 DAX40-06-1 「知の共有化」システムの標準的な構築プロセス

# はじめに

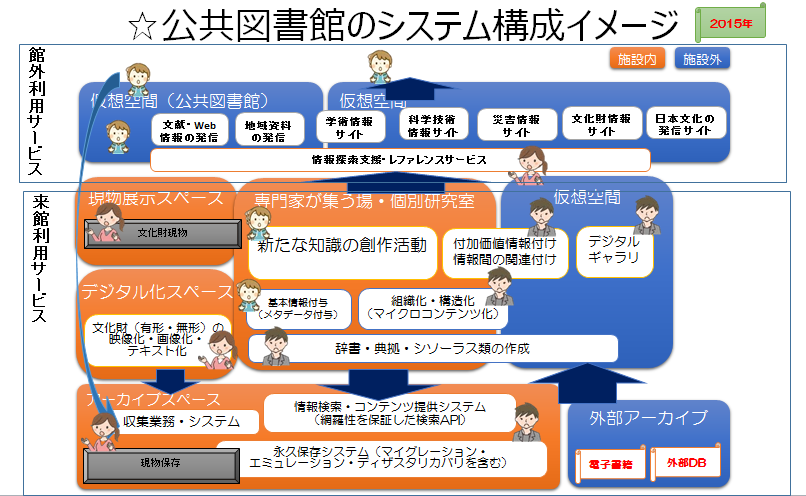
## 概要

## 改版履歴

### 2020年3月12日 DAX40-01の詳細を分冊化

# 現在の公共図書館等のシステム構成イメージ

## Subtopic



# 概要

## サービスの高度化のために、 システム構築・運用に当たっては、 次世代技術を活用することが望ましい

### AIと人間の能力と役割の一般論として、 AIが実用段階に達した今、 今まで人が担ってきた部分の作業も、 精密化するとAIを活用したほうが効率的なことが多々ある。

## システム構築・運用に当たっては、 必要なスキル・知識を持った人材の確保が必要である

### アーカイブ機関において、 システムの調達・構築・運用のスキル・知識を持った人材が 不足しているのが現状

### 構築のタスクに必要なスキル・知識を 選択的に習得することが効果的である

### 参考例の１つとして「iコンピテンシ・ディクショナリ」

## 適正な調達を行うために、 開発タスクとドキュメントをひな形に進めることが効果的である。

### 参考例の１つとして「政府機関での調達の標準ガイドライン」

### その前提となっているシステム開発標準が「共通フレームワーク」

# システム開発標準としての「共通フレーム2013」

## 共通フレームとは

### ソフトウェアの構想から開発、 運用、 保守、 廃棄に至るまでのライフサイクルを通じて必要な作業項目、 役割等を包括的に規定した共通の枠組み。

### 何を実施するべきかが記述されている、 「ＩＴシステム開発の作業規定」である。

