ソフトウェアレビューシンポジウム
JaSST Review'18

graat

アーキテクチャのレビューについて

2018/12/14

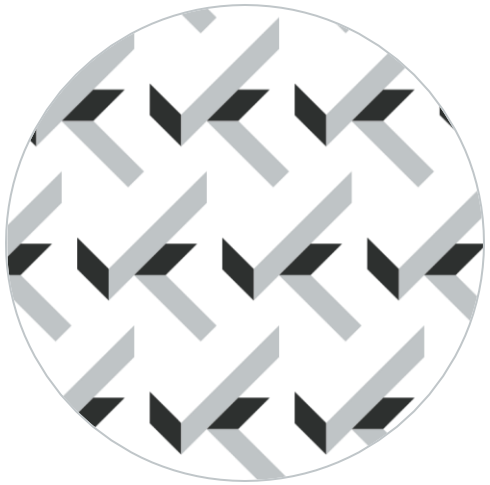
鈴木雄介
Graat　代表
日本Javaユーザーグループ　会長

自己紹介

鈴木雄介

- Graat（グラーツ）
  - グロース・アーキテクチャ＆チームス株式会社
  - 代表取締役 社長
  - http://www.graat.co.jp/
- 日本Javaユーザーグループ
  - 会長
  - http://www.java-users.jp/
- SNS
  - http://arclamp.hatenablog.com/
  - @yusuke_arclamp

アジェンダ



- アーキテクチャとは
- アーキテクチャの役割
- アーキテクチャ設計レビュー
- アーキテクチャ設計レビューの難しさ
- まとめ

graat

アーキテクチャとは

アーキテクチャとは

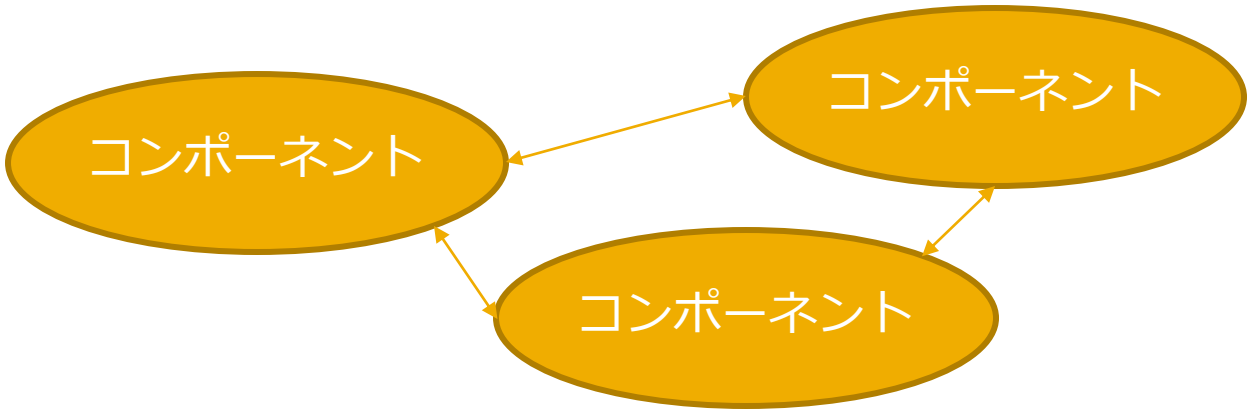
定義

- architecture 〈of a system〉
  - ある環境におけるシステムの基本的な概念や性質のことであり、そのシステムの要素と関係、そして、設計と進化の原則が具現されている
    - ISO/IEC/IEEE 42010
    - http://www.iso-architecture.org/
    - 参考：All about ISO/IEC/IEEE 42010 (r5)

アーキテクチャとは

システムの要素と関係性

- システムは複数のコンポーネントの組み合わせで成立する
  - » コンポーネント：部品、特定の機能を果たす単位
- システムが1.どのようなコンポーネントで構成され 2.それらがどのように連携するのか？



アーキテクチャとは

アーキテクチャのレベル感

- 様々なレベルにおいて要素と関係性が存在する
  - 企業全体システム
  - 企業内/システム間
  - システム内/サービス間
  - サービス内/フレームワーク
  - コード

アーキテクチャとは

システムの要素と統合の方針

- システムの目的にあわせ、
- システムをどのような要素で構成し、それらの要素をどのように連携させるのか
  - 将来的な変更に対して、どのように対応するのか？
- を示した方針

アーキテクチャ設計とは

定義

- architecting
  - システムのライフサイクルを通じて、アーキテクチャを適切に実装、保守、改善することを想起、定義、表現、文書化、伝達、証明するプロセス
    - ISO/IEC/IEEE 42010
    - http://www.iso-architecture.org/
    - 参考：All about ISO/IEC/IEEE 42010 (r5)

graat

アーキテクチャの役割

アーキテクチャの役割

現代的なシステム開発

- 機能を実現する手段が沢山ある
  - 言語、OSSライブラリ、商用製品、クラウドサービス…
    - かつ、各コンポーネントは常に進化している
- 他システム連携も増えている
- 環境の変化スピードが早い
  - 新しい機能が欲しい、ユーザーが増えていく

