非機能要件記述と
アーキテクチャ記述
ガイド

概要編

独立行政法人　情報処理推進機構
ソフトウェア・エンジニアリング・センター
エンタプライズ系ソフトウェア開発力強化推進委員会
要求・アーキテクチャ領域

非機能要件とアーキテクチャＷＧ

2010 年 3 月

< 空欄 >

目　次

< 空欄 >

1. はじめに

現代世界においてビジネスを含めた社会生活を営むにあたり、それを支える社会システム基盤はコンピュータシステムを活用した機器や基盤となる情報システムに支えられていると言っても過言ではない。そのためコンピュータシステムに障害の発生があって何らかの形で機能しなくなった場合、国民生活に支障をきたし、またビジネスでの信用の失墜につながるなど社会的な影響度が増大していると考えられる。

これらを解決するために情報システムの信頼性の確保が必要であるが、一方でシステムの巨大化・複合化、パラダイムシフトや IT 技術の進歩の速さ、社会システム・ビジネスシステムの急激な変化等に対応が求められる。そこで社会やビジネスにおける適切なゴールとリスクの洗出しを行い、それらに呼応した情報システムの提供が求められる。

これらの情報システムの実現には、社会システムやビジネスシステムにおけるゴールやリスクからシステムの屋台骨であるアーキテクチャに至るトレーサビリティの確保が求められる。トレーサビリティの確保は信頼性の高いシステムの実現に寄与するだけでなく、優先度の低いリスクやゴールがあった場合、それらの導出結果を可視化することで、品質過剰なアーキテクチャの発見等コスト削減にも寄与できる可能性がある。情報システムの高度化・大規模化が進むに伴って、非機能要件[1]（Non Functional Requirements、以下、NFRと略す場合がある）の重要性がさらにクローズアップされている状況にある。

一方で、高品質の IT システムを短期間で構築するニーズの高まりにより、ベストプラクティスとして参照アーキテクチャを活用したパターン駆動アプローチでソリューションアーキテクチャを設計するITアーキテクトが多くなっている。参照アーキテクチャを安易に踏襲したソリューションアーキテクチャを設計すると、前提条件を満たさない制約がある場面や環境では全体最適のバランスが崩れ、期待した品質要件を満足しないリスクが包含されてしまう可能性がある。非機能要件はアーキテクチャの決定に重大な影響を及ぼすものである一方で、様々な状況の変化に対応し、リスクを考慮した非機能要件を定義するためには、今まで以上に難しい課題を解決する必要がある。

経済産業省　ソフトウェア開発力強化推進タスクフォース　要求工学・設計開発技術研究部会および非機能要求とアーキテクチャワーキンググループでは、非機能要件を正確に記述し、IT システム全体の構造を決めるアーキテクチャ設計にどのようにインプットするかという命題をもって、伝統的に曖昧なまま進められていた要求プロセスにメスをいれ、「非機能要求記述ガイド」（以下、2008 年度版非機能要求ガイドと略す）を 2008 年 7 月に提供した。独立行政法人 情報処理推進機構ソフトウェア・エンジニアリング・センター（以下 IPA SEC と略す）は、2008 年 7 月にエンタプライズ系ソフトウェア開発力強化推進委員会要求・アーキテクチャ領域にその後継の非機能要件とアーキテクチャワーキンググループ（以下、本 WG と略す）を設置し、2008 年度版非機能要求記述ガイドをベースにアーキテクチャ設計に影響すると考える非機能要件および制約の導出方法と記述方法について検討し、アーキテクチャに関する記述に直すことを検討してきた。

本ガイドでは、1) 企画・要件定義工程おける非機能要件記述を対象にして、アーキテクチャに関連する非機能要件を説明し、コントロールケースを用いた非機能要件の記述アプローチを示す。2) 設計工程

[1] 前回のガイドでは同じNon Functional Requirementsを「非機能要求」として訳していたが、当ガイドでは共通フレーム 2007 の定義に合わせて「非機能要件」と訳す。

におけるアーキテクチャ記述を対象にして、参照アーキテクチャを活用したパターン駆動アプローチの考慮点を示す。

本ガイドの想定読者

本ガイドは IT アーキテクトを目指す技術者のみならず、ビジネスゴール・ビジネスリスクと IT アーキテクチャの関係性を理解したい技術者も読者として想定している。

本ガイドの位置付け

2008 年 7 月に IPA SEC が公開した経済産業省ソフトウェア開発力強化推進タスクフォース非機能要求とアーキテクチャワーキンググループによる 2008 年度版非機能要求記述ガイドでは、非機能要求の記述ルール確定に位置付け、その記述ルールをまとめた。本ガイドでは、対象とする非機能要件およびコントロールケースを拡充した一方で、非機能要件と IT システムのソリューションアーキテクチャとの関係性を説明している。