### 目的は、 ソフトウェア開発に関係する人々（利害関係者）が、 「同じ言葉で話す」ことが出来るようにするため。

### ウォーターフォール、 スパイラル、 プロトタイプ、 アジャイル系すべての開発方法論に共通したもの。

## 共通フレームの特徴

### （１）超上流の重視

### （２）モジュール性の採用

### （３）責任の明確化

### （４）責任範囲の明確化

### （５）工程、 時間からの独立性

### （６）開発モデル、 技法、 ツールからの独立性

### （７）ソフトウェアを中心としたシステム関連作業までを包含

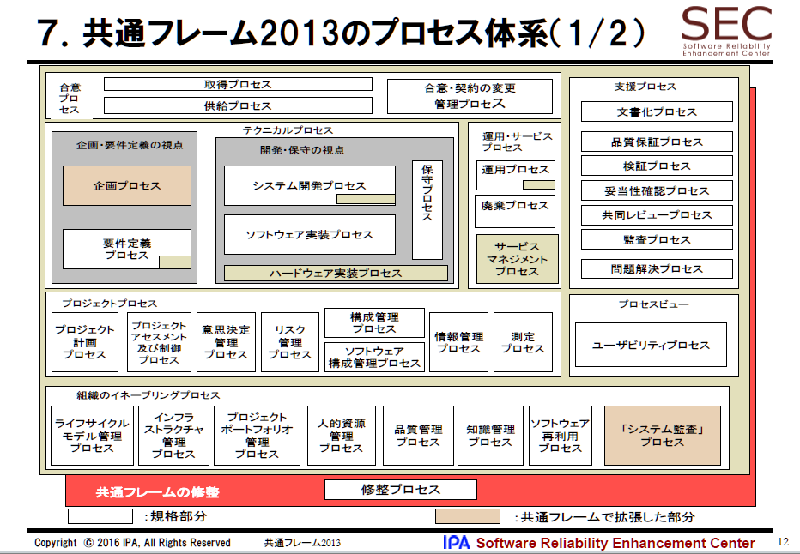
### （８）システムライフサイクルプロセスとの整合性

### （９）文書の種類、 書式を規定しない

### （１０）修整（テーラリング）の採用

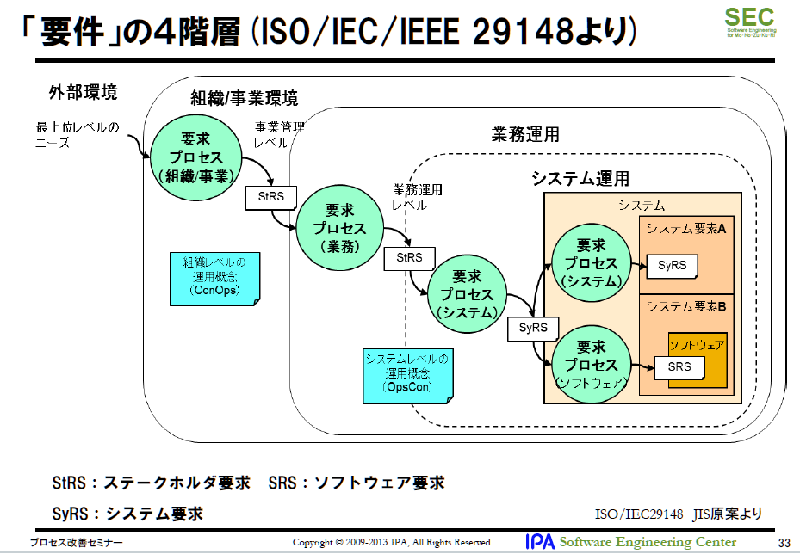
## 共通フレームのプロセス体系

### Subtopic



## 「要件」の4階層

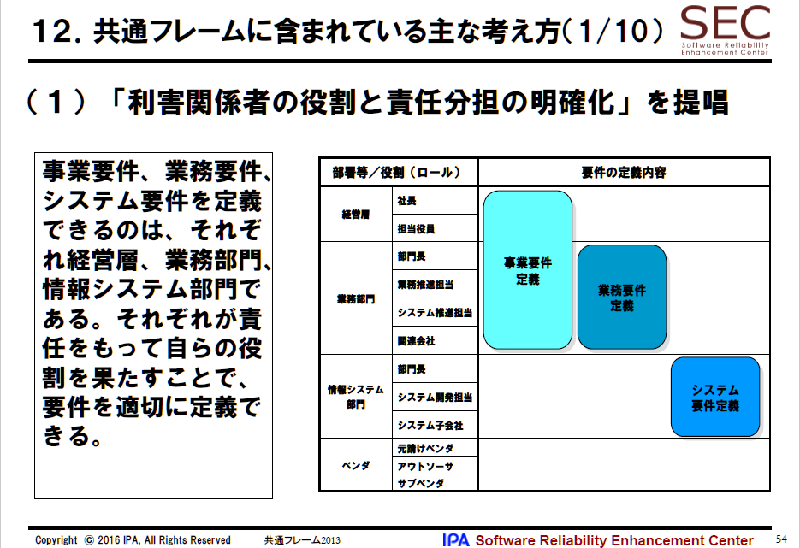
### Subtopic



## 共通フレームに含まれている主な考え方

### （１）「利害関係者の役割と責任分担の明確化」を提唱

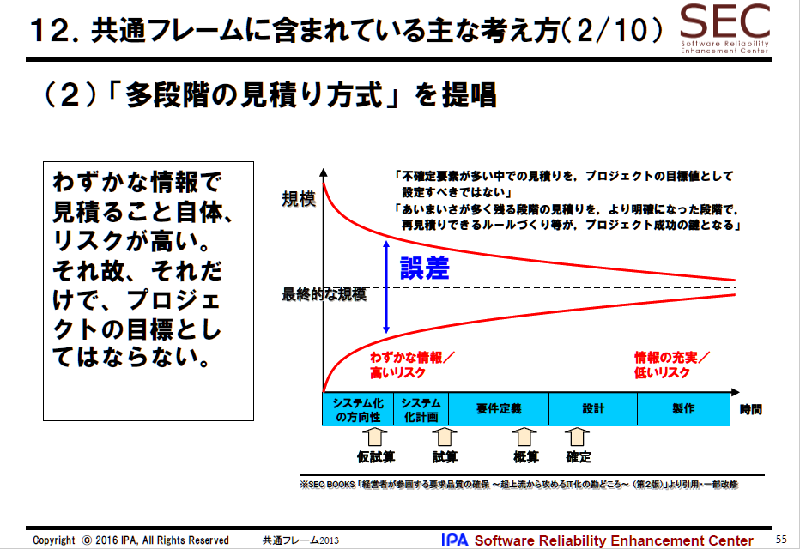
* + - 1. Subtopic



* + - 1. 事業要件、 業務要件、 システム要件を定義できるのは、 それぞれ経営層、 業務部門、 情報システム部門である。
      2. それぞれが責任をもって自らの役割を果たすことで、 要件を適切に定義できる。