アーキテクチャの役割

組み合わせでシステムを作る時代

- 「巨大でモノリシックなシステム」ではなく「適度に分割され、入れ替え可能なシステム」
  - 進化が進んでいくとマイクロサービス

- アーキテクチャが重要に！
  - システムの分割と統合の方針がライフサイクルを通じた根幹

良いアーキテクチャとは何か？

アーキテクチャの役割

良さを非機能で判断する

- 機能実現は前提として非機能を重視すべき
  - 非機能を中心としたシステム評価
- どのような組み合わせが適切に非機能を実現するのか？
  - 「保守性」「性能」「可用性」「セキュリティ」「使用性」…
  - 特に他システムやクラウドのような「非機能を伴うコンポーネント」の扱いが重要になる

アーキテクチャの役割

非機能が保証されないと…

- リリース時点で問題がなくてもライフサイクルの中で
  - 複雑さによる保守性悪化
  - 性能劣化や階層障害の発生
  - 部分的な機能停止がシステム全体の停止を招く
  - ユーザービリティの限界
  - …

アーキテクチャの役割

アーキテクチャがシステムの非機能を保証する

- アーキテクチャをきちんと設計されればライフサイクルを通じて適切に機能が提供できるようになる
  - 最初に方針を定めることが重要

- アーキテクチャの評価は「非機能の適切さ」で評価可能
  - 単体の品質ではなく、組み合わせて想定通りに動くのか？

