth and breadth of knowledge and skills
- Provides architecture leadership in a collaborative manner



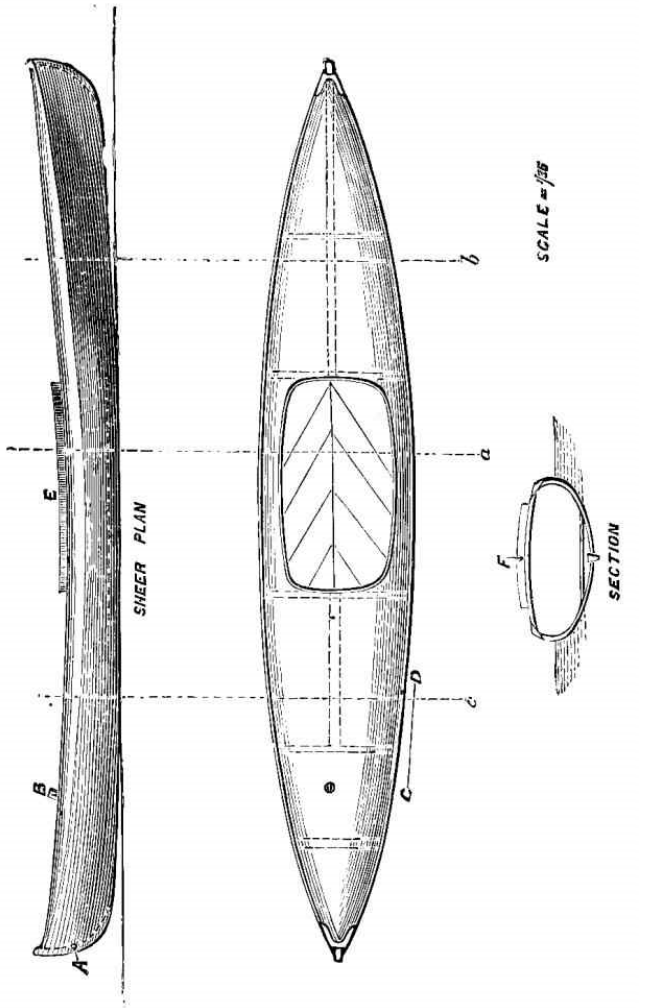
Responsibilities

- 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
- DONTs:
  - Dictate the architecture, preventing others from having their say.
  - Guard the architecture as personal property.

