アーキテクト～期待と現実～

JEITA 組込み系ソフトウェアワークショップ 2013
早稲田大学 創造理工学部 経営システム工学科
岸知二

基本編

ソフトウェアアーキテクチャ

- ソフトウェア設計の主要なテーマ
  - 各種の定義があるが本資料では以下のように定義

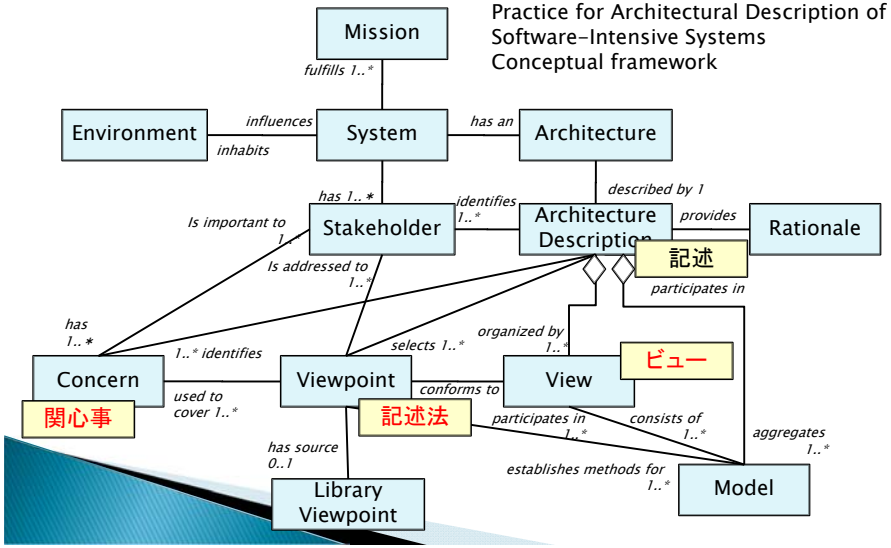
- ソフトウェアおよびその開発を支配するソフトウェア構造や構造化の原則
  - ソフトウェア開発の方針や方向付けを行うために必要なソフトウェア構造
  - 具体的な構造だけでなく構造化原則(抽象的な構造やガイドライン)を含む
  - 概念的な構造もソフトウェアアーキテクチャに含める

アーキテクチャ設計の難しさの例

- 多面的:
  - 様々なステークホルダーの関心事に対応
  - 関心事毎に捉える側面が違う(ビュー)
- 横断的
  - それぞれのビューは一般にクロスカッティング
  - どう構造化するかという手法が未成熟
- 変動的
  - 時間の中で環境と技術は変化する
  - しかし一旦具現化されたものは慣性をもつ

アーキテクチャ記述の概念フレームワーク

IEEE-Std-1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems Conceptual framework



ISO/IEC 9126

ISO/IEC 9126, 1991: Information technology – Software product evaluation – Quality characteristics and guidelines for their use
JIS X0129 ソフトウェア製品の評価 – 品質特性及びその利用要領

- 6つの評価観点と品質評価の基本プロセスを規定
  - 今後はISO/IEC 25000 SQuAREへと移行

品質特性	概要	副特性
機能性 functionality	明示的及び暗示的必要性に合致する機能を提供する能力	合目的性、正確性、相互運用性、標準適合性、セキュリティ
信頼性 reliability	指定された達成水準を維持する能力	成熟性、障害許容性、回復性
使用性 usability	指定された条件下で利用されるとき、理解・習得・利用でき、利用者にとって魅力的である能力	理解性、修得性、運用性
効率性 efficiency	使用する資源の量に対比して適切な性能を提供する能力	時間効率性、資源効率性
保守性 maintainability	修正のしやすさに関するソフトウェアの能力。修正は是正、向上、変更への適応を含んでもよい	解析性、変更性、安全性、試験性
移植性 portability	ある環境から他の環境に移すためのソフトウェア製品の能力	環境適用性、設置性、企画適合性、置換性

