価値ベースデザイン

システム企画開発運用における原理原則

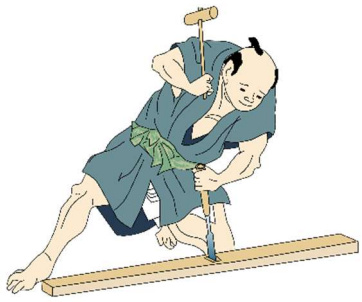
LEXAR Corporation | contact@lxr-j。com

概要

企業がシステムを導入するようになって数十年、DX がもてはやされてからもかなりの年数が経ちます。しかし、その企画開発運用プロセスは未だに経験と勘により進められています。本書は、その経験と勘の世界を科学の世界に変えるために執筆されたものです。
多くの日本企業は国際競争力を取り戻す必要があります。しかし日本の DX は欧米に比べて遅れていると言われています。その DX の遅れを取り戻すためには今のように時間・コスト・人財をかけていては追いつきません。
これを解決する唯一の手段、それが科学に立脚して KAIZEN を積み重ねることです。DX は非連続な発展だから KAIZEN ではないという方もいるかもしれません。非連続な発展はビジネスモデルが違うものであり、DX 云々とは別の話です。
本書はそんな KAIZEN 型 DX を短期間・低コスト・少ない人財で継続的に発展させるための原理原則を示したものになります。

生き残るために必要な科学

社会が発展するのに科学が必要であることに異論がある人はあまりいないのではないでしょうか。では、現在のシステム企画開発運用プロセス、実際の製造と内部テスト以外の上流工程が科学的なのか。システムの専門家ではない方は信じられないかもしれませんが、現在のプロセスは職人芸に支えられています。要件定義や基本設計のような工程ひとつとっても曖昧で、人によって定義や捉え方が違います。いやでも、ITIL とか COBIT とか TOGAF などいろんな標準があるじゃないか。はい、あります。しかし、それはベストプラクティスと言われ金言集であるため、実験科学的です。つまりある前提条件のもと有効ですが、このようなベストプラクティスはその前提に関する言及はありません。

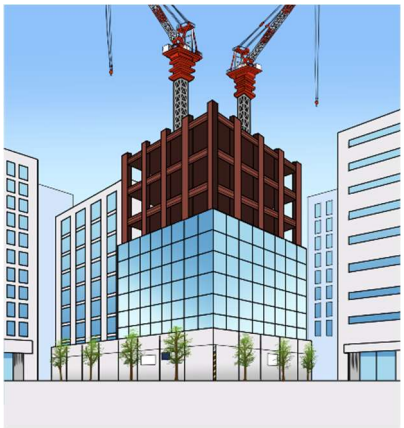


非常に属人的なプロセスは多くの負の遺産を残してきました。

- ・システムのブラックボックス化
- ・高額なコンサルティング費用
- ・ベンダーロックイン
- ・改善したくてもすぐできない動きの思いシステム
- ・大規模なシステム障害やセキュリティ事故
- ・ただ 5-10 年で入れ替えているだけなのに高額なコスト

このような負の遺産は、非科学的であり、その職人がいるうちはいいのですが、いなくなるとまた新たな職人の育成から入り、永遠と車輪の再発明を繰り返しています。現在、システムに関する社会課題に挙がっているものは、殆どがこの非科学的な世界が原因であることに多くの方が気付いていません。

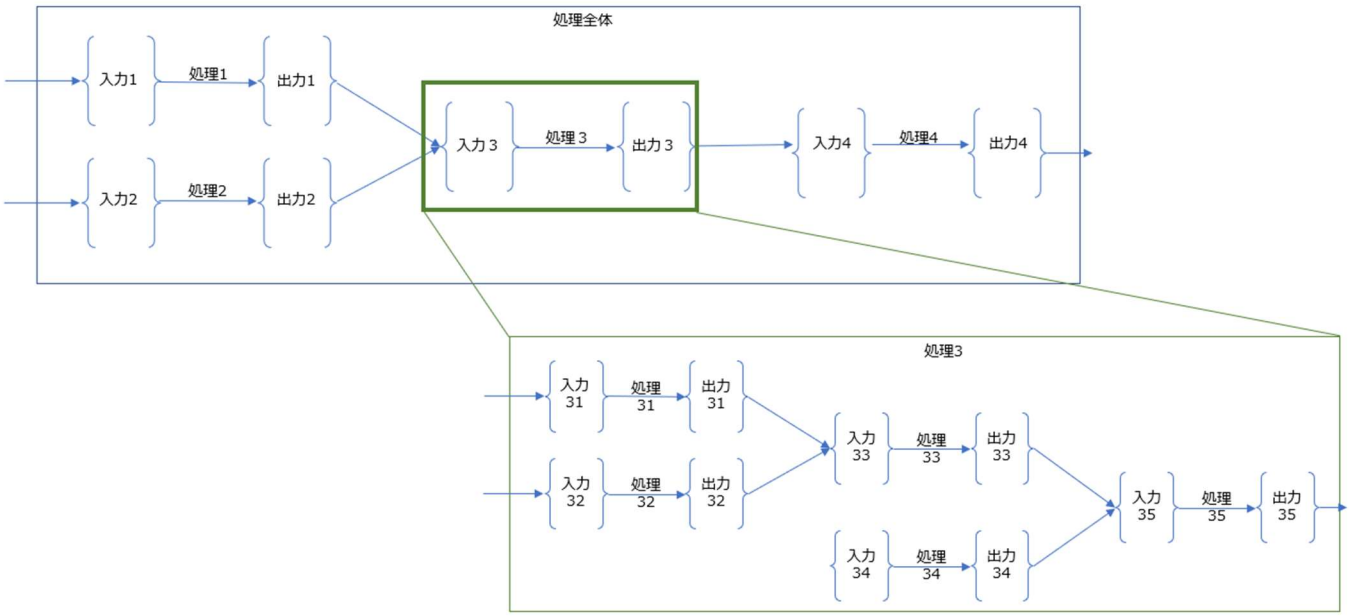
正しい原理原則に基づいた理論科学的な IT プロセスの構築が出来れば、現在社会課題になっているような問題の大半が解決します。
2025 年の崖で警鐘を鳴らされたブラックボックス化問題やデータ活用が進まない、生成 AI 活用が進まない問題などです。
そして最大の問題は、1 度システムを入れたら 10 年も 20 年塩漬けとなり、その間システムを含めた業務を改善しないことです。社会の変化は大変早くなっており、数年後どうなっているかわからない状況です。そんな中で投資回収期間が 10 年を超えるようなシステムを入れて、改善もせず業務を遂行して、今後の大競争時代に勝てるのでしょうか。



「最も強い者が生き残るのではなく、最も賢い者が生き残るのでもない。唯一、変化に対応できる者である」という言葉があります。今の時代に 10 年も 20 年もシステムを塩漬けにするのではなく、変化に強く常に改善し、生成 AI のような新技術に素早く対応するにはどうすればよいのか。
その解は、ただ 1 つ。正しい原理原則に基づいた科学的な IT プロセスを構築することです。生き残るには、科学の力が必要です。

