SS2009 WG7 User まとめ

2009/6/17～19

◆テーマ

1. 企業の中で IT 部門に求められること
2. インソース・アウトソース
3. ユーザとベンダーのマインドのズレ
4. アカデミアとの連携
5. ユニケージ手法研究
6. 要求獲得における PRINCE モデル
7. IPA/SEC の取り組み
8. その他
9. 今後の課題

◆テーマ毎の議論内容

1.企業の中で IT 部門に求められること

各企業における環境、様々なステークホルダーの視座などにより異なるだろう。

企業の IT 部門と経営者の間で、マインドのズレがあるかも知れない。また現場の利用者との間にもギャップがあるだろう。これらのギャップ・ズレを調整するファシリテータとしても、IT 部門は重要な位置にある。

様々なギャップもそうだが、予算中心で動く企業の文化にも問題があるだろう。

また、企業の中での IT 部門の位置づけについても変化が必要である。上場企業の中の非 IT・コアコンピタンス企業において、IT 部門の人間が社長となっている事例が少ない、ということも問題である。

2.インソース・アウトソース

PP を見ても、インソースすべきは、コアシステムであるか否かという点は共通であった。

アウトソーサーの選択に関しては、信頼できるかどうかが重要である。できる・できないでなく、どんな提案ができるか？というような視点でもアウトソーサーを判断すべきである。

という中で、アウトソースすることに関して、できるだけコストダウンすることが重要という考えもある。これは、前述のコアシステムか否か（コンテキスト）に依存するだろう。アウトソースして問題ないと考えられるコンテキスト分野に関しては、徹底的にコストを抑えるという方針もある。

しかし、どこをインソースで行い、どこをアウトソースするかという戦略・ポリシーが議論されないといけない。人的リソースが不足している企業が多いという背景の中、戦略を持たずコア業務をアウトソーシングしている企業が多いことは問題である。

3.ユーザとベンダーのマインドのズレ

ユーザ企業と SI ベンダー間のマインドのズレは、立場が違うので当然かも知れないが、長期的見て Win-Win の関係を模索しなくてはならない。

ギャップを埋めるためには、その間に存在するファシリテータが重要である。

ユーザ企業と SI ベンダーのギャップは、なにか障害が発生したときに顕著になるのかも知れない。だから、SLA（Service Level Agreement）が重要になる。事例として、アメリカの大停電が挙げられる。この事例では、そもそも電力に対して住民が、その費用の支払も含めてコミットしていた。だから大きなクレームは無かった。対して銀行のサービスなどは、利用者との間に SLA がないので、1 分間サービスが止まっても苦情がくる。

また、日本ではあまり例はないが、概算の人月見積りではなく、成功報酬型の契約を行う方式も、マイ

ンドを合わせるのに有効かも知れない。

立場の違い、マインドのズレというものは様々である。利用する側・提供する側は当然であるが、プロダクトを提供・開発するベンダーについても、ソフトウェア開発の請負なのか、パッケージソフトウェアの開発なのかで異なる。パッケージソフトウェアの開発については、利用者が多いので要求も多岐に渡る。パッケージソフトウェアの機能追加に関するポリシーも重要で、無制限にその要求を取り込んでいくと、ソフトウェアは肥大化・硬直化し、リファクタリングできない存在になってしまう。

また、ユーザとベンダー、それぞれが抱える問題点・弱点を認識する必要がある。ユーザ企業においては、技術力の低下・思考停止となりベンダーへの丸投げなど、体質改善をしなければならない。

ベンダーにおいては、多重下請け構造・モデリング技術等のソフトウェア工学音痴・現場知らず、などがあるだろう。

4.アカデミアとの連携

残念ながら日本では連携が弱いといえる。

アカデミアとユーザは、この業界の中では両極に位置する。そして、だからこそ相互補完することには意味がある。ユーザ企業の現場で必要とされる特殊な要件であっても、それを実現する方式を、理論ベースから立ち上げることがアカデミアにできるからだ。

大学のみならず、高校・中学・小学などでの取り組みはどうか？

日本ではほとんど例はないだろうが、韓国では、小学校でソフトウェア工学を教える試みが検討されている。これは 2～3 年前の韓国の教育会議で議論されていた内容で、実際に小学生にこのような教育が実施されているか不明であるが、ソフトウェア工学に対する日韓の取り組みの違いを感じる事例であった。

また、日本でも、企業に所属していて、大学でソフトウェアを学ぶ人もある。

5.ユニケージ手法研究

技術的な視点では以下のような特徴がある。

シェルだけで Application を作成する。
必要なコマンドは自作し、パイプでストリーミングすることにより高いパフォーマンスを出す。
RDB を使わない。
データが最も重要で、マスター更新の情報も含めて、生データを全て保持する。
（1 エンター、1 ファイル）

開発プロセスに関しては、以下のような特徴がある。

現在の Agile プロセスに非常に近い方式である。
ユーザと同じ目線に立ち一体化して、開発を行う。

上記のような特徴の中で、今回の SS2009-user では、特にその開発プロセスが重要であると考えられる。SI ベンダーでも、企業の中の IT 部門も、実際の利用者を外から眺める第三者の視座で情報システムを見る傾向がある。しかし、USP は同じ目線であり、職場に於いて一体化し、一緒に悩み・考えるスタイルだ。前述の様々なギャップを埋めるためにも、ユーザ企業の中の IT 部門は、第三者的な視座でなく、同じ目線で利用者と会話する必要がある。