### （２）「多段階の見積り方式」を提唱

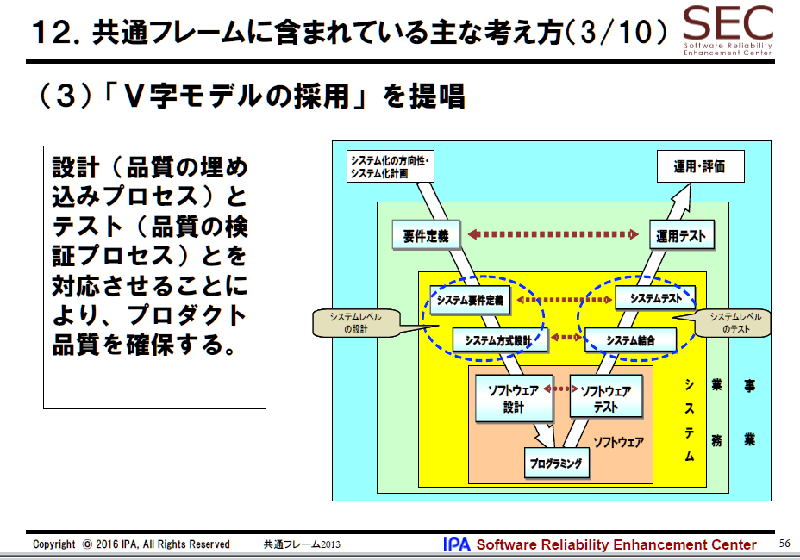
* + - 1. Subtopic



* + - 1. わずかな情報で見積ること自体、 リスクが高い。それ故、 それだけで、 プロジェクトの目標としてはならない。

### （３）「Ｖ字モデルの採用」を提唱

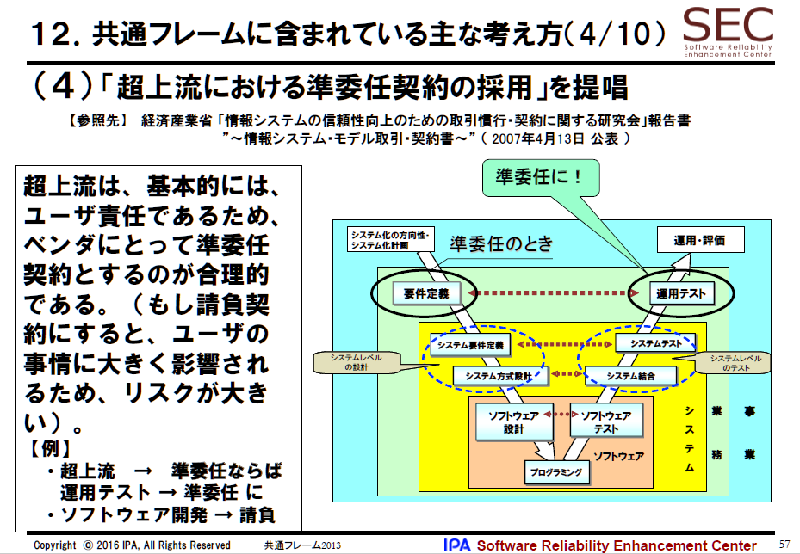
* + - 1. Subtopic



* + - 1. 設計（品質の埋め込みプロセス）とテスト（品質の検証プロセス）とを対応させることにより、 プロダクト品質を確保する。

### （４） 「超上流における準委任契約の採用」を提唱

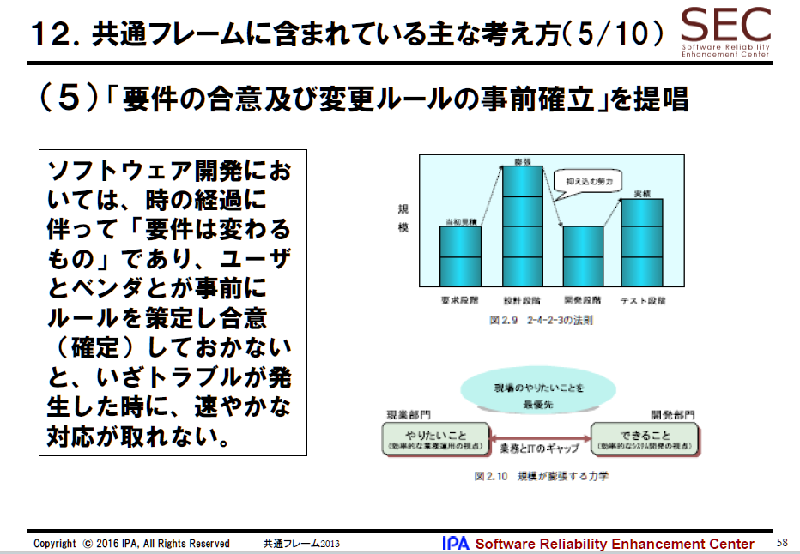
* + - 1. Subtopic



* + - 1. 超上流は、 基本的には、 ユーザ責任であるため、 ベンダにとって準委任契約とするのが合理的である。（もし請負契約にすると、 ユーザの事情に大きく影響されるため、リスクが大きい）。
      2. 【例】
         * ・超上流→ 準委任ならば運用テスト→ 準委任に
         * ・ソフトウェア開発→ 請負

### （５） 「要件の合意及び変更ルールの事前確立」を提唱

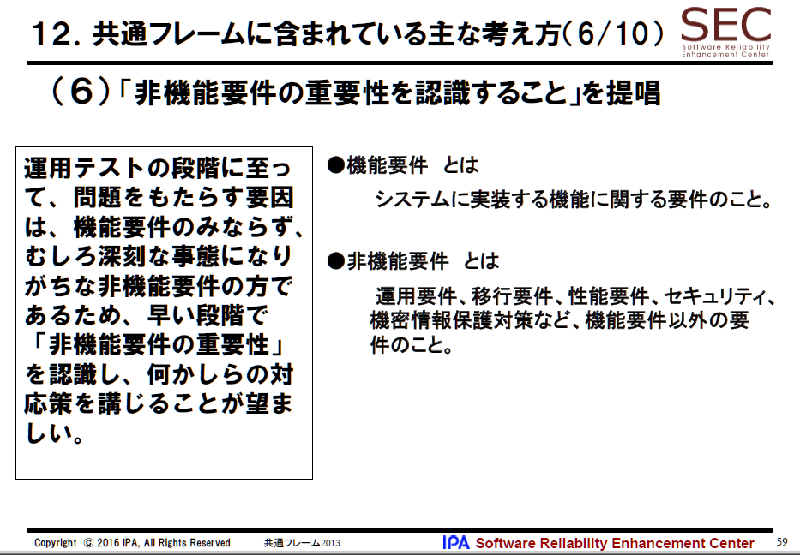
* + - 1. Subtopic



* + - 1. ソフトウェア開発においては、 時の経過に伴って「要件は変わるもの」であり、 ユーザとベンダとが事前にルールを策定し合意（確定）しておかないと、 いざトラブルが発生した時に、 速やかな対応が取れない。