科学に必要な写像

科学には実験科学と理論科学があります。全社はベストプラクティスでそれだけでは課題が解決出来ないことを時代が証明してきました。今必要なのは普遍的な理論科学です。理論科学の理論とは論理的ということであり、論理的とは入力があって処理があって出力がある関係が明確であり、処理内容の筋道が通っていることです。これを大学数学では写像と言います。理系では大学１年で習います。Y=F(X)も写像です。こちらは中学生でならいますね。それを少し具体化したものが写像です。



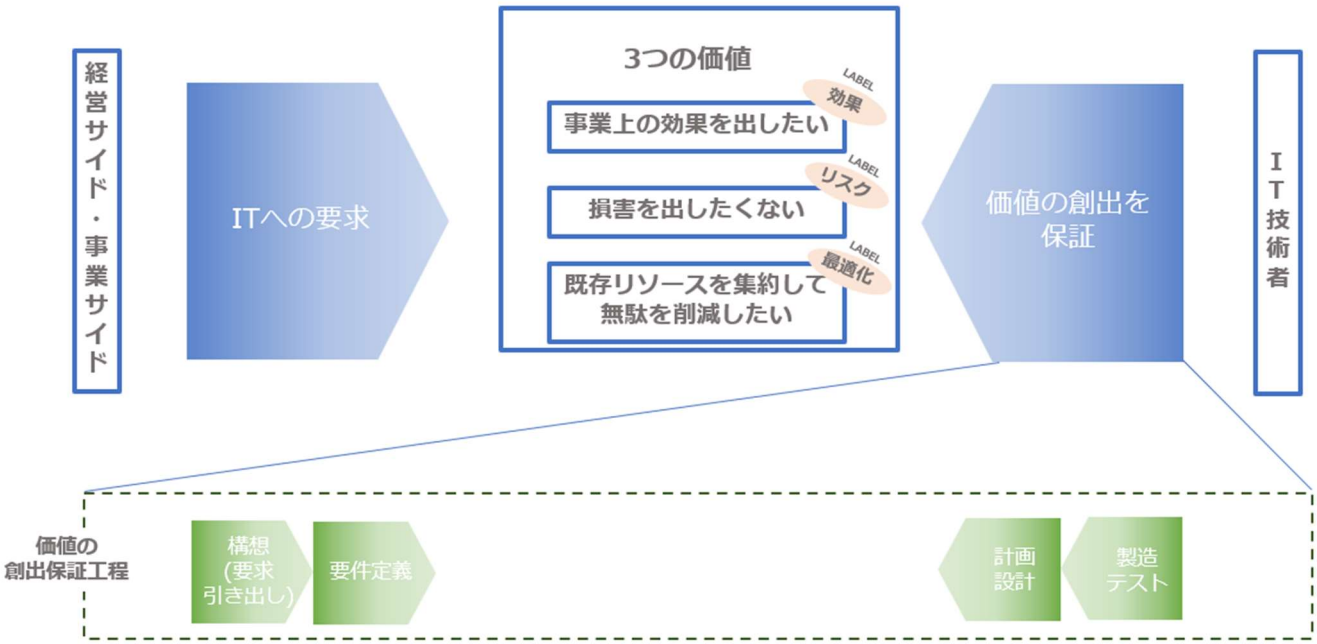
写像と難しく言いましたが、要は入力と出力・処理の関係性に矛盾がないか。上図のように写像にもレベルがあって、抽象度の高いものと詳細なものがあり、それぞれが論理的に整合している必要があります。

抽象化と段階的詳細化

人間は１度に処理できる情報量が限られるため、複雑なものを理解するには、抽象化する必要があります。この抽象化と詳細化の組み合わせは非常に重要です。なぜなら理解出来なければ情報をいくら整理しても意味がないからです。この理解というのは、ある人やあるチームだけが理解できるのも駄目で、極力幅広く誰でも理解できるレベルの言葉・表現にすることで、学習コストを省略できるため、幅広く効果がでやすくなります。

価値の定義

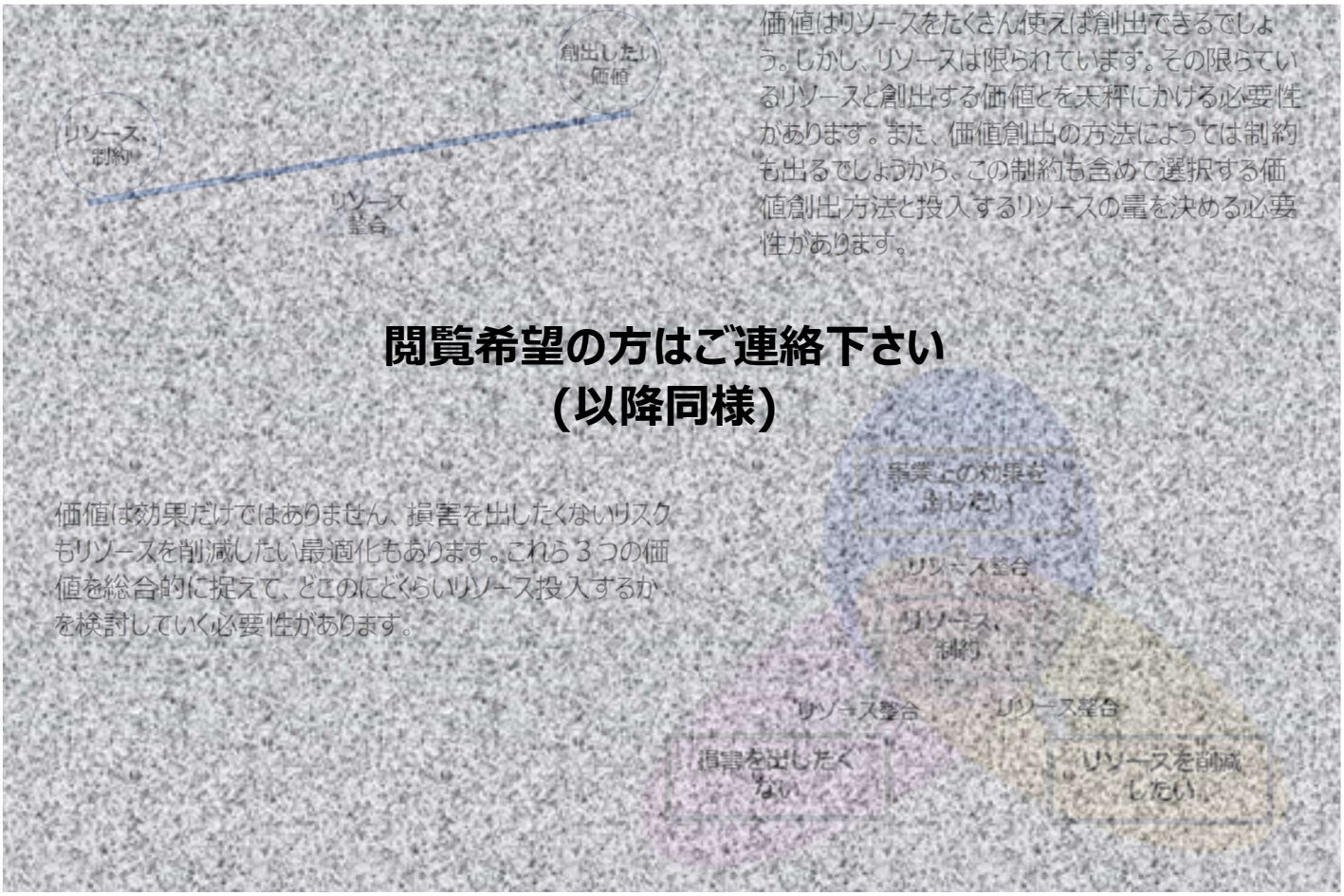
システムの企画開発運用プロセスの原理原則は最上流の要求定義から運用保守に至るまで一貫した原理原則のもと、理論展開できる必要があります。まず、私たちは何のためにシステムを作るのか、それは価値を創出するためです。その価値とは何でしょうか。その価値とは３つあります。最初に考えるのは売上を向上したり、作業効率を向上するためでしょう。つまり効果：ベネフィットを創出することが１つの価値です。二つ目は、その効果が享受出来ないリスクやセキュリティなど何かしらの損害を受けることを避けることです。三つ目は、個別にサーバーを立てていたものを会社全体で集約するなど全体で最適化することコスト等の効果が出るものです。実は、私たちがシステムを作ったり維持する目的は、この３つ以外にありません。
これが原理原則の最も抽象度の高い定義です。ここから段階的に詳細化していきます。



