2013年度　第 7 回特別講義　レポート

日時	2013 年 12 月 20 日（金）　10:00～12:00
会場	日本科学技術連盟・東高円寺ビル　2階講堂
テーマ	「システムズエンジニアリングにおける運用と品質」
講師名・所属	山本　修一郎氏（名古屋大学教授）
司会	鷲崎　弘宜氏（早稲田大学准教授）
アジェンダ	1.システムズエンジニアリングとは 1)90 年代の社会システム 2)Soft Systems Methodology 3)21 世紀 systems Methodology 4)Enterprise Systems Engineering 2.関連知識 1)運用概念の事例 INCOSE ConOps 2)要求工学標準 IEEE29148 におけるシステム要求 3)システム運用知識の抽出方法（実例からの考察） 4)アーキテクチャフレームワーク事例 TOGAF 3.保証ケースによる品質の確認手段 1)非機能要求グレードと保証ケース 2)TOGAF の dependability 3)ITIL と保証ケース 4)テストと保証ケース 4.まとめと今後の課題
アブストラクト	システムズエンジニアリングで重要となる関知識として、システ ムの運用概念、要求工学、アーキテクチャフレームワークの事例として、INCOSE ConOps、IEEE29148 要求工学標準、TOGAF を概観する。 次に、保証ケースによるシステム品質の確認手法として、筆者らの研究事例を紹介する。具体的には、非機能要求、アーキテクチャ設計、運用プロセス、テスト十分性に対する保証ケースの適用法について説明する。最後に、今後の課題について展望する。

司会者：鷲崎氏からの講演への着目点のご案内

昨今システムズエンジニアリングというキーワードを耳にする機会が多い。

伝統的なソフトウェアエンジニアリングの捉え方には最近限界があり、例えばソフトウェアシステム全体における要求であるとか、設計であるとか、或は運用であるとか、ソフトウェアの中に閉じていないシステム全体のとらえ方が今大切になってきている。
また、産業界などでもIEEE26262とか、個別のキーワードであったり規約等の取り組みが最近出てきている。本日はこれらをきちんと俯瞰して、システムズエンジニアリングという全体の位置付けと、その中での個々の技術を深堀していただくための非常に良い機会であると考える。

＜講義の要約＞

◆はじめに
本　　本日はシステムズエンジニアリングとソフトウェアエンジニアリングという観点で話を進めていく。
日本においては、本日話すようなシステムズ工学の書物は少ない状況である。システムズ工学というと制御理論とかの本が多いが、むしろプロセスの議論である。問題とソリューションを反復的に関係つけていくという考え方が、欧米のエンジニアリングの概念であるということを紹介する。またシステムズエンジニアリングに関連する知識ということで、エンタープライズアーキテクチャの話も紹介する。運用と品質という観点から最近取り組んでいるアシュアランスケース（保証ケース）、これは ISO26262 では安全性というような呼び方がされているが、基本は同じなのでこういった基礎的な話もする。

◆1. システムズエンジニアリングとは
1）90年代の社会システム
　社会システム方法論は、90年代にイギリスで考案されたものです。
IEEE1220 systems engineering process については、入力から要求分析→要求確認→機能分析→機能検証がありその全体をコントロールするシステム分析と制御である。この部分はマネージメントと言った方がよいかも知れない。そして、設計検証を経由して出力が出る。よって、ソフトウェアの開発プロセスとほとんど同じと思えるが、これがシステムズエンジニアリングのプロセスである。

2）Soft Systems Methodology(ソフトウェアシステム方法論)
IEEE software systems engineering process　は所謂 V モデルである。システム分析→システム設計→ソフトウェア要求分析→ソフトウェアデザインそして反対側に Testing が並ぶ。システム全体の分析をして、その後にソフトウェアの分析をする。これはソフトウェアだけのモデルだが、このカウンターパートとしてハードウェアの分析設計もある。

3）21世紀 systems Methodology（システムズエンジニアリングの情報）
・システムエンジニアリングの情報
　システムズエンジニアリングの基礎的な概念が良く整理されているサイトがある。
「 http://se.rdy.jp/incose_thema.html」慶応大学の白坂先生が整理されたものである。本日この後

述べる CONOPS の内容も説明されている。

・社会システム
　社会システムについて分析すると、自然科学と人文科学に分けられる。日本では人材を理系と文系に分けるが、システム方法論の中では自然科学と人文科学という分け方がされている。自然科学は因果関係を分析するが、人文科学は目的関係に着目して統合（インテグレーション）を分析する。そしてソフトウェア開発は人文科学に近いのである。つまり、ソフトウェア開発者は理系の方が多いが、この考え方だと、実際にやっている仕事は文系の仕事だと考えられる。
　以上からわかることは、社会システムは自然科学と人文科学を統合する必要があるということである。

・システムの階層
　システムには階層があるが、最近良く目にする System of Systems(以降 SOS と表す)がある。SOS は、自律したシステムが互いに相互連携して構成される複合システムである。
　エンタープライズは SOS を更に包含する様なより大規模で複雑なシステムである。人、プロセス、技術、システム、更にそれ以外の資源を複数の組織、地域を跨って相互結合して、ある共通のミッションを実現するような構造体である。

最近は従来のシステムズエンジニアリングよりも更に複雑な問題を解決するための新たな技術が、エンタープライズシステムズエンジニアリングと言われている。よって、企業の為のシステム工学ではなくて、社会全体で相互結合されるようなシステムをどうやって開発していくかと言うことである。
　そう言う意味では最近のネットワーク環境はまさにエンタープライズシステムだと言える。通信事業者が提供するネットワークの上に更にそれを使うアプリケーションサービスロバイダーが居るので、そのプロバイダーのアプリケーションをたくさんの人が使うことによって障害がおきる事例が多く出てきている。これは、どのようなアプリケーションサービスプロバイダーが居て、アプリケーションサービスをエンドユーザがどのように使うかという事を、通信サービスプロバイダーがある程度マネージメントしなければならない時代になったと言う事である。例えば、30年前頃には通信を NTT が独占していたので、NW 全部が計画通りに動いていたわけだが、そんなことは全くできない時代になったと言う事である。
　そういう意味で、SOS とかエンタープライズとは何かについて我々がイマジネーションを働かせておかないと、システムの品質が保証できない事があるという事である。そのための新しい技術がシステムズエンジニアリングである。これは従来のトラディショナルなシステムズエンジニアリングではなくて、新しい時代のシステムズエンジニアリング、例えばエンタープライズエンジニアリングが必要になったと言う事である。