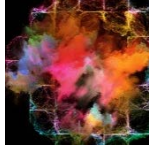
#3. Understand Your Product

Understand Your Product

Understand Your Product



#4. Create an Architecture Vision

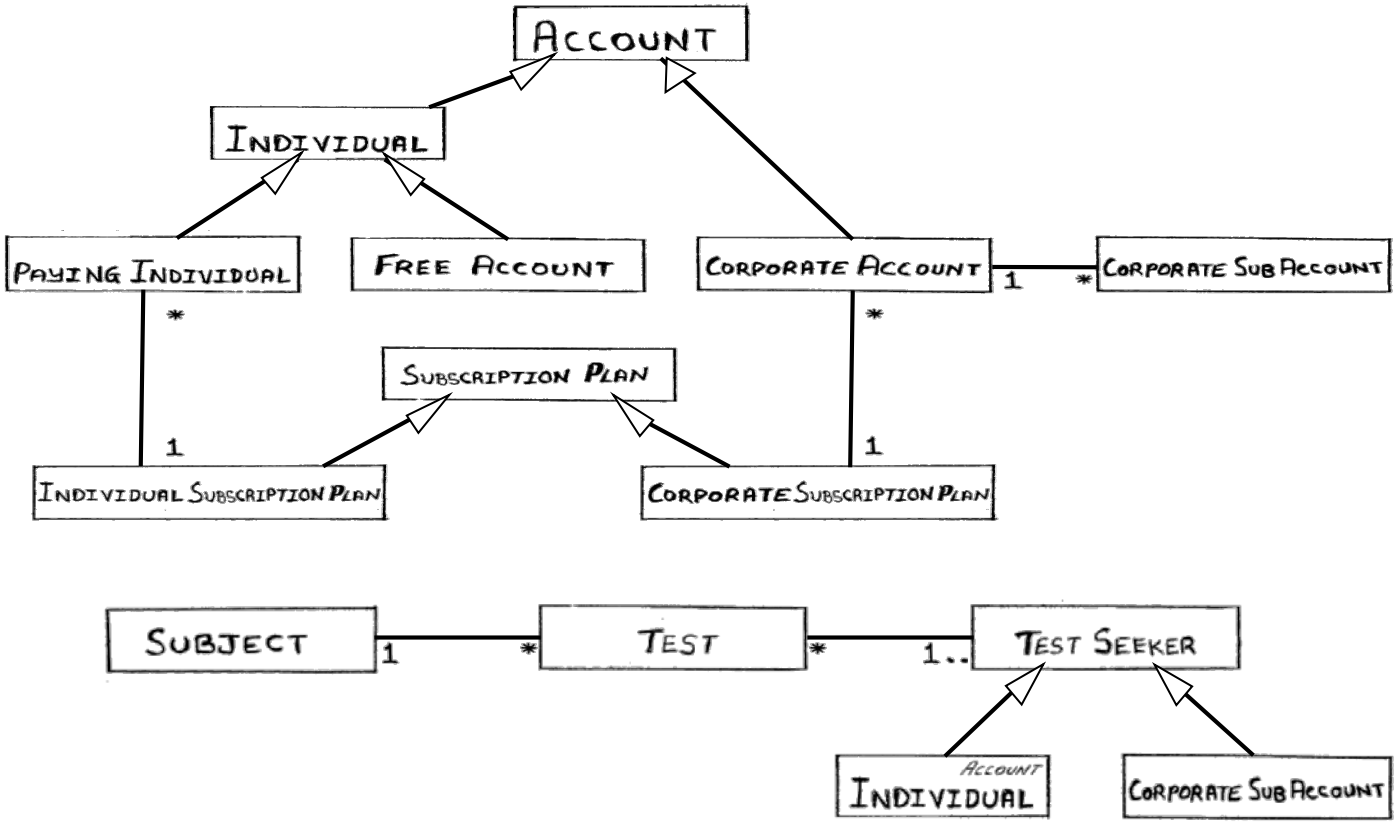


Create an Architecture Vision

- Create a shared vision of the architecture
- When?
  - Sprint zero.
- How?
  - Architecture / technical workshop.
- Who?
  - Team as well as the Product Owner (architecture must be based on requirements).

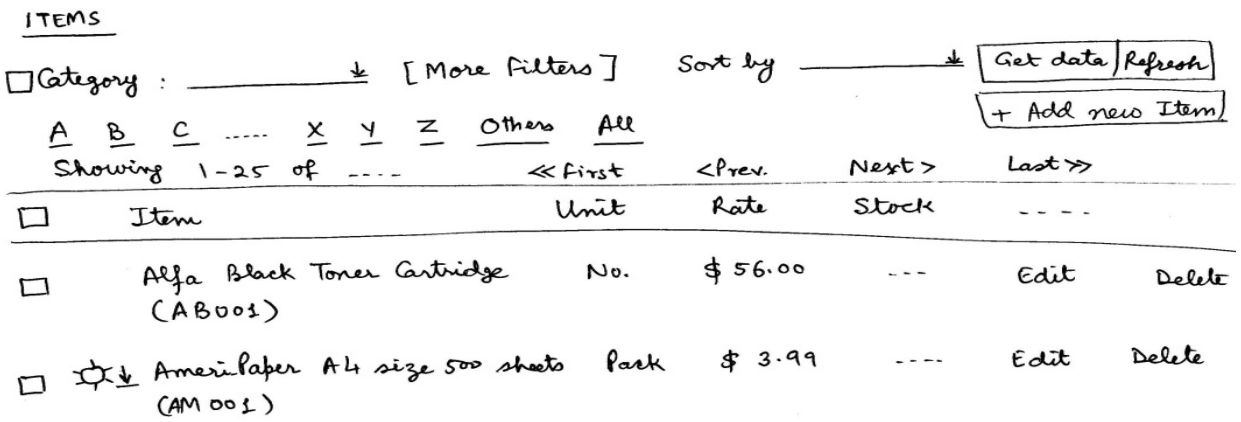
Architecture Workshop: Activities

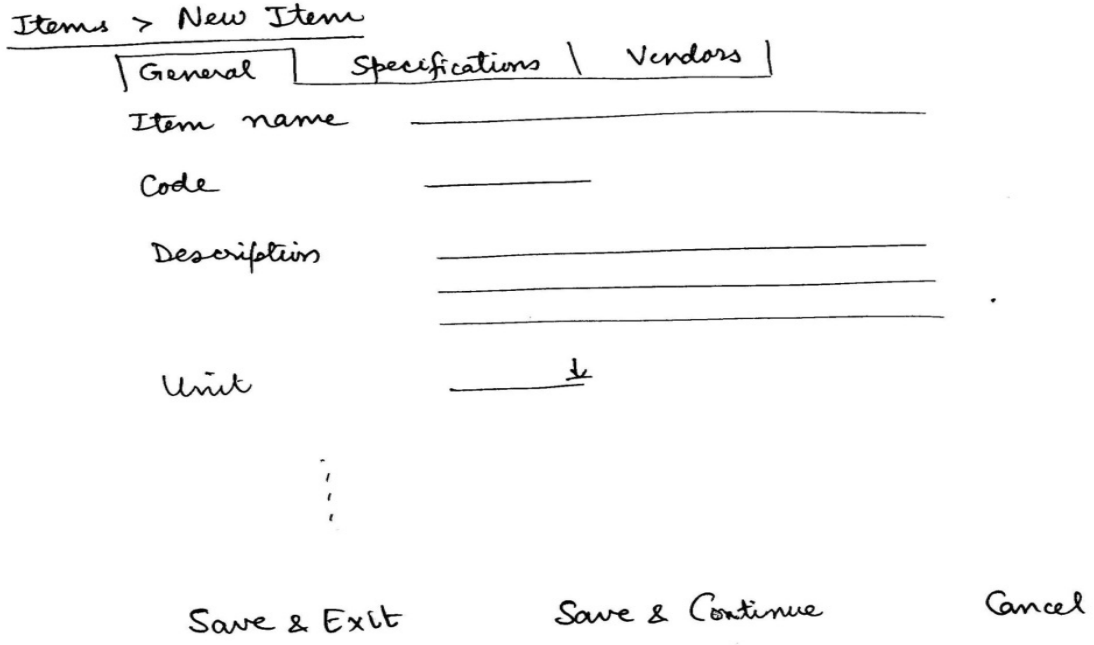
- Using whiteboard / paper…
- High level domain modeling