工程とは価値の段階的創出ステップ
要件定義や基本設計などのいわゆる工程は、この価値を創出するステップであると言えます。一般にプロジェクトがある程度の規模になってくるとその期間は１年や２年という長期間になってきます。これだと長期だとうまくいかなかったからやり直しという訳にはいきません。取り返しのつかない状況を避けるには、この工程毎にやるべきことを正しく定義して、しっかり作り込んでいく必要があります。そのような意味から、この工程の定義こそがプロジェクトの成否の大きなキーファクターとなります。

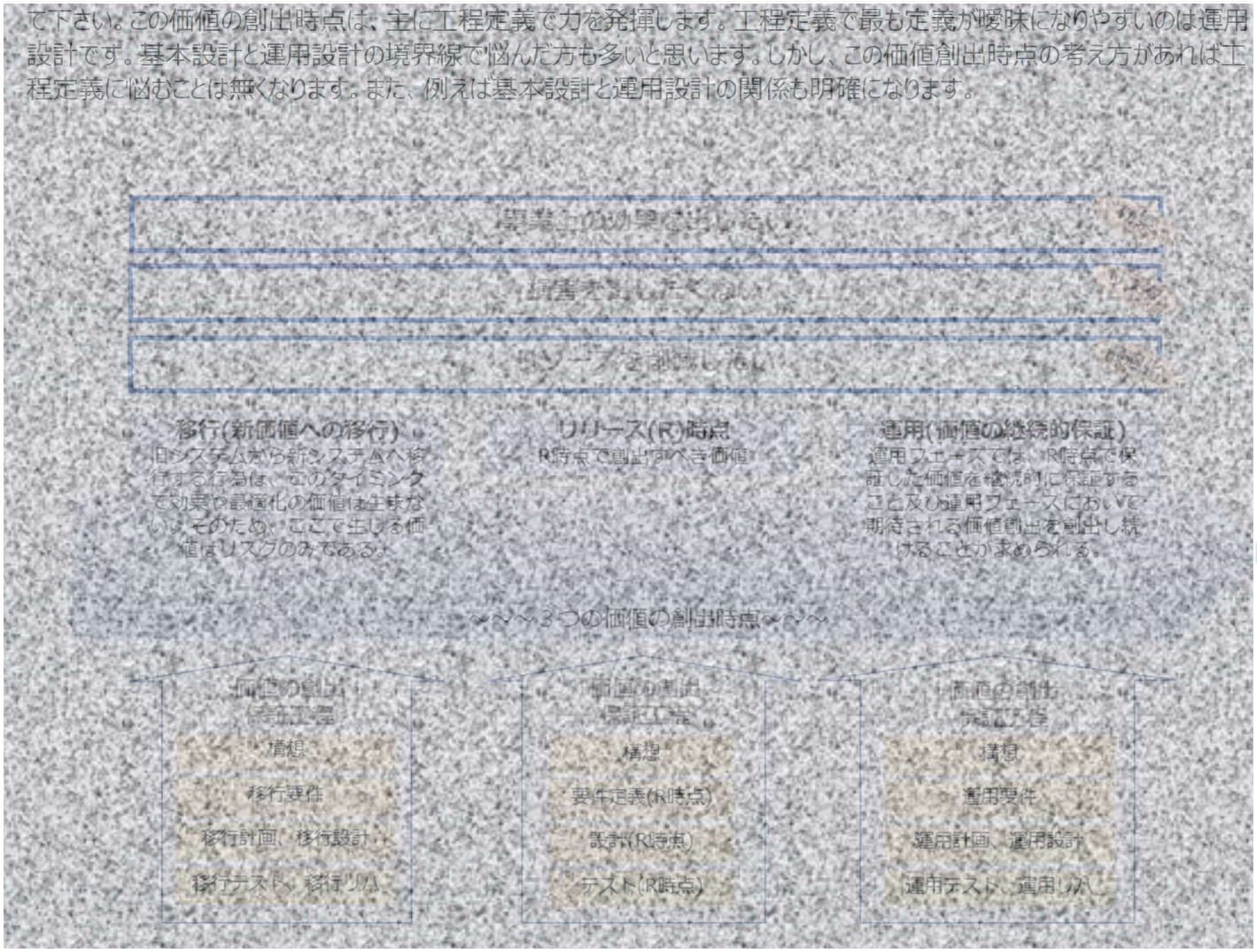
リソース整合

価値を創出するには、リソースが必要です。リソースとは人財・資金・時間など企業が持つあらゆるものです。



価値創出時点

価値には創出するタイミングがあります。イメージしやすい例として、リスクをイメージして頂くと良いと思います。システムをリリースした時点で担保されているリスク対策、システム移行時に担保されるリスク対策、運用時に施されるリスク対策をイメージして下さい。この価値の創出時点は、主に工程定義で力を発揮します。工程定義で最も定義が曖昧になりやすいのは運用設計です。基本設計と運用設計の境界線で悩んだ方も多いと思います。しかし、この価値創出時点の考え方があれば工程定義に悩むことは無くなります。また、例えば基本設計と運用設計の関係も明確になります。