本ガイドの内容は、本 WG 内で検討・整理した記述アプローチをまとめたものである。この取組みが、実際のプロジェクトにおける非機能要求が効率的、網羅的に記述することが可能となり、ひいては IT システムの信頼性確保に寄与することを願う。

2. 非機能要件記述

(1) 非機能要件記述のプロセス

非機能要件記述とアーキテクチャ記述のベースとなる、本ガイドで説明する非機能要件記述に関するアプローチは次のとおり(図 2-1)。本アプローチは繰り返し実施することを前提とし、意思決定したアーキテクチャが実現可能なものかどうかを検証する。ある非機能要件の要求レベルや場面が変わったことで、影響を受ける要件もある。そして実現可能性が確認できたアーキテクチャを関係するステークホルダとの間で合意する。これらを繰り返し、実施することでより精微なアーキテクチャに仕上げる。

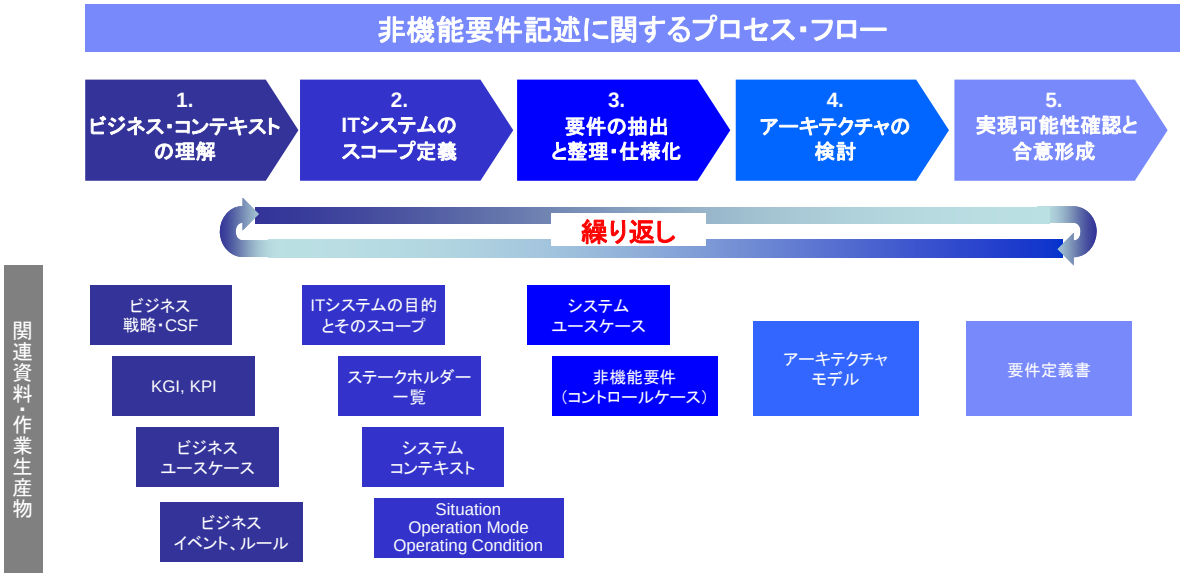

図 2-1 非機能要件記述プロセス・フロー

① ビジネス・コンテキストの理解

ITシステムが必要とされるビジネス環境について理解し、ビジネスイベントやビジネスルールを整理する。目標を代替特性で表現し、主要成功要因の達成度を評価するために重要項目達成目標 KGI (Key Goal Indicator)や重要業績評価指標 KPI(Key Performance Indicator)を整理する。

② IT システムのスコープ定義

IT システムが対象としている範囲を定義する。IT システムの目的とスコープを定義し、システムコンテキストを作成する。場面(Situation)の検討を行い、刺激(イベント)の発生源からの何らかの刺激により、場面が変化することを考える。運転モード(Operation Mode)の検討を行い、例えば full(正常運転)、degraded(縮退運転)、limited(制限運転)、alternative(代替運転)を主なモードを、各機能のサービスレベルの違いとして定義する。さらに稼動条件(Operating Condition)について、usual(通常時)、surge(予想可能なピーク)、burst(予測不可能なピーク)を適用する。システムのおかれている場面によりさらに異なる非機能要件を取り得ることがある。

③ 要求の抽出と整理・仕様化

システム・ユースケースの作成および非機能要求の抽出と整理を行う。あるシナリオで実行することを想定した場合、ある場面におかれた IT システムが、ある運転モードと稼動条件の下、どのような

非機能要件が求められるかを定義する。ここで整理されたシナリオ、場面、運転モード、稼動条件をコントロールケースとして記述する。インパクト（Impact）の定義として、定義された非機能要件が満足されない場合、どのような影響があるかを検討し、コントロールケースに記述する。特に KPI やビジネスへの影響が大きいものについては注意する。