### （６）「非機能要件の重要性を認識すること」を提唱

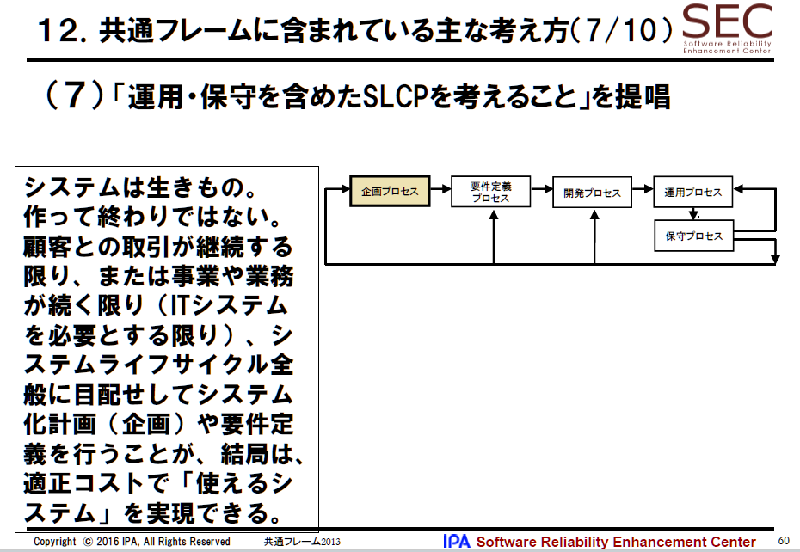
* + - 1. Subtopic



* + - 1. ●機能要件とは
         * システムに実装する機能に関する要件のこと。
      2. ●非機能要件とは
         * 運用要件、 移行要件、 性能要件、 セキュリティ、 機密情報保護対策など、 機能要件以外の要件のこと。
      3. 運用テストの段階に至って、 問題をもたらす要因は、 機能要件のみならず、 むしろ深刻な事態になりがちな非機能要件の方であるため、 早い段階で「非機能要件の重要性」を認識し、 何かしらの対応策を講じることが望ましい。

### （７） 「運用・保守を含めたSLCPを考えること」を提唱

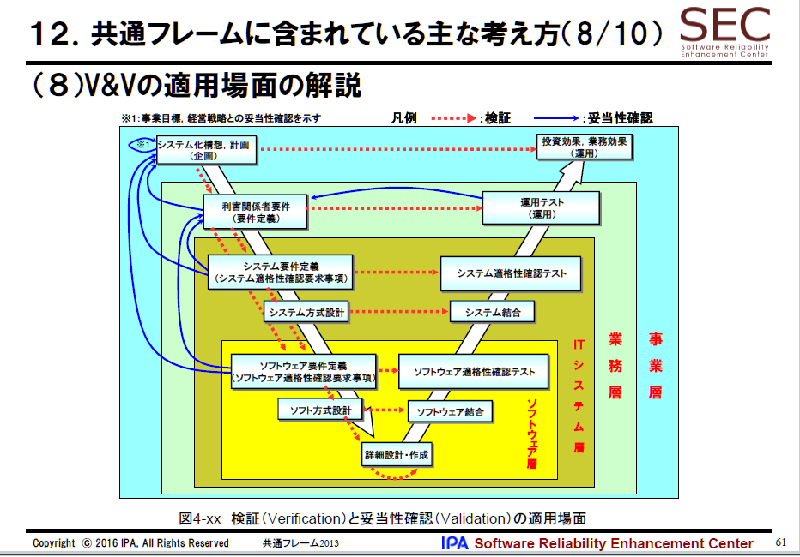
* + - 1. Subtopic



* + - 1. システムは生きもの。作って終わりではない。顧客との取引が継続する限り、 または事業や業務が続く限り（ITシステムを必要とする限り）、 システムライフサイクル全般に目配せしてシステム化計画（企画）や要件定義を行うことが、 結局は、適正コストで「使えるシステム」を実現できる。

