Engineering Tripos Part IIB FOURTH YEAR

Paper 4M21: OBJECT ORIENTED SOFTWARE DESIGN

SOFTWARE SYSTEMS AND ENGINEERING

Crib for Examples Paper 4

1. (a) Method:

   - a mockup of the online banking system that implements main scenarios
   - observation and qualittive interviews by user experience specialist
   - video and screen capture
   - session time: 45 mins

   User Group:

   - since online banking is targeted at everyone, a user group representing all demographics is required (age groups, gender, professions)
   - assuming the online banking service is being aimed at existing users of other online banks, the user group should consist of users who are already using some online banking service, they could further be segmented into
     - casual users (use it less than once a month)
     - active users (use it at least 1-2 times a month)
     - very active users (use it at least once a week)

   (b) The results of the study:

   - conclusions based on observations of users by the specialists to identify for example a common source of confusion such as
     - if all users managed to navigate through the screens without requiring further assistance
     - if users managed to correct mistakes (typos) easily
     - if users at any point stopped to read the instructions for a significant period of time
   - qualitative user feedback, questions regarding the user perception of the user experience, such as
     - how users felt about using online banking service
     - whether they would recommend the online banking service to their friends
   - quantitative results: ranking/scoring by user groups (chart/tables)
     - the amount of time users spend on each screen in each scenario overall and by user group

- ranking each scenario on the scale from -3 to 3 on the following criteria

  efficiency (can I do what I want to do in a quick efficient way?)

  clarity (was it clear to me how to do what I need to do?)

  reliability (was I confident in results of my actions)

2. (a) In the Waterfall model, the step of identifying Requirements and producing the corresponding specification happens at the very beginning of the software project. This introduces the major risk as all other parts of the process are fully dependent on the correctness and fullness of the Requirements specification. In reality, it is often not feasible to produce a definitive Requirements specification as they are not always fully understood by the customer and also it is difficult to predict future changes in the technology, environment and business practices - something that is today is not necessarily "correct" tomorrow.

   Furthermore, as the Waterfall model provides no feedback from any of the steps to any of the preceding steps, the cost of change (whether due to the implementation or design or specification) is very high.

   (b) Designing software used for life critical applications requires a very high level of predictability, availability and resilience, i.e. regardless any malfunction, misuse etc. the software should not put a life at risk. Typically, the software and its design process will also be a subject of compliance to standards that capture some of the practices to minimize the risk of life threatening error occurring. Such development process can be less suitable for some of the Extreme Programming practices:

   i. Daily Deployment - the lifecycle of the life critical software is much more stable. As any change introduces risks, even if the system is in active development, it will not be practical to be updating any existing deployment.

   ii. Shared Code - the collective ownership of the code is also more difficult as e.g. the code related to the medical diagnostics will be specialised and can include implementations of advanced computational techniques and as such should be maintained by the experts.

   iii. Single Code Base - the lifecycle of the medical diagnostics systems can span over decades and will likely to have requirements to support multiple versions running on legacy hardware/software, as such having separate branches can be necessary.

   iv. Team Continuity - it maybe also difficult to maintain the same team continuity due to the extended system lifecycle, instead the supporting processes could be in place to ensure effective knowledge transfer between new/old team members.

   v. Incremental Deployment - this would be difficult to maintain as the release process is costly (each release has to pass through compliance with corresponding regulations) and not practical for the core functionality of the application: the system is either be ready to diagnose patients or it is not.

vi. Stories - while they could be used for facilitating daily/weekly project management, it is likely that the full "standard" requirements specification is also required for the compliance and auditing.

3. There are four main test types: Unit tests, Integration/Component tests, Functional tests, Usability tests, Performance tests that could be used.

   Unit tests - these tests should cover individual small software modules (e.g. class methods) to test their correct behaviour. These tests are used to check that a class method can be called with given parameters it produces the result expected, as well as that e.g. given invalid parameters the method handles the error as expected. In the photo sharing application an example of such unit tests could be:

   – testing a method that is used to calculate how many "followers" the user have: get the followers count, add a number of followers, get the followers count again and compare it with the expected number
   – testing a method for a new user registration: call it with invalid characters for username ? check that the error condition is raised.

   Integration tests - they check how parts of the system interact with each other and help to test if the entire subsystem work correctly. An example would be to test the process of submitting the photo from the Mobile App to the Server, the test script should test

   – using the Mobile App UI, select an existing image from the phone and submit it to the user?s account
   – the Server component should receive the image and persist it
   – check that the image has been received by the Server, that it is the correct image (no corrupted in transfer) and the Server has persisted the image and associated it with the user account

   Functional tests - each feature provided by the system should be tested that it is implemented and performs according to the specification. For example, a feature of submitting an image to the server should have functionality of:

   – tap on a "Post" Button to submit an existing image, navigate to the images saved on the phone and allow to choose an image, confirming the selection
   – submit the image to the Server by tap on "Submit" Button
   – show the confirmation that the image has been posted successfully

Usability tests - these can be used to discover how real users interact with the product: how they feel about it, can they find the features of the product when they need them as well as to gage the perceived utility of the product: is it a must have app or is it something users can leave without, how often would they use it? The usability test designed for the photo sharing application can ask users to complete a number of tasks that could be typical: register, share an image, follow another user.

Quantitative data can be collected on e.g. how long does it take the user to share an image.

Qualitative interviews can ask how users felt about the application, did they find easy to do things they needed to do and what part of the application they liked the most.

When developing a new software product for the open market (i.e. no specific requirements from the customers), the usability testing helps the ?customer discovery? process as it helps to determine what features of the application users engage with and see as most valuable and focus the development effort accordingly.

Performance tests - these tests should make sure that the system meets user expectations (as measured in usability tests) in terms of performance and availability of the service. The performance can be tested both on the client side: how long does the App take to submit an image to the server (on different types of phones with different types of network connections) and on the server side: how many concurrent connections can the system handle before the performance deteriorates.

4. Once the cause is identified for each of 5 Whys, the corrective actions should ensure that all factors (human, technology, process) that contributed to the problem have been addressed. The actions should require effort proportional to the damage caused and the risk of causing a similar problem again. In the case of the failed Support URL, these actions could be as follows:

   – To address the rejected submission the team needs to submit an update/clarification to the review team
   – The team also needs to fix the failure reported by the review team and setup the URL on the server.

   Without using 5 Whys and corrective actions at each step, the team would stop after this two steps. However, in this case there is an equal chance of this problem happening again. The following further steps help to reduce this risk.

   – The problem should have been spotted in testing before the application was submitted ? adding automated checks for Support URL validity will ensure that an alarm is raised should it be unavailable for any reason
   – Similar to code reviews, implementing a review process for all data required for the submission will help to spot any issues

- Implementing an automated app submission process (including a step of checking the Support URL) will make the submission process more predictable and repeatable, eliminating a human error from future submissions.