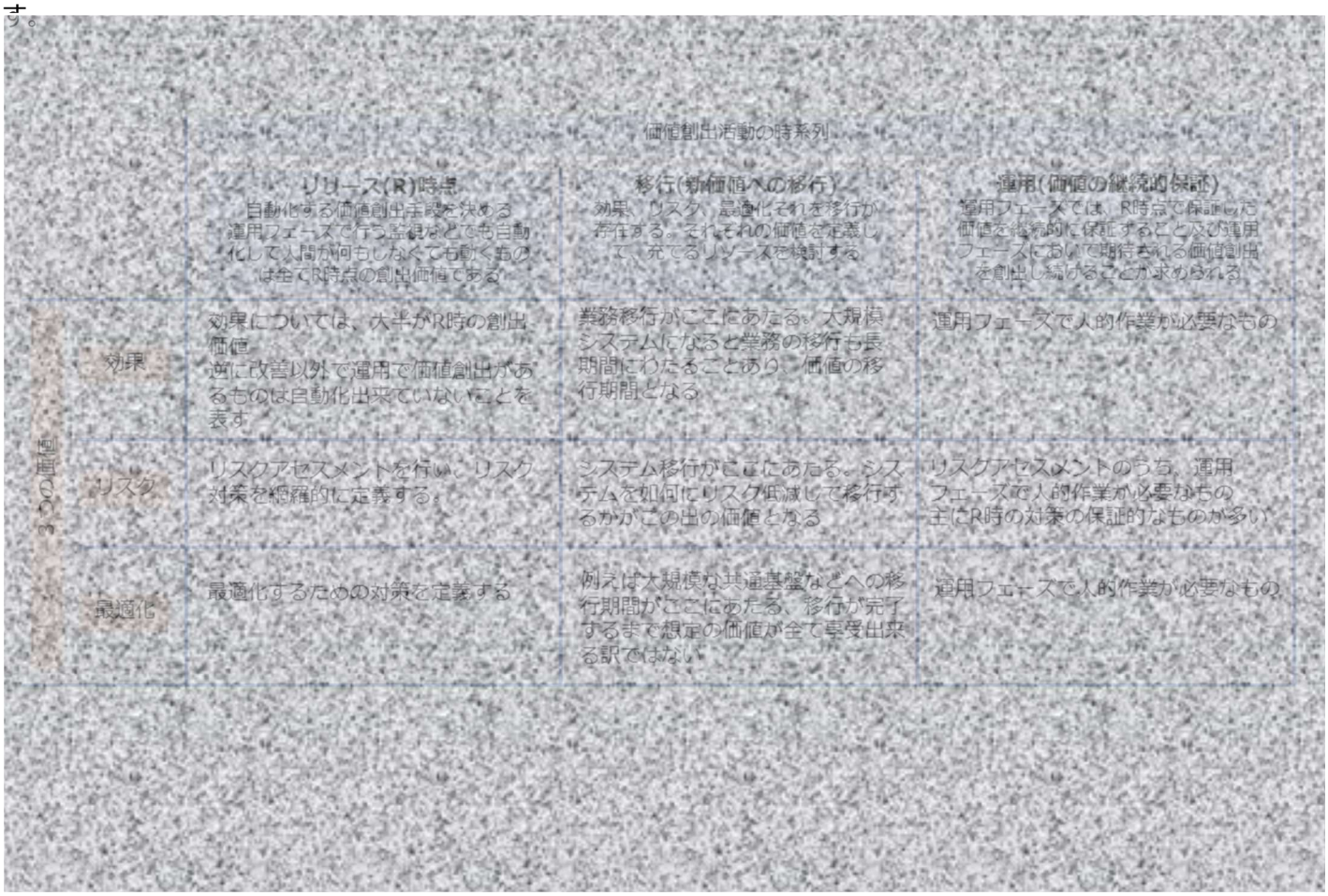


工程の導出ロジック

工程定義を導き出すには、IT が創出する価値や価値創出時点に加えてハイレベルな工程の定義を掛け合わせて考える必要性があります。掛け合わせのパターンが多いため、掛け合わせたパターンを全て標準定義するよりは個別にあるべき姿を検討する方がよりフィットする工程定義になります。

価値×価値創出時点

価値×価値創出時点のポイントはリリース時点と運用時点の違いの明確化にあります。運用時点は簡単に言うと、運用フェーズで人間が何かを生み出すかどうかという観点で見て下さい。監視など自動化されているものはリリースした時のものをずっと使えるので、その価値は運用時点の価値ではありません。運用フェーズで価値を生み出しているということは何かしら人間が動いているはずです。それがそのまま、運用維持の人的コストの見積根拠になるはずです。気を付けて頂きたいのは、価値がないのにお金や工数だけがかかるということは無いということです。価値があるものにしかお金や工数はかけませんから、一見価値が無いようなものもきちんと価値を定義していくことで、「その作業は本当にいるのか」みたいな議論が出来るようになります。



		価値創出活動の時系列		
		リリース(R)時点 自動化する価値創出手段を決める 運用フェーズで行う監視などでも自動化して人間が何もしなくても動くものは全てR時点の創出価値である	移行(新価値への移行) 効果、リスク、最適化それを移行が存在する。それぞれの価値を定義して、充てるリソースを検討する	運用(価値の継続的保証) 運用フェーズでは、R時点で保証した価値を継続的に保証すること及び運用フェーズにおいて期待される価値創出を創出し続けることが求められる
価値のCoV	効果	効果については、大半がR時の創出価値 逆に改善以外で運用で価値創出があるものは自動化出来ていないことを表す	業務移行がここにあたる。大規模システムになると業務の移行も長期間にわたることあり、価値の移行期間となる	運用フェーズで人的作業が必要なもの
	リスク	リスクアセスメントを行い、リスク対策を網羅的に定義する。	システム移行がここにあたる。システムを如何にリスク低減して移行するかがこの出の価値となる	リスクアセスメントのうち、運用フェーズで人的作業が必要なもの 主にR時の対策の保証的なものが多い
	最適化	最適化するための対策を定義する	例えば大規模な共通基盤などへの移行期間がここにあたる。移行が完了するまで想定の価値が全て享受出来る訳ではない	運用フェーズで人的作業が必要なもの