・5階層のシステム工学
　この新しいシステムズエンジニアリングの一つの手法として、Hitchins 氏らが提唱しているのが「5階層システム工学」である。プロジェクトが作り出すもの、そして事業・エンタープライズが生み

出すもの、産業界全体が生み出す物、さらに社会経済的なシステムといった枠組み（階層）が提唱されている。

・外部ループ・内部ループ

Hitchins 氏の著書に「外部ループ・内部ループ」の図が提唱されている。問題空間に対してソリューションを設計するのが外部のループ、そのソリューションの中の特定のシステムの設計をするのが内部ループである。そういう意味で新たな SOS に対するシステムズエンジニアリングなどでは、外部ループを中心に考えるということである。

つまり、社会経済的な問題を解決するシステムをどのようなシステム群によって解決していくかということである。このシステム群の中にはコントロールできないものがある。例えばネットワークサービスプロバイダとかである。この考え方はオープンイノベーションの考え方ともつながると考えられる。つまり、自分が全てのシステムを完璧に設計できる時代ではないということである。外部パートナーが作ったコンポーネンツをうまく活用して我々の解くべき問題を解決するシステムのアーキテクチャを開発する時代になったと言える。

・発展と変革

Hitchins 氏らの方法論の中で問題の解き方が提唱されている。まず問題に対してシステム方法論を使って理想的なシステムの解を作る。つまり変革(Revolution)である。一度この解が得られれば、今あるシステムとこの理想的なシステムを使って、問題を起こしていたシステムを置き換える活動が出てくる。でもそのシステムもやがて現実のシステムになっていく。よって次に新たな問題が出てきた時には新たな理想的なシステムとの差を見て改善できる場合は発展（Evolution）させる。それもできなくなった場合にはまた変革がはじまる。こういうようにしてシステム自体も継続的に発展させていくというプロセスがある。これも一種の古いシステムと新しいシステムから構成される SOS と考えることができる。

・システム科学の類型

もうひとつの新しいシステム工学の本の中で、システム科学、複雑系のシステムの科学がある。これとエンタープライズシステムズエンジニアリングと対比させて分析した図がある。組織化された複雑性と組織化できない複雑性、組織化された単純性と組織化できない単純性を類型として示している。

組織化された単純側はニュートン力学の方程式で表すことが可能なものである。対する組織化できない複雑性は、例えば気象などのように式で表すのは難しいが統計学が使える部分である。つまり、バラバラなのでランダム性をうまく使う事で予測できるということである。しかし組織化された複雑性、まさに Enterprise of systems engineering の領域であるが、組織化されているために、そしてその組織化が均一でないために、そのゆがみで複雑になっていく。しかしながら統計的に平均的な特性の分析はできない。よってこの組織化された複雑性の部分も新しいシステム工学が必要な領域である。

・扱いにくい問題の統治

Enterprise of systems engineering とは何かと言うと、「技術」と人間を扱うような「非技術」とを統合しようとするものです。これはあたかも自然科学と人文科学の対比に良く似ていて、この両者を統合しようとするものです。古典的システム工学（TSE:Truditional system engineering）は技術の事しか考えていませんでした。
つまりソシオテクニカルなシステムを分析する技術が必要になります。

・分析と合成
　Rebovich 氏らは、分析と合成の表を提唱した。これは自然科学と人文科学の対比に良く似ている。分析は物事の内部に入り込んでいき、システムを分解して分析した後に再結合する。対して合成は全体の振る舞いや性質を分析して説明する。個別に分けることができなく、全体が絡み合った時に不具合が起きる。全体が絡み合うと言う事はどういう事かを合成的にシステム全体を考え、環境とどう相互作用をするかを分析する。これが合成的な考え方である。

4）Enterprise Systems Engineering （以降、ESE と略記）
・ESE の対象領域
　Ribovich 氏著の ESE に関する書籍が出たのは 2011 年であり、この辺りから議論がされている。例えば「POET：Political、Operational、Economical、Technical」という標語があるが、これは「政策」「運用」「経済状況」がシステムに影響を与えている事を示している。この POET についてしっかりと考えて戦略的な技術計画が必要であるということである。
　エンタープライズの統制においては、個人と組織の成功基準のバランスが重要であり、個人の成長とコミュニティをどのようにして統治し確立していくのかというテーマがある。ESE のプロセスとしては組織学習が関係する。ビジネスプロセス、エンタープライズシステムズ、内部プロセスを明確にするということである。

◆2. 関連知識（具体的な関連分野の知識）
CONOPS とは、Concept of Operation と言うことで、ITIL V3 においても記載されている。最近標準化された新しい要求工学の規格である ITIL V3 ISO/IEC29148 の中でシステム要求の開発プロセスを紹介する。
　数年前に遭遇したスパコンのトラブルの時に作った運用要求の記述手法を紹介する。
　エンタープライズアーキテクチャのフレームワークとして TOGAF（The Open Group Architecture Framework）を紹介する。

1）運用概念の事例 ConOps
・システム工学の概念設計と CONOPS(Concept of Operation)
　システム工学の概念の「概念構成要素」と「問」と「説明」について表にしたものを示す。システム工学の教科書で、A.Kossiakoff 氏らによる System Engineering Principles and Practis,2011 年版からの抜粋である。
　Why:「なぜシステムを運用せねばならないのか？」が運用要求、What/How mach:「何を運用するのか？」が機能・性能要求、How/Who:「システムの集合体がどのように運用されるか」が運用概

念、Where/When:「どこでこのシステムは運用されるのか？」が運用コンテクストである。
この中で、運用概念（How/Who）において、特定のシステムの運用概念が CONOPS である。ここで注目すべき事項は、SOS に対する運用概念ではなく、特定のシステムの運用概念であるという事である。
　CONOPS 自体は古い概念であり。1980 年代頃から提唱されている。ただし、日本では注目されてこなかった経緯がある。

・CONOPS の構成要素
　構成要素は以下である。
　目的すなわちなぜこのシステムが必要なのか、他のシステムとの関係、情報の入力源と出力先、それ以外の関係や制約の要素を持つ。
　CONOPS の簡単な説明にはデーターフローダイアグラムが適する。このフローの最上位レベル、つまり、システムと外部環境との相互作用を明示した図であり、これが CONOPS の構成要素と対応している。

・Concept of Operations(CONOPS)文書
 CONOPS は最終的には成果物である。作る時期は、要求定義の初期に作成し、システムがどこでどのように使われるのか、何をなぜするのかを書くのだが、「なぜ」は目的なのでミッション定義がここに書かれる。つまり、重要な最上位の性能要求や目的を明らかにするものである。