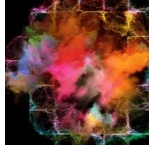
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping

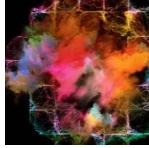




Architecture Workshop: Activities



- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Identify desired architecture qualities

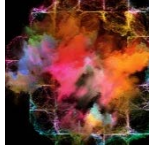


Architectural Qualities: Examples

- 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

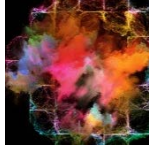
Stock Trading System:

- Security
- Dependencies on external systems
- Performance

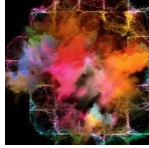
Online Travel Agency:

- Dependencies on external systems
- Usability
- Internationalization

Architecture Requirements

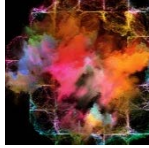


- Be specific
- What exactly do we mean by an architecture requirement, in a given context?



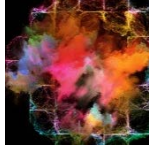
Portability Requirements: Examples

- 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.



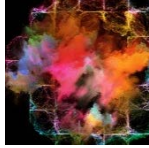
Security Requirements: Examples

- 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



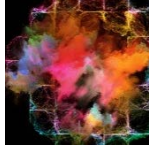
Usability Requirements: Examples

- 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

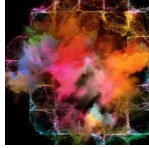
- System uptime should be at least 99.9%
- When a system service fails, an alert to the Infrastructure team should be raised.
- Even when one service of the system becomes unavailable the other services should continue to run as far as possible.
- It should be possible to deploy newer versions of the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



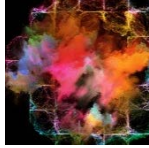
Integrity Requirements: Examples

- 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

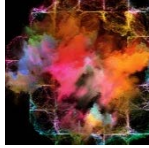


- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Identify desired architecture qualities
- Strategies for achieving desired qualities; don’t commit



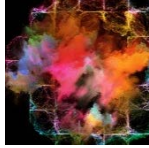
Portability: Strategies

- Separation of UI logic, business logic, data access logic in separte components
- Layering, partitioning



Modifiability: Strategies

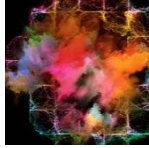
- Loose coupling between components
- Proper encapsulation
- Layering, partitioning
- Model-View-Controller
- Application framework providing common features and services, extention points for providing story-specific behaviour
- OO principles and design patterns
- Anticipating changes, providing hooks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



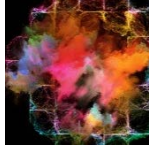
Concurrency: Strategies

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

Usability: Strategies



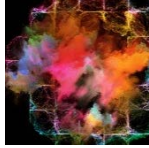
- Separation of the UI from the rest of the application
- Giving feedback about what the system is doing
- Letting the user issue commands such as Save as Draft, Cancel, Undo, show multiple views
- Design patterns: Command, Memento
- Maintaining a model of the task, or the system, or the user



Performance: Strategies

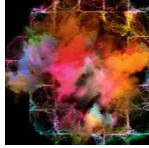
- Infrastructure planning: server capacity, clustering, failover, virtualization, network bandwidth
- Managing event rate
- Quantum of communication among components, layering
- Database: indexes, partitions, stored procedures
- Non-architectural aspects:
  - choice of algorithms
  - implementation of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

Security: Strategies

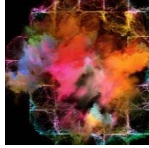


- Authentication of users
- Single sign-on
- Authorization of users, limiting access
- Audit trail
- Intrusion detection system

Integrity: Strategies

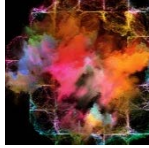


- Periodic run of batch programs to check integrity of derived data against raw data
- Discover patterns, fix integrity problems automatically, maintain history of such changes



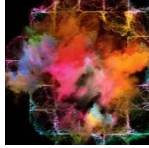
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don’t commit
- Cross-cutting requirements



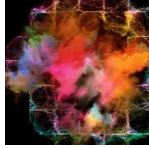
Cross-Cutting Requirements: Examples

- Audit trail
- Alerts for important events that need attention
- Centralized error logging
- Excel export from all browse windows



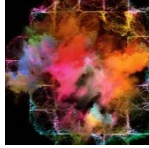
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don’t commit
- Cross-cutting requirements
- Other considerations for architecture



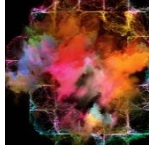
Other Considerations: Examples

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



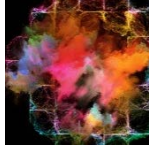
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don't commit
- Cross-cutting requirements
- Other considerations for architecture
- Validate (at every stage) with the customer