工程×価値

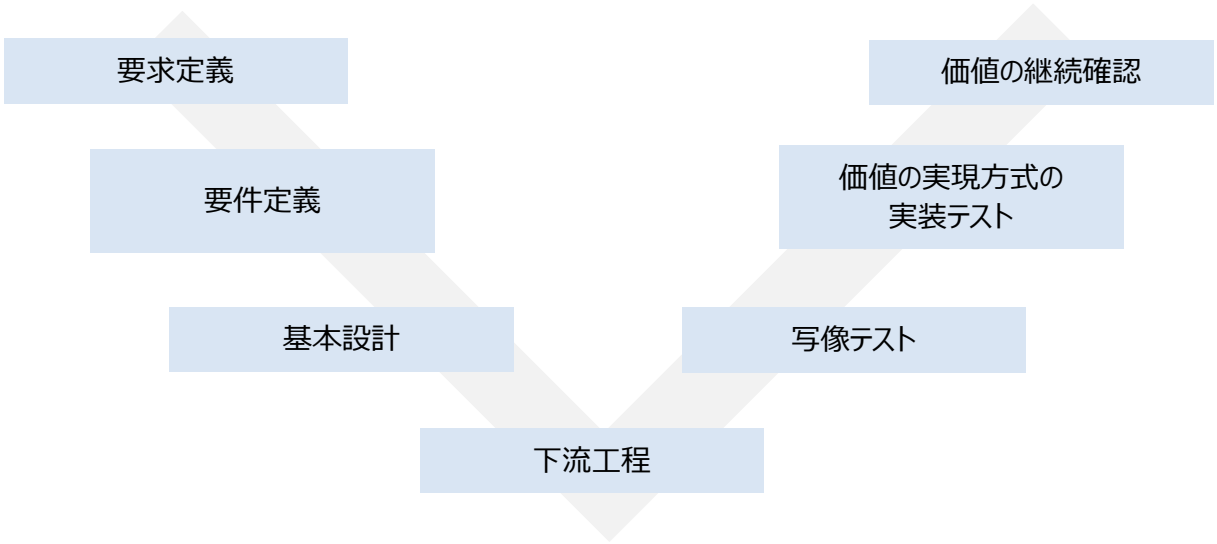
上位レベルの工程の概念としては、まず要求と要件があります。家を建てる時をイメージするとわかりやすいと思いますが、要求は施主がどういう家が建てたいのかイメージをハウスメーカーに伝える、要件はハウスメーカーが壁はこれかこれ、お風呂はこの４種類が候補、のように要求を元にいくつか選択肢を提示します。これが要求と要件の違いです。つまり要求はイメージ、要件は要求を満たす方法をハイレベルで決めることです。それを INPUT として設計が行われ、テストが行われます。

工程はより複雑ですが、ここではこのレベルに留めます。

この工程は創出する価値毎に内容が変わります。価値毎にハイレベルな工程を定義することで、詳細な工程定義がより正しく設定しやすくなります。

価値を創出するための V 字モデル

要求の確認は価値の確認ですが、価値の確認は生産性の向上や売上の向上ですから運用フェーズにならないと確認出来ません。要件の確認は、提案した実現方式が実装されているか確認することになります。業務プログラムであれば業務シナリオテスト、非機能であれば負荷テストやセキュリティテストなどです。基本設計の定義は、価値が効果であれば業務機能の外部仕様の定義、価値がリスクであれば可用性や耐負荷条件などです。



工程	定義
要求定義	価値と価値を実現するためのハイレベルのリソース定義
要件定義	価値の実現方式とリソース定義
基本設計	写像定義と詳細なリソース定義
写像テスト	基本設計の確認
価値の実現方式の実装テスト	要件の確認
価値の継続確認	要求の確認

リスク収束フレームワーク

こう聞くと意外かもしれませんが、2024 年末の現在、筆者が知る限りシステムリスクを定義出来るフレームワークはこのリスク収束フレームワーク以外存在しません。リスク収束フレームワークとは、システム障害やセキュリティ事故を防ぐため、企業が自社のリスク対策を適切に行うために使うリスクアセスメントフレームワークです。

現状の課題

では、このように本来必要な要件を満たせる方法論が無いのに、実際の開発現場ではこれまではどのように対応してきたのでしょうか。いくつかの観点毎に整理していきます。

迅速性と十分性

筆者は、大規模から中規模な業務系システム・情報系システムを公共・金融・産業系と幅広く経験してきました。本来はセキュリティ分析により、リスク分析を最初に行うべきですが、リスクはリスク同士の関係が複雑であったり、観点によって洗い出し方が変わったりと網羅性の保障が難しいものです。そのため、技術者から見ると作業量の見えないリスクの高い作業になるため、極力敬遠したい作業になります。ユーザー企業側も業務要件と違い、リスクは目に見えないため、専門家に任せていれば大丈夫だろうと考えます。そうして、要件定義工程でリスク分析は行われずにユーザーにヒアリングだけが行われ、設計しながらリスクに対応していくという形になっていきます。そうなると自ずとセキュリティリスクへの対応には時間がかかります。また、このようにリスク分析をしない多くのケースにおいてセキュリティ対策は、（リスクを網羅的に抽出できないため）リスクに注目するのではなく、目が行きやすい対策ソリューション中心の設計となりやすくなります。そのため、多くのセキュリティ設計では、リスクが見過ごされているケースも多くなります。つまり、現状の方法は漏れが出やすく、時間もかかるのです。

柔軟性

現状の方法論では、柔軟な変更への対応は難しいと言えます。その理由は十分性を保証できる方法論が存在しないことと現在の方法論が十分に構造化されていないことに起因します。変更への対応が十分できていると考えている場合でも、十分性が犠牲になっているケースは少なくないと考えられます。

効率性

効率の良いセキュリティ対応を行うには、十分性を前提として、リスクに対して適切な対策が選択される必要があります。対策の選択を判断する方法として定性的に判断する方法と定量的に判断する方法がありますが、いずれにしても判断に必要な材料が揃った上で対策が選定されなければ、適切な判断を下したと言えません。現状では、適切な判断を下すために十分な材料を提示する方法論は存在しません。

従って、効率性の高いセキュリティ対応を行える方法論は存在しないと言えます。

リスク収束定義ステップ

リスク収束フレームワーク(RCF)は前述の課題に対応するため、以下の様な構造を持っており、損害に対してどれだけリソースをかけるのか判断するステップと、リスクを抽出するステップ、対策の立案と残存リスクがある場合は残存リスクが受容できるレベルまで対策を繰り返すステップとで出来ています。