・プロダクトラインに対する CONOPS
　プロダクトラインについても CONOPS が記述できる。ここで、プロダクトラインは何かと言うと特定のシステムではなく製品系列だが、それに対する CONOPS の拡張も提案されている。これについてはカーネギーメロン大学の SEI のページに公開されている。

（http://resources.sei.cmu.edu/asset_files/TechnicalReport/1999_005_001_16745.pdf）
SEI においては、「どのように使われるのか」「プロダクトラインに参加するのは誰か、組織、活動はどんなものが有るか」「プロダクトラインの運用がどういう順序で定義されているかについて運用モデルのプロセスは」について記述すべきであると定義されている。

・CONOPS 文書の位置付け例
　CONOPS 文書の位置付けの例の図がカーネギーメロン大学の SEI のページで公開されている。
（http://www.sei.cmu.edu/productlines/ppl/concept_of_operations.html）
Product Line の CONOPS から、Production Plans へ、一方は Business Case、Scope、Requirement がつながっている。
　この体系で CONOPS の位置から、そもそもこの Product Line のビジネスシナリオがどうなっているか及び生産計画を説明するための文書であることがわかる。

・プロダクトラインのための CONOPS 文書の構成例
　CONOPS 文書のテンプレートを示す。
（http://www.sei.cmu.edu/productlines/ppl/concept_of_operations.html）
章立ては「概要」「アプローチ」「背景」「組織的考慮事項」「技術的考慮事項」「推奨事項」「Q&A 等の付録」となっている。

・CONOPS 文書の内容例
実際の CONOPS 文書に書くべき内容を説明したものが下記にある。Nicole Roberts らにより公開されている。
（http://www.dtic.mil/ndia/2008systems/7191roberts.pdf　P16 参照）
これは、Product Line ではなくて、システム一般に対する CONOPS 文書のテンプレートとして使えるものである。システムが置かれる外部環境を明確にする事と、システムの目的、問題定義等を明らかにするものである。

・CONOPS の定義対象
CONOPS でよく書かれている内容が下記に示されている。
（http://www.dtic.mil/ndia/2008systems/7191roberts.pdf　P11 参照）
システムの利用シナリオ、環境とシステムの相互作用、システム自体が多くを占め、システムの詳細については多くは書かれていない。つまり、システム利用とシステムを取り巻く環境及びシステム事態の高水準の機能を記述したものが CONOPS である事がわかる。

・CONOPS の活用工程
CONOPS をどういう工程で利用するかが下記に示されている。
（http://www.dtic.mil/ndia/2008systems/7191roberts.pdf　P11 参照）
要件定義、システム設計、試験計画で利用されている事がわかる。

2）要求工学標準 IEEE29148 におけるシステム要求
・ISO/IEC/IEEE 29148:2011
　IEEE29148 が捉える要求が下記に示されている。
　（ http://www.iso.org/iso/catalogue_detail.htm?csnumber=45171）
　この中にも CONOPS が入っている。最初に作るのは「システム要求」ではなくて「ステークホルダ要求」である。つまりお客様の要求を定義する。この場合、エンドユーザは不特定多数なので把握できないという意見が必ず出るが、エンドユーザの代弁者の力を借りてステークホルダの要求を定義する事は可能である。
次にシステム要求、ステークホルダの要求を解決するためのシステムの要求を明らかにする。次にこれに沿ってソフトウェアの要求を開発する。この中で、システムの運用については、お客様がシステムをどう使うかを明らかにしておく必要がある。ここでは、CONOPS は二つあり、一つは「組織の観点での運用概念」、もう一つは「システムの概念での運用概念」である。組織の観点とは、

ビジネスプロセスである。システムだけではなくそれを含んで業務全体をどのようにオペレーションするのかが組織の観点の CONOPS である。

・ステークホルダ要求（StRS）の構成例
「ビジネスマネジメント要求」「ビジネス運用要求」「ユーザ要求」「提案システムの概念」の章で CONOPS が出てくる。ソフトウェアエンジニアリングの観点から言うと、CONOPS はステークホルダ要求のドキュメントの中に含まれる事がわかる。

・システム要求の構成例
システムのコンテキスト、ユーザ要求が書かれているが、この中には ITIL V3 29148 には CONOPS は出てこない。これは、ステークホルダ要求で CONOPS が明確になっているので、その次のシステム要求の中では CONOPS は省略されている事がわかる。

3）システム運用知識の抽出方法（実例からの考察）
　数年前に発生したスパコントラブル事例からの考察をした。
運用者にヒヤリングしたところ、運用マニュアルが無かった。そこで、現場の運用者でも作れるような運用知識を再構築する事となった。

・運用の限界
　問題が発生した場合にその証拠を保存していない場合が多い。発生後しばらくしてから異常に気付く事が運用上のトラブルである。例えば、昨今のクラウドサービスのデータベース紛失の問題は、無くした時には気が付かず、お客様が使おうとした時に使えないということになる。この場合何があったかの証跡が残っていなければ分析もできない。これは運用中だが、運用停止の場合には何か想定外の事が起きるという可能性の分析もできない。これは運用手順が無いからである。運用手順があればその手順上のリスクの分析が可能である。つまり、手順が無い場合は、全て都合良くやった事になっているわけである。
次に運用組織の文化については、情報隠蔽している組織が新聞で暴露されると、そんな馬鹿なことをやっているからと批判するが、自分自身の胸に手を当てると自分もそうなっている可能性がある。説明責任も曖昧である。日本では説明責任を果たしたと言われるのはテレビの前で社長が頭を下げた時だが、それは説明にはなっていない。つまりこういった状況を明確にするには、様々な主体があるが関連する対象との間の依存関係を定義しておく必要がある。本来定義すべきことができていないからこういった問題が発生する事となるわけである。

・運用前の要求品質の確保とその記載方法
運用要求に対し誰がどういう状況の時に何を定義してしその手順を実施するのか、その時の入力情報と出力情報、そしてそれを実施した時の結果はどこに報告するのか。また、事後状況としてどういう状況が成立すれば良いかについて表で書きませんかという事を提案した。これについては現場の保守者に同意していただき進めることができた。

・運用要求定義票と運用要求の抽出法
（http://www4.atpages.jp/sigksn/conf07/SIG-KSN
（http://www4.atpages.jp/sigksn/conf07/SIG-KSN-007-02.pdf　P3 参照）
一部「事前状況」「運用規制」「事後状況」「役割分担」については ITIL のガイドブックには記載が無く、新たに追記した。
表にした事により定義の抜けが明確になる。よって、これを記載する事で可視化ができ抜けを防ぐ議論ができる事となる。

