

Paper 3F6: Software Engineering and Systems

UI Design and Software Management

Solutions to Examples Paper 4*User Interface Design*

1. A use case specification should include: a definition of the user's goal; list of any preconditions; criteria for successful completion; and list of the main steps in the activity flow.

- (a) **goal** transfer £100 from the current account to the savings account

preconditions user has £100 in current account

success criteria accounts show amount transferred

activity flow

- i. view summary of account totals, and verify current > £100
- ii. select "transfer money" operation
- iii. enter source account
- iv. enter target account
- v. enter amount to transfer
- vi. display summary of transaction and if ok confirm
- vii. if summary not ok edit transaction details
- viii. view summary of account totals, and verify transfer has completed

- (b) **goal** pay a bill of £36.50 to BT plc, customer number EA3482828

preconditions user has £36.50 in current account

success criteria current account shows payment of £36.50 to BT plc

activity flow

- i. view summary of account totals, and verify current > £36.50
- ii. select "make payment" operation
- iii. select payee from list of payees
- iv. if payee not listed, create a new payee giving customer number
- v. enter amount to pay
- vi. display summary of transaction and if ok confirm
- vii. if summary not ok edit transaction details
- viii. view summary of account totals, and verify payment has been recorded

- (c) **goal** close all accounts and transfer any outstanding balance to an account

preconditions None

success criteria Accounts are closed and balance transfer confirmed

activity flow

- i. select "close accounts" operation
 - ii. check confirmation request and confirm
 - iii. enter target bank account name
 - iv. enter target bank sort code
 - v. enter target bank account number
 - vi. check summary of closure balance and target bank details and if ok confirm
 - vii. if summary not ok edit transaction details
2. A prototype of the internet banking system described in Q1 has been implemented:
- (a) a usability test requires a panel of trial users. Tests should be conducted in a room where each user can be observed, and questioned on their actions. All user interface actions should be logged. Each user should be given a set of tasks to perform, these tasks should cover all of the use cases identified at the specification and design stages.
 - (b) metrics and measurement. The usual metrics are ease of use/learnability, speed of operation, robustness, recoverability and adaptability. In addition, the overall user subjective impression is important. Some of these metrics can be computed from the logged data (eg speed of operation, robustness) others can be inferred from user behaviour. Users are typically given questionnaires and asked to rank ease of use, etc on a 5 point scale.

Software Management

3. (a) A code inspection is much more formal than a walk-through.
- (b) There are 5 possible cases, any four of these will suffice
- i. list is initially empty
 - ii. x is smaller than any number in the list
 - iii. x is larger than any number in the list
 - iv. x is already in the list
 - v. the normal case i.e. none of the above
- (c) The walk-thru should consider each of the special cases and make any other relevant observations. Points that should be picked up are
- case i appears to be ok.
 - case iii looks faulty. If an item is added to the end of the list, it appears that the succ pointer of the original end of the list is not updated.
 - case iv appears to be ok.
 - case ii and the general case are handled by the same code. Case ii appears ok, however, as with case iii, in the general case the preceding item's succ pointer is not updated.

Note that a code review aims only to identify potential sources of bugs not to fix them.

- (d) If the integer numbers i were known to lie in the range $0 < i < N$, the `insert` function could be simplified by initialising it with two *sentinel* items. One with a value of 0 and one with a value of N. This would remove the three special cases and simplify the code.

4. The main cases to test are

- (a) required element is less than first element in the array
- (b) required element is greater than last element in the array
- (c) required element is first in the array
- (d) required element is last in the array
- (e) required element is near the middle of the array

Since the search does a *binary chop* of the array, it would also be prudent to check retrieval at the exact centre of the array for both the case N is even and N is odd (since the routine could be used for other values of N).

A suitable test harness would need to populate the array and then check retrieval of various elements. A simple version of this is given below but more elaborate versions could easily be derived.

```
void SearchTest()
{
    const int N=100001;
    int i,array[N];
    // fill array with consecutive numbers
    for (i=1; i<=N; i++) array[i]=i;
    // now check retrieval for each of the test cases
    if (binary_search(array,0,N) != 0) reporterror(0);
    if (binary_search(array,N+1,N) != 0) reporterror(N+1);
    if (binary_search(array,1,N) != 1) reporterror(1);
    if (binary_search(array,N,N) != N) reporterror(N);
    if (binary_search(array,N/2,N) != N/2) reporterror(N/2);
}
```

Note that it would be simple to check all values ie

```
for (i=0; i<=N; i++)
    if (binary_search(array,i,N) != i) reporterror(i);
if (binary_search(array,N+1,N) != 0) reporterror(N+1);
```

and this would be a good solution provided that compute time was not an issue.

Steve Young
February 2008