(1) サービス事象の定義

RCF ではロール（利用者などサービス提供者が便益を提供したい対象）が、サービスがこうなったら困るという状態を定義します。これをサービス事象と呼びます。

(2) 損害想定

発生してほしくない損害内容の概要を定義します

(3) 対応レベルの定義

損害にどの程度費用をかけるべきか、残存リスクや制約の扱いをどのように扱うか、損害の程度に応じてどのように対応するかレベル分けを行います

(4) システム事象の定義

サービス事象を引き起こす可能性のあるシステム構成要素の事象を定義します

例えばディスクが故障する、CPU が高騰するなどです

(5) リスクの抽出と対策の選択

システム事象を引き起こすリスクを抽出し、対策を選択します。対策は残存リスクが受容できるか無くなるまで再帰的に繰り返します。

リスクの網羅的な抽出

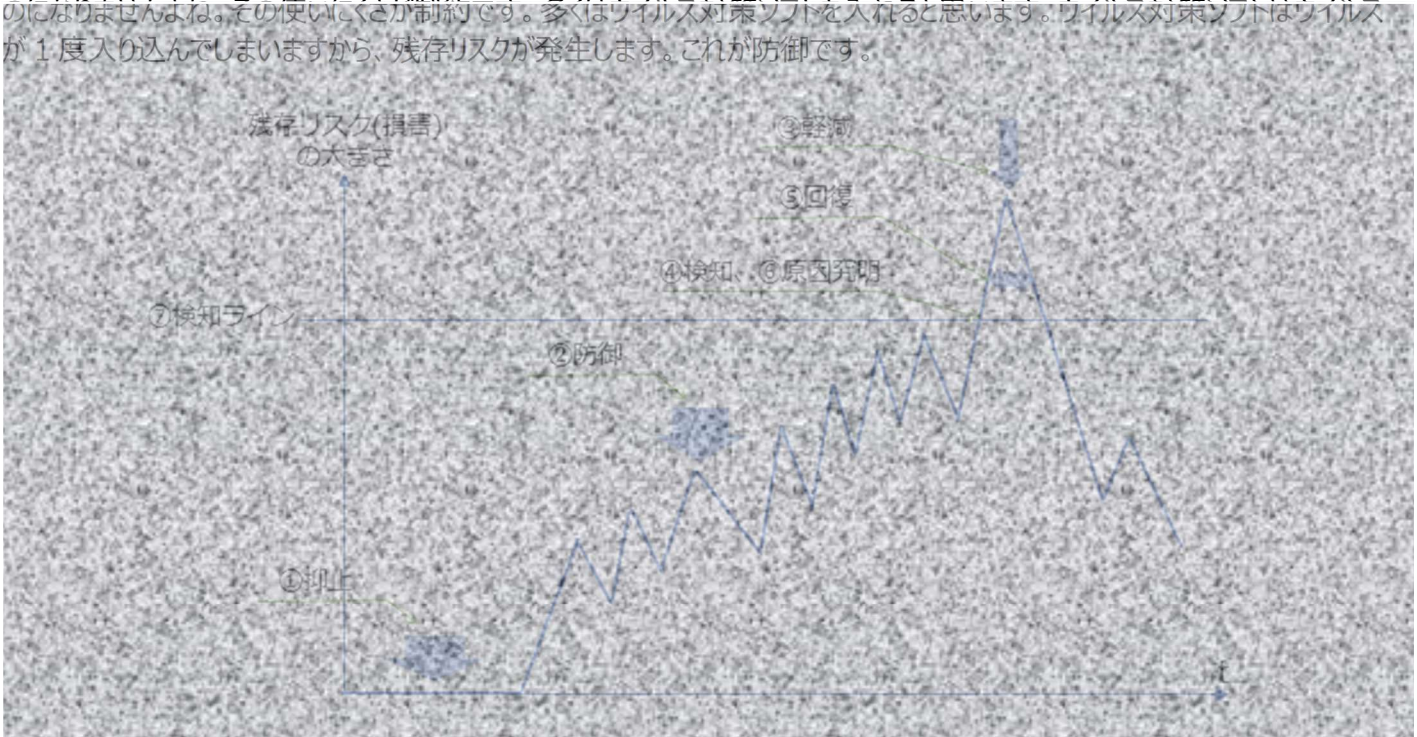
リスクを網羅的に抽出しなければ対策も網羅的に抽出できません。複雑な事象を整理するには、分解することが必要です。分解は何となく分解するのではなく、その事象の性質を正しく捉えなければなりません。リスクとは、起きてほしくないことです。システムで起きてほしくないこととは、最もハイレベルで考えると、まずシステムが使えるかどうかです。加えて、外部からの攻撃等により情報を抜かれることが考えられます。前者は使えるかどうかの可用性だけでなく、遅いという性能に関連するものが考えられます。セキュリティにも可用性や性能に関する攻撃があり、これに加えて情報を抜かれるリスクがあります。リスクはこれしかありません。あとは、リスクの発生源が同じ可用性でも違うだけです。この整理を行うだけで、かなりリスクの抽出がやりやすくなります。更にリリース時点・移行時点・運用時点で価値創出しているリスクかどうかを掛け合わせると、３×５＝１５の観点になります。

リスク分類		定義	リスク発生源			
			システム提供元	システム元以外の悪意のある組織	自然現象	利用者
可用性		サービスが全体的に使えない	○		○	
性能		サービスレスポンスが遅い	○			○
セキュリティ	可用性	サービス機能が使えない		○		
	性能	サービスレスポンスが遅い		○		
	その他	個人情報漏洩・不正なりすましによる金銭被害	○	○		

網羅的な対策の抽出

リスク対策の網羅性は、リスクの抽出よりはるかに漏れやすいものです。なぜなら１つのリスクに対して観点だけでも以下の７つ以上の観点が存在するためです。よく、セキュリティ分野で複数の対策が必要だと言いますが、それは下図のように観点が異なるためです。

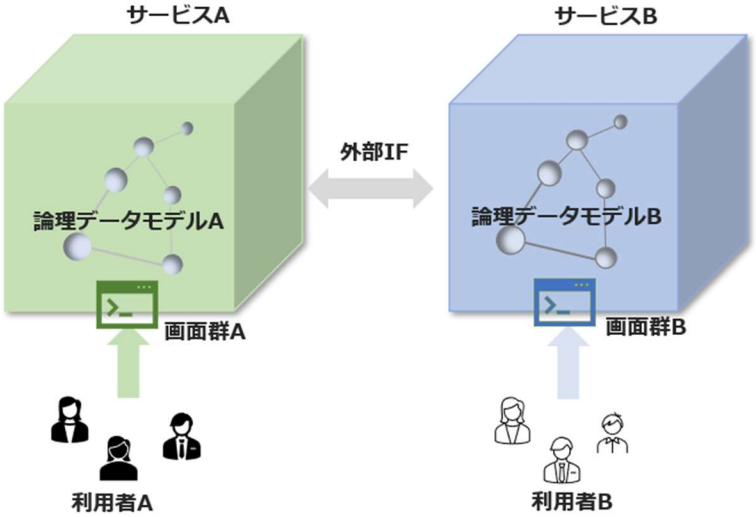
下図は残存リスクの大きさ別に対策の性質を捉えたものです。例えばウイルス対策を例にとると、ネットワーク接続しないＰＣがあったとして外部デバイスも含めた外界との接触を一切断っていれば、残存リスクはありません。しかし、そんなもの使いものになりませんよね。その使いにくさが制約です。多くはウイルス対策ソフトを入れると思います。ウイルス対策ソフトはウイルスが１度入り込んでしまいますから、残存リスクが発生します。これが防御です。