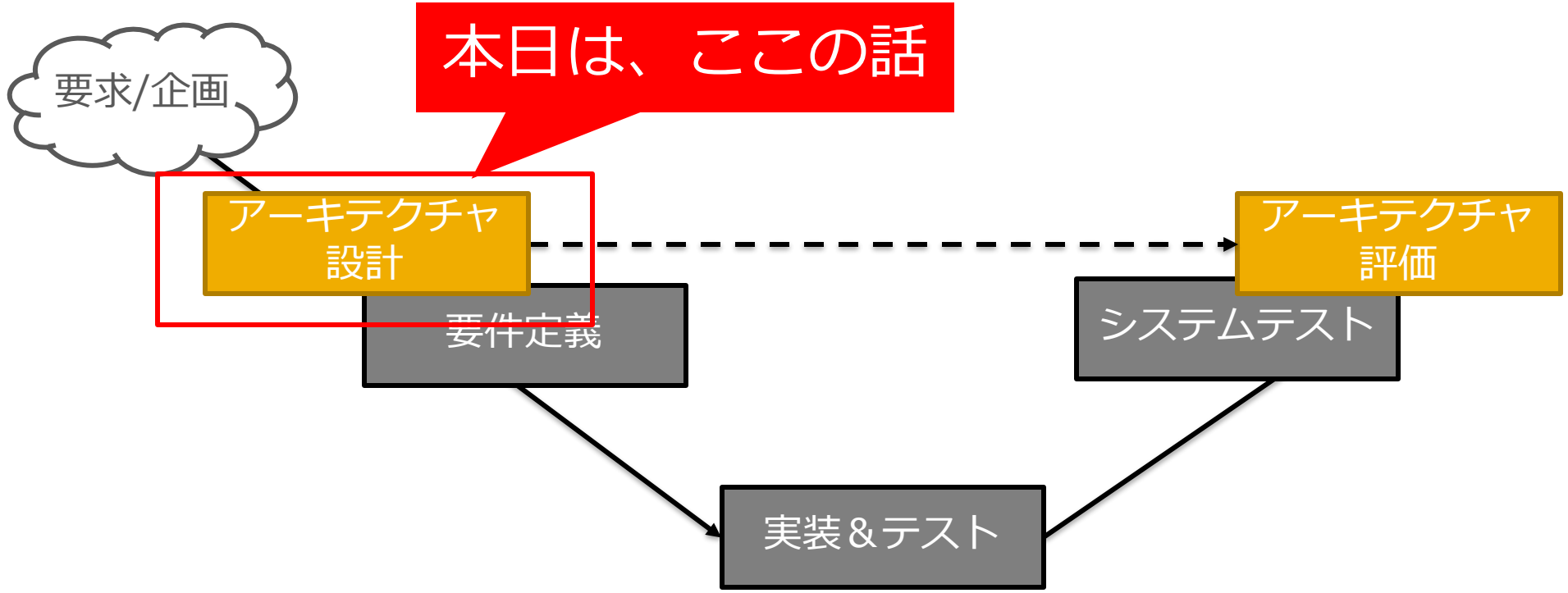
graat

アーキテクチャ設計レビュー

アーキテクチャ設計レビュー

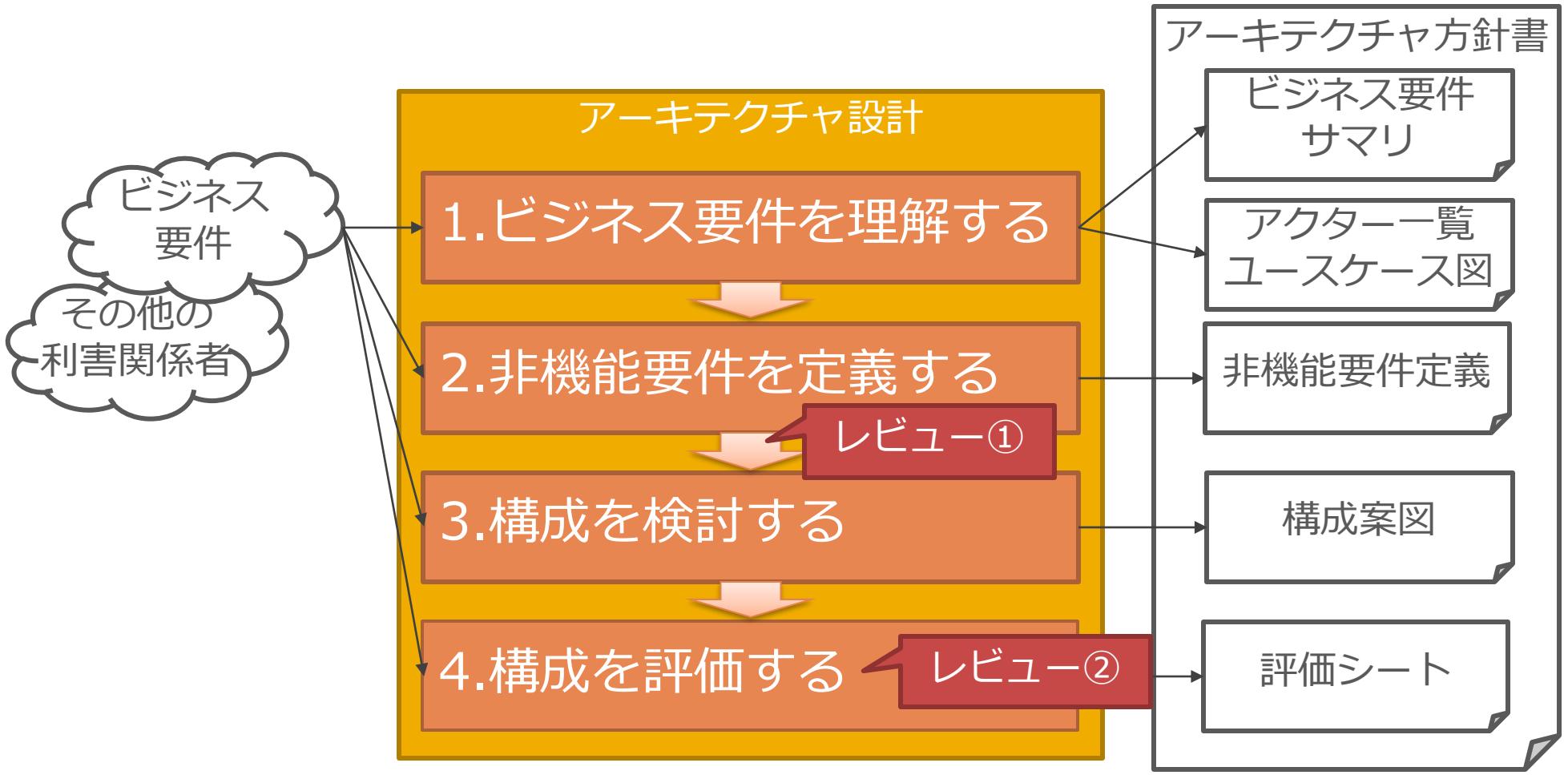
アーキテクチャ設計のレビュー

- アーキテクチャの設計工程でレビューを適切に行う
  - » アーキテクチャ評価まで行ってから気付いても遅いので



アーキテクチャ設計レビュー

設計プロセスと成果物



アーキテクチャ設計レビュー

2つの観点

- ①非機能要件の定義
  - » そのシステムに求められる非機能要件を定義する

- ②構成案の評価
  - » 構成案が非機能要件を適切に満たすのかを評価する

アーキテクチャ設計レビュー

①非機能要件を定義する

- 非機能要件として整理を行う
  - 様々な非機能定義がある
    - 例：経産省が提供するソフトウェアの品質メトリクスセット
      - ✓ http://www.meti.go.jp/policy/it_policy/softseibi/#p01
    - 例：IPAが提供する非機能要求グレード
      - ✓ https://www.ipa.go.jp/sec/softwareengineering/std/ent03-b.html
  - これが正解、というのは存在しないので自分で使いやすいリファレンスを持っていると便利