④　アーキテクチャの検討

要件の優先順位の検討を行い、IT システムの存在意義を左右するユースケースや、実現しないと影響が大きい、コントロールケースを選定する。非機能要件関係記述による仕様の完全性とアーキテクチャの検討を行う。IT システムに重要な別の非機能要件と相関関係にある非機能要件を認識し、要件間の不整合、ユースケース、コントロールケースの関連性を定義する。そしてアーキテクチャ上の意思決定を行う。具体的にはコントロールケースで定義されている条件（シナリオ、場面、運転モード、稼動条件）と原理原則、アーキテクチャパターンを紐付け、アーキテクチャ設計上の意思決定を行う。

(2)　アーキテクチャ設計に関連する非機能要件

2008 年版非機能要求記述ガイドでは、Zou‐Pavlovski[1]によって導入されたコントロールケースを拡張し、特に効率性及び回復性についてコントロールケースを用いた記述方法について述べた。

本ガイドでは、2008 年版非機能要求記述ガイドをベースに、対象とする非機能要件の種類を 10 種類に拡張した。 これらの非機能要件は、本 WG でのディスカッションを通じ、IT システムのアーキテクチャへの影響が大きく、アーキテクチャ構築時に特に考慮すべきものとして優先的に採り上げた。 また、「制約」は、IT システム構築プロジェクトの前提条件であるが、非機能要件として整理するか否かは議論が分かれるところである。 しかし、IT システム・アーキテクチャ構築時には、必ず考慮すべき項目であることから、本ガイドでは、次の 10 種類の非機能要件と同様に整理した。

- 可用性(Availability)
- 障害許容性(Fault tolerance)
- 回復性(Recoverability)
- 効率性 (Efficiency)
- 適時性(Timeliness)
- 最新性(Data Currency)
- 管理容易性(Manageability)
- セキュリティ (Security)
- 性能拡張性(Scalability)
- 機能拡張性(Extensibility)
- 制約(Constraints)

各非機能要件について、定義、説明、メトリクス（指標）、メトリクス記述例、関連シナリオについて説明することで具体的に定義を行なう。付録 A に回復性の定義例を示す。

(3) コントロールケースによる非機能要件記述

コントロールケースとシナリオ

機能要件に対する記法にはユースケース等が存在するが、非機能要件に対する記法で広く利用されているものは存在していない。Joe Zou と Christopher J. Pavlovski はコントロールケースにより非機能要件を記述することを提案している。本 WG では非機能要件の記法として、Zou-Pavlovski のコントロールケースを拡張することを検討した。

Zou-Pavlovskiによって提案されたコントロールケースの概略と、本WGでのコントロールケースの拡張内容について説明する。

・Zou-Pavlovski のコントロールケース

Zou-Pavlovski は非機能要件の記法としてコントロールケースを提案している。コントロールケースは想定される稼動条件(Operating Condition)、稼動条件下で守るべき品質、品質が守られなかった場合のリスクといった要素で構成される。

コントロールケースにより、ステークホルダに対してリスクを明らかにし、リスクを和らげるためにどのような非機能要件が必要なのかという点を明らかにできる。

ユースケースとコントロールケースは並行して作成するのが自然である。ユースケース作成時に、そのユースケースが想定する稼動条件を洗い出し、その稼動条件に対して必要な非機能要件を表現したコントロールケースを作成する。つまり、ユースケースとコントロールケースは稼動条件を介して関連付けられているとも理解できる。

・本 WG におけるコントロールケースの拡張

本 WG で個々の非機能要件について議論を進める中で、すべての非機能要件を Zou-Pavlovski のコントロールケースで記述できることがわかった。可用性、効率性のようなシステム利用者や外部システムといったアクターと関わりがある非機能要件はユースケースと関連付けて導出が可能である。一方、機能拡張性、セキュリティのようなアクターに直接影響しない非機能要件は、個別のユースケースと関連付けて導出することは難しいことがわかった。 そこで本 WG は、その様な非機能要件に対するコントロールケースをユースケースだけではなく、ミスユースケースやチェンジケースといった種々のシナリオと関連付くものとして拡張することを提案する。コントロールケースと関連付くシナリオには以下のようなものが挙げられる。

表 2-1　コントロールケースと関連するシナリオの一覧

名称	概要
ユースケース	想定内の利用シナリオ
ミスユースケース	想定外、誤操作、故意、悪意の利用シナリオ
チェンジケース	将来発生しうる外部環境変化のシナリオ
フェイルケース	障害時のシナリオ

非機能要件ごとに、導出元となりやすいシナリオの種類が決まっている。前項で挙げた非機能要件に対して、導出元となりやすいシナリオとの関係を整理したものが図 2-2となる。

