HF Boot Camp

Day -1

General "How-to"

- When you receive the problem- Do not think about implementation at all.
- Three step process:
    - What is needed - Keep defining the problem until you understand the business problem fully.
    - What is existing code/infrastructure for this feature? First you need to do some retrospect on what pieces of the puzzle already exists? You have to understand the existing code for this feature to come up with the best possible solution.
    - How am I going to test it. This is the most important piece - if you know how can I write an automated test case (if possible) or how do I test this feature - indicates how well you understood the problem.

I am ready to Implement!

- Writing Object Oriented Code - you are tempted to think procedurally - lack of any conforming tool - engineers often misconstrue the true meaning of OO.
- You start with your scrapbook code and soon enough that scrapbook code is a class and pile on more junk methods - hotch/potch of methods and attributes - viola I am OO!
- Define your classes carefully.
- Identify your entities you will deal with. Start to think about what behaviour can they have, what attributes can they have?
- Translate all attributes to variables.
- Translate all behaviours to methods.

How to approach a feature?

- Keep defining the problem until you have a clear vision of what Seth/Arnold/Lisa wants. Once all your questions have been answered following approach is what we take always.
- We follow a bottom up approach.
- Given a problem: DB --> BL --> UI
  - Find out what DB entities you will need.
  - Draw out the table structure
  - Write out the DAO
  - POJO
  - Given a problem you know what business logic you will need - example - you need to save an order, search the order by certain criteria, update an order, delete order etc. For all such methods write appropriate DAO methods.
  - Next you focus on BL. Wrap the DAO methods with BL.

contd..

- BL methods are the glue between UI and DAO. This layer should have the maximum amount of code, if you find yourself writing more in UI or DAO - step back and re-think your strategy.
- BL is where you will write your story.. Your code should read like a novel - flows eloquently not mix-match hotch-potch of methods. So naming your methods will help some one else read it like a story.
- When you had been given the demo UI - you already know how many clicks/actions were present.
- For each click - start writing a Action class - extending from EHRAbstractAction if no JSON, AbstractJSONAction is returned type is JSON. This is applicable for EHR only. For PM follow along PM Actions workflow.
- Think of all parameter names - Create ParamEnum in param package.

Contd..

- Map the ParamEnum to appropriate POJO, invoke BL method, return URL or JSON. Review the AbstractAction Template method.
- Now is the turn of JSP. Look at the static HTML - save as JSP.
- Decide whether you are going the Context route or Jquery/JSON route - this will influence whether your actions will be JSON or non JSON.
- Lot of time there is Context involved, In the converted HTML static page, think where will you put dynamic values - start substituting them with <%=ctx.getSomething() %> - Note at this time you do not have the actual method, niether the Context class- you are just thinking and typing what you would need.
- Once the bare minimum frame for JSP is ready - start the Context class - put in all the getter methods you placed in the JSP.
- JavaScript - think OO as well.
- Fire away testing..You are done!

Fax Service - Concept to Completion!

- We were going to use myfax.com web services to incorporate fax capability into our EHR.
- So we are dependent on the myfax capabilities.
- So the journey begins with "what can myfax webservices do" - what APIs are available to us.
- Their restriction will dictate what application constraint we have to enforce.
- So first step is - Engineers implement/study their web services. Document all the necessary restrcition for Arnold to build upon.
- This link provides entire documentation - but the first part was added by the engineer upon which Requirements team(Arnold/Seth) built the feature on.
- https://sites.google.com/a/healthfusion.com/outboundfaxservice/home
- Based on the web services capability and our feature adoption you come up with the table strcuture.
- https://sites.google.com/a/healthfusion.com/outboundfaxservice/project-updates

Contd..

- The business logic will revolve around the basic feature - send fax, receive fax, map error messages, display queue - see OutboundFAXBL.java
- One thing helps me organize my thought process is evernote - its free and great! Try it for your scratch pad as you think in your head - jot down your key bullet points/approach - to give shape later.

Conclusion

- Focus on the problem definition the most.
- Can you do a through local testing, starting from how to recreate the issue? Can you write the steps to test clearly for QA to follow?
- Think about whose responsibility it is when it comes to class design
  - Which layer does this belong to - DB/BL/UI
  - What kind of objects should this feature have?
  - What kind of correlation between the objects be? One to many, one to one?