Web アーキテクトの仕事

Web アーキテクトの仕事の基本的なプロセスと成果物，心得を説明します。

仕事のプロセスは 3 ステップ

Web アーキテクトの仕事は，突き詰めれば「非機能要件に対する現実解を導き出すこと」です。ただし，この現実解のシステム構成や構造は，Web アーキテクトが 1 人で導き出すわけではありません。

例えば，インフラ製品を提供するハードウエア・ベンダーやミドルウエア・ベンダーの担当者，もしくは特定技術分野に精通した社内外のスペシャリストから，情報提供・技術検証・導入コンサルティングなどを適宜受けることになります。また，開発プロジェクトに参加している PM や技術者たちとも相談しながら，全体の構成と構造を考えなければなりません。

つまり Web アーキテクトは，システムに対するステークホルダー（利害関係者）の中心となり，技術的な解決策について繰り返し対話・調整・思考する役割を担います（**図 1**）。そのため Web アーキテクトには，技術力に加えて調整能力も必要とされるのです。

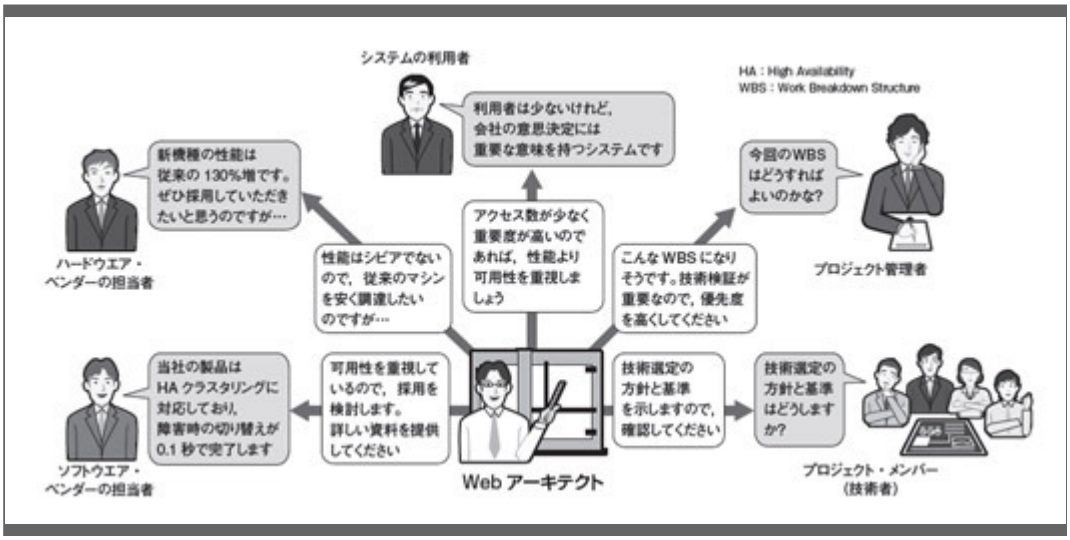

図 1●Web アーキテクトはステークホルダーの中心にいる

Web アーキテクトは，Web システムの構成やアーキテクチャを決めるうえでのリーダーである。利用者，ベンダー担当者，プロジェクト管理者，プロジェクト・メンバーといった複数の利害関係者（ステークホルダー）とコミュニケーションし，方針などを明確化する必要がある

こうしたWebアーキテクトの仕事の基本的なプロセスは，次の三つのステップで表すことができます。

ステップ1:システムの利用者や所有者が漠然と抱える機能以外の要求を抽出して整理する

ステップ2:コストや納期を考慮しながら妥当な範囲で技術的な解決策を洗い出す

ステップ3:解決策についてシステムのステークホルダーの合意を得る

以下，各ステップについて掘り下げます。

ステップ1:非機能要件を抽出して整理する

非機能要件は，システムの利用者や所有者から明確に示されることは多くありません。漠然と伝えられたり，間接的に表明されたりすることがほとんどです。Webアーキテクトは，想像力を働かせて，言外に隠された要求を読み取ることが欠かせません。