概念フレームワーク (1/2)

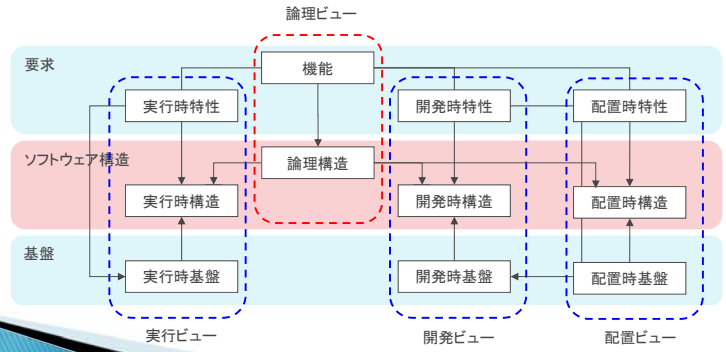
- 品質特性はビューと対応づけて理解できる
  - ビューに応じたソフトウェア構造と基盤が存在
  - 対応するステークホルダを考えると有用

ビューに応じた要求・基盤・ステークホルダの例

ビュー	要求	基盤の提供する構成要素	ステークホルダ
論理	機能	(基盤は存在しない)	顧客、ドメインエキスパート、開発者
実行	性能、可用性、信頼性	実行スレッド、通信チャネル、スケジュール、共有リソース	開発者、システムインテグレータ
開発	拡張性、修正容易性、再利用性	モジュール、インタフェース	開発者、テスタ、ライブラリアン、管理者
配置	変更容易性	定義ファイル、実行ファイル	システムエンジニア、開発者、保守担当者、運用者

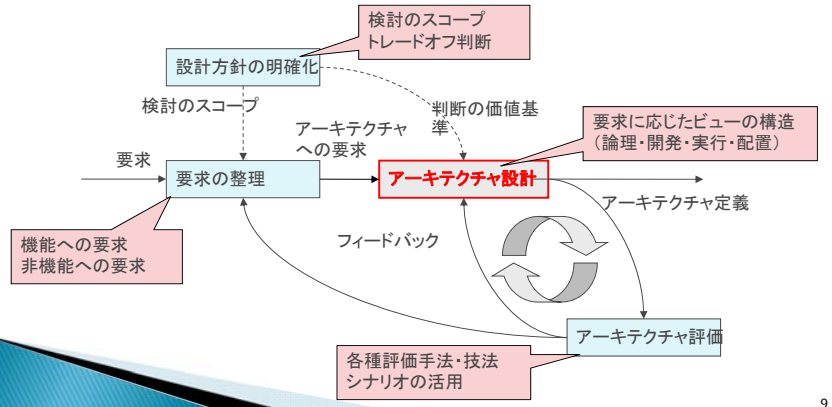
概念フレームワーク (2/2)

- 要求、ソフトウェア構造、基盤の関係を模式的に示したもの(Ranが提唱、一部改変)



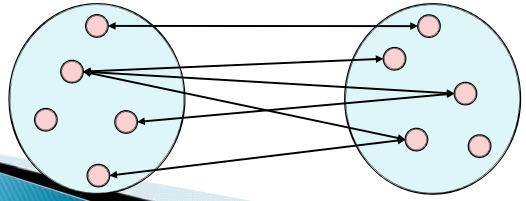
アーキテクチャ設計の枠組み

- アーキテクチャへの要求と設計方針に基づき、複数の観点から設計と評価を繰り返す



クロスカッティング

- Scattering:
  - ひとつの関心事がある表現上で散在すること
- Tangling:
  - 複数の関心事がある表現上でからみあうこと
- Crosscutting
  - ふたつの表現間で、関心事のScatteringやTanglingが起こっていること