アーキテクチャ設計レビュー

①非機能要件を定義する - 機能適合性

- ビジネスチームとの調整になる可能性が高い項目
  - 非機能的な「すぐに」「早く」「確実に」などについては、なぜその要件が必要なのかを確認する
  - ステップ案などで段階的に改善できることを示すのもよい
- 基本的には技術的制約を前提にする
  - 技術に無理をさせると保守性が下がることが多い
  - 「できない」というよりは「コストが高い」という説明を

アーキテクチャ設計レビュー

①非機能要件を定義する -性能効率性/可用性

- クラウド/仮想化によって柔軟になった項目
  - ただし、いまだにレガシー環境はあるので気は抜かないこと
- キャパシティプランニングは時間軸を意識する
  - インフラを変更する時間軸（作業時間/費用確保時間）で考える
    - 3ヶ月で可能なら3ヶ月、1年ごとなら1年後、5年ごとなら5年後を考える
  - ビジネス側の要求精度は確認する
    - キャパシティが足りないのは問題だが、多すぎるのもエコではない

アーキテクチャ設計レビュー

①非機能要件を定義する -セキュリティ

- 最初に関連部署と関連法令の確認を行うこと
  - あとからセキュリティ関連の考え方が変わるのは厳しい
- 考えることが多岐にわたるため、どこの話かを整理する
  - 認証認可
  - データの保護（ストレージ、ネットワーク）
  - 不正アクセスやアタックへの対策、監査、監視

アーキテクチャ設計レビュー

①非機能要件を定義する -使用性

- 社内向けかコンシューマー向けでポイントが異なる
  - 社内向けは「必要十分」
  - コンシューマー向けは「差別化要素」
- 特にスマホやデバイスを利用する場合は注意が必要
  - 使用性確保のために技術的な制約が出る可能性がある
- マニュアル整備なども意識するとよい

アーキテクチャ設計レビュー

①非機能要件を定義する -保守性/移植性

- ライフタイムコストに影響するため、とても重要
  - 特にテスト性を高めるのは有効
  - システム連携において独立したスケジュールでテストができるかどうかは大きな違い
- ログ～監視の流れは標準化が必要
- 複数環境は意識する
  - ローカル、開発、検証、ステージング、本番

アーキテクチャ設計レビュー

①非機能要件を定義する – レビュー観点

- システムに対して適切に設定されているか？
  - 非機能的に抜けもれがないか？
    - 意図的な抜けもれも含めて明示することが大事
  - 将来性が考慮されているか
    - ライフサイクルにおける非機能要件の変化
  - ステークホルダーの意見が整理されているか？
    - この時点では、ある程度の矛盾や不可能性は許容する

アーキテクチャ設計レビュー

②構成案の評価

- 構成案を作成し、非機能要件から評価する
  - 可能なら3案作るのが望ましい
    - 可能な限り非機能要件を実現する案
    - 技術的制約やコストを最優先にした案
    - その間

No	構成案	非機能1	非機能2	非機能3	コスト
1	構成案1	○	○	○	×
2	構成案2	△	×	△	○
3	構成案3	△	△	○	△

アーキテクチャ設計レビュー

②構成案の評価 – レビュー観点

- バランスが取れた判断になっているか？
  - 100点満点になることはありえない
    - 非機能要件に矛盾が一切ない、ということがありえるか？
  - 落としどころの意図を確認する
    - 技術的オーバーキルではないか？
    - 「誰か」の意見に寄りすぎていないか？