非機能要件＼シナリオ	ユースケース関連	ミスユースケース関連	フェイルケース関連	チェンジケース関連
可用性	●			
障害許容性			●	
回復性			●	
効率性	●			
最新性	●			
適時性	●			
管理容易性	●			
セキュリティ		●		
性能拡張性				●
機能拡張性				●

図 2-2 非機能要件と関連性の高いシナリオ

本ガイドでは、前項に挙げた非機能要件と関連性の高いシナリオであるユースケース、ミスユースケース、チェンジケース、フェイルケースを対象とした。

コントロールケースの説明例を付録 B に示す。また非機能要件記述例を付録 C に示す。

3. アーキテクチャ記述

(1) 本ガイドのアーキテクチャ記述のスコープ

本ガイドでは、開発プロセスとして、企画・要件定義工程、およびアーキテクチャの設計工程を対象に、複雑な対立関係がある非機能要件の解決に向け、パターン駆動アプローチによるITシステムのソリューションアーキテクチャを記述する参照アーキテクチャ活用事例研究結果を提示する。

アーキテクチャの定義

アーキテクチャ（Architecture）に関しては、様々な定義が存在し統一された標準が確立していないため、本ガイドでは、企業や公的機関(以下、エンタープライズ)が価値創造や組織活動の支援に利用するITシステムのアーキテクチャに限定する。本ガイドが対象にするITシステムは、情報技術に基づいた情報システムであり、IT 製品(ハードウェアやソフトウェア)を基盤として主にアプリケーションソフトウェアを開発対象とするソフトウェア中心システム [2]である。ソフトウェア中心システムのアーキテクチャ記述の推奨プラクティスであるIEEE1471-2000は、アーキテクチャを「システムの基本的になる組織的な構造であり、コンポーネント群、コンポーネント間の相互関係と環境との関係、設計と改良を管理する原則により構成される。」と定義している。

アーキテクチャは、設計の一種であるが、全ての設計がアーキテクチャではない。例えばシステムの構成要素であるコンポーネントに実装するアルゴリズムや入出力するデータの構造などは、コンポーネントの設計であり、システム全体の組織構造のアーキテクチャではない。米国カーネギーメロン大学ソフトウェアエンジニアリングインスティテュート（以下 CMU SEI と略す）では、システム要件の満足に対して重要でなく、コンポーネントの設計者や実装者に意思決定を委ねることができる設計をアーキテクチャと区別して「アーキテクチャ以外の設計(Non-architectural design)」と定義している（[12]）。

本ガイドでは、開発対象のアプリケーションソフトウェアのコンポーネント設計ではなく、アプリケーションソフトウェアの基盤となるハードウェアやシステムソフトウェア、さらに、利用・運用される環境も含めたIT システムのアーキテクチャを対象とする。

アーキテクチャのスコープ

IT アーキテクチャは、その対象領域のスコープにより、一般にエンタープライズアーキテクチャとソリューションアーキテクチャに大別される。エンタープライズアーキテクチャは、エンタプライズの全体最適化を目的として、IT 標準や参照アーキテクチャなど、個別の IT システムが遵守すべきアーキテクチャガバナンスである。本ガイドでは、エンタープライズアーキテクチャは、ソリューションアーキテクチャを設計する上で遵守する非機能要件の制約に位置け、アーキテクチャ記述では、個別の問題領域を対象としたIT システムのソリューションアーキテクチャを対象とする。

アーキテクチャのビューポイント

IT アーキテクチャでは、全てのステークホルダのビューポイント[2]の様々な関心事を単一のビューでアーキテクチャ記述[2]を作成することが困難である。そのため、様々なビューポイントカタログが提案されているが、業界標準として確立されたものが存在していない。

本ガイドでは、主要なビューポイントカタログや IT スキル標準の「IT アーキテクチャ・メタモデル・セマンティックス解説」[4]も考慮して、主要なビューポイントを独自に分類し、ビューとビューポイントとの関係を示している。

(2) 参照アーキテクチャを活用したパターン駆動アプローチ

参照アーキテクチャは、特定の問題領域を対象にしたソリューションアーキテクチャの成功事例に共通して見出された典型的なノウハウを一般化したソリューションアーキテクチャのテンプレートである。ソリューションアーキテクチャの成功事例は、機能要件だけでなく、非機能要件(品質要件や制約)の複雑な対立関係を全体最適で解決しているため、そのテンプレートである参照アーキテクチャには、様々な非機能要件に関連するアーキテクチャのパターンが含くまれている。

パターンの定義

パターンとは、特定の文脈において繰り返し発生する問題に対する実証済みの解決策を、いくらか抽象化してまとめた記述であり、ノウハウや定石あるいは典型的な手本ととらえることができる。パターンは規則ではないため、必ず従わなければならないものではないが、既知のパターンと文脈が合致する場合はその解決を再利用することで適当な結果が得られることが期待できる。