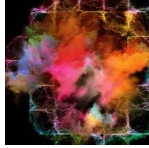
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don’t commit
- Cross-cutting requirements
- Other considerations for architecture
- Validate (at every stage) with the customer
- Potential technical risks



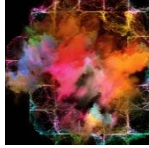
Potential Technical Risks: Examples

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



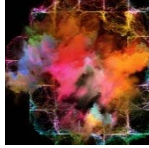
Architecture Workshop: Activities

- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don't commit
- Cross-cutting requirements
- Other considerations for architecture
- Validate (at every stage) with the customer
- Potential technical risks
- Take stock of existing, reusable architectural assets



Architecture Workshop: Activities

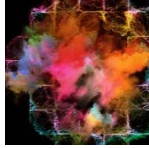
- Using whiteboard / paper…
- High level domain modeling
- UI prototyping
- Desired architecture qualities
- Strategies for achieving desired qualities; don't commit
- Cross-cutting requirements
- Other considerations for architecture
- Validate (at every stage) with the customer
- Potential technical 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.

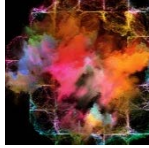
Product Backlog



Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.

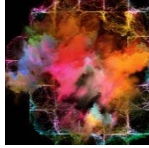
#5. Build Architecture Through Stories

Product Backlog

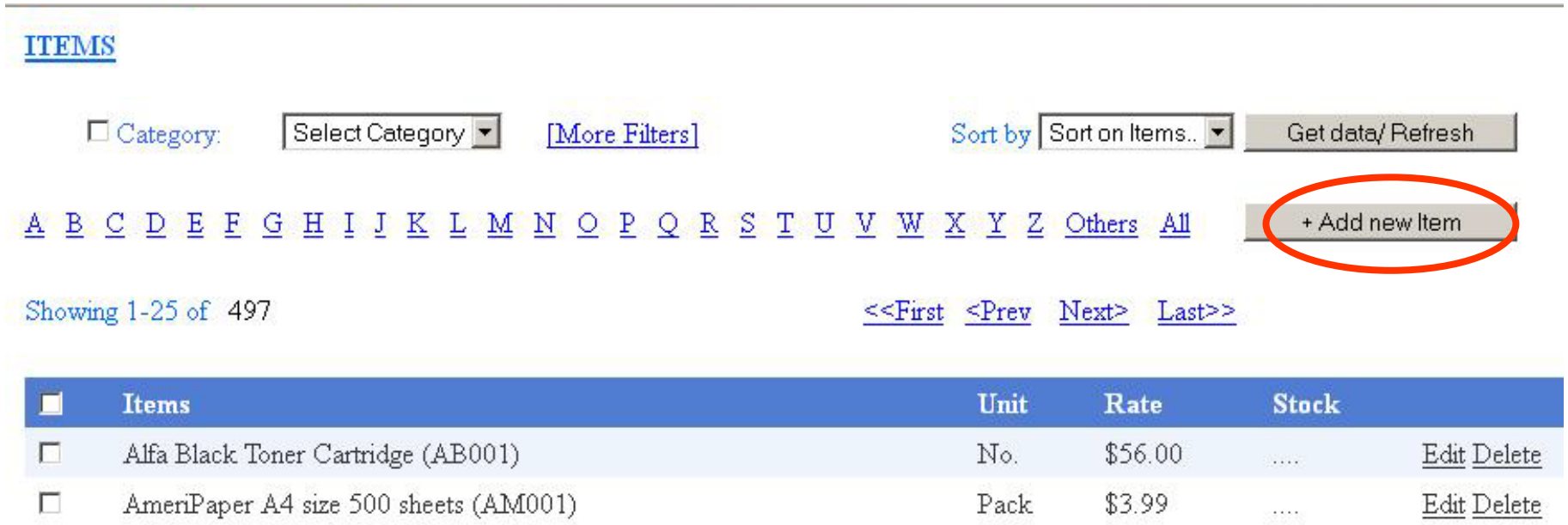


Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.