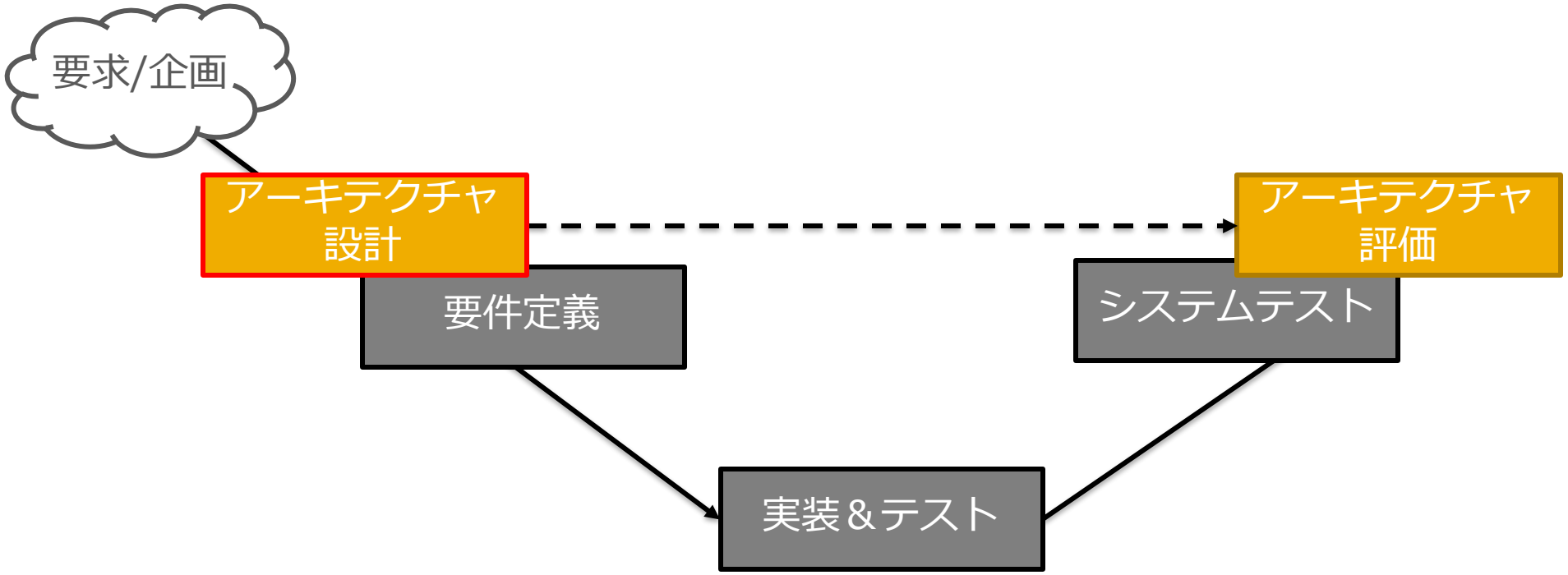
graat

アーキテクチャ設計レビューの難しさ

アーキテクチャ設計レビューの難しさ

アーキテクチャ設計は常に事前的である

- 何もないのに正しくレビューできるものなのか？
- つまり、常にリスク（不確実性）がある



アーキテクチャ設計レビューの難しさ

リスクと戦う

- 技術的リスク
  - 採用するコンポーネントが技術的に規定された品質が出るか？組み合わせたときに問題ないか？
- システムの成長リスク
  - システムのライフサイクルにおける成長がどうなるのか？

アーキテクチャ設計レビューの難しさ

技術的リスク

- 「その構成で非機能要件が満たせるか？」
  - 正しく○△×を付けられているのか？
- 「リスクに気付く」のは経験に寄るところが多い
  - 多くの人にレビューをしてもらうのがよい
- リスクがあるなら検証して明確化する

アーキテクチャ設計レビューの難しさ

技術的リスク

- リスクの検証方法や回避方法を理解しておく
  - すべてのリスクを検証している時間はない
  - どの程度のリスクであれば、どの程度の検証をするのか？
  - そもそも、リスクの許容も必要

補足：技術的リスク

リスク検証の種類

検証方法	実施内容	例
机上検証	初期段階から実施でき、コストも時間もかけないが検証精度としては最も低い。製品のドキュメントからアーキテクチャのコンセプトを理解する	• 事例やホワイトペーパーで、製品コンセプトを確認し、適用業務とのFit&Gapを検討する • 想定構成で有識者による確認を行う
PoC	Proof Of Concept/概念実証。技術的な課題に対して、コンセプトレベルでの実現性を確認する。非機能的な課題の洗い出しを行うことが目的となる。	• 利用を想定する技術的な要素について、その技術の実現性を確認するための簡易的な試作を作成する
プロトタイプ	試作品としてシステムを構築する。機能の流れを限定的に作成することが多く、その場合は、エラー処理などが考慮されないため、完全な検証にはならない。	• 動作する一部機能を作成し、部分的な環境（本番はサーバ4台だが1台のみなど）で動かしてみる
先行実装	設計工程の後半、あるいは実装工程の先頭で実機能を開発する。この結果、テスト工程を早めに行うことができる。本番環境を用いた検証を目的とする。	• プロジェクト計画上、先行して実装作業を行い、本番環境での性能試験を実施する • 本番環境で一通りの設定を行い、想定される動作をするかを早期にテストする
テスト	実装が開始されてから、非機能要件上の課題に対して検証可能な機能が作成できたら試験を行っていく	• システムテスト前の性能試験、セキュリティ検査 • 本番データを使った移行リハーサル

補足：技術的リスク

リスク対応の種類

リスク対応	判断基準	対応策の例
回避	リスク要因が根本的で、大きな対応コストがかかる場合などはリスクを回避する	・要件を取り下げる ・システム構成を不採用にする
受容	発生頻度が低い、影響が小さい、発生しても短時間で復旧するなどの場合はリスクを受容する	対応なし
代替	発生頻度が低いものの、対応コストが高く影響が大きい場合にはリスク発生時にとりうる代替手段を用意する	・システムを使わずにマニュアル運用にする ・データ転送を別経路（媒体渡し）にする
低減（要件）	主に要件要素によってリスクの発生を低減する。	・要件を部分的に変更してリスクの発生確率を減らす ・発生時にシステム機能が制限されることを許容するなど。
低減（システム）	主にシステム要素によってリスクの発生を低減する。	・コストをかけてリソース増強や構成変更を行う ・ベンダー側の体制強化を依頼する