アーキテクチャ設計におけるパターンがアーキテクチャパターンであり、これまでに様々なものがカタログ化されて分類および公開されている。IT システムのアーキテクチャの設計において、扱う全ての機能要件および非機能要件が全く独自のものであるという状況は稀であり、部分的あるいは全面的に過去の成功事例と類似していることが多い。

過去の成功事例に共通して見出された典型的なノウハウを一般化し、実装技術や製品に依存しない抽象化されたアーキテクチャパターンを体系的にまとめたアセットが参照アーキテクチャと呼ばれている。

非機能要件とアーキテクチャパターンの関係

CMU SEI の研究成果である ADD や ATAM[8]によると、アーキテクチャパターンには、主目的の品質特性をよくするアーキテクチャの仕掛けがあり、品質特性に基づく設計の根本 [9]と呼んでいる。また、欧州では、アーキテクチャパターンから識別した解決策の本質的な仕掛けをアーキテクチャの根本［10] と呼んでいる。これらの用語は、日本語として馴染みが薄いため、本ガイドでは、これを基本パターンと呼ぶこととし、「特定の場面や状況に想定されるリスクや制約を前提として、主目的の品質特性に貢献するアーキテクチャパターンが品質を良くする本質的な仕掛けである。」と定義する。

ATAM によると、基本パターンには、センシティビティポイントとトレードオフポイントがあることが知られている。センシティビティポイントは、基本パターンを構成するコンポーネントやその関係の特徴として

見出され、主目的の品質特性に強い影響があるアーキテクチャの変数になっている。たとえば、キャッシュの場合、データアクセスの応答時間を短くしてパフォーマンスの品質特性を良くする効果を大きくするためには、キャッシュ有効時間が長いこと、および、キャッシュのデータ容量が大きいことによりキャッシュヒット率が高くなることが前提条件になる。この前提条件を満たさない場合には、意図したパフォーマンスの品質特性が良くならない。これら前提条件は、アーキテクチャの変数であり、センシティビティポイントになる。

トレードオフポイントは、対立関係にある複数の品質特性に影響するアーキテクチャの変数である。一般に、センシティビティポイントは、他の品質特性を悪くするトレードオフポイントにもなっていることが知られている。たとえば、キャッシュの場合、キャッシュの有効時間がマスタデータの更新間隔よりも長くなると、キャッシュデータの鮮度が古くなり、データの最新性の品質特性が悪くなる。このトレードオフは、クラウドコンピューテイングで話題になっているCAP 定理(Consistency, Availability and Partition Tolerance Theorem)にも当てはまっており、キャッシュサーバーにデータを物理的(空間的)に分散させ、マスターデータサーバーの処理能力やサービス提供時間に依存せずに、大量のユーザに何時でも良い応答時間でデータアクセスサービスを提供するためには、データの最新性、つまり、データの一貫性を犠牲にする必要がある。

(3) アーキテクチャパターンの事例研究

高品質の IT システムを短期間で構築するニーズの高まりにより、ベストプラクティスとして参照アーキテクチャを活用したパターン駆動アプローチでソリューションアーキテクチャを設計する IT アーキテクトが多くなっている。参照アーキテクチャは、特定の問題領域を対象にしたソリューションアーキテクチャの成功事例に共通して見出された典型的なノウハウを一般化したソリューションアーキテクチャのテンプレートである。ソリューションアーキテクチャの成功事例は、機能要件だけでなく、非機能要件(品質要件や制約)の複雑な対立関係を全体最適で解決しているため、そのテンプレートである参照アーキテクチャには、様々な非機能要件に関連する基本パターンを組み合わせたアーキテクチャパターンが含くまれている。一般に、参照アーキテクチャは、実装技術や製品に依存しない抽象化されたコンポーネントの機能仕様のセットとコンポーネントの論理的な関係や物理的(空間的)な配置を示したモデル図として提供されるが、非機能要件の対立関係、非機能要件と基本パターンの因果関係、全体最適が成立する前提条件などが明文化されていないことが多い。そのため、参照アーキテクチャを安易に真似てソリューションアーキテクチャを設計すると、前提条件を満たさない制約がある場面や環境では全体最適のバランスが崩れ、期待した品質要件を満足しないリスクがある。

本 WG では、このリスクを緩和するために、TOGAF で紹介されている参照アーキテクチャの一つである Patterns for e-business (以下、P4eb) [11] を題材としたアーキテクチャパターンの事例研究を実施した。この事例研究では、ATAM[9]のアプローチを参考にして、参照アーキテクチャに暗黙的に含まれている基本パターンと非機能要件の因果関係を以下のようなステップで可視化している。

① 参照アーキテクチャから品質特性に影響する基本パターンを抽出し、主効果、副効果、悪影響を分析して品質特性別に分類する

② 基本パターンが主目的の品質特性に貢献するセンシティビティポイントを識別し、品質特性を良くする原理を明確にする

③ 対立関係にある複数の品質特性に影響するトレードオフポイントを識別し、他の品質特性を悪くしている原理を明確にする

参考文献

[1] Joe Zou and Christopher J. Pavlovski: “Modeling Architectural Non Functional Requirements: From Use Case to Control Case”, Proceedings of IEEE International Conference on e-Business Engineering 2006(ICEBE’06) (2006).