・運用活動知識の構成と運用活動定義票の付随資料例
運用活動定義表を記述すれば全てかというと、これは一部である。特定のタスクは定義表一枚一枚に記載するが、その相互のつながりがわからない。よって、定義表を書けばそれを使う手順、シナリオの必要性が見えてくる。
更に、この活動の必要性の根拠としてのドキュメントが必要となる。そこで規約を作る事となる。これら、付随資料を整理すると「体制」「管理台帳」「ルール」「システム構成」「書類」を大項目とした付随資料の Tree ができる。つまり、運用活動定義票を作ると、これに紐づいて抜けがある事に気が付くので芋づる式にドキュメントがそろっていく。
恐ろしい事にこういうものが無くても運用できていたと言うことはどういうことかというと、問題がない時はいいがトラブルが有ったら対処できないのが現実である。よって、この定義表は簡単なものでは有るが、埋めていけばいいので文章を書かなくて良いし、運用知識の可視化ができた。従来は経験がある者にしか運用ができなかった事がわかる。

4）アーキテクチャフレームワーク事例　TOGAF（The Open Group Architecture Framework）
・主な EA フレームワーク
Enterprise Architecture(以降　EA と記述)は、1987 年に IBM のシステムジャーナルにおいて Zachman が提唱している。これが世界を変えた論文である。単に解説的な論文であったが、整理が如何に大切かという事である。

・Zachman フレームワーク
（http://en.wikipedia.org/wiki/Zachman_Framework）
単なる表であるが非常にわかりやすく定義されている。この時点ですでに「運用」も入っていた。これがベースとなって、複数の企業に存在するシステムを最適配置し、また最適な計画を立てる仕組みが EA である。つまり、EA は SOS なのである。大企業だとその企業の中にシステムが 100 以上あるのが普通で 1000 のシステムがある場合もある。その場合、戦略的な計画が無ければマネージメントは不可能である。つまりこの戦略計画がシステム設計する以上に非常に重要である事は明白である。

・変換分析
現状からあるべき姿へ発展させるためにはどうしたら良いか。EA の基本は現状の理解とあるべき

姿を明示して、このギャップを埋める活動が EA による開発である。
Hitchins らの提唱の説明で述べたように、現状のシステムをあるべき姿のシステムで置き換えるという考え方である。

・TOGAF の主要なアクタ関係
TOGAF は The Open Group Architecture Framework である。Open Group は Linux の標準化を中心に活動しているグループである。主要なアクターは、ステークホルダ、スポンサー、アーキテクチャ委員会、EA チーム、実装プロジェクトであり、役割を持った複数の人達の間の関係が整理されている。よって、TOGAF は複数のシステムを最適設計していくための仕組みなので、アーキテクチャを整理する上で複数システムを前提にして総合的な最適性を追求するものである。

・TOGAF のステークホルダ
TOGAF のアプローチは必然的に社会技術的なアプローチとなるので関連する組織は多く有る。よって、単純になるからと言って、システムのアーキテクチャ設計の技法と解釈するとうまくいかない。なぜならアーキテクチャビジョンをまず作り、そのビジョン段階でビジネスシナリオを検討する。つまり、このシステムを実現するもしくは品質良く作るだけが目的ではないという事である。

・ADM（Architecture Development Method）フェーズと技法の関係
（https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P68）
TOGAF に沿った開発手法は初期から SOS を対象にしている。次に変革即応性評価つまりケイパビリティ評価である。これはその企業組織を改革したいと思っているかということである。システム開発のケイパビリティも同じだが、能力のない者にはシステムは作れない。

よって、プロジェクトを成功させる秘訣は一つしか無い。目的を実現できる能力を持った者を集める事である。そうでないと途中で放棄される。もう一つのやり方は、自分の組織だけでやるのであれば、そのメンバーの能力に応じた目的だけを定義する事である。こういう議論を開発現場ですると夢が無いと言われるが、実際は非現実的な夢を見れば失敗するのである。では、自分達の能力を超えた夢を達成するために何をしたら良いかと言えば能力を持った外部パートナーと協業するべきである。

こういった考慮をするために最初に作るのはビジネスアーキテクチャである。ビジネスアーキテクチャとは、ゴールの定義とビジネスプロセスの定義である。これはまさに CONOPS でやるところである。つまり、運用回りの明確化、情報システムのアーキテクチャ、テクノロジーアーキテクチャ（情報技術基盤）の設計である。情報システムアーキテクチャの中で、データアーキテクチャとアプリケーションアーキテクチャを定義する。次に、機会とソリューションとの対応でどういう可能性があるか分析する。これがわかるとこの中でトランジションアーキテクチャが明確になる。変換分析では、現状から有るべき姿をどう実現するかを分析する。ここでケイパビリティ計画が必要となる。つま

り、実現可能なリソースがあるか、さらに配置はどうするかということである。

次に移行計画を作成し、実装ガバナンスへ入り、実際のシステム開発に入る。最後にアーキテクチャ変更管理を実施するがここが最重要である。通常システムを作ったら終わりと考えていないだろうか。作り上げた次の段階から運用が始まるわけで、システムのゴールを定義したがこの定義したゴールがシステムの運用によって、本当に達成されたかどうかをモニタリングする必要がある。

この時に実装されたシステムが目的を達成できていなかったら、二つの場合がある。一つはシステムの実装が誤っている。二つ目は目的を間違えている。つまりビジネス環境の中でこういう目的を立ててシステムを作ったが、その目的は仮説でしかない。ビジネスが変化していれば、あるいはビジネスの捉え方を誤っていれば目的が正しく達成できない事が有る。その時は目的を変える必要が有る。つまり、新たな目的のためにアーキテクチャビジョンを再設計する必要が有る。よって、TOGAF でも Hitchins 氏らの提唱の紹介で説明したような外部ループがここで回っているという事である。

TOGAF はアーキテクチャとかソリューションをしっかりと定義するので、それをビルディングブロックとしてリポジトリに入れておき再利用するようにするという仕組みを提案している。よって、Zachman の EA と比較するとかなりにプロセス寄りになっていることがわかる。具体的にどのようにアーキテクチャを開発するのかという方法が ADM である。これら各フェーズが繰り返されるわけである。

・TOGAF のエンタープライズ連続体
(https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P75)
アーキテクチャ、ソリューション、実装があり、アーキテクチャ連続体は、固有なものから汎用のものまでが連続的に構成されている。これは、個別の知識から汎用の知識までが連続的に発展していくという考え方である。こういうコンセプトが欧米にはあり、よって欧米は標準化が重要との意識が強い事がわかる。