多元的な関心事の分離

- Multi-Dimensional Separation of Concerns
  - 世界をひとつの視点で捉え、ひとつの階層でモデル化することは困難
  - 支配的な関心事（構造)をもたず、各関心事は独立して存在
    - “tyranny of the dominant decomposition”
  - 各関心事は重複したり関係を持ったりする
- サブジェクト指向、Hyper/Jなどの流れ
  - ハイパースライス(hyperslice)
    - 関心事をカプセル化する手段
  - 合成規則(composition rule)
    - ハイパースライスがどのようにひとつのハイパースライスに合成されるかを示す

Tarr, P, et al. “N Degrees of Separation: Multi-Dimensional Separation of concerns”, Proceedings of ICSE ’99. Most Influential Paper (2009)

ソフトウェアの進化

- ソフトウェアは時間とともに進化する
  - 新たな要求や技術変化が起こると、レガシーなシステムをその環境に適合するように再設計するなどする必要。
  - ソフトウェア保守の必要性

As large-scale programs such as Windows and Solaris expand well into the range of 30 to 50 million lines of code, successful project managers have learned to devote as much time to combing the tangles out of legacy code as to adding new code. Simply put, in a decade that saw the average PC microchip performance increase a hundredfold, software‘s inability to scale at even linear rates has gone from dirty little secret to industry-wide embarrassment. (Williams, Sam. 2002)

レガシーソフトウェア

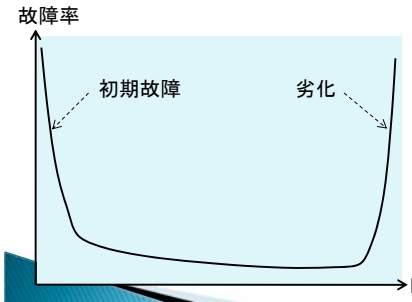
Legacy software

- 過去に作られいまだに業務の中核を支え、不可欠なソフトウェア
- 多くの場合品質に問題がある
  - 拡張による複雑化、ドキュメントの不備、不十分な変更管理、など
- 変更を加える必要性が生じると様々な困難を伴う
  - 技術やビジネスの変更、相互運用性の確保など
- どのように進化させるかが課題に

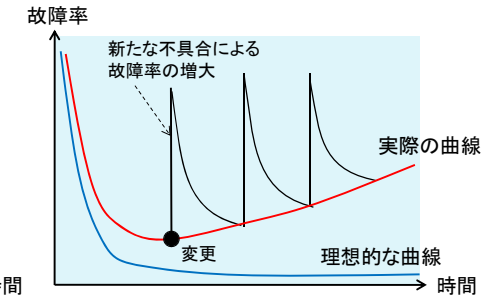
故障率曲線

- ハードウェアとソフトウェアとでは異なる

ハードウェア
・劣化により「バスタブ曲線」を描く



ソフトウェア
・本来の定常になる前に変更が加えられ
徐々に故障率の最小値レベルが上がる



ロジャー S.プレスマン著、西康晴他監訳、実践ソフトウェアエンジニアリング、日科技連

アーキテクチャ不整合

- 再利用資産はそれが使われるアーキテクチャ上の想定を持っており、使う側のアーキテクチャがそれと整合しないと再利用が困難になる
  - 再利用を行う際にはこのアーキテクチャ不整合を起こさないように気をつけなければならない

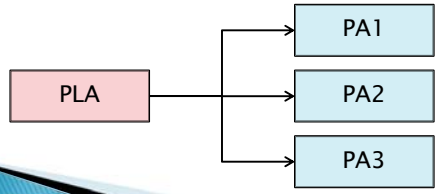
アーキテクチャ不整合の例

・扱うデータの構造や表現が異なる
・コンポーネント間の通信モデルが異なる
・データフロー処理なのに部品はイベントドリブンを想定
・オンデマンドで生成したいのに、起動時に生成が必要
・利用するOSやミドルウェアが違う

Garlan, D. et al: Architectural Mismatch: why reuse is so difficult, IEEE Software, 1995.

