System development in relation to civil engineering failures

by Champake Mendis

New creations are produced in various fields. It may be in civil engineering sphere as well as software engineering sphere. These two fields cannot be matched exactly but examples can be drawn from civil engineering to software engineering to avoid the similar failures in project management in software production.

Before the advent of new project Management techniques, outcome of several projects had failed. For instance we will consider the construction of world famous monuments in Australia like the Sydney Harbour and Opera House.

The Sydney Harbour Bridge was completed according to the specification and on schedule in 1932. While the requirements for the bridge had been very clear from the outset implementing the new design proved to be more difficult than expected. As a result the bridge casted 9 million Australia dollars (A$) to build - more than double the original estimated of A$4 million. Since its completion, the bridge has been adjusted to carry eight rather than six lanes of car traffic, and A$3 million annually to maintain.

Subsequently the design and construction of another Sydney landmark began. Actual construction of the Sydney Opera House began in 1959 and was estimate to take 5 years to complete and at an estimated cost of A$7 million. The opera house was actually completed and officially opened in 1973 at a total cost of A$102 million! After completion, one of the dressing room/areas for the main concert hall was converted into a Playhouse, an underground car park complex was constructed at a cost of A$40 million, and a 50-year maintenance programme costing A$20 million is under way.

Given the above statistics, in a software engineering context they would undoubtedly be used to confirm the gravity of the 'software crisis'. Why then are they not used to support the existence of a 'civil engineering crisis'? To answer this question, it is also worth exploring the nature of software that makes it so susceptible to association with feeling of crisis. We will consider the software engineering environment.

Nature of software

Software is less perceptible than the products of other engineering disciplines and its physical manifestation - units of description (specifications, programs and documentation) - appears easier and cheaper to change than the hardware on which it runs. Therefore, there is a great temptation to make casual changes to software, and many inexperienced software developers succumb to this temptation. In the apparently analogous civil engineering context while change in requirements and products are often also made the consequences of such changes are almost always well understood.

Practice at NBRO

The system is initiated with formulating objectives of the system proposed by the NBRO. The client is given the option to modify the objectives to the requirements of their establishment. The objectives are compared with existing systems to design a prototype.

The prototype system is installed at the client location and the client is requested to study the system and propose any changes to the formats and reports present in the prototype.

This process takes about six months as client is to propose feasible alterations to the system. The final product is derived out of the prototype.

This practice is newer methodology when considering existing liner methods earlier in existence.

It can be even considered as flow method where the System Analyst/ is to be active through out System Development process. Programmer constantly attend to the changing requirements of the clients. Implementation Engineer functions start only at the derivation of the final product.

OBJECTIVE —
PROTYPING —
REFINEMENT —
FINAL PRODUCT

Clients role

The most important factor is the input from the client. The client is given the option to suggest the necessary changes at the appropriate time prescribed by the developer. The client should be capable to identify the feasible alteration to the system. It is expected that the co-ordinating officer in the client department to have a knowledge of computers. This helps the developer to co-ordinate the process in an efficient manner to achieve the systems objectives.

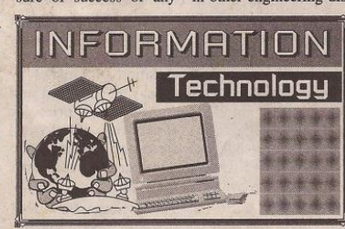
The well qualities of a well designed software can be evaluated using several factors. Basically the inputs, processes and the outputs should match the user environment. System developments also depend on the developer's environment.

Perceived crisis

Trueman Raynolds has ntroduced a triangle shape scenario as follows.

A system that is fast and good can not be produced to a lower cost. A system which is cheap and good can not be fast also. Similarly when two points are considered, the other point need to be ignored.

In the Sri Lankan context this cannot be proved as most expensive software is available to a very low price due to software piracy. This in a way helps the software Industry but harms the valuable effort spent on development. The measure of success of any



product is quality, which ultimately is a measure of user satisfaction. Users are satisfied if their requirements have been met.

In reality, however, it is often the perceived quality that determines the success of a product. In other words, it is user satisfaction of perceived user requirements that determines a user's view of a product's quality. On reflection, this is intuitive and consistent with what most user and developers report as causes of the software crisis: changing requirements, continuous adaptation of software to meet these changing requirements, and the maintenance of user satisfaction in the process.

The software industry has found it particularly difficult to maintain user satisfaction in an evolving environment.

This is in contrast to civil engineering, where user expectations, and hence user satisfaction are more stable.

It is difficult to identify solutions to the problems highlighted by the numerics, because of the different ways in which such statistics can be interpreted.

For example, even though it is acknowledged that there is a higher cost of fixing errors discovered later in a software product's development life cycle, many developers are still spending more time, effort and money on software maintenance. Focusing on different kinds of maintenance, for example adaptive, preventative, corrective, perfective is indeed both valuable and necessary, but there is an even greater need to focus on requirements engineering activities.

Requirements engineering lies at the junction between the customer with a need and the developer with a solution, and is therefore critical to the development of quality products that meet user needs. However, requirement engineering is also equally critical for future product maintenance; that is, the requirements for the development and maintenance of an existing software product are just as important to elicit, specify and validate as the requirements for a new product. Conversely, maintaining and evolving large requirements documents is crucial to supporting requirements tractability and change management of evolving software products.

Inter-disciplinary relevance

While it would be unfair to assert that software engineering has ignored advances made in other engineering disciplines, it is fair to ask why there has been little progress made in the way of systematic inter-disciplinary technology transfer.

This is not to say that traditional engineering concepts have not influenced software engineering - software reuse, architecture and process modelling are some examples where they have; rather, it is the subsequent development of these concepts into software development practices and techniques that is in question.

The reverse of this situation is not true. For example, both structured and formal methods developed largely by the computing community have been successfully adapted and applied in electronic engineering to specify electronic instruments.

It can be accepted that the figures presented at the beginning of this article was that 'software engineers are not doing too badly compared with civil engineers' or 'civil engineers are just as bad at building their products as software engineers are at building software'. While this may indeed be true, it certainly does not appear to be the perception of the general public or the software community at large. Spectacular software systems failures such as the London Ambulance Service dispatching system of the Denver Airport baggage control system help little to improve this perception.

It can be accepted that such failures were due to project management errors where there was a failure to recognise the role and impact of the software systems, and particularly inadequate understanding of the critical paths of the respective projects. However, given that such project will continue to be commissioned, perhaps the software industry should make an effort to elevate the role and involvement of the software (requirements) engineer to include or at least advise on project management, where critical choices that impact the outcome of the project are made, but the correct dialogue between the consultant and the client cannot be left out.

(The writer is the Head of Computer Division, National Building Research Organisation attached to Ministry of Housing and Urban Development, Hony. Secretary Association of Professionals NBRO, the views expressed here are his).

Mobite
Safe Drivin

A Free Hands-Free Vehicl

Take advantage of our special X'mas offer
and such a pleasure !

FREE ! AT ALL MOBITEL BRANCHES, SELECTED SINGER OUTLETS ISLANDW AND SANDESHAYA.

A HANDS-FREE VEHICLE KIT WORTH R EVERY PURCHASE OF AN ERICSSON M

Choose from a stunning range of world fam phones which come with a MOBITEL connec And get yourself a free gift of a HANDS- (Includes cigarette lighter charger, microph speaker)