Story 1: Creating List of Items

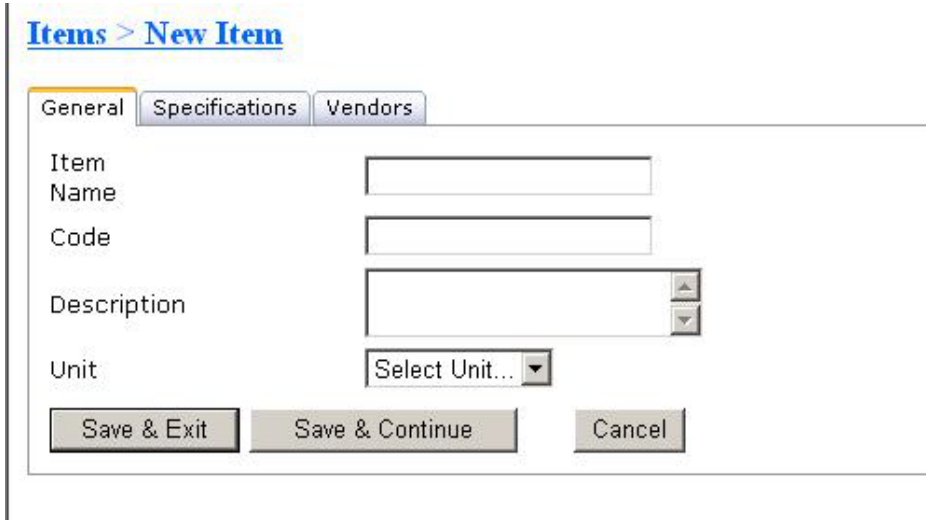


Implement the UI and the Story



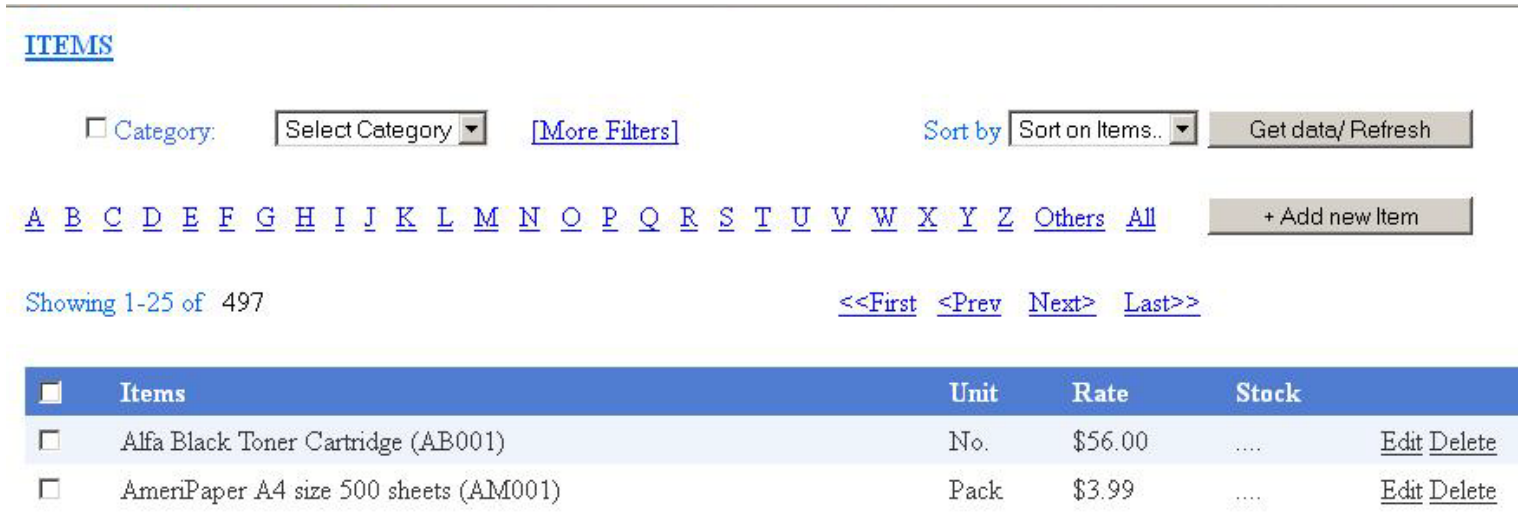
Form_load:
    fetch data from Items table
    populate the grid
New_click:
    display blank entry form

Implement the UI and the Story



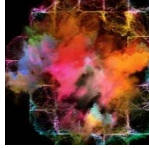
SaveAndExit_click:
    validate data
    INSERT into Items values …
    exit form
etc.

Implement the UI and the Story



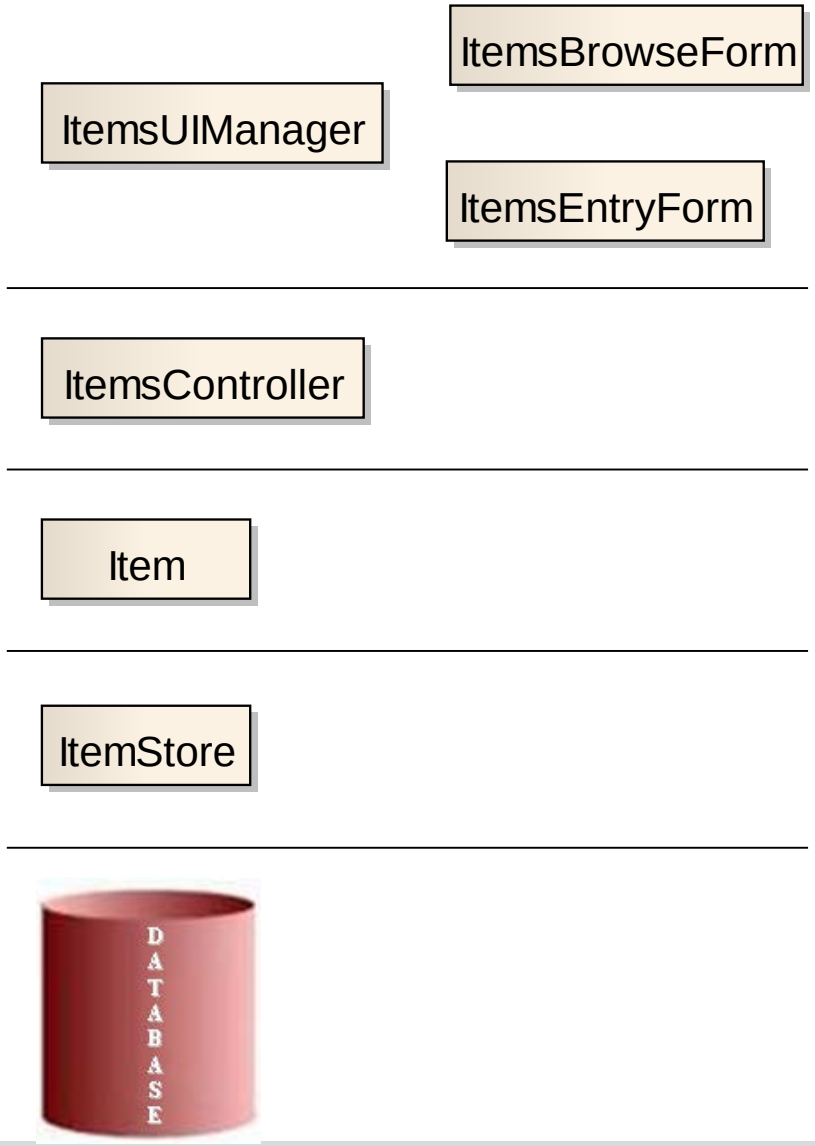
Form_load:
    fetch data from Items table
    populate the grid
New_click:
    display blank entry form
    //…on return from the entry form
    re-populate the grid from Items table
etc.

