Architectural Concerns When Selecting an In-House Integration Strategy – Experiences from Industry

Rikard Land[*], Laurens Blankers[#], Stig Larsson[*], Ivica Crnkovic[*]

[*]*Mälardalen University, Department of Computer Science and Electronics*
PO Box 883, SE-721 23 Västerås, Sweden
{rikard.land, stig.larsson, ivica.crnkovic}@mdh.se
[#]*Eindhoven University of Technology, Department of Mathematics and Computing Science*
PO Box 513, 5600 MB Eindhoven, Netherlands
l.blankers@student.tue.nl

1. Introduction

Consider the scenario where two or more software systems have been developed in-house, for different purposes. Over time, the systems have been evolved to contain more functionality, until a point where there is some overlap in functionality and purpose. The same situation occurs, only more drastically, as a result of company acquisitions and mergers. A new system combining the functionality of the existing systems would improve the situation both from an economical and maintenance point of view, and from the point of view of users, marketing and customers.

To investigate this problem of *in-house integration*, we carried out a multiple case study, consisting of nine integration projects in six organizations from different domains (here labelled *A-F*). For details on methodology, a presentation of the data sources, and the complete copied out interviews, see [3]. Elsewhere we have analyzed the case study data from a process point of view [4], and discussed the possibilities for reuse in this context [2]; the present paper investigates issues of importance to an industrial architect [5] and focuses on how to select a high-level integration strategy.

2. Selecting a Strategy

The following in-house integration strategies have been identified:

No Integration (NI) Do nothing – this requires no extra effort or resources in the short term, but can consequently not give any return on investment.

Start from Scratch (SFS) Discontinue all existing systems and initiate the implementation of a new system. The new system will likely inherit requirements and architecture from the existing systems [2].

Choose One (CO) Evaluate the existing systems, choose the one that is most satisfactory, and discontinue all others. The chosen system will likely need to be evolved before it can fully replace the other systems.

The fourth option is to reuse parts from more than one existing system and re-assemble them into a new system, and we present two such types of merge, distinguished by the time required to implement them:

Instant Merge (IM) The existing components are rearranged with only minor modifications or development of adapters.

Evolutionary Merge (EM) Continue development of all existing systems towards a state in which architecture and most components are identical or compatible.

Selecting a strategy is naturally influenced by many factors. A reasonable starting point would be to early focus on questions and issues that could rule out some strategies. The two concerns we have found are:

Architectural Compatibility The notion of *architectural mismatch* is not new [1]. If the architectures of the existing systems are not very similar, *Instant Merge* can be excluded, and if the systems are very dissimilar also *Evolutionary Merge*. In reality, compatibility is not so easily categorized and must be assessed in each new situation. Compatibility issues found in the cases were: similarity of component roles and high-level structures [2]; data models (cases *F1*, *F2*, *F3*); similarity of underlying frameworks, i.e. how components are defined, for example 'processes communicating via files' (*F1*), certain hardware topologies (*C*), and identical development and deployment environments (*F3*). Similar ancestry and standards may imply a certain amount of compatibility (*C*, *D*, *F2*) [2].

Retireability Retiring a system may be considered more or less feasible, based on influences such as: investments made (*B*, *F1*, *F2*) and the satisfaction of various stakeholders: architects (*A*, *C*, *E1*, *F2*), users

and customers (*B*, *C*, *D*, *F2*, *F3*), management (*A*, *B*, *E2*, *F1*, *F2*, *F3*). Affection to 'ones own' system also influences the will to retire it. If only some or none of the systems are considered possible to retire, the strategies *Start from Scratch* and *Choose One* can be excluded. Retireability is typically re-evaluated given the resulting set of possible strategies, until an acceptable balance is found between the estimated cost of integration and the problems caused by retirement (*A*, *B*, *C*, *E2*).

Table 1 summarizes the exclusion of strategies based on these concerns (black denotes exclusion). Table 2 shows the results for *Architectural Compatibility* and *Retireability* in the cases, and Table 3 the resulting exclusion of strategies (black background) and the chosen strategy (circles).

3. Discussion and Future Work

The proposed scheme, with five strategies and two concerns to exclude strategies, may seem trivial and self-evident. It seems that some of the cases however were not aware of this, and unnecessarily spent time and energy without significant progress (cases *C* and *F2* in the tables). These concepts should therefore be useful for the software architect to focus analysis and discussions.

There are several research directions for the future. First, the strategies are not so distinct in reality and could be further refined; for example, although the overall strategy may be *Start From Scratch* or *Choose One*, some parts may still be reused from several existing systems. Second, we would like to further investigate the notion of (in)compatibility: further studying problems reported from practice would form a basis for how compatibility can be assessed, and also suggest specific patterns or mechanisms to overcome incompatibilities. Third, we would like to see guidelines on when and how to involve different stakeholders in order to negotiate retireability and finally select a strategy.

We would like to thank all interviewees and their organizations for sharing their experiences and allowing us to publish them.

4. References

[1] Garlan D., Allen R., and Ockerbloom J., "Architectural Mismatch: Why Reuse is so Hard", In *IEEE Software*, volume 12, issue 6, pp. 17-26, 1995.

[2] Land R., Crnkovic I., Larsson S., and Blankers L., "Architectural Reuse in Software Systems In-house Integration and Merge - Experiences from Industry", In *Proceedings of First International Conference on the Quality of Software Architectures (QoSA)*, Springer, 2005.

[3] Land R., Larsson S., and Crnkovic I., *Interviews on Software Integration*, report MRTC report ISSN 1404-3041 ISRN MDH-MRTC-177/2005-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, 2005.

[4] Land R., Larsson S., and Crnkovic I., "Processes Patterns for Software Systems In-house Integration and Merge - Experiences from Industry", In *Proceedings of 31st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2005.

[5] Sewell M. T. and Sewell L. M., *The Software Architect's Profession - An Introduction*, Software Architecture Series, ISBN 0-13-060796-7, Prentice Hall PTR, 2002.

Table 1: The exclusion of possible strategies

		SFS	CO	EM	IM
Architectural Compatibility	Very high				
	Modest				■
	No			■	■
Retireability	All				
	Not all	■			
	None	■	■		

Table 2: Concerns per case

	Retireability	Compatibility
A	All	No
B	Not all (One)	No
C (initially)	None	Somewhat
C (final)	Not all (Either)	
D_{HMI}	Not all (One)	No
***D_{Server}**	(?)	Somewhat (?)
E1	All	Somewhat
E2	Not all (One)	Somewhat
F1 (initially)	No	No
F1 (final)		
$F2_{Pre}$	All	No/Somewhat (?)
***$F2_{2D}$**	All	Somewhat
$F2_{Post}$	All	No
***$F2_{3D}$**	None	Somewhat
F3	All	No/Somewhat (?)

Table 3: Possible and desired strategies, per case

	SFS	CO	EM	IM
A	o		■	■
B	■	o	■	■
C'	■	■		o
C''	■	o		■
D_{HMI}	■	o	■	■
***D_{Server}**	(?)	(?)	o	(?)
E1	o			■
E2	■	o		■
F1'	■	■	o	■
F1''	■	o	■	■
$F2_{Pre}$	o		(?)	■
***$F2_{2D}$**		o		■
$F2_{Post}$	o		■	■
***$F2_{3D}$**	■	■	o	■
F3	o		(?)	■

Notes: For cases C and F1, there are two rows each to describe how decisions changed over time. Cases D and F2 has been divided according to subsystem boundaries. An asterisk indicates that the integration is still at the planning stage.