例えばWebシステムの所有者が，「今回のWebシステムは海外拠点の業務支援を目的に日本の本社に設置します。各国の業務時間内だけ利用できれば十分です」と説明したとしましょう。Webアーキテクトはこの説明から，「システムは止められないかも」とか，「夜間バッチによってオンライン性能が低下するかも」と思いを巡らさなければなりません。

なぜなら，各国に時差があるからです。拠点が日米だけであれば2拠点とも業務時間外になっている時間帯があるかもしれませんが，拠点が日米欧にあるとそうした時間帯は実質的にほとんどなくなるでしょう。拠点ごとにシステムを独立させられないとすれば，このシステムは計画停止が困難になり，可用性や保守性などの要件が厳しくなります。

Web アーキテクトはヒアリングしているとき，非機能要件に相当する内容が含まれているかもしれない，と気を配らなければなりません。ヒアリング内容を可用性や運用性などの視点でとらえ，その場で非機能要件なのかどうかを確認します。先程の「各国の業務時間内だけ利用できれば十分」という言葉は，例えば「24時間365日の無停止稼働になる」という可用性の要件や，「複数の機器を順番にアップデートさせることになる」といった運用性の要件にとらえ直します。

このようにWebアーキテクトは，ヒアリングの対象者が何気なく口にする言葉の中から隠された要求を読み取って，非機能要件を抽出します。具体的な解決策は後から考えることになりますが，予算から考えて明らかに過大・過剰な非機能要件が挙がっている場合には，抽出段階でその旨を説明し，非機能要件の絞り込みや調整を試みるべきでしょう。また，セキュリティ要件はシステムの利用者や所有者が主体的に明示することは難しいので，考えられるリスクを具体的に示しながら，それが顕在化した場合にどう困るのかを確認しつつ要件とするかどうかを判断しましょう。

抽出した非機能要件は，情報システムに求められる品質特性について定めた標準規格「ISO9126-1」に基づいて整理します。要件をISO9126-1の品質属性に分類することで，解決策を考えやすくなります。

ステップ 2:解決策を洗い出す

次に，整理した非機能要件ごとに解決策を洗い出します。解決策は最終的に，次のステップ 3 で各ステークホルダーの意見を聞きながら合意・決定しますので，この段階で一つに絞り込む必要はありません。むしろ，多様な選択肢を漏れなく洗い出すことを意識しましょう。

解決策を考えるアプローチは複数あります。Web アーキテクトは，与えられた予算と納期を考慮しながら，いろいろなアプローチを複合的に検討し，実効性のある解決策を導き出さなければなりません。

代表的なアプローチは，この連載で多く採り上げた“システム的”に解決策を探る方法です。「可用性」「パフォーマンス」「セキュリティ」「運用性」といった要件ごとに，要素技術や適用方法を考えていきます（**表 1**）。一般に若い技術者はシステム的な解決策にとらわれがちですが，実際にはすべての非機能要件にシステム的な解決策を適用できるわけではありませんし，システム的な解決策の効率がよいとも限りません。

表 1●非機能要件を満たすための要素技術やプロセスの例

非機能要件の種類	要素技術やプロセス
可用性	FT（無停止）サーバー
	データ同期技術
	フェールオーバー技術
	HA（High Availability）クラスタリング
	負荷分散クラスタリング
パフォーマンス	スケールアップ
	スケールアウト
	キャパシティ・プランニング
	サーバー負荷分散装置（ロード・バランサ）
セキュリティ	セキュア・プログラミング
	脆弱性テスト
	SSL（Secure Sockets Layer）
	ファイアウォール
	侵入検知システム（IDS）
	侵入防御システム（IPS）
	Web アプリケーション・ファイアウォール（WAF）
	認証（利用者・端末）
	可逆暗号
	不可逆暗号
運用性	サーバー集約
	サーバー仮想化
	リソース・プール
	自動インストール・サーバー
	ソーリー・サーバー