[2] ISO/IEC 42010:2007 Systems and software engineering -- Recommended practice for architectural description of software-intensive systems, 2007.

[3] The Open Group: TOGAF Version 9, The Open Group Architecture Framework (TOGAF), 2009.

[4] IPA ITスキル標準 プロフェッショナルコミュニティ ITアーキテクト委員会, IT アーキテクチャ・メタモデルセマンティクス解説 Ver1.0, 2007.
http://www.ipa.go.jp/jinzai/itss/activity/architect_com.html

[5] Paris Avgeriou and Uwe Zdun: “Architectural Patterns Revisited – A Pattern Language”, Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005) (2005). http://hydra.infosys.tuwien.ac.at/Staff/zdun/publications/ArchPatterns.pdf

[6] 鷲崎弘宜, 丸山勝久, 山本理枝子, 久保淳人 (深澤良彰 監修): “ソフトウェアパターン - パターン指向の実践ソフトウェア開発”, 近代科学社, 2007.

[7] Deepak Alur, John Crupi, Dan Malks, （中野明彦, 佐野祐一郎, 宮田泰宏, 土屋聡一郎訳）: "J2EE パターン: 明暗をわける設計の戦略", ピアソンエデュケーション, 2002.

[8] Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation, Technical Report, Software Engineering Institute, Carnegie Melon University, CMU/SEI-2000-TR-004 ESC-TR-2000-004 (2000).

[9] Gallagher, B.P.: “Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architecture: A Case Study”, Technical Note CMU/SEI-2000-TN-007 (2000).

[10] Uwe Zdun at al.: Modeling Architectural Patterns Using Architectural Primitives, OOPSLA ’05, October 2005.

[11] http://www.ibm.com/developerworks/patterns/

[12] CMU SEI: http://www.sei.cmu.edu/architecture/start/glossary/

[13] 電子情報技術産業協会, ソリューションサービス事業委員会: 民間向け IT システムの SLA ガイドライン第三版, 日経 BP 社, 2006.

付録A.　　非機能要件の説明例

非機能要件の「回復性」の定義を例として示す。

(a)　定義

回復性とは、障害によって下がってしまった業務レベルを指定されたレベルにまで復旧するために、直接に影響を受けたデータ、コンポーネントやハードウェアを回復する能力である。

(b)　説明

回復性を定義するためには、システムへの影響だけではなくビジネスへの影響を分析することが必要である。障害発生後ビジネスを再開するまでに、どのくらいの時間がかかるか（Recovery Time Objective：復旧時間目標）や、どこまで障害発生直前の状態のデータを用いて復旧できるか（Recovery Point Objective:復旧時点目標）を定義する。

また、障害復旧後に達成すべきビジネスレベル（Recovery Level Objective:復旧レベル目標）を設定することができる。例えば、すべてのサービスを正常に回復する場合だけでなく、サービスレベルを落とした縮退モードの場合や一部のサービスを停止する制限モードなどを設定できる。