ユニケージ手法のアーキテクチャは、事務処理系プログラム（Cobol など）の基本的なソフトウェアアーキテクチャに近い。いままで、Cobol のようなソフトウェアが適用可能なドメインに Agile 開発はそぐわないと思われていた感があるが、そうではない。適材適所で開発アーキテクチャ・プロセスモデルを適用するという、良い事例であるだろう。

6.要求獲得における PRINCE モデル

PRINCE モデルとは、戦略的な要求獲得計画を立案することを目的に、要求の獲得時期と、要求獲得率の成熟度をタイプに分けてモデル化したものである。タイプには、早期成熟型（E-Type）・中期成熟型（M-Type）・後期成熟型（L-Type）・予測できない突発型（U-Type）がある。

このモデルは、産学が連携して開発されたモデルである。参加した企業・大学は、日立・富士通・東芝・NTTData＋九工大・阪大・専修大・筑波大、である

このモデルを適用するにあたっては、開発案件も様々なタイプ・性質が影響するだろう。それによってPRINCE モデルの、どの Type に合致するかが、わかるかも知れない。Type がわかれば、開発プロセスの選定（WaterFall・Agile・Incremental・UP など）や、コスト見積りなどに有効である。

開発案件のタイプとしては、以下のようなものが考えられる。

新規システム or 再構築システム
戦略要求 or 支援要求
基幹系 or 情報系 or コミュニケーション系
コア or コンテキスト
機能要求・非機能要求　　などなど

また、利用者やプロマネ、開発・設計者の成熟度にも拠るだろう。また、パッケージソフトウェアの開発や、それに関するアドオン・カスタマイズ開発などの案件においても依存する。

また、PRINCE モデルの分析において、要求の粒度を調整することは重要な課題であるが、ユースケースおよびシナリオ（代替シナリオ・例外シナリオ含む）を受け皿に分類することで調整することができる。

7.IPA/SEC の取り組み

重要インフラ情報システムに関する、信頼性研究会の報告書が作成された。
http://sec.ipa.go.jp/reports/20090409.html
まず、信頼性の要求水準を見極める。これには、レベルⅠ～Ⅳまでの区分がある。

レベルⅠは、社会的影響が殆どないもの、
レベルⅡは、社会的影響が限定されるもの、
レベルⅢは、社会的影響が極めて大きいもの、
レベルⅣは、人命に影響、甚大な経済損失があるもの、である。

そしてこの報告書の指針及びチェックリストで診断することによって、強化すべき対策があるか、過大投資の可能性があるか、ということを見極めることができる。

8.その他

直接的に今回の SS2009-user の直接的なテーマではなかったが、議論に上がった内容を記述する。

教育について。

USP では、教育教材を策定中である。そして教育とは一種のスクリーニングだと考えている。これは、スポーツや芸能の文化と同じく、このスクリーニングを通過した人間だけが、エンジニアと認められる、という考え方である。

また、あるメーカー系企業においては、新卒の人員をまず IT 部門に配属させる、という事例もある。いかなる職種に配属されようが、IT リテラシは必須であるという考え方である。

BeaconIT 殿の ETL ツール（Waha! Transformer）

ユニケージ手法に通じるものがある。Waha! Transformer には、View Filter という単機能なアイコンが用意されている。これはユニケージ手法のコマンドと基本的な考え方が似ている。

双方とも、共通的に使用する部品を、単機能で無味乾燥的なものに抑えた。OOA におけるクラス・オブジェクトのメソッドは、有意味で自然言語的であることに対して対照的である。そして、その単機能で無味乾燥的な部品で、ファイルを様々に処理していき、結果を得る。処理に関しては、Waha! Transformer においてもパイプでストリーミングすることが可能である。

9.今後の課題

ユーザ企業の IT 戦略・ベンダーとの関係などに関して、今後の課題について記述する。

・ ユーザ企業の体質改善

おおよそ 10 年になるだろう。ユーザ企業は自分たち自社のソフトウェアを作ること・技術を習得することを止め、ベンダーに依存する体質となってからだ。人員は高齢化し、モチベーションも低下している。

多くの企業の IT 部門は、自社の業務に明るい。この業務知識を、自分たちでそのままモデル化したり、コード化することができれば、ベンダーに業務説明をする、という非効率な作業を削減できる。

そのために、どのようなアプローチで何から始めればいいのだろうか？

・IT 音痴な経営者、ビジネス音痴な CIO

上記は、一般企業においてよく当てはまる構図だろう。IT 部門と経営者のギャップに関する議論も多くでたのだが、さて、何をどうファシリテートすれば、この Gap が埋まるのだろうか？

・ソフトウェア工学音痴な SI ベンダー

モデルで議論できないエンジニア、ゼネコン化している大手企業などなど、SI ベンダーも体質を変えなければならないだろう。

・管理主義・官僚主義な日本の企業体質

特に昨今の法規制が、管理・官僚主義を加速している。ソフトウェアというものは、もっと柔軟で温暖的な存在であるはずだ。

しかし、ユーザ企業のエンジニアの多くの仕事は、組織間の利害調整などの、非技術的な作業ばかりだ。このような状況では、技術的なイノベーションは起きないだろう。