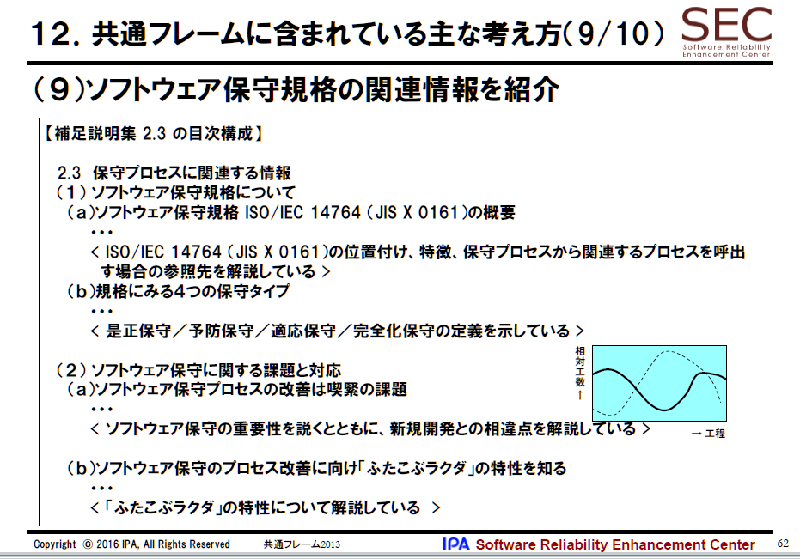
### （８）V&Vの適用場面の解説

* + - 1. Subtopic



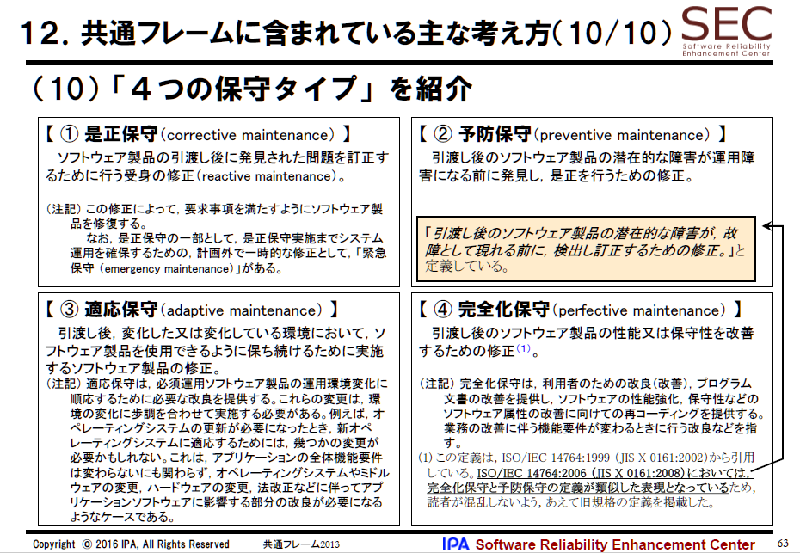
### （９）ソフトウェア保守規格の関連情報を紹介

* + - 1. Subtopic



### （10）「４つの保守タイプ」を紹介

* + - 1. Subtopic



* + - 1. 【①是正保守（corrective maintenance）】
         * ソフトウェア製品の引渡し後に発見された問題を訂正するために行う受身の修正（reactive maintenance）。
         * （注記）この修正によって，要求事項を満たすようにソフトウェア製品を修復する。 なお，是正保守の一部として，是正保守実施までシステム運用を確保するための，計画外で一時的な修正として，「緊急保守（emergency maintenance）」がある。
      2. 【②予防保守（preventive maintenance）】
         * 引渡し後のソフトウェア製品の潜在的な障害が運用障害になる前に発見し，是正を行うための修正。
         * 「引渡し後のソフトウェア製品の潜在的な障害が，故障として現れる前に， 検出し訂正するための修正。」と定義している。
      3. 【③適応保守（adaptive maintenance）】
         * 引渡し後，変化した又は変化している環境において， ソフトウェア製品を使用できるように保ち続けるために 実施するソフトウェア製品の修正。
         * （注記）適応保守は，必須運用ソフトウェア製品の運用環境変化に順応するために必要な改良を提供する。 これらの変更は，環境の変化に歩調を合わせて実施する必要がある。 例えば，オペレーティングシステムの更新が必要になったとき， 新オペレーティングシステムに適応するためには， 幾つかの変更が必要かもしれない。 これは，アプリケーションの全体機能要件は変わらないにも関わらず， オペレーティングシステムやミドルウェアの変更， ハードウェアの変更， 法改正などに伴って アプリケーションソフトウェアに影響する部分の改良が 必要になるようなケースである。
      4. 【④完全化保守（perfective maintenance）】
         * 引渡し後のソフトウェア製品の性能又は保守性を改善するための修正(1)。

(1) この定義は，ISO/IEC14764:1999 （JIS X 0161:2002）から引用している。 ISO/IEC14764:2006 （JIS X 0161:2008）においては， 完全化保守と予防保守の定義が類似した表現となっているため， 読者が混乱しないよう， あえて旧規格の定義を掲載した。

* + - * + （注記）完全化保守は， 利用者のための改良（改善）， プログラム文書の改善を提供し， ソフトウェアの性能強化， 保守性などのソフトウェア属性の改善に向けての再コーディングを提供する。 業務の改善に伴う機能要件が変わるときに行う改良などを指す。

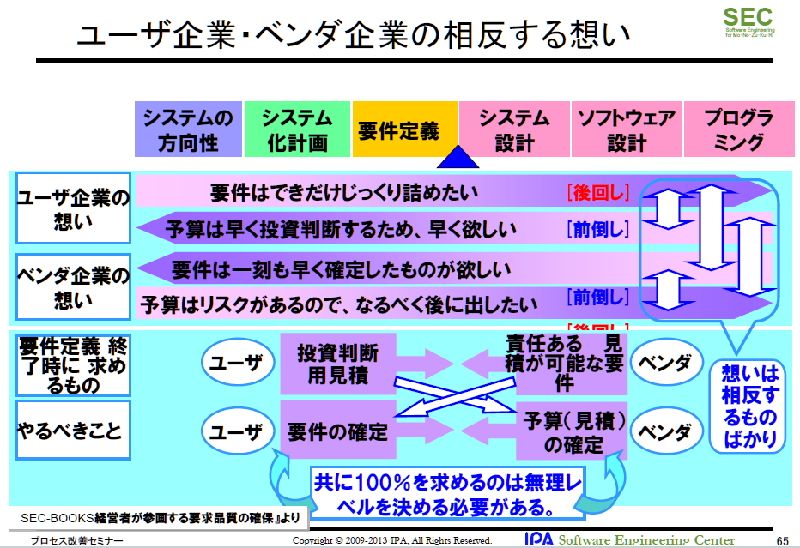
## 実務に活かすIT化の原理原則17ヶ条

### 原理原則１７ヶ条の構成

* + - 1. 基本的な考え方：
         * 原理原則を理解しやすくするため、 原理原則の基になる考え方を説明したもの
      2. 行動規範：
         * 原理原則の基づいて、 受注者・発注者のそれぞれが具体的にどのように行動すべきか示したもの
      3. 原理原則条項：
         * 原理原則は「超上流」において必要とされる事柄を、 格言のように短く表現したもの
         * 原理原則【１】 ユーザとベンダの想いは相反する
         * 原理原則【２】 取り決めは合意と承認によって成り立つ
         * 原理原則【３】 プロジェクトの成否を左右する要件確定の先送りは厳禁である
         * 原理原則【４】 ステークホルダ間の合意を得ないまま、 次工程に入らない
         * 原理原則【５】 多段階の見積りは双方のリスクを低減する
         * 原理原則【６】 システム化実現の費用はソフトウェア開発だけではない
         * 原理原則【７】 ライフサイクルコストを重視する
         * 原理原則【８】 システム化方針・狙いの周知徹底が成功の鍵となる
         * 原理原則【９】 要件定義は発注者の責任である
         * 原理原則【10】 要件定義書はバイブルであり、 事あらばここへ立ち返るもの
         * 原理原則【11】 優れた要件定義書とはシステム開発を精緻にあらわしたもの
         * 原理原則【12】 表現されない要件はシステムとして実現されない
         * 原理原則【13】 数値化されない要件は人によって基準が異なる
         * 原理原則【14】 「今と同じ」という要件定義はありえない
         * 原理原則【15】 要件定義は「使える」業務システムを定義すること
         * 原理原則【16】 機能要求は膨張する。コスト、 納期が抑制する
         * 原理原則【17】 要件定義は説明責任を伴う