(c)　メトリクス

回復性のメトリクスには、RPO、RTO、RLO などがある。

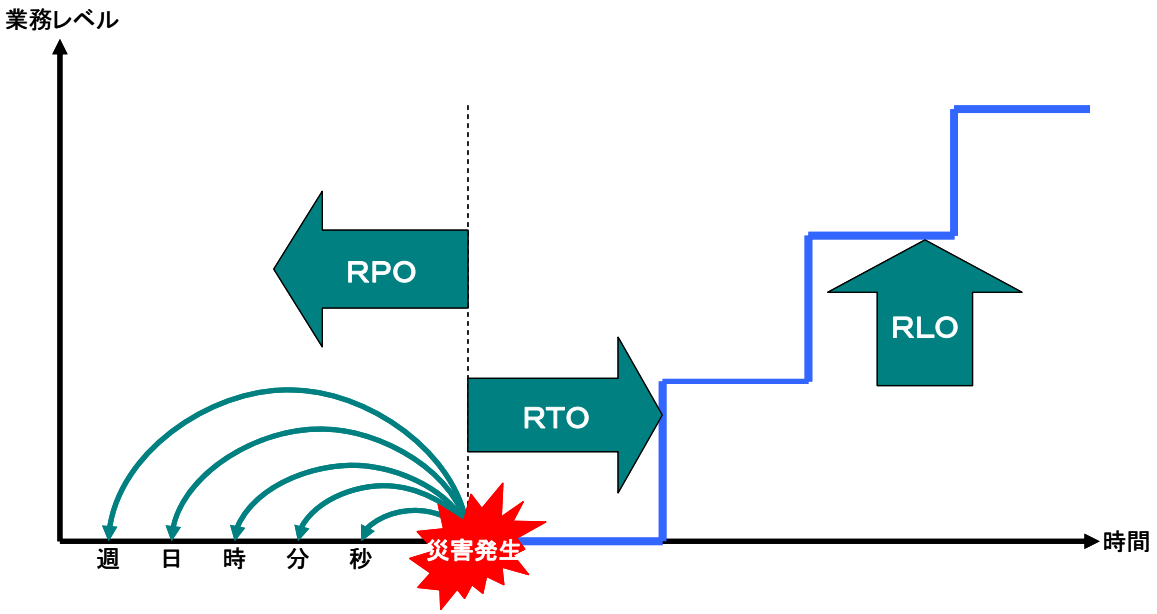

図 A-2 回復性のメトリクス

■RPO【Recovery Point Objective：復旧時点目標】

障害が発生した場合に、過去のどの時点までのデータを復旧できればいいかという目標が RPO であり、時刻で表される。RPO は障害回復時点で失われている可能性のあるデータ量と関係があり、どの程度までデータ損失を許容するかの目標と言い換えることもできる。

RPO はゼロに近いほどデータ損失が少なくなるが、比例してコストも高くなる。データやトランザクションの損失によるビジネスへの影響を考慮して決める必要がある。

■RTO【Recovery Time Objective：復旧時間目標】

障害発生後、いつまでにデータ、システムやサービスを復旧させるかという目標が RTO であり、障害から復旧までの時間で表される。システム停止やサービスの中断が許容される程度（時間）と言い換えることもできる。

RTO はゼロに近いほど早く復旧するが、比例してコストも高くなる。システム停止によるビジネス損失との兼ね合いで決める。停止していても何らかの手段でビジネスが継続できるのであれば、RTO を長く設定することができる。

■RLO【Recovery Level Objective:復旧レベル目標】

障害発生後、どの程度の状態でサービスを再開するかという目標が RLO であり、運転モードとサービスレベルなどで表される。ただし、想定する業務によっては他のメトリクスを用いることもある。

運転モードの例を次に示す。

- Full Service mode: 通常モード
- Degraded mode: 縮退モード(容量縮退、品質低下)
- Restricted/Limited mode:制限モード(機能制限)
- Alternative mode:代替モード(マニュアル運用など)

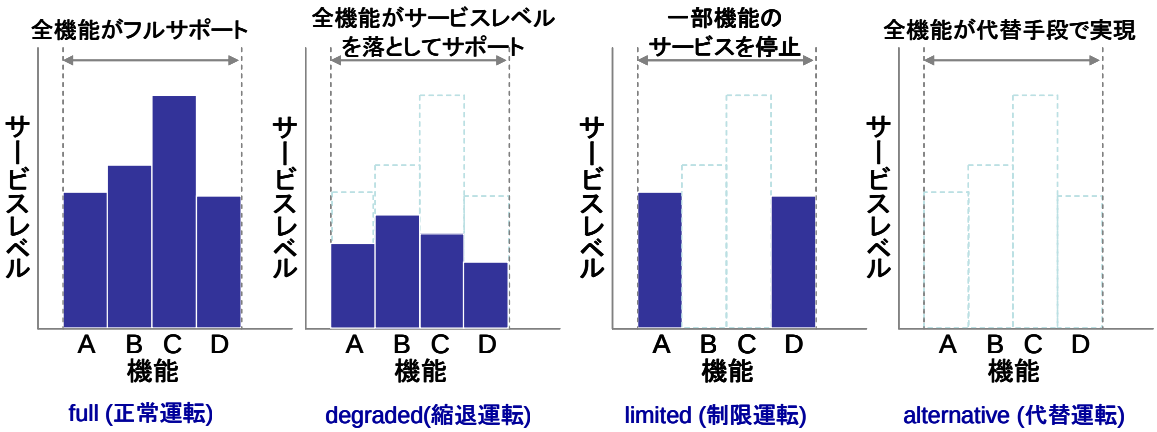

図 A-２　運転モード

サービスレベルは、縮退、制限する品質特性により下記のようにレベルを表記すると効果的である。

- 提供可能な機能数の充足率（%）、または、提供可能な機能のリスト
- 提供可能な品質が低下する度合い（%）、低下率
- 提供可能な資源・容量の度合い（%）、縮小率