・ケイパビリティとアーキテクチャ
(https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P76)
管理、構成、要素、成果物のカテゴリの各々の中身をうまく組み合わせて、全体をできるだけ標準的に一貫性がある形で設計していくという事が、TOGAF の考え方である。

・TOGAF アーキテクチャ要求仕様の構成と開発工程
(https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P60)
基本項目としてゴールが有り、重要なものとしてアーキテクチャの原則方針、アーキテクチャを設

計する場合に守らねばならない方式がある。
この枠組みは日本人にとっては難しい。実際、要件定義、設計の基本原則が無いからこそ人によって視点がずれていき問題を起こす。海外ではこういったアーキテクチャの構成を大切にしている。システム個別で設計してはバラバラになってしまうような所を中核として、これだけは絶対に守らなければならない所をまず先に書くという事である。

・FEA の概要（紹介のみ）
（https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P64）
Federal Enterprise Architecture を紹介する。

・Gartner フレームワークの工程（紹介のみ）
（https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P64）

・アーキテクチャフレームワークの比較
（https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16195&folder_id=4605　P64）
各フレームワークの特徴をフレームワークで比較したものである。用語体系が突出しているのが Zachman であり、TOFAF の V9 は全体を平均的にカバーしている。Gartner のモデルは、ビジネス価値と参照モデルに重点を置いている事がわかる。

◆3. 保証ケースによる品質の確認手段
保証ケースによりどのように品質の保証ができるかについて研究結果を説明する。

1）非機能要求グレードと保証ケース
　保証ケースによって、どのようにソフトウェア品質を保証できるかについて説明する。

・保証ケース（D-Case）の例
（https://acs.is.nagoya-u.ac.jp/index.php?module=User&action=DownloadFile&id=6797&file_id=16673&folder_id=4806　P19）
ゴール　ストラクチャリング　ローテーションと言い、イギリスのキムケリー氏が提唱した手法である。まず、図形の説明であるが、四角はゴール、丸四角は前提、菱形は戦略、丸は証拠である。ゴールが成立する事を証拠で説明するという事である。しかしながら、ゴールが直接説明できない事があるので、戦略により上位の大きなゴールを下位の複数のサブゴールに分解して、そのサブゴールであれば証拠があるので成立する、という事を説明している。つまり、証拠を使って主張が客観的に論理的に説明できるかを議論するための図が保証ケースである。
　なぜ、「主張」「戦略」「証拠」の3つに説明できるかを示すために、コンテクスト（前提）がある。この前提条件の元でこの上位の主張は3つに分解できるという事である。

・D-Case の5つの役割
保証ケースの事を、Dependability Case と呼んでいる。この役割は以下の四つの要素であるので、お客様との間で合意を形成する場合に便利に使える。(1)主張を明示的に書ける、(2)証拠を明示的に定義できる、(3)主張の前提条件を明示できる、(4)物証により主張を論理的に説明できる、(5)標準的な表記法なので客観的な説明が可能である。

・D-Case 実装評価研究会：4回開催、述べ 130 名が出席（http://www.dcase.jp）
　D-Case は初心者には書きにくい。初心者でも書けるようにするに、初心者の目線での習熟評価のため「実証評価研究会」を立ち上げた。この研究会において参加者からの意見をいただき普及活動を実施している。これは文部科学省の研究開発プロジェクトであるが、2014 年 3 月で終了となる。よって今後は次の書籍を参考にすると良い。「主張と証拠　2013/4　山本修一郎著」、「実践 D-Case 2013/3　松野裕・山本修一郎著」

・D-Case 活用事例
（http://dcase.jp/example.html）
保証ケースであるが、どの位使われつつ有るのかをしめしたものがこの事例である。自動車のエンジン制御、衛星制御、ロボット制御、金融パッケージミドル等の動作保証を客観的に説明するために用いられている。

・安全性ケースを作成する上での課題
まず、ゴールには何を書いたら良いか。日本人には馴染まない。なぜかと言うと、日本の教育は問題に起因した解き方しか教えないし、作文も感想文を書かされるのみ。よって、自分の意志を持って何かを主張するという作文をしていないからである。そこで戦略からいくつのサブゴールに分解すれば良いのか、また証拠も残してないので何を書くべきかわからない。つまり、展開幅とかどこまで深度を求めるかもわからない。また、各要素の間の関係も線で結んだものの、その正否が判断できない。

・議論分解パターンの構成
保証ケースを書くのに、初心者でもわかるようにしたものが議論展開パターンである。これを分類するパターンとして、「説明対象システム」「説明する議論の構造」「どういう証拠を使うのか」「正しい事が確認された再利用時の確認のし方」がある。この4つのカテゴリーに分けてパターンを整理する事を提唱している。

・議論分解パターンの分類
議論分解パターンを用いて、保証ケースを作成する上での分解問題の種類と、それに適用できる保証ケースの種類を対応付けることができるので、保証ケースの分解方針と解決策の再利用が期待できる。

・パターン事例

議論のパターンをまとめると50を超えた。これだけの事例があればほとんどの問題は記述可能であるし、記述対象毎にパターンをつくったものを見ると、記述モデルの構造が決まっているのでその構造があればそれに対して保証ケースを書けば良い事が解ってきた。更に記述法が増えれば新しい記述パターンを追加すれば良い事がわかってきた。よって、さらにパターンを拡充しようとしている。

・パターンの記述例
2014 年 3 月に予定されているパターンの国際会議 AsianPLoP 2014 に関連して、パターンの記述実態をスライドとして追加した。パターン毎に情報を記入しており、なぜそのパターンが必要なのかという状況、解決策が配置される前提条件等を持ってパターン化した。これに基づいて記載したパターン記述例を紹介する。分解問題、分解状況、解決策、例えば複雑な問題に対して D-Case を作る必要が有る。

分解状況はアーキテクチャが明確になっているときである。アーキテクチャが明確になっていない場合はアーキテクチャ分解パターンは使えない。解決策はシステムのアーキテクチャが二つのサブシステムによって構成されているとすると、サブシステム間に相互作用が有る。その場合、システムが保証されると説明されるためには、少なくともサブシステムの両者が保証されていると説明されていること、更に、サブシステム間の相互作用が保証されているという事が説明できる事が条件である。そのときの前提はシステムアーキテクチャの定義である。

また、各説明要素は and 条件である。なぜかと言うとシステムの安全性や妥当性を保証するためのものだからである。つまり、条件が成立する時もしない時もシステムは安全でなければならないからである。