アーキテクチャ設計レビューの難しさ

システムの成長リスク

- システムは成長するのか？衰退するのか？
  - 未来の予想は誰にもできない
  - 稟議書に書かれた数字が正しいかは「微妙」
- できれば、時点で最適化されているのが望ましい
  - 主にコスト面で（ビジネスの継続性に影響するから）
  - インフラは仮想化によって調整幅は増えた

アーキテクチャ設計レビューの難しさ

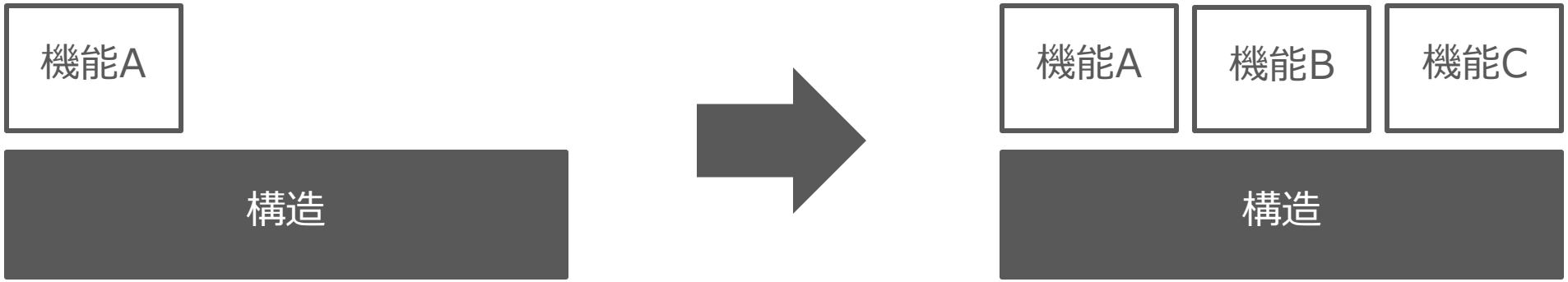
システムの成長リスク

- 未知なる未来に対して戦略があるか？
  - 予測型
    - 予測確度が高いことを前提に効率化する
  - 犠牲型
    - 予測確度が低いことを前提に非効率を受け入れる
  - 拡張型
    - 予測の曖昧さを受け入れる

アーキテクチャ設計レビューの難しさ

システムの成長リスク - 予測型

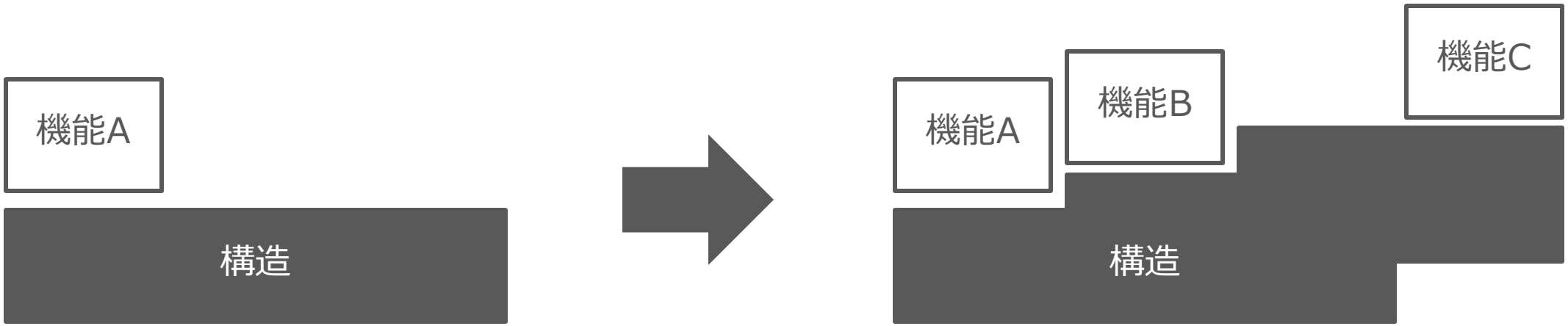
- 追加される機能に対して同一の構造を割り当てる
- 最も効率的（※予測が正しければ）