Product Backlog



Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.

Story 2: Layered Architecture



Distributed Architecture

ItemsBrowseForm

ItemsUIManager

ItemsEntryForm

ItemsController

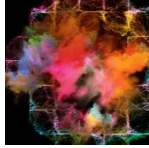
Item

ItemStore

Acceptance Test

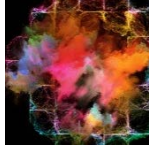
- ✓ 1. .....
- ✓ 2. .....
- ✓ 3. .....
- ✓ 4. .....
- ✓ 5. .....
- ✓ 6. .....

Looking Back

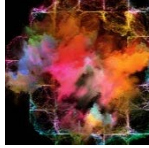


- Miniscule architecture evolved, proven with the code
- Architectural stability and state of completeness: very low

Product Backlog



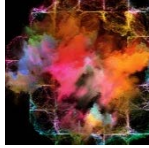
Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.



Story 3: Creating List of "Account Heads"

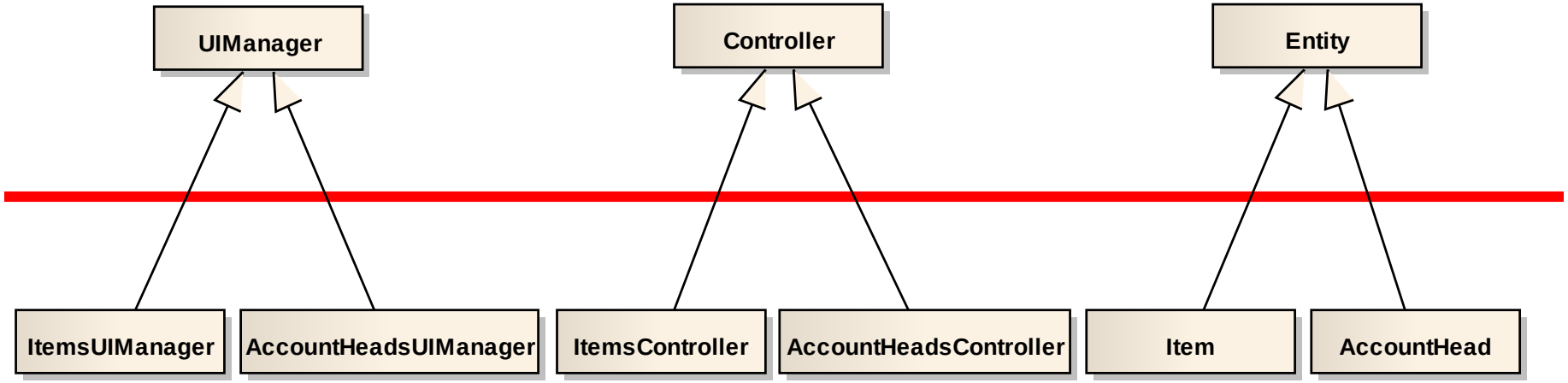
- Write acceptance tests.
- Design the UI.
- Start implementing the story.
- Similarity in implementation with the earlier story...
  => take up the next story also

Product Backlog

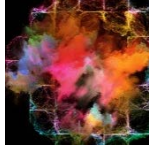


Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.

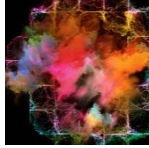
Story 4: Application Framework Evolution



Product Backlog



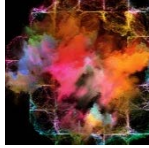
Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.



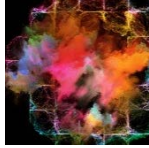
Story 5: Test Framework, "Tax Codes" Story

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

Product Backlog



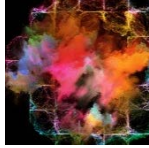
Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.