条件選択に対する信頼性も用意されている。矛盾を解決する時のパターン、代替案と比較するパターン、参照モデルのパターン、DEOS プロセス等が用意されている。なお、DEOS プロジェクトとは、システムが通常運用されている時も安全で、システムが変革対応する時も安全、更にシステムが障害を起こした時も迅速対応できる、つまりシステムが全体で信頼できるという考え方である。よって、DEOS プロセスは運用を前提としたような考え方になっている。

IPA が数年前に非機能要求のグレード表を作ったが、その非機能要求に応じた形で、確かにこのサービスは非機能要求を満たしているという事を示すというためのパターンである。 Availability（可用性）をシステムは達成していますという事をまず主張する。次に Availability の複数の各品質特性を達成している。その品質特性を達成している事を試すために、特性評価指標が定義されており、その特性表か指標が達成されている事が主張になり、その主張を満たすための情報が証拠になる。達成結果つまりテスト結果証拠文書等がエビデンスになる。

・Dimensions of Assurance Integrity Level（保証の一貫性レベル）
どの程度まで保証ケースによってシステムの信頼性を保証したのか、そしてその次元の考慮であ

る。議論の網羅性、エビデンスの信頼度、証拠の信頼性、前提条件の完全性と言った次元が必要である。よって、保証ケースの研究課題であげた、「書かれた保証ケースの妥当性」の吟味がいるという事である。

・欠陥分析手法と安全性ケース
「保証ケースは昔からやってきた事の繰り返しではないのか」との意見を良く聞くが、回答はノーである。従来は欠陥分析、つまり「ここへこういう対策をする」で終わっているはずである。保証ケースは安全性を保証する手法なので欠陥分析手法によって明らかになった問題原因、それに対する対策案、システムの中に組み込まれていてその証拠はここに有るというのを説明するのが安全性ケースである。具体的には、欠陥がどこに有るかと見て200個並んだとすると、その100の欠陥に対して安全対策がそのシステムの中で実現していますという安全を示す証拠を示すのが安全性ケースである。

2）ISO26262 と安全性ケース
・ISO26262 による安全性ケース
ISO26262 は 10 分冊有り、その中の Part10 に安全性ケースをしっかりと書くべきという事が述べられていて、Goal Structuring Notation つまり、今まで保証ケースの例で書いてきたものである。これが安全性ケースを記述する手法として記載されている。また、重要な点は、プロダクトについての議論と、プロセスについての議論をするべきであるという事である。プロダクト面だけで安全であるという事は不十分であり、製造プロセスや運用プロセスも安全であるという事を確認すべきであり、それを保証するドキュメントをつくるべきである。
システムの安全性を保証するために、安全性ケースを書くとシステムのリスクが発見できたとすると、システムの設計を変更する、それに応じて安全性ケースも更に変更する。このサイクルが重要である。

・ISO26262 の概要
ハード製品開発とソフトレベル製品開発が有り、その前にシステムレベル製品開発がある。その前の段階において、機能安全性の管理が必須である。

3）非機能要求グレードと保証ケース
・名古屋大学ポータルサイトの NFR(非要求)グレード
　信頼性特性のカテゴリとして「可用性」「性能・拡張性」「運用・保守性」「移行性」「セキュリティ」「環境・エコロジー」が定義されており、各々に特性仕様項目が有り、その指標の合計は162項目がある。

・主張の定量評価尺度定義法
「非機能特性を達成している」という主張を各特性項目毎に分解して説明からその証拠、つまり特性指標の達成報告までを説明するための一つの定義多ターンである。

4）O-DA 標準の概説（TOGAF の dependability 拡張としての O-DA 技術標準）
TOGAF の ADM を損来の高いものにする取り組みをしており、20131 年 8 月に国際標準で TOGAF が認められた。その内容について紹介する。

・O-DA（安全・高信頼性検証国際標準）を発表　8/7 日（米国時間 6 日）
（http://www.opengroup.or.jp/pdf/pressreleaseFinal_Update20130806.pdf）
（http://www.opengroup.or.jp/pdf/nikkeisangyo20130808_O-DA.pdf）
TOGAF について日本の機関が貢献したのはこれが最初の標準である。

・O-DA 標準の構成
「定義」は保証・高保証アーキテクチャ・保証ケース、次に「O-DA フレームワーク」は障害対応サイクル・変化対応サイクル・通常運用サイクル・保証ケースの説明がある。基本的には用語の説明と形式手法の説明である。これは証拠として使う。証明がされている事によってシステムは信頼できるという事が証明可能となるという事である。

・Open Dependability through Assuredness Framework（変化対応サイクル）
O-DA のサイクルには、要求プロセス、アーキテクチャ設計・開発がある。その時にそれぞれのプロセスが信頼できるという事をケースとして保証する。障害対応サイクルでは、問題検知がされたら原因究明し迅速対応する。目的環境変化対応では、ビジネス環境が変わると目的がそぐわなくなるので、新たに要求プロセスを再度回す事に成る。これも二重サイクルでは有るが、一つのシステムに対してこの DEOS のプロセスが有るという事である。前述した Hitchins 氏らの提唱した二重サイクルとは異なる。Hitchins 氏らのサイクルは、内部サイクルが特定のシステムのサイクルである。でもよく見るとこのサイクルは運用を前提にしているので、この運用の中で複数のシステムが運用されている可能性もある。すなわち、システム自体が Open 化されているとすると、このモデルは SOS に対して適用すれば Hitchins 氏らのモデルに近くなるとの見方も有る。

・高信頼性（Assuredness）の言葉の意味・定義
アーキテクチャの実装すべき要求が満たされている。その事をあるレベルの証拠によって定義しているという事について、ステークホルダが合意していますという事が Assuredness である。よって、ステークホルダが認めた要求をアーキテクチャが達成しているという証拠があるという事である。つまりステークホルダが合意した要求になっているという事である。

・高保証アーキテクチャの言葉の意味・定義
Assurable な要求を達成したアーキテクチャーが高保証アーキテクチャである。そのために、保証ケースを使うという事が規格の中で述べられている。

・説明責任の言葉の意味・定義
サービスを継続的に提供できてかつ説明責任を遂行できる能力がディペンダビリティである。説明責任能力がある事によって説明責任を定義している。説明責任が信頼性の構成要素に成ってい