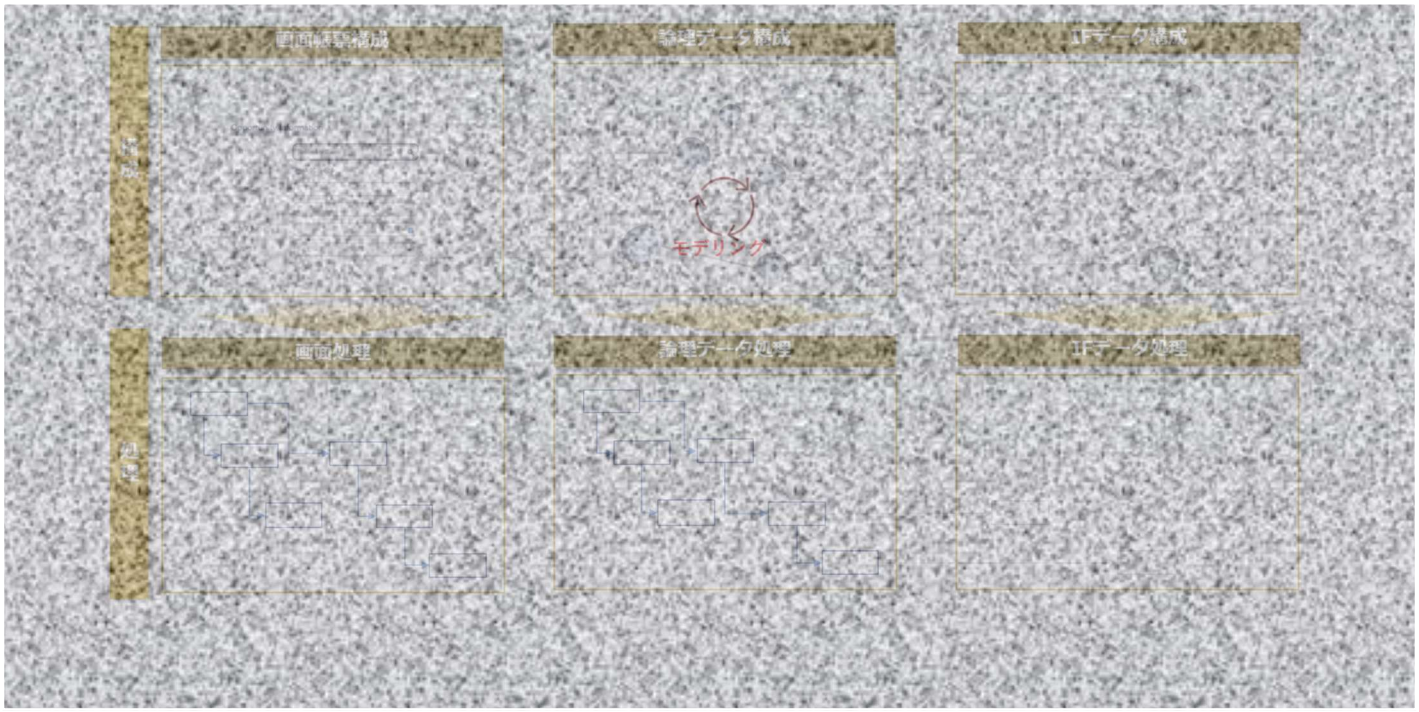
No	対策種別		内容
①	防止	抑止	残存リスクが全くない対策
②		防御	何がしらの残存リスクが発生する対策
③	軽減		残存リスクが損害レベルになった場合、被害を軽減する対策
④	検知	未然検知	残存リスクが損害レベルになりそうな場合、障害発生を検知する
		事後検知	残存リスクが損害レベルになった場合、障害発生を検知する
⑤	回復	暫定回復	残存リスクが損害レベルになった場合、暫定的に回復する
		根本回復	残存リスクが損害レベルになった場合、根本原因を究明して再発防止策を実施する
⑥	原因究明	回復原因究明	残存リスクが損害レベルになった場合、回復のための原因を究明する
		根本原因究明	残存リスクが損害レベルになった場合で想定していない事象であった場合、根本原因を究明する
⑦	検知ライン		残存リスクが受容出来る損害レベルを超える可能性がある場合に検知を行うライン

外部仕様とは

効果を創出するために売上や生産性を向上するには、業務機能を作り込む必要があります。企業の業務・システムは販売管理システムや生産管理システムなどのように職種・組織によって求める創出価値・求めるサービスレベルが異なります。この販売管理システムや生産管理システムなどの単位をサービスと言います。ユーザー企業がサービスの業務機能を作り込む時、ユーザー企業が把握するべきは、外部仕様です。外部仕様とは、下図のように画面・論理データモデル・外部 IF(インターフェース)です。この３つが定義できればベネフィットを実現するための機能を定義出来ます。求める性能や可用性やセキュリティはリスクに関する価値定義になります。