Story 6: Framework, Handling Concurrency

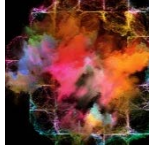
- Implementing database record locking in the framework, instead of in individual stories.
- Locking strategies:
  - Pessimistic locking
  - Optimistic locking
  - No locking
- Who chooses the locking strategy?
- And how do they choose it?
- How do we help them make a choice?
- By writing scenarios with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

Product Backlog



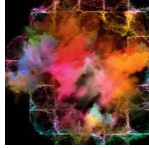
Product Backlog	
Sr.No.	**Story**
1	Maintaining the list of "Items" for stock and sale
2	Implementing Layering and Partitioning in "Items" story
3	Creating the list of "Account Heads"
4	Refactoring the "Items" and "Account Heads" stories to extract common behaviour into a "framework" for the application
5	Testing the framework by extending it to implement a third story (Tax Codes), and ironing out issues.
6	Implementing concurrency handling in the framework and testing
7	Account Heads: for "Customer" types, entering additional details
8	Framework: Including support for Save as Draft, Undo, Redo.
9	Framework: Audit Trail.

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

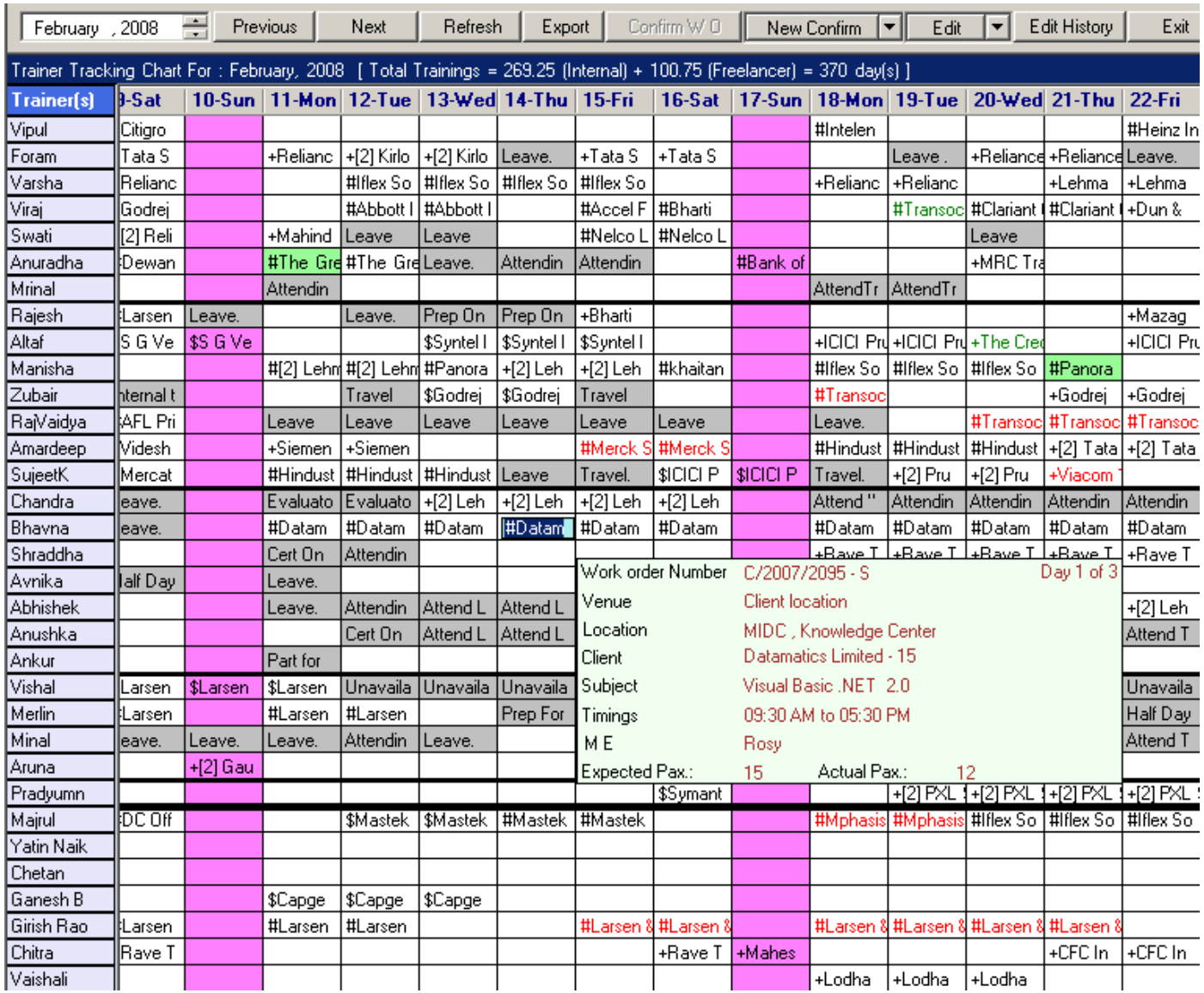
#6. Model and Implement Incrementally



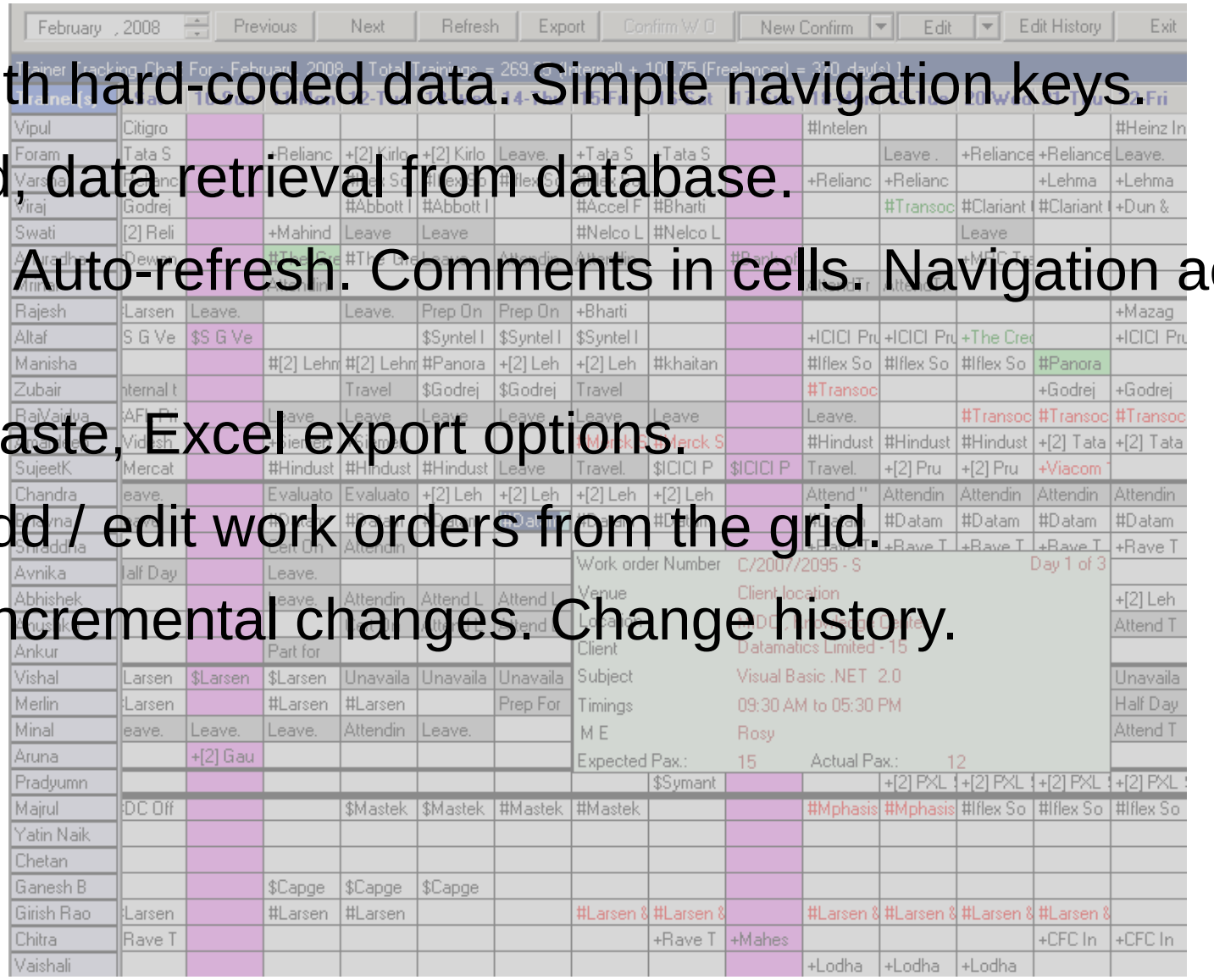
Model and Implement Incrementally

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

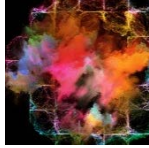
