BARBIERI EXTREME PROGRAMMING APPROACH FOR SYSTEMS INTEGRATION (BXPSI)

GIONATA BARBIERI[*] (gionatabarbieri@gmail.com)

Abstract

This article aims to introduce, to propose and to describe a new modeling approach for project managers and software architects about systems integration's solutions. The goal is pursued through an algorithm whom matches up peculiar Agile software life cycle concept (Extreme Programming) with a Waterfall methodology, serviceable especially in case of XSD modifications and bugs. Moreover this article presents new key performance indicators and cost parameters to manage each work activity and to regulate the billing phase of Change Requests.

Keywords

Extreme Programming; XP; Software Development Life Cycle; Agile; Waterfall; Systems Integration; Enterprise Application Integration; EAI; Service Oriented Architecture; SOA; Middleware; XSD; XML; Enterprise Service Bus; ESB; JMS; Project Target; Macro Task; Micro Task; Change Request; BXPSI.

An "*Extreme Programming*" (XP) approach typically is characteristic of *Agile software development* methodology but it could be applied in a more general context like *Waterfall software development* scenario. Many times it happens but not in a concise and well managed way for middleware technologies and systems integration development.

The complexity of software (SW) development about systems integration (SI) and enterprise application one (EAI) needs to be leaded by time-box releases constituted of short development cycles in all the integrity of SW production. That necessity is represented exactly with the use of an XP approach.

The base unit of middleware (MW) and EAI-SI solutions is an *XML Schema Definition* (XSD) to define the interfaces' pattern in order to validate an Extensible Markup Language (XML) data message. So, the goal of this article is to refine a new XP approach to contrast any sensible problem linked with the changes of XSD definitions that are downstream of a functional analysis, act that means an intensive new working opera for developers conditioned by strong impacts in the program code.

It is desirable, as a fundamental condition of a correct functional design, that cases as above do not happen but, nevertheless deprecable, there are situations in which it is possible to realize it (i.e. new urgent customer requirements).

We can easily structure a complex project in the following parts: a *Project Target* (PT) achievable with several *Macro Tasks* (MT) each of which shaped by *Micro Tasks* (µT). These last ones represent the XP kernels of our interest that are tested through *Micro Tasks (integration) Tests* (µTT). After all the test phase for each µT, it is obvious that the *Macro Tasks (integration) Test*

(MTT) is implicitly executed. Each µT unit test is considered in this article as part of the µT entity (and of its development), so it is important to not create confusion about the adopted terminology.

A PT defines the problem domain and the expected implementations directly depending on the shared requirements analysis. With a MT, in this context, we usually realize a systems integration unit and its µT represent an operation, a function, an action provided by the systems integration unit.

In Fig. 1 we have a picture example of an inter-communication among several systems, using an *Enterprise Service Bus* (ESB) related to a MW. The systems A and B communicate through XML-based messages defined by apposite XSDs. So, in example continuing, we can name *A2B 1* and *A2B 2* etc. these ones, in the line-way from system A to B. Any scenario constituted of information XML-based messages, as described above, could have whichever complexity form, independently from both ESB medias and the transportation layer, but the focus of discussion is identical (many times in a typical complex EAI solution case we have deployed engines that use JMS – Java Message Service – queues to conduct all our messages).

However, instead of aiming to develop all the expected µTs of a MT, it is better to organize the works step by step, developing program code for each µT, before completing the MT *in toto*. It means that, comparing with our example, first there will be the SW coding of the µT phase related to, or, handled by *A2B 1* SI XML-based messages, second, just with the termination of the previous phase, will be possible to execute the SW µT development of the code associated to *A2B 2* SI XML-based messages and finally, in an iterative mode, until will be coded the last µT, in according to a sort of an agile XP method that slices the MT in a cadenced µT SW development.

After the first step (µT development associated to *A2B 1* SI XML-based messages) and before than the second one (µT development associated to *A2B 2* SI XML-based messages), this XP approach reckons on the µTT for the first phase, that allows ourselves to consider the next step only if result is regular. So we have a sequence of these time-slots, or an algorithm:

- I: µT-1 development
- II: µTT-1
  - II.a: µTT-1 KO return to I step to fix the bugs
  - II.b: µTT-1 OK jump to III step
- III: µT-2 development
- IV: µTT-2
  - IV.a: µTT-2 KO return to III step to fix the bugs
  - IV.b: µTT-2 OK jump to V step

…

- n-th: µT-n development
- n-th+I: µTT-n
  - n-th+I.a: µTT-n KO return to n-th step to fix the bugs
  - n-th+I.b: µTT-n OK >>>>>> MT is completed and tested!