アーキテクチャ設計レビューの難しさ

システムの成長リスク - 予測増改築型

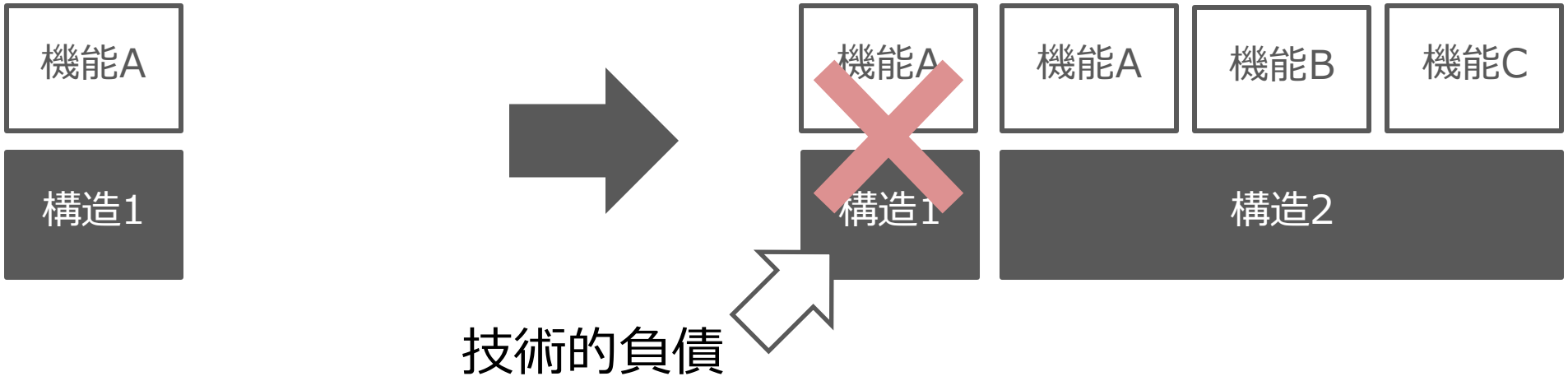
- 予測型を長期間維持してこじらせたパターン
- 保守性が悪く、コストが増加する（別名：温泉旅館型）



アーキテクチャ設計レビューの難しさ

システムの成長リスク - 犠牲型

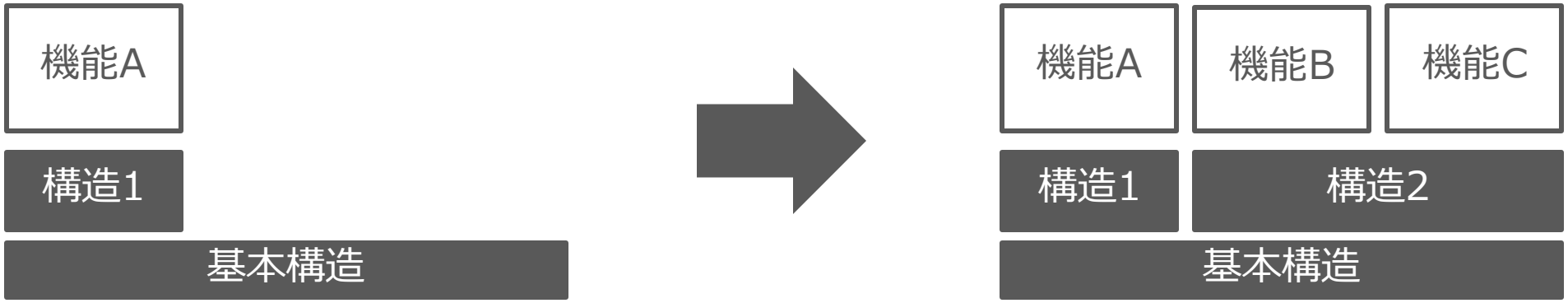
- 最初に作る構造は最低限にしておき、のちに再整備する
- 機能移植にコストはかかるが長期保守にメリット