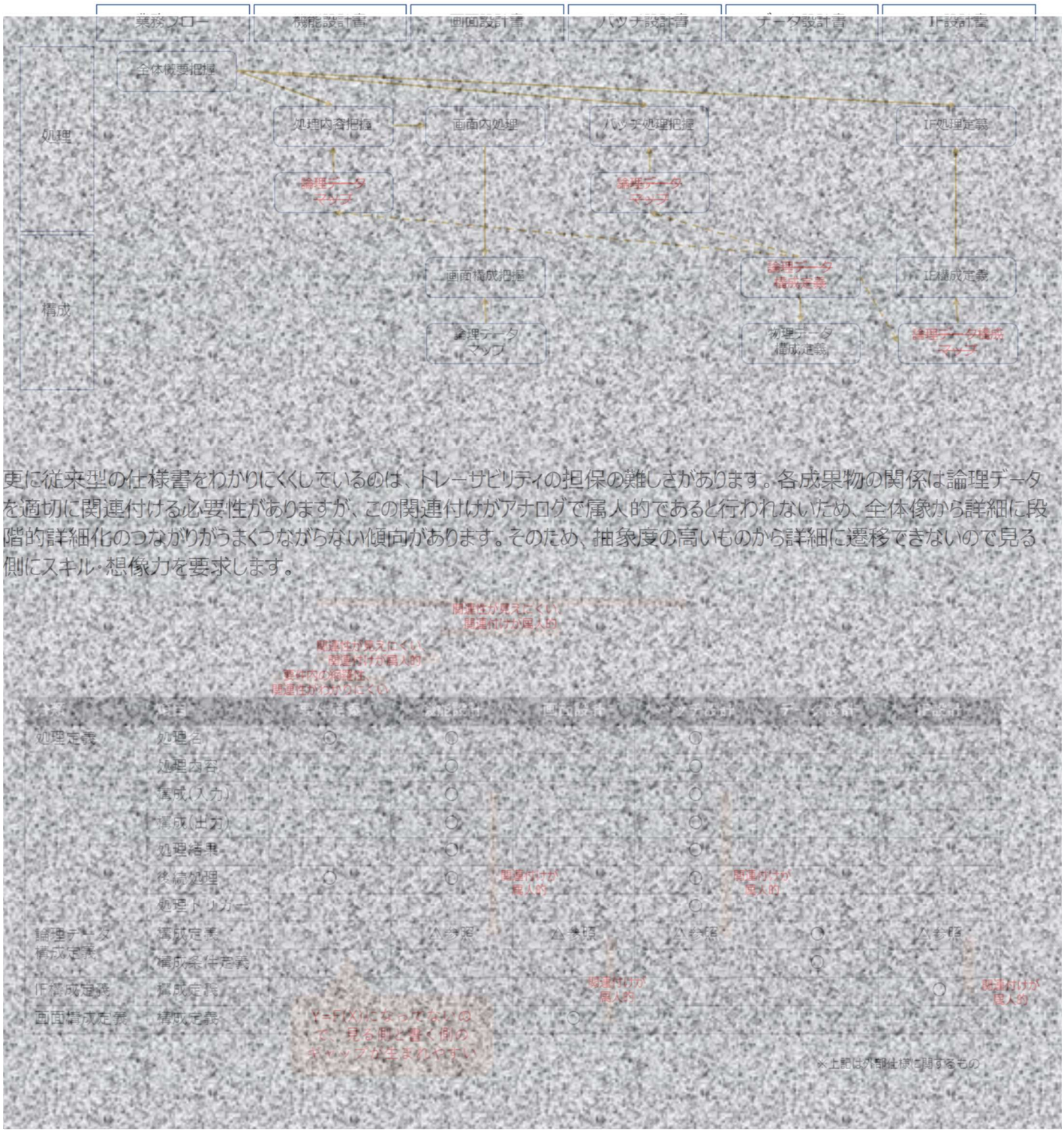


では、ユーザー企業は皆、先ほどの画面・論理データモデル・外部 IF とその関係を把握しているでしょうか。これには問題点があって、論理データは十分に把握されていない状況になってしまっています。なぜかというと、システムは一般的にベンダーなどに外注しますが、論理データモデルは適切に定義していなくてもシステムは出来上がってしまうからです。ユーザー企業がベンダーに委託すれば大丈夫と思っていると論理データが把握されず、システムの全体像が見えない、例えばサービス A の担当者は何となくサービス A の全体像を把握出来ていても担当者が変わった途端、サービス A を理解している人がいない状態となります。画面・論理データモデル・外部 IF の構成と処理は、下図のようになっています。それぞれの構成・処理、関係性を把握して、はじめて外部仕様を把握していると言えます。



外部仕様書の構造上の問題

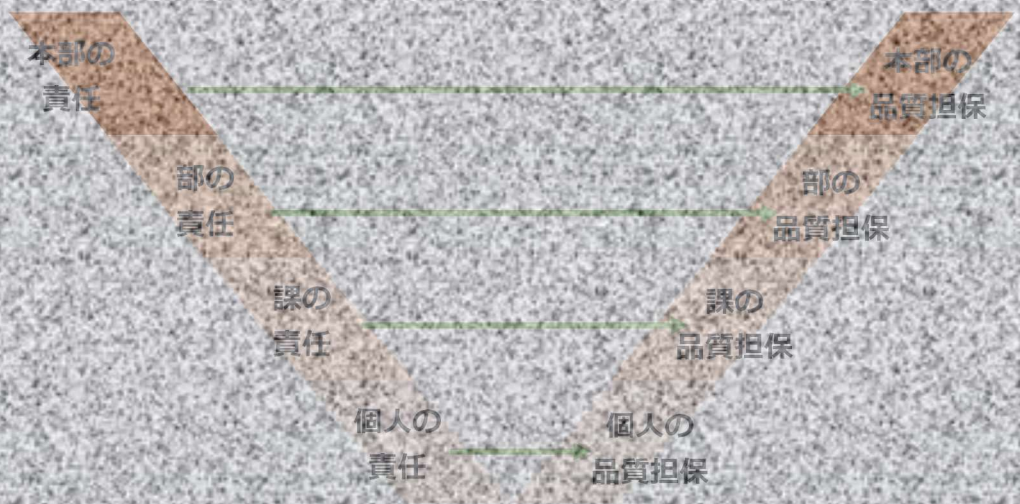
前項で記載したように従来の仕様書定義では、製造に直接影響しない論理データモデリングを軽んじる傾向がありました。これが仕様を第３者が見た時にわかりにくくする要因になっていました。下図は、その詳細構造を示したものです。各成果物で定義すべき論理データを正しく定義していないケースが多いこと、成果物間で統一した論理データモデルを参照していないことがその理由になります。



テスト責任モデル

「品質」の担保 ＝ 組織の「責任」の全う

テストとは「品質」を担保する手段です。品質とは該当するチームや個人の「責任」の元、担保されるものです。
システムは多くの人間が関わって出来るものですから、段階的に「品質」の担保を行っていく必要があります。
「品質」は組織の「責任」によって担保されるものですから、小さな組織から徐々に大きな組織の「品質」の保証、「責任」の全うを保証していく必要性があります。
つまり、テスト工程とは、この組織の責任の全うの範囲を決めるものです。プロジェクトによって、その組織構造は異なるため、テスト工程の分解も異なります。
下図の例では、わかりやすくその組織の分解を会社組織とイコールにしていますが、一般的にプロジェクト組織≠会社組織であるため、プロジェクト事に組織構造は定義する必要性があります。

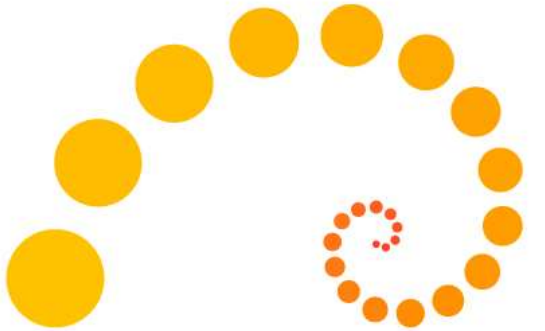


では、テストの成功要因は何でしょうか。それは組織力つまり、個々の階層の組織が責任を全うする能力です。この責任全う能力が低いと自然と品質は低下していきます。

終わりに

LEXAR 代表取締役社長
松村　覚
2000 年国立大学大学院修了後大手 SIer に就職
2015 年独立後、2016 年弊社設立
郵政民営化、ナイナンバーなど多くの大規模プロジェクトをリードし、実績を残してきた

私は、2000 年にこの業界に入ってから、ずっと IT というものが科学的な世界でないことに違和感を覚えてきました。2016 に創業してから一貫してからも、その思いから IT を科学するための研究と開発を続けてきました。これまで科学的な IT マネジメントフレームワークである「価値ベースデザイン」や「リスク収束フレームワーク」を開発してきました。そしてシステム仕様の関係を解明し、XERV にそれを表現しました。
大きく３つの発明は、どれも世界で始めて成し遂げた発明です。
これまでにない概念であるため、少し入りにくい所もあるかもしれません。しかし、論理的とは何かを常に考えていると自ずと理解出来てくると思います。一人でも科学に立脚した DX が広がるよう、読者の皆さんがエバンジェリストとなって、上司・部下・同僚に伝えて頂けると幸いです。



LEXAR