### 原理原則【１】 ユーザとベンダの想いは相反する

* + - 1. Subtopic



* + - 1. ITシステムの企画・開発の現場では、 ユーザ企業とベンダ企業の相反する想いがあります。 例えば、 ユーザ企業は、 要件はできるだけじっくり詰めたいし、 予算は早期の投資判断を求められるので、 最終費用を早く確定してほしいとの想いがあります。他方のベンダ企業の想いはまったくその逆です。 これがお互いにとってそもそもの不幸の始まりとなります
      2. 開発規模（工数）に見合った、 最低限の工期を確保できなければ顧客満足を満たす開発はできません。 受注者には開発規模に見合った工期を主張することが 求められます。

### 原理原則【２】 取り決めは合意と承認によって成り立つ

* + - 1. 証拠のない口約束のように、 決まったと了解していることが、 それ以降の都合で無責任に変更となり、 残念な思いをする、 ということはよくあります。
      2. 決め事は可能な限り文章に残し、 承認ルール（主体と方法）の確認をして、 信頼度を高めなければいけません。
      3. 承認は合意に基づいていることが必要です。

### 原理原則【３】 プロジェクトの成否を左右する要件確定の先送りは厳禁である

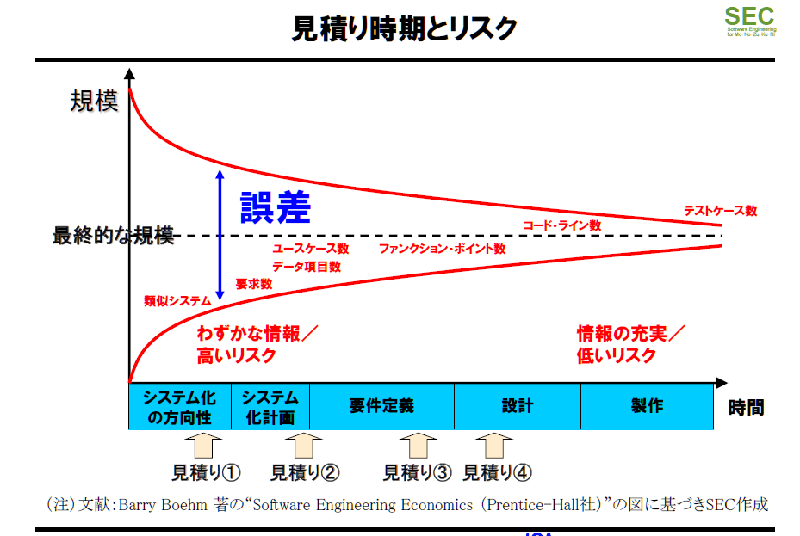
* + - 1. 要件定義は開発全体の成否を左右重要な工程です。 曖昧な要件のまま開発が始まると、 プロジェクトが失敗するリスクが大きくなります。
      2. 特に、 システムの出来を左右する要件に高いリスクを抱えたまま、 プロジェクトを進めることは危険です。 あせってベンダに開発を依頼しても、 先に進めず、 かえって時間・コストがムダになることもあります。
      3. 解決の目処が立つまでは、 先に進まない勇気も必要です。

### 原理原則【４】 ステークホルダ間の合意を得ないまま、 次工程に入らない

* + - 1. プロジェクトを起こした業務企画担当者は、 プロジェクト責任者として、 これらステークホルダの方針、 意見、 課題などについて、 漏れなく綿密に把捉し、 できることとできないことをIT担当者、 ベンダとともに切り分け、 業務要件として取りまとめていく責任を果たす必要があります。
      2. ステークホルダもまた、 システムの供給側に立つ場合は、 積極的にシステム開発要件の策定に参加し、 利用者ニーズを確実に把握して、 正確にシステム機能に反映していくことが必要です。

### 原理原則【５】 多段階の見積りは双方のリスクを低減する

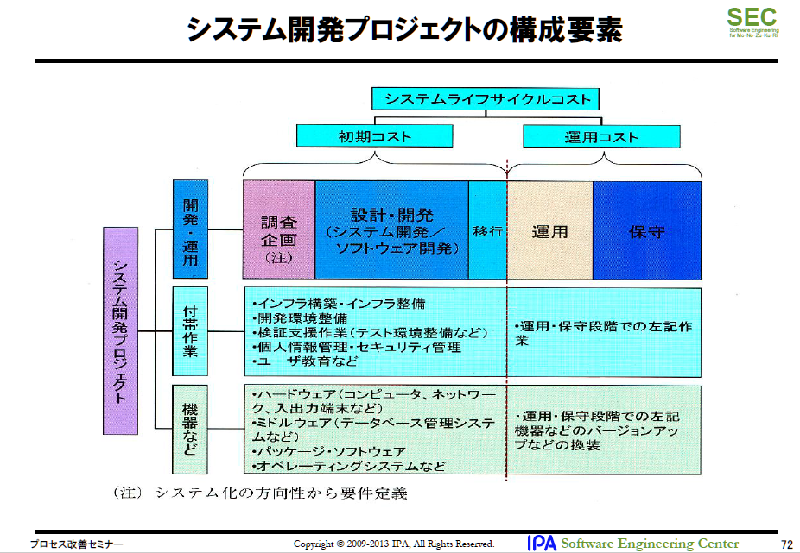
* + - 1. Subtopic



* + - 1. 不確定要素が多い中での見積りを プロジェクトの目標値として設定すべきではありません。
      2. あいまいさがある段階の見積りを、 はっきりした段階で見積り直せるルールづくりなどが プロジェクト成功の鍵となります。
      3. 要件の不確定さやプロジュクトの特性・リスクに応じて、 適切な契約方式（多段階契約、 インセンティブ付契約など）を選択することにより、 発注者・受注者の双方にメリットが生まれます。多段階とは、 受注先をその都度変えるということではなく、 固まり具合に応じて見積り精度をあげていこうということです