To complete BXPSI algorithm we can use a bug-tracker as help to schedule the roles in the MT.

We have considered now message flows A2B, but the same way is for B2A or X2B etc..

If a post-development or a post-testing XSD modification about a µT is required, each consequent impact on related XML-based messages is minimized if we adopt BXPSI algorithm, because we can insert this new task in an exactly step of the algorithm, having tested every µT of a MT. At this point, there are two possibilities: first (*optimal scenario*) - each µT is not bonded to others; in this case there is only a new development, corresponding to new task qualified by the XSD modification, afterwards to test, so there is a substitution action of the interested time-slot

within algorithm chain; second (*sub-optimal scenario*), in which the impacted µT is wired to others, so the fixing-oriented development intervention and subsequent tests involve all the µTs, starting from the i-th µT to the n-th µT. Both scenarios permit to save and to optimize time in terms of SW code development and of SW test phases because the action is related just to one time-slot (a total growth of MT development+MTT all-out time * 1/n) with the optimal one, or because the action is related for few (n-i) time-slots with the sub-optimal one (a total growth of MT development+MTT all-out time * $(n\text{-}i)_{th}/n$). Following an analytical representation:

$$\text{if} \quad (MT\ development\ Time\ +\ MTT\ Time) = MT\ Time$$

- usually (worst) scenario:
  $(MT\ development\ Time\ +\ MTT\ Time) = MT\ postmodification\ Time$
- sub-optimal scenario:
  $(MT\ development\ Time + MTT\ Time)\frac{(n-i)}{n} = MT\ postmodification\ Time$
- optimal scenario:
  $(MT\ development\ Time + MTT\ Time)\frac{1}{n} = MT\ postmodification\ Time$

However procedures linked to BXPSI algorithm are extensible to any form of bug to fix in the SW code.

Finally is useful for any XSD modification/bug, caused by functional analysis-design or exchange interfaces documents lacks, to define cost criteria to use in a BXPSI approach.

We can consider two cases: 1.) gratis action; 2.) billed action.

The *gratis action* is represented by the scenarios in which the XSD modification/bug is signaled during the technical µT analysis-design or mostly if the impacts of the XSD modification/bug are fixable in a pre-determined or agreed technical µT design time fraction. It is advisable to establish a bound of 0.5 percentage-points like technical µT design time fraction.

The *billed action*, comparable with a *Change Request* (CR), is represented by the scenarios in which the impacts of the XSD modification/bug exceed the agreed yet percentage for the gratis action state. Following analytical formulation and an example, having defined *%TA* (the agreed technical µT analysis-design bound in percentage terms, or equivalently the gratis action supported percentage), $\overline{\%TA}$ (the negation of the gratis action supported percentage, or equivalently the billed action supported percentage), *cTA* (billed costs overall associated to technical µT analysis-design and to µT SW development and related unit tests), *cCR* (new expected billed costs for a CR, related to the billed action). Useful it is Fig. 2.

$$\overline{\%TA} = 1 - (\%TA)$$

$$cCR = cTA * \overline{\%TA} \qquad [currency]\ .$$

With the values of *%TA=0.3* and *cTA=10k$* the results are:

$$\overline{\%TA} = 0.7$$

$$cCR = 7k\$\ .$$

In conclusion BXPSI algorithm is convenient in a SI context for four reasons: I) to contrast functional analysis-design or exchange interfaces documents lacks; II) to regulate CRs with analytical and certain models, especially in a typical work scenario in which there are contracts between a client SW governance team and a provider SW team, agreed through a negotiation phase for apposite indicators; III) to insulate criticalities and problems with impacts within our SW code, using light and standardized procedures organized in regular algorithm steps; IV) to parallelize SW developments about similar but loose μTs (so in a condition of optimal scenario) with a best effort.