Example: Trainer Tracking Chart



Model and Implement Incrementally

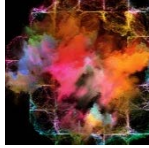


- Static grid with hard-coded data. Simple navigation keys.
- Dynamic grid, data retrieval from database.
- Cache table. Auto-refresh. Comments in cells. Navigation across months.
- Cut, Copy, Paste, Excel export options.
- Options to add / edit work orders from the grid.
- Tracking of incremental changes. Change history.



Model and Implement Incrementally

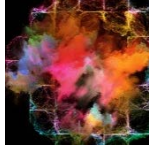
- At the start of each iteration, during the sprint planning meeting, have discussions on incremental modeling, design changes.
- Apply architecture and design patterns as required, gently.
- Don’t worry about getting your architecture right on the first day.
- Test-driven development.
- Refactoring.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



To Summarize

- Evolve architecture, collaboratively
- Establish a clear architecture vision
- Just enough architecture:
  - Identify and fulfill the architecture requirements
  - Establish and validate the architecture with the help of stories
  - Understand the significant elements and how they fit together
  - Identify and mitigate the key risks

Questions?



SATURN 2017

13th Software Engineering Institute Architecture Technology User Network Conference

Thank You

Pradyumn Sharma

pradyumn.sharma@pragatisoftware.com

Twitter: PradyumnSharma