### 原理原則【６】 システム化実現の費用はソフトウェア開発だけではない

* + - 1. Subtopic



* + - 1. 見積り範囲がソフトウェア開発のことだけを指しているのか、 インフラ整備（システム基盤整備）などのような付帯作業も対象にしているかなど、 スコープを明確にしていくことが大切です。
      2. 発注者は、 何をお願いし、 何を自分で行うのか、 一方、 受注者は自分の提供する作業やサービスは どの範囲なのかをお互いに明確にしておくことが重要です。

### 原理原則【７】 ライフサイクルコストを重視する

* + - 1. 開発コスト、 運用・保守コストのバランスを考えなければなりません。 大切なことはライフサイクルコストを意識することです。
      2. 例えば、 運用性・保守性を高めるポイントとして以下があります。
         * －メンテナンスフリー
         * －拡張性の容易さ確保
         * －モニタリング・トレーサビリティの確保
         * －障害発生時の調査、 リカバリーが容易な設計
         * －OS・ハードウェアのバージョンアップ対応

### 原理原則【８】 システム化方針・狙いの周知徹底が成功の鍵となる

* + - 1. 超上流のフェーズで、 システム化の方針・狙いを浸透させておかないと、 各人が勝手気ままに要件を考えるため、 仕様の統一に時間がかかり、 最初の構築だけでなく、 その後の維持・保守においても費用と時間が増大することになります。
      2. システム化の目的はコンピュータやプログラムではなく、 事業目標を達成するための情報システムの構築なのです。

### 原理原則【９】 要件定義は発注者の責任である

* + - 1. 要件定義とは、 どのようなシステム、 何ができるシステムを作りたいのかを定義することです。 それはあくまでも発注者の仕事であり、 発注者の責任で行うものです。 要件定義があいまいであったり、 検討不足のまま、 受注者に開発を依頼した場合、 その結果として、 コスト増、 納期遅れ、 品質低下を発生させるおそれがあります。 その責任を受注者に負わせることはできません。
      2. 受注者が支援する場合であっても、 要件定義で作成した成果物に対する責任は発注者にあります。

### 原理原則【10】 要件定義書はバイブルであり、 事あらばここへ立ち返るもの

* + - 1. ベンダ企業を含むステークホルダ間の合意のベースとなるのは常に要件定義書です。 設計工程以降よりも、 むしろ、 要件定義の合意形成時点での吟味が重要です。 「決定先送り型」の要件定義では、 あいまいな海図に基づく航海のようなもので、 早晩プロジェクトが破綻します。
      2. ステークホルダ間の合意は、 名目的な合意ではなく、 実質的な合意であることが不可欠です。

### 原理原則【11】 優れた要件定義書とはシステム開発を精緻にあらわしたもの

* + - 1. 要件定義工程では、 業務要件を整理・把握し、 その実現のためのシステム機能要件をしっかり固めます。あわせて性能、 信頼性、 セキュリティ、 移行・運用方法などの非機能要件、 既存システム接続要件、 プロジェクト特有の制約条件も洗い出します。また、 将来の方針を見込んで稼働環境を定めることが大切です。流行に流されず、 ルールを定めることです。

### 原理原則【12】 表現されない要件はシステムとして実現されない

* + - 1. この原則は、 建築における施工主と工事業者の関係にあるように、 発注と受注における常識です。しかし、 情報システム開発においては往々にしてこの原則が成立しない場合があり、 「行間を読め」、 「言わなくても常識」、 「言った言わない」など表現されない要件が、 両者のトラブルの原因になります。

### 原理原則【13】 数値化されない要件は人によって基準が異なる

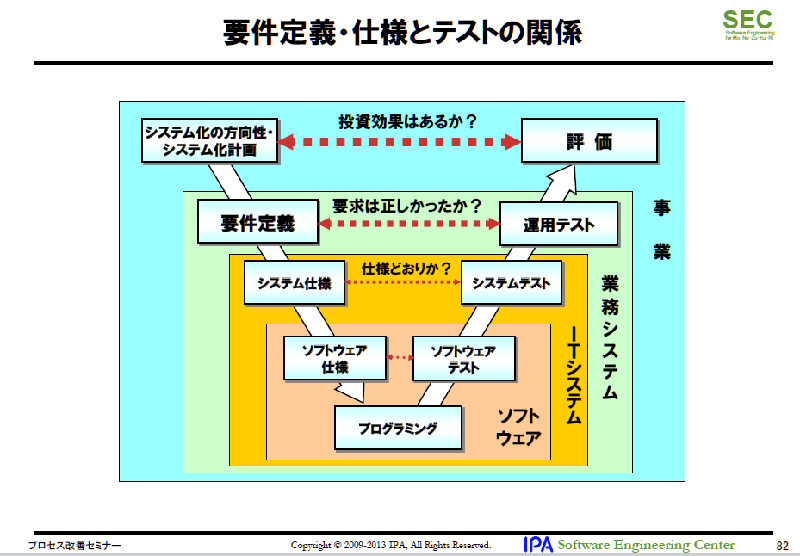
* + - 1. 要件定義では、 定量化できるものは、 極力、 数値化します。数えられないものは定義できません。「大きい、 小さい、 速い」だけでは、 人によって「ものさし」が異なります。
      2. 数値化されていても誤りはあります。例えば、 使用する単位が違えば結果は大きく変わります。単位まで含めて確認し、 決めなければなりません。