例えば運用性の一部として，「障害時に原因の判別や修正個所の特定が容易であること」といった要件や「稼働後の修正や変更が容易であること」といった要件を求められた場合には，要素技術やその適用方法を検討するよりも，一貫性を保ったアプリケーション構造にする方が重要になってきます。こ

の場合には，前回「アプリケーション・アーキテクチャ」で説明したように，構造化や標準化などを徹底することが肝心です。これは，システム的な解決策というよりも，“アプリケーション的”な解決策です。

“開発プロセス的”な解決策もあります。パフォーマンスの要件であれば，設計フェーズで「キャパシティ・プランニング」を，テスト・フェーズで「負荷テスト」を，それぞれタスクとして追加すれば，要件を満たしたかどうかを確認できます。「使いやすいシステムにしてほしい」といったユーザビリティの要件が示された場合には，設計フェーズではインタラクション・デザインやペーパー・プロトタイピングを，テスト・フェーズでは利用者による操作検証を，それぞれ実施することを考慮します。

Web アーキテクトは，短期間で実効性を上げられるように，臨機応変に解決策を検討することが欠かせません。そのためにはシステム技術以外の技術・手法を理解することが肝要です。

ステップ 3：解決策に対して合意を得る

ステップ 2 で考えた解決策を各ステークホルダーに示し，どの解決策を採るのか合意・決定します。Web アーキテクトはこのとき，解決策ごとのトレードオフを十分説明しなければなりません。

例えば AP サーバーの可用性の要件に対して，ステップ 2 で「HA（High Availability）クラスタリング」と「スケールアウト」の 2 種類の解決策を考えたとします。スケールアウトは，拡張性が高く，専用のクラスタリング・ソフトがいらないというメリットがあります。ですがサーバーの台数が増えると，AP サーバー・ソフトのライセンス料がかさんでコストが上がりかねないというデメリットもあります。

こういったメリットとデメリットをステークホルダーに十分に示し，デメリットについてはさらに解決策を考えることが重要です。例えば，ライセンス料によるコスト増大を避けるために，無償のオープンソース・ソフトを利用することが考えられます。ただしオープンソース・ソフトは，一般に商用製品と同等のサポートを期待しづらいので，ステークホルダーの意向を十分に考慮し，どの解決策をどのように組み合わせるかを調整します。

最終的に，選択した解決策を組み合わせたとき，全体のバランスが取れているどうかを確認します。例えば**図 2** 左のような構成は，負荷が増大してサーバーを増設するたびにファイアウォールの設定を変更しなければならず，運用後に問題を起こしやすい悪い設計と言えます。Web アーキテクトは全体に与える影響に配慮しなければなりません。図 2 左の例は同右のように，AP サーバーを増設してもファイアウォールの設定を修正しなくてよい構成にすべきでしょう。

