DE/ION/006

ICL Pathway Memorandum

To: Phil Hemingway, Dave McDonnell, Steve Doyle, Les Ong, Brian Orzel, Martin Smith, Steve Warwick, Alan Ward

CC:

From: Chris Humphries

Date: 12 March 1999

Re: EPOSS Product Improvement Options

Introduction

The following is a summary of the discussions of the workshops held on 25th February and 9th March, and also includes some post-25/2 workshop input from Les Ong.

Candidate product improvement measures

The following measures were identified as possible ways of improving the EPOSS product to enhance its maintainability and to reduce the risk of severe operational problems.

Stock unit dll

This is too large and complex and would be better split into a number of separate components, e.g. GUI, business rules, stock unit and user admin, logon/logoff, stock unit balancing, weekly cash account.

Reporting

This is too generic and too complicated. Any attempt to modify it in one place results in a proliferation of knock-on effects in other places. This makes it hard, risky and time-consuming to make modifications to the product. A number of requirements to make such changes will inevitably arise in live service.

Either there should be one reporting engine for each of a number of areas, or a new mechanism should be written for translating between message store data and report formats, based on an analysis of the underlying messages and data.

Attribute grammar

Document and take out any redundant attributes (not primary or secondary mappings). The latter would entail reliance on the agent documentation of the attributes used by the agent code.

Cash Account

Rewrite cash account report as a separate report. Bring it into line with the POCL view of the cash account and align the two reference data models. Possibly do the rewrite in C for performance.

Error handling

Implement error trapping, especially in the area of peripheral handling.

Business rule validation

Make the application enforce compliance of inputs with business rules.

Logic threads

Make sure that the flow of control through the programs correctly implements the specified processing. Eliminate dead ends etc. caused by e.g. business data errors.

Comments

Insert/review code comments.

Tidy-up code

Remove commented-out code and obsolete code. Use any available tools to identify code that logically cannot be executed.

Modularise code

Separate business rule code and GUI code where this currently occurs in the same object.

Document

Provide documentation of the revised system (this will require a certain amount of documentation of the existing system as a pre-requisite for change).

Error messages

Tidy up and make consistent the existing error messages. Currently the messages use upper case inappropriately, are not always accurate and are in need of general improvement.

Menu navigation

Make consistent the presence/absence and usability of the Prev Button.

Evaluation criteria

Each of the above improvement measures was evaluated against the following set of criteria. The first two criteria tend in favour of implementing a measure and the remaining three tend against.

Benefit

The benefit of implementing a particular improvement measure in terms of the product's enhanced maintainability (time/effort/risk), stability, and robustness.

Do nothing risk

Risk that if a particular improvement measure is not implemented, a severe software problem will arise in live operation that is difficult or impossible to manage and recover from. This could arise in the initially released system or from an error implementing a subsequent change to the software. There is also the risk that, due to poor maintainability, a business-critical change could not be implemented within the required timescale.

Destabilisation risk

The risk that implementing a particular improvement measure will destabilise the product, leading to an position that is difficult or impossible to manage and recover from.

Migration risk

The risk that, for a particular improvement measure, the process of migrating in live service from the old product to the new product embodying the measure will encounter unforseen difficulties, leading to an position that is difficult or impossible to manage and recover from.

Cost

The time, effort and budget required to implement the measure.

Evaluation matrix

Each criterion is evaluated high (H), medium (M) or low (L). The cost bands are, for code, unit test and link test, L: 0-10 man days M: 10-30 man days H: >30 man days.

	Benefit	Do nothing risk	Destabilisation risk	Migration risk	Cost
Stock unit dll	H	H	H	M	H
Reporting	H	H	H	H	H
Attribute grammar	H	L	L	H	L
Cash Account	L	L	H	H	H
Error handling	L	M	M	M	H
Business rule validation	L	L	M	M	M
Logic threads	L	M	L	M	M
Comments	M	L	L	L	H
Tidy-up code	M	L	M	L	M
Modularise code	H	L	H	L	H
Document	M	L	L	L	H
Error messages	M	L	L	L	L
Menu navigation	M	L	H	L	H

First cut evaluation

A first-cut evaluation of the above data is as follows.

Unweighted score moduli:
H=3
M=2
L=1

Score signs:
+ve for criteria tending in favour of implementing a measure
-ve for criteria tending against implementing a measure

Since there are two criteria tending in favour three tending against, a weighting factor of 3 is applied to positive scores and 2 to negative scores.

This gives the following scorecard:

	Benefit	Do nothing risk	Destabilisation risk	Migration risk	Cost	Total
Stock unit dll	9	9	-6	-4	-6	2
Reporting	9	9	-6	-6	-6	0
Attribute grammar	9	3	-2	-6	-2	2
Cash Account	3	3	-6	-6	-6	-12
Error handling	3	6	-4	-4	-6	-5
Business rule validation	3	3	-4	-4	-4	-6
Logic threads	3	6	-2	-4	-4	-1
Comments	6	3	-2	-2	-6	-1
Tidy-up code	6	3	-4	-2	-4	-1
Modularise code	9	3	-6	-2	-6	-2
Document	6	3	-1	-2	-6	0
Error messages	6	3	-2	-2	-2	3
Menu navigation	6	3	-6	-2	-6	-5

This gives the following ranking.

	Benefit
Error messages	3
Stock unit dll	2
Attribute grammar	2
Reporting	0
Document	0
Logic threads	-1
Comments	-1
Tidy-up code	-1
Modularise code	-2
Error handling	-5
Menu navigation	-5
Business rule validation	-6
Cash Account	-12

FUJ00121099

Second Workshop Discussion

The above evaluation was accepted as a starting point for discussion.

The below table summarises the points raised against each measure.

Measure	Benefit	Comment
Error messages	3	Desirable but not a measure to enhance product maintainability and robustness.
Stock unit dll	2	Too much effort to do all at once. May be possible to split out modules one at a time, but a design is needed first.
Attribute grammar	2	Most of the benefit comes from the documentation. Redundant attributes is desirable for space saving, but does not add much toward maintainability and robustness.
Reporting	0	More beneficial than Stock unit, but also needs prior design before decision can be made what to do in this release and what, if anything, can be done later.
Document	0	Should be undertaken as modules are opened for development work. Waste of time documenting existing stock unit and reporting modules.
Logic threads	-1	Too risky. Deal with through error processing.
Comments	-1	Implement as modules are opened for development work.
Tidy-up code	-1	Removal of obsolete code too risky. Removal of commented-out code to be implemented as modules are opened for development work.
Modularise code	-2	Not worth it.
Error handling	-5	Implement high level error trapping immediately and detail error trapping as modules are opened for development work.
Menu navigation	-5	No.
Business rule validation	-6	No.
Cash Account	-12	No.

Conclusion

A Design activity will be undertaken to design the product improvement implementations for Stock Unit and Reporting. It will then be decided what development can be achieved within Release 2+ timescales. This work will be scheduled. Some remaining work may be planned for later releases.

A technical author will be engaged to document the attribute grammar. The best person to do this would be Trish Morris, since she already knows the subject area and the Counter Development team.

FUJ00121099

High level error processing will be implemented across the product.

The following measures will be implemented progressively, as modules are opened for development work:

- Comment code.
- Remove commented-out code.
- Implement low level error handling.
- Provide LLD module documentation.