### 原理原則【14】 「今と同じ」という要件定義はありえない

* + - 1. 「今と同じ」でも要件定義は必要です。
      2. そもそも同じでよいなら再構築する必要はありません。 よくないから再構築するというところから発想したいものです。
      3. 現行システムの調査をする場合は、 システムの機能を洗い上げ、 新システムの実像を明確にするだけでは不十分です。 現行システムをどう使っているか、 という点から調査をしなければなりません。
      4. 「そもそも今の要件はどうなっているのか」を問い直し、 場合によっては具体的な要件にまで導くことも必要です。

### 原理原則【15】 要件定義は「使える」業務システムを定義すること

* + - 1. 要件定義は、 業務にとって「使える」、 「役に立つ」、 「運用できる」システムを定義することです。
      2. 発注者は、 それまでのやり方にとらわれることなく、 むだな業務や非効率な手順を客観的に評価し、 新業務をゼロベースで再設計することが大切です。
      3. 要件定義の場に参加して、 議論が横道にそれたり、 枝葉末節に陥らないように助言するのは受注者の役割です。また、 受注者は、 要件として定義したものが、 システム化計画で想定したコストや期間と比べて過剰なものや、 逆にあまりに多くの費用を要さずとも実現可能な要件は、 勇気を持って変更を進言しなくてはなりません。
      4. Subtopic



### 原理原則【16】 機能要求は膨張する。コスト、 納期が抑制する

* + - 1. システム開発のコストは実現する機能ではなく、 工数に比例しますから、 どのくらいの作業が残っているのかをきちんと把握しながら、 機能との折り合いをつけて作業を進める必要があります。 このバランス感覚を プロジェクトメンバー全員が持っていなければ意味がありません。
      2. プロジェクトの背景や目的に応じたシステム化の範囲を検討し、 「ついでにこの範囲も」という考え方は本来の目的を見失うので絶対に避けましょう。

### 原理原則【17】 要件定義は説明責任を伴う

* + - 1. システム開発における万全なる準備は、 正確な要件という情報の次工程に向けての伝達です。 分が次工程に伝える必要のある情報について、 要件確定責任だけでなく説明責任を負う必要があります。
      2. システム開発の受託側から見た原則は「受託した要件として、 書いてあるものは実現させる。書かれていないものは作らない。」ことです。
      3. もちろん、 プロジェクトのスタート地点で、 すべてを誤りなく責任をもって確定することはできません。 「要件の行間を読め」ということを要求してはいけません。
      4. 基本的には当たりまえの前提や例外処理であっても 漏れなく伝達する必要があります。

## 【参考資料】

### 超上流から攻めるIT化の原理原則１７ヶ条【小冊子形式】

<https://www.ipa.go.jp/files/000005109.pdf>;

### 原理原則／行動規範【表形式】

<https://www.ipa.go.jp/files/000060917.pdf>;

# ユーザのための要件定義ガイド～要求を明確にするための勘どころ～ 【2017年3月】

<https://www.ipa.go.jp/sec/publish/tn16-008.html>;

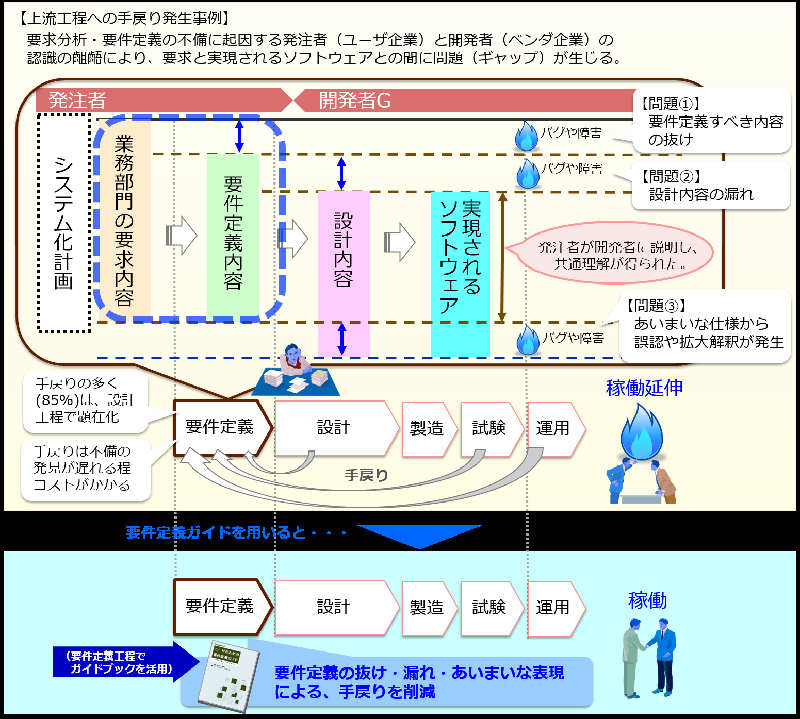
## システムに求められる要求を確実にシステム要件へ

### ユーザ企業からベンダ企業に要求が正しく伝わっていないと、 開発プロジェクトの現場で実装する機能は要求を正しく反映したものにはならない。 この問題の解決には、 ユーザが要求を抜け・漏れなく定義するために実施すべきことを 明確にすることが重要。

### そこで、 ユーザ企業とベンダ企業の知見やノウハウをまとめた“勘どころ（コツ）”を示した。

### 本ガイドブックは、 主にユーザ企業でITシステムの要件定義を実施する読者を対象に、 要件定義において発生する問題と、 その解決方法をまとめました。

### Subtopic



## システム構築の上流工程強化

## ITシステムに関わるIT技術とユーザ企業の現在

## ソフトウェアエンジニアリングの各領域における取り組み

## 各領域に取り組む背景