るという定義である。

・保証内容メタモデルの言葉の意味・定義
要求が拡張されており、証拠付き要求という概念が必要になっている。つまり、要求は単独で定義すれば良いわけではなくなっており、信頼性の有る要求、つまり達成されたという証拠のついた要求が必要である。通常の開発では要件定義を最初にやりその者は終われば居なくなるが、システム開発が終わり運用中に確かにその要求は保証されている、達成されているという証拠が必要である。つまりこの証拠は運用されないと残らないわけである。また運用中にこの要求が達成されなくなる事があるとも言える。よって、要求は常に存在しているという定義に拡張されているわけである。開発中から運用中を含め常に監視対象になっているという事である。これがないと要求が達成されなくなったという事を確認できないわけである。

・高信頼 ADM の概要
O-DA では、前に述べた ADM（アーキテクチャビジョンからアーキテクチャ変更管理まで）が全て高信頼化される必要があるという事を述べている。
これをどう実現するかというと、アーキテクチャビジョンでは「ディペンダビリティのスコープの定義」「主張の定量的評価尺度の定義」「保証ケースの作成能力」等を定義する。ビジネスアーキテクチャ、情報システムアーキテクチャ、技術アーキテクチャ、ソリューションの４つを保証ケースを使って、確かにそのアーキテクチャが妥当であるという事を確認する事が求められるようになった。

運用についても運用管理の保証ケースを作る事が明記されている。よって、運用している時もこの運用は問題なく実行されているという証拠を残す必要があるという事である。つまり、運用プロセスが定義されている事が条件なので、運用要求記述票の説明で述べたように、運用活動とは何か、運用活動におけるリスクとは何かを明確に定義して、運用リスクを解消する対策がとられているという事を確認する必要がある。これが達成できれば、保証ケースが作成できる事になる。そのために、ITIL についての保証ケースを作る必要がある。

５）ITIL と保証ケース
ITIL に対する保証ケースを使った品質保証の仕組みを説明する。

・運用活動の妥当性確認例
現状の ITIL は非常に拡張されていて運用だけではない。サービス戦略、設計、移行、運用、サービス改善まで含んでいる。これは TOGAF とも非常に似てきている。開発と改善のサイクルが絶対に必要だという事がわかる。なぜかというと、TOGAF にも有るし ITIL にも有るからである。

・IT サービス継続性管理の活動
サービス設計の中に記述されている「IT サービス継続性管理活動」が有り、この内容を使いどのように妥当性を確認するのかを紹介する。
　４段階に整理され各々の段階毎の活動内容を示す。段階とは、「開始」「要求と戦略」「導入」

「継続的な運用」である。この各段階に対して全65項目の具体的な活動項目が決まっている。

・開始段階のチェックリストの例
各段階の活動項目全てを満たす必要があるので、チェックリストができる。各活動項目毎にどういう状況が達成される必要が有るのかという事を明らかにするとチェックリストが作成できる。

・IT サービス継続性管理に対する開始段階に対する D-Case
各活動項目のチェックリストに対してサービス継続性管理を、先に述べた「主張」「前提」「戦略」「証拠」の4段階に分解するパターンができる。このようなパターン化をするとシステマティックに運用プロセスの妥当性評価ができる事が解る。

6)テストと保証ケース
(http://www4.atpages.jp/sigksn/conf13/SIG-KSN-013-04.pdf)
テストの十分性、どこまでテストすれば良いのかに対して保証ケースが適用できるかについて説明する。

・D-Case とテストの関係
全てのプロセスの妥当性を保証しようとすると、要件定義や設計だけではなくて製造やテストもその工程の十分性の確認方法が問題になる。

・テスト十分性に対する保証ケース作成手順
テスト項目だけを見ても、そのテストが妥当かの判断はできない。どこまで遡る必要が有るかというと、要求である。つまり、要求に対してこのテスト項目で本当に要求がシステムによって実現されたかを確認できるかを確認する必要がある。すなわち、システムが問題なく動くという事を確認する事ではなくて、システムが要求を実現している事を確認する必要が有る。

・要求記述表
(http://www.bcm.co.jp/site/youkyu/youkyu108.html)
要求仕様に着目して要求記述表を記述しておく事が大切である。この要求記述表は ISO29148 の記述項目を使っている。
要求記述表は、通常1行しか書かない場合が多いが、ISO29148 で要求されているのは、「システムの主体は誰か」「何を対象にシステムが実行されるのか」「実行の契機」である。つまり何を対象にその要求は実行されるか、その要求が実行される契機は何かという事を書けないとテストはできない。機能として、状態変化とかイベントの入力からどういう出力を出すかという事を分析する事でありこれが正確な要求分析表である。

・要求逸脱分析表
この要求が記述できたら、次に要求の欠陥が発生するのはどこだとの事に着目して要求逸脱分析をする。これにより作られるのが例外系のテスト項目である。

パラメータは要求記述表の各記述項目のそれぞれに対して、HAZOP(Hazard and Operability Study)のガイドワード(逸脱の条件)を用いて、必要とされる以外の情報が入る可能性があるとか、部分的で完全ではないとかの逸脱の条件を分析して記載する。その逸脱の重要性をリストアップして重大な事項についてはテストをする。それ以外はテストしない。なぜかというと、システマチックに網羅的にあげる事ができるようになるため、この条件網羅の結果は非常に多数となる。その結果多数のためにテストができなくなるからである。

・テスト十分性確認 D-Case(一般形)
(http://www.bcm.co.jp/site/youkyu/youkyu108.html)
トップレベルのテスト十分性の D-Case のテンプレートを示す。まずは要求毎にテストを実施する。次に正常系・例外系を分類する。異常系要求逸脱表で挙げた物を全て並べていく。

7)具体例
・要求仕様の例
　要求仕様の例として、ビデオレンタル会員がビデオを借りる場合を例としてあげる。通常正常系しか記述されていない。これについてテスト項目を考えてみる。

・要求記述表の例
この要求を記述表に書くと要求仕様には記載されていない、条件・機能・制約の明確化が可能となる。こういう用にして考慮漏れが無いかを洗い出す。

・要求逸脱分析表の例と・テスト十分性確認 D-Case(具体例)
そもそも、システムが起動できないとかメニュー以外の画面が表示された場合とかの逸脱条件の洗い出しが可能となる。条件項目が挙がったら、お客様と議論をして必要な条件を選択し合意すれば良い。これで必要なテスト項目の決定ができる。

◆Q&A
Q1_ITIL と保証ケースの説明で、IT サービスについての分析をしてそこから D-Case が導出される部分について、D-Case とその前の分析のつながりが解らない。D-Case のゴールはどういう流れで出てきたのか。

A1_段階毎に分類した活動の大分類があるが、まず活動内容に分解する。その活動内容の結果全てが最下段の証跡になる。同様に全ての段階毎の活動結果を証跡として並べた物であり、この条件全てを満たせば、サービスの継続性が維持できる事を証明できるということである。