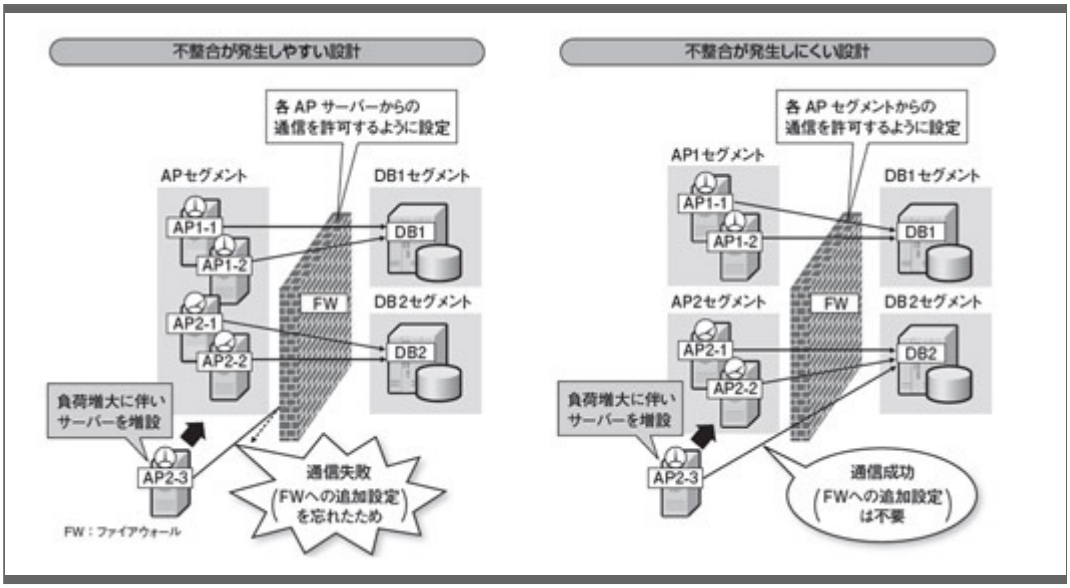

図 2●不整合が発生しやすい設計と発生しにくい設計

「スケールアウトでキャパシティを拡張する」という同じ方針でも，設計次第で不整合の生じやすさは変わる。サーバー増設に伴う追加設定が必要な左の設計に比べ，追加設定が不要な右の設計は不整合が発生しにくい

このほか，解決策の積み重ねによって，予算以上のコストが発生してしまう場合もあります。その場合，ステークホルダーに予算の上積みや，要件に対する妥協を求めることになります。要件に優先順位をつけ，解決策を判断することが欠かせません。

近年、「IT アーキテクト」という職種が注目度を増しています。情報システムが社会インフラとしてなくてはならない存在になっている現在、その良否、出来不出来が直接生活に響いてくる場面も増えてきています。その情報システムのアーキテクト(建築家、設計者)という立場である IT アーキテクトとはいったいどのような職業なのでしょうか。

IT アーキテクトが求められている背景や果たすべき役割

IPA(独立行政法人 情報処理推進機構)が定める IT 関連の職業に従事する人材が持っている(持つべき)スキルを規定した ITSS(IT スキル標準)V2 では、IT 人材の職種を 11 に分類しています。その中に「コンサルタント」「IT スペシャリスト」「プロジェクトマネジメント」等の職種に並んで「IT アーキテクト」があります。"IT"の文字が示すように技術的な側面が強調されており、"スペシャリスト"とも異なるグローバルな観点を持つ職種ということがわかります。IT アーキテクトの職種説明には以下のように記述されています※。

> ビジネスおよび IT 上の課題を分析し、ソリューションを構成する情報システム化要件として再構成する。ハードウェア、ソフトウェア関連技術(アプリケーション開発技術、メソドロジ)を活用し、顧客のビジネス戦略を実現するために情報システム全体の品質(整合性、一貫性等)を保った IT アーキテクチャを設計する。設計したアーキテクチャが課題に対するソリューションを構成することを確認すると共に、後続の開発、導入が可能であることを確認する。また、ソリューションを構成するために情報システムが満たすべき基準を明らかにする。さらに実現性に対する技術リスクについて事前に影響を評価する。

この記述から、IT アーキテクトはシステムが解決すべき課題を分析して「要件」にし、その技術的な解決策(ソリューション)を構成する「アーキテクチャ」を設計する職種であることがわかります。ソリューションは個々の課題を解決する手段なので、それらを整合性ある形で纏め上げる枠組みを設計することが役割となっています。また、システムの品質や、開発可能性、技術リスクの評価等が責務として挙げられており、システム全体の構成を考えるディレクターといった立場といえます。

この職種内容を見ると、現在の情報システム構築が抱える問題が透けて見えてきます。大規模開発を行って鳴り物入りで開発したものの、エンドユーザの要望を反映していないため、まったく使われず埃をかぶっているシステム、性能や負荷レベルをまったく考慮せずに開発したため、カットオーバした途端にダウンしてしまうシステム、最新/最高レベルの技術を採用したため開発者にノウハウがなく、試行錯誤の末に結局完成しなかったシステム……。このような、巷でよく聞かれるバランスや全体見通しに欠けたシステムの発生を未然に防ぎ、顧客が望み、開発者が安心して開発でき、利用者が喜ぶシステムを設計するのが、IT アーキテクトの職務といえます。