Below it is attached a URL link to set up a BXPSI free on-line calculator tool for Project Managers and Software Architects involved in SI activities:

http://gbtlc.altervista.org/bxpsi.html

Rome (Italy) – December 7th, 2012 – © by Gionata Barbieri

(*): *Gionata Barbieri* is Telecommunication Engineer – ICT Science (Master's and Bachelor's degrees at "Federico II" University of Naples – Italy). He has got a post-degree Master (Middleware Academy – UIIP at Biogem Campus in Ariano Irpino – Avellino – Italy) and the Senior Certification enabling to exercise the profession of ICT Engineer (provided by "Federico II" University of Naples and Ordine degli Ingegneri della Provincia di Napoli). Currently he works as TIBCO Specialist Consultant and R&D Software Engineer for Meware SRL in Rome (Italy).

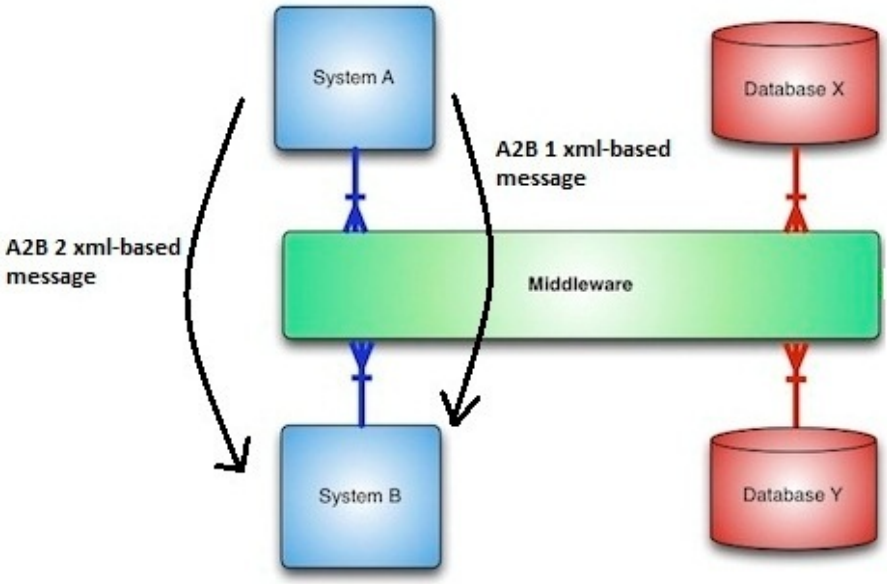
System A
Database X
A2B 1 xml-based message
A2B 2 xml-based message
Middleware
System B
Database Y

Fig. 1

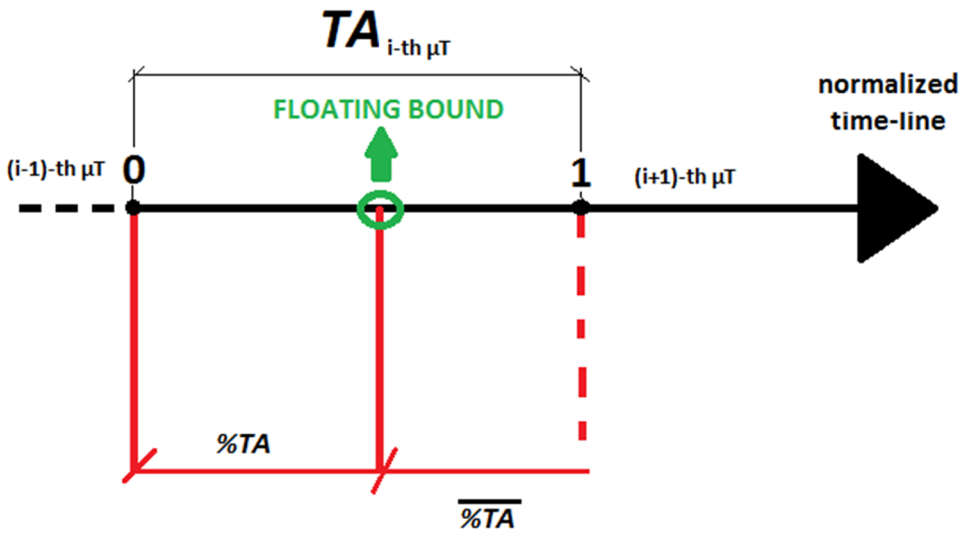
TA i-th µT
FLOATING BOUND
normalized time-line
(i-1)-th µT
0
1
(i+1)-th µT
%TA
%TA

Fig. 2