Q2_システムのある観点で分解して、and で満たしていくという考え方のみだと、Open system までの保証はできるが、Enterprise になると更に複雑な相関関係も考慮が必要なので、そこまではカバーできないように考えるがいかがか。冒頭のシステムの階層の説明で、「Enterprise」「System of systems」「System」の階層が示されているが、SOS までは、システムとシステムの相関であるので

個々のシステムも満足、つまり dependable かどうか確かめていけば全体としても dependable だという考え方と理解した。しかし、Enterprise は更に複雑との理解だがいかがか。

A2_結局は階層性が有ると言う事なので、Enterprise の中に複数の SOS がある事である。環境とも相互作用があるし、システムと環境の作用も有る。環境の中自体はわからないとしても、環境とシステムの間の関係の分析はある程度できるし、それを分析しておかないと想定外の事が起きた時にどうするかがわからない。リスクマネージメントの根本はシステムが想定した条件の下ではただしく動作する事が確認できる。想定している範囲で条件からの逸脱が有っても対処している。つまり例外入力に対して対応ができる。それは例外処理が有るからである。では最後に何が残るかというと、想定した例外以外の何かが起きる Case でそれは書けない。では起きた時にどうするかというと運用プロセスで対応する。よって、それが最後の歯止めになる。運用プロセスにおいてシステムが困難な状況に陥ったら、「このようにしてここまでは何とかして継続性を保証します」と記載されていれば、その訓練はできる。「何が起きるか解らないが何か重大な異常が起きた時にはこのようにしてシステムの継続性を保証します。」という証拠は作成可能である。よって、このようにすれば Enterprise 系のシステムでも Assurance Case を適用できる。

Q3_システムエンジニアリングとして運用についてお客様に対し契約をアプローチするが、契約に結びつかない事が多い。特にお客様側が想定していない状況を条件としても交渉が難しい状況である。よってこういう運用についての提案は誰が出すかが重要と考える。ステークホルダを誤ると突然否定される恐れもある。よってこういった取り組みはお客様側と共に検討していく以外無いのか。

A3_その通りである。正しいお客を見つける必要が有る。間違ったお客様がステークホルダだと思い込む場合もある。逆に言えば、間違っていると判断した場合には適切なステークホルダを探しに行く必要が有る。

Q4_Dependability Case はあくまでも証拠があるという事であって、運用するシステムは正しいとか安全だと言う保証ではないと考えて良いのか。つまり現実の運用の中で確かに安全であるとか Dependable であるという事は、D-case などに照らして判断して、間違いが有れば見直して行くという事か。もしも前提が間違っている場合は、Case の見直しもあり得るのか。

A4_その通りである。システムが安全だという説明ドキュメントが Assurance Case である。説明の中にはシステムが安全だという証拠がある。という事である。つまり、なぜ安全だと言えるのかという事を説明する文書である。前提が間違っていれば Case を見直す。必要な議論が抜けている場合も有るし、必要な証拠がそろっていない場合も有る。最悪の場合証拠が偽造の場合もある。

Q5_証拠能力の内容で、SNS とか ID サービスでは「如何に受けているか」とか「如何に沢山の人が使っているか」という尺度が、品質が良いという尺度よりもビジネス要件として重要である。これを本日の手法で検証しようとした場合、アクセス数をエビデンスとして使うべきと考えるが、企画の段

階で取れるアクセス数と、サービス立ち上げ後の階で把握できるアクセス数との間では、ユーザの層の違いとかで中身が異なる。その場合証拠として正しいかどうかが危ういと見られる恐れが有る。このように同じ手法で把握したとしか主張できない場合もある。その場合は企画の段階で何を正とするかを合意した上で進める必要があるという事か。

A5_そうならないように企画段階でしっかりと Assurance Case を書くという事である。つまり、アクセス数がこの手法によって収集されますと定義する事である。
　アクセス数の把握条件が異なっていた場合でも、そのアクセス数によって確かめられる上位の主張が正しいのかどうかという議論ができる。実装する前にそういう Assurance Case を書く事により、この主張はこのアクセス数によって保証できるのかどうかを事前に議論した方が良い。アクセス数を収集できれば良いという考えだと、どのような実現手段でも良いという事である。つまり、実装された時のアクセス数が本来とるべきアクセス数と乖離していたという事を提起する事である。

Q6_安全性は大事なコンセプトだが、「謝ってしまえば良い」というサービスも多く有る。そういう風土に対して金券を配れば良いとか、値段の交渉にも発展する場合もある。また、電子書籍を出した時にユーザが APL を Install できなかった時に、社長が NG は2割か3割でしょと言ってしまったものとかもある。こういった事例は社会をダメにしていると思う。こういった事例も含めプロセスだけでなく社会全体の状況を見ていく必要が有ると考える。しかしながら風潮としては謝ってしまえば良いという方が強まっていると感じるがいかがか。

A6_コストパーフォーマンス的には謝って済む場合はその方が良いかも知れない。逆に、それによって失う信用もあるのでそういった事について議論すべきと考える。つまり信頼性を保証すべき対象の選択について保証ケースを使って議論をすべきと考える。よって、全てを徹底的に実施するという事ではないと考える。すなわち、保証対象の選択という議論がある。もう一つは、謝る謝らないに依らず罰則規定が有るようなものもある。つまり法律は守る必要が有る。この範囲で守っていますという事についての説明をする必要が有る。そしてその事について Assurance Case を書くことができる。

◆感想
昨今の複雑化するシステム運用環境において、その実態の認識は有るものの抜本的な対策は構想も出来ず、個々のサブシステム単位での対策に終始しているのが運用現場・開発現場で良くある状況との認識です。この重大でかつ複雑な課題に対してシステマティックな俯瞰手段と解決に向けた検討経緯と具体的な手段までを紐解いていただき、非常に感動的な内容でした。特に安全高信頼性検証のホットな標準化内容として O-DA 標準のご紹介をいただけたことは、我々としてこれを契機として今後の普及への命題をいただけたものと思います。
　さらに、今回の講演内容の現場への適用手段としてテンプレートを紹介いただけた事で理論から実適用への導きもしていただけました。
今後の課題としまして、今回の命題である Hitchins 氏らの定義する外部ループを捉えた俯瞰による改革もしくは改善の活性化には、サブシステムを開発する立場ではなく、外部ループを仕切れる

ステークホルダの思考を変革していく事が重要な課題であるとの認識です。
内容の濃い貴重な御講演に深く感謝いたします。