アーキテクチャ設計レビューの難しさ

システムの成長リスク - 拡張型

- 構造そのものに拡張性を持たせる
- （※天才に限る）



graat

まとめ

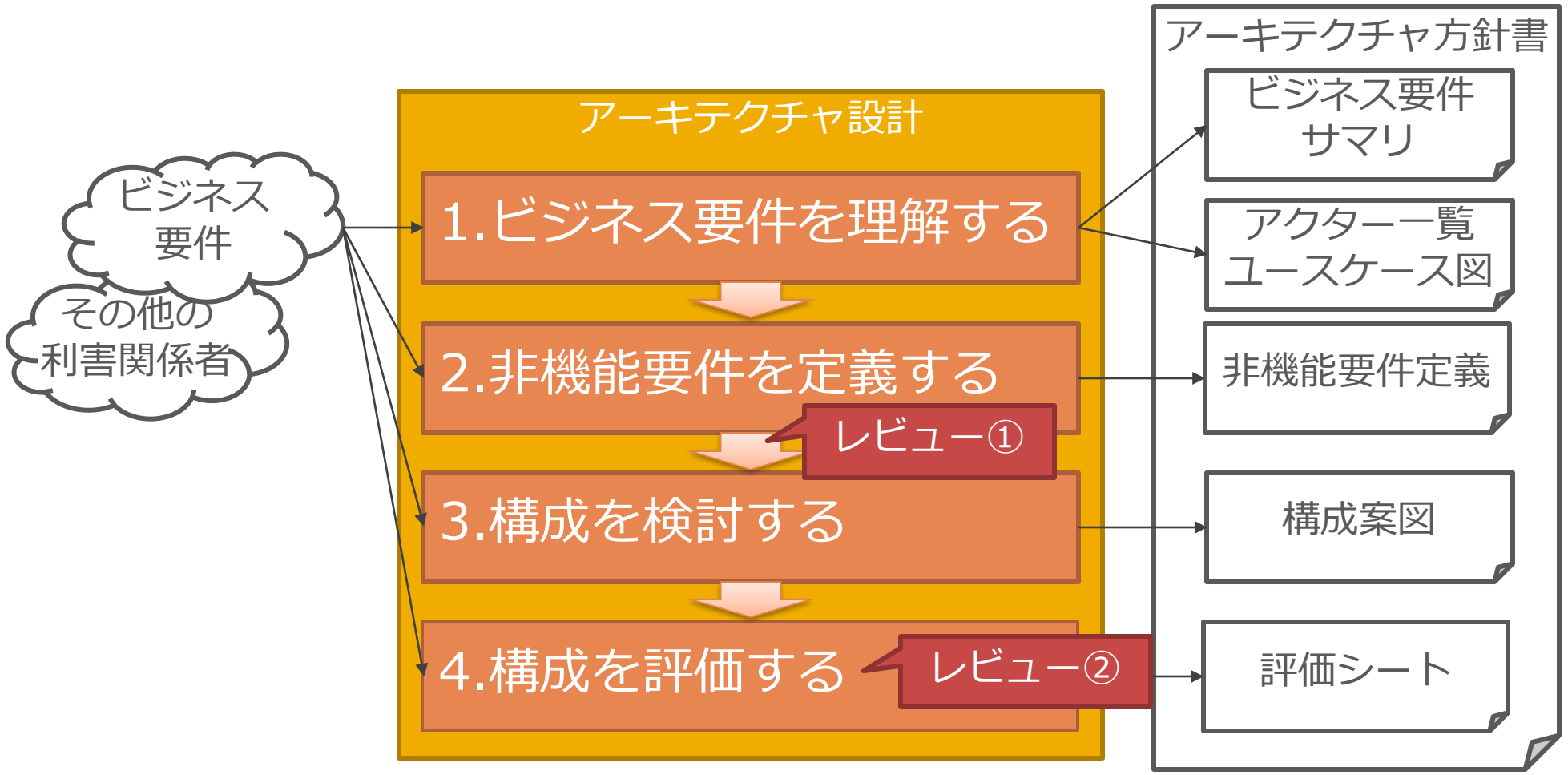
まとめ

アーキテクチャの役割

- 組み合わせでシステムを作る時代
  - まさに組み合わせを考えるアーキテクチャ設計は重要
- 非機能要件で良さを定義する
  - アーキテクチャがシステムの非機能を保証する
  - アーキテクチャの評価は「非機能の適切さ」で評価可能
    - 単体の品質ではなく、組み合わせて想定通りに動くのか？

まとめ

設計プロセスと成果物



まとめ

①非機能要件を定義する

- そのシステムに求められる非機能要件を定義する
- レビュー：システムに対して適切に設定されているか？
  - 非機能的に抜けもれがないか？
    - 意図的な抜けもれも含めて明示することが大事
  - 将来性が考慮されているか
    - ライフサイクルにおける非機能要件の変化
  - ステークホルダーの意見が整理されているか？
    - この時点では、ある程度の矛盾や不可能性は許容する

まとめ

②構成案の評価

- 構成案が非機能要件を適切に満たすのかを評価する
- レビュー：バランスが取れた判断になっているか？
  - 100点満点になることはありえない
    - 非機能要件に矛盾が一切ない、ということがありえるか？
  - 落としどころの意図を確認する
    - 技術的オーバーキルではないか？
    - 「誰か」の意見に寄りすぎていないか？

まとめ

アーキテクチャ設計は常に事前的である

- 何もないのに正しくレビューできるものなのか？
- つまり、常にリスク（不確実性）がある
  - 技術的リスク
    - 採用するコンポーネントが組み合わせたときに問題ないか？
    - リスクの検証方法と回避方法の理解
  - システムの成長リスク
    - システムのライフサイクルにおける成長がどうなるのか？
    - 戦略的な構成の理解

まとめ

アーキテクチャ設計レビューには信念が必要

- 必ずしも技術的に優れていることが重要ではない
  - すべてを理解するスーパーマンはいない
- むしろ、傾聴し、整理し、働きかける力が重要
  - より多くの人を判断に巻き込むことが大事
- 「誰かの意見」に従っていないか？が大事
  - リスクと戦うには信念がいる（誰かのせいにするは簡単）