18 Example of SIP

• Solo Iteration Process (SIP)
• Software begins by a very simple initial development
• Functionality is added one step at a time by software changes (SC)
  • initial development + 11 SC’s
Point of Sale

- Keeps inventory
- Keeps records of cashiers
- Allows sale of items at terminal
- Records store cash balance
- Keep track of fluctuating regular prices and sale prices
Sequence of tasks

1. **Initial version**: single item sold, only cash payment, single price, etc.
2. Expand inventory to support multiple items.
3. Support multiple prices with effective dates.
4. Implement promotional prices.
5. Support the log-in of a single cashier.
7. Add cashier session.
Sequence of tasks (cont.)

8. Keep detailed sale records such as item sold and date/time of sale.
9. Support multiple items per transaction.
10. Expand concept of cash payment to include cash tendered, change, and keep track of these values with regards to a specific sale.
11. Implement credit card payment.
12. Implement check payment.
1. Initial version (one class)
Task 1. Expand inventory

• Concept location, impact analysis
  • trivial

• Prefactoring
  • two extractions
    • class Inventory
    • class Item
After prefactoring

Store
- balance : double
- inventory : Inventory

Inventory
- item : Item

Item
- inventory : int
- price : double
- tax : double
Completing SC

• Actualization
  • new fields in the Item class
    • UPC, item name, current quantity
  • class Inventory
    • a new data structure to hold a collection of items

• Change Propagation
  • field Item is removed
Class diagram after Task 1

```
Store
- balance : double
- inventory : Inventory
+ getBalance() : double
+ processSale() : double
+ resetStore() : void
+ Main() : void

Inventory
- inventory : Item

Item
- upc : long
- name : string
- inventory : int
- price : double
- tax : double
+ calcSubTotal() : double
+ calcTotal() : double
```
Task 7: Add cashier session

• Explicit concept: “Session”
• Each login will start a new session
• Session data
  • login/logout times
  • number of transactions
  • cash totals
Before SC
Concept location

- Static dependency search begins at class containing `main()`
Impact analysis

• Highlighted classes represent the impact set
Prefactoring

- Extract class Session from CashierRecord
Actualization

- Add new fields
  - logout time
  - total cash
  - total number of transactions
- Create methods to increment cash and transaction totals
Change Propagation

- **CashierRecord**
  - Changed to keep collection of sessions

- **Cashier**
  - Supporting methods were added

- **Store**
  - `commitSale()` method changed to update current session data

- **Inventory** Visited, not changed
Testing

• 7 test classes before SC
  • 55 assertions

• 8 test classes after SC
  • 1 new test class to test Session (SessionTest)
  • relevant test methods moved from CashierRecordTest to SessionTest
  • 65 total assertions after SC
After SC
Task 9: Multiple Items per Sale

• System currently only supports one item type per sale.
• This SC expands on the sale concept to include multiple line items for a sale.
Concept Location

• Examine class dependencies, starting with `Main()` method in the Store class
  • located in Sale class, as the `saleItem` field.
    • explicit concept
    • we will expand the primitive concept
Impact Analysis

• Shaded classes are in the impact set
Prefactoring

- SaleLineItem class is extracted from the Sale class
Actualization

• SaleLineItem instance in the Sale class is changed to a collection of line items.
• quantity field added to SaleLineItem.
• Supporting methods added to the SaleLineItem class.
Change propagation

• Sale class
  • SaleLineItem instance changed to a collection of objects.
  • support methods added.

• Store class
  • processSale method changed to remove inventory for all line items.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Total Classes</th>
<th>Before</th>
<th>New</th>
<th>Changed (propagation)</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: initial</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2: inventory</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3: multiple prices</td>
<td></td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4: promo prices</td>
<td></td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5: cashier login</td>
<td></td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6: multiple cashiers</td>
<td></td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>7: cashier sessions</td>
<td></td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>8: detailed sale</td>
<td></td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>9: multiple line items</td>
<td></td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>10: payment</td>
<td></td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>11: credit payment</td>
<td></td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>12: check payment</td>
<td></td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>
# Case study data

<table>
<thead>
<tr>
<th>Phase</th>
<th>Explicit / Implicit Concept</th>
<th>Prefactoring</th>
<th>Postfactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: initial</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2: inventory</td>
<td>explicit</td>
<td>extract class</td>
<td>extract method move method</td>
</tr>
<tr>
<td>3: multiple prices</td>
<td>explicit</td>
<td>extract class</td>
<td>none</td>
</tr>
<tr>
<td>4: promo prices</td>
<td>implicit</td>
<td>none</td>
<td>extract method</td>
</tr>
<tr>
<td>5: cashier login</td>
<td>implicit</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>6: multiple cashiers</td>
<td>explicit</td>
<td>extract class</td>
<td>extract method method rename class</td>
</tr>
<tr>
<td>7: cashier sessions</td>
<td>explicit</td>
<td>extract class</td>
<td>none</td>
</tr>
<tr>
<td>8: detailed sale</td>
<td>explicit</td>
<td>extract class</td>
<td>none</td>
</tr>
<tr>
<td>9: multiple line items</td>
<td>explicit</td>
<td>extract class</td>
<td>none</td>
</tr>
<tr>
<td>10: payment</td>
<td>implicit</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>11: credit payment</td>
<td>implicit</td>
<td>extract superclass</td>
<td>none</td>
</tr>
<tr>
<td>12: check payment</td>
<td>implicit</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>
# Case study testing overview

<table>
<thead>
<tr>
<th>Step</th>
<th>Test Classes</th>
<th>Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>68</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>71</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>77</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>89</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>98</td>
</tr>
</tbody>
</table>
Tool support

- JUnit – unit testing framework for Java
- Abbot – functional testing framework for graphical user interfaces
- Refactoring browser
  - built in refactorings in Eclipse
  - we need more (extract class)