(d) メトリクス記述例

■RPO【Recovery Point Objective：復旧時点目標】

①トランザクションに関しては障害発生の 0.5 秒前までに戻ること。マスタデータに関しては 1 日前の状態に戻ること。アーカイブデータに関しては 3 日前の状態に戻ること。

または、

②障害直前にコミットされた時点まで回復すること。コミット前の処理中のトランザクションはすべて破棄する。

■RTO【Recovery Time Objective：復旧時間目標】

①サービスが全停止の時は、1時間以内に復旧を完了する。

②サブシステム単位のサービスが停止の時は、2時間以内に復旧を完了する。

③軽微のエラー場合は、翌日対応する。

■RLO【Recovery Level Objective:復旧レベル目標】

①基本的に全機能を通常稼動の状態で回復することを目標とする。しかし，RTO 時間内に通常稼動が無理な場合は，縮退・制限運転を検討する。

②業務の優先順位を付けて、優先順位が高いものを早く復旧させる。

関連シナリオ

フェイルケース

付録B.　　コントロールケースの定義例

コントロールケースのひとつであるチェンジケース（Change Case）の定義を例で示す。

(a)　定義

チェンジケースとは、今後発生する可能性のある外的環境(場面：Situation)やシステム環境(状況：Condition)の変化に対応し、既存システムからの変更や拡張を明らかにするために用いる特定のシナリオである。

(b)　説明

チェンジケースは、現在の外的環境(場面：Situation)やシステム環境(状況：Condition)と、将来の環境の差が分かるように記述する。

外的環境の変化の例として、

- 法改正
- 業務のやり方、業務フローの変更
- 組織変更
- 商品・サービスのカテゴリの追加／変更
- 想定利用者の変更

などのようなビジネス側面の変化と、

- 技術の進化

などのようなテクノロジー側面の変化がある。

システム環境の変化の例として、

- 他システムとの追加接続
- システム資源の変化

などがある。

その他、チェンジケースには、変更が発生する時期、チェンジケースが発生する可能性、変更発生時の影響の度合いを記述する。

関連する非機能要件

- 性能拡張性
- 機能拡張性

付録C.　　非機能要件の記述例

シナリオとコントロールケースを使った非機能要件記述例を示す。

scenarioDefinition
（シナリオ一覧）

xxケース番号	アクター	ユースケース名称
UC01	Webユーザー	取引申込を登録する
UC02	Webユーザー タイマー	取引申込をバックエンドシステムに依頼する
UC03	勘定系システム	取引結果を保管する
UC04	Webユーザー	取引結果を確認する
UC11	Webﾌﾛﾝﾄｴﾝﾄﾞｱﾌﾟﾘｹｰｼｮﾝ	取引申込をスケジュールに登録する
UC12	スケジューラー	取引を処理する
UC13	（スケジューラー）	取引結果をフロントエンドシステムに通知する

(a)　　シナリオ一覧

controlCase （コントロールケース）	内容
ccid （コントロールケースID）	CC-E1
ccName（名称）	利用者視点の性能
ccDescription（説明）	利用者の求める性能
ccSituation （外部環境）	インターネットチャネルの利用頻度に関わりなく、普段、ピークとも。バックエンドシステムが非稼働のときも含む。
ccAssociateConstraints （関連する制約）	ITポリシーによるスループット制限値：5000トランザクション/秒
ccAssociateScenario （関連するシナリオ）	UC01取引申込を登録する　UC04取引結果を確認する
ccRiskDescription （リスクの定義）	ピークに合わせ

context（コンテクスト）	operationMode
	operatingCondi
changeCase （チェンジケース）	from Situation
	to Sistuation

NFR type	内容
時間効率性	応答時間　：サ の利用者待ち時
時間効率性	平均スループッ
資源効率性	主記憶使用率

impact （影響）	
likelyhood （発生頻度）	常時

controlCase （コントロールケース）	内容
ccid （コントロールケースID）	CC-E3
ccName（名称）	処理集中時のための資源利用率の制限
ccDescription（説明）	ピーク時や一時的な処理集中などに備えて、資源に余裕を持つ。
ccSituation （外部環境）	普通
ccAssociateConstraints （関連する制約）	
ccAssociateScenario （関連するシナリオ）	特定のユースケースはなし。
ccRiskDescription （リスクの定義）	

context（コンテクスト）	operationMode	Full(正常運転)　Limited(制限運転)
	operatingCondition	usual
changeCase （チェンジケース）	from Situation	
	to Sistuation	

NFR type	内容
資源効率性	主記憶使用率：80%までとする。

impact （影響）	
likelyhood （発生頻度）	

(b)　コントロールケースを使った非機能要件記述例

図 C-1　非機能要件記述例