ソフトウェアプロダクトライン

- 共通の管理された特徴を持ち、特定のマーケットやミッションのために、共通の再利用資産に基づいて作られる、ソフトウェア集約的なシステムの集合 (SEIの定義)
- プロダクトラインは共通のアーキテクチャ上で構築
  - プロダクトラインアーキテクチャ: すべてのプロダクトのベースとなる包括的なアーキテクチャ



PLA: Product Line Architecture
PA: Product Architecture

ソフトウェアエコシステム

- 複数のベンダが異なった目的で関わる
- アーキテクチャの良し悪しが極めて重要

"A software ecosystem consists of a software platform, a set of internal and external developers, a community of domain experts and a community of users that compose relevant solution elements to satisfy their needs
"A software ecosystem provides the set of solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions"
(Jan Bosch)

アーキテクチャの効用

- アーキテクチャはなぜ重要か
  - 品質特性に大きな影響
- だが、何をするかによって利きが違う
  - 単発システム： まあ重要
  - 再利用、プロダクトライン： かなり重要
  - プラットフォーム・エコシステム： 致命的に重要

組織編

アーキテクチャ設計と組織

- アーキテクチャ設計の役割を規定するのは組織
  - 役割は体制とプロセスの中で位置づけられる
- 一般に予算はプロジェクト単位でつく
  - プロジェクト横断的な問題は運用が難しい
  - 予算面、権限面、指示系統面..
- 複数プロジェクトにまたがる組織問題は難しい
  - 再利用資産ですら、うまく使ってもらえない
  - アーキテクチャが抽象構造ならなおさら

再利用の成熟度

- 再利用の導入や進化の状態を表すもの
  - RiSEなどが提案
- 再利用などの実践には組織の成熟度が必要
  - アーキテクチャ設計にも類似性

Ad-hoc	Basic	Initial	Organized	Systematic
個人的活動 散発的	部分的適用 技術的活動	組織が認識 規則・標準	開発組織横断 プロセス統合	企業横断 戦略活用

→ より高い

RiSE: Reuse in Software Engineering, 再利用に関するブラジルの研究機関

組織の文化

- 価値観
  - 何を重視し、何にどこまで投資するか
- アーキテクチャの価値
  - 即効性というよりも中長期的視野での価値
  - コスト削減というよりリスク管理の投資

- アーキテクチャのどういう側面に価値を見出すか
  - 中期的な価値、リスク管理的な側面にコストを払う体質か

育成編

アーキテクト

- アーキテクチャの設計・徹底・保守

	役割	知識	資質
技術面	モデリング、トレードオフ分析、プロトタイピング技術、トレンド分析	分野知識、技術的な成功要因、開発手法・モデリング	創造性、調査能力、洞察、現実的、抽象化能力、リスク管理
ビジネス面	ビジネス戦略の技術戦略への反映、競合・市場動向把握	組織のビジネス戦略と実践、競合のビジネス戦略と実践	将来への洞察、企業的発想
組織面	コミュニケーション、ビジョンの徹底、ビジョンの維持	組織の力学、組織的・個人的な要求	他の立場からの視点、論理的、建設的
コンサルティング面	開発者のニーズを引き出す、開発者をサポートする、メンタリング	ヒアリング・コンサルティング技術	他人の成功に協力できる、人柄、よき教師

Bredemeyer, D.などより

人の育成

- SEI, SEC等体系、既に充実したもの多数
  - 非常に多くの知識やスキルが求められている
  - むしろtoo much?
  - その組織にとってのミニマムが重要
- 組織がアーキテクトを役割として定義しているか
  - 組織が定義できないなら育てられるか？
  - 個人の側のスキルアップの問題だけではない
- 役割の成果を評価しているか
  - 「いると便利」というだけで育つのか
  - 中期的な効果をどう評価するのか

組織の育成

- 組織の重要性
  - 役割は組織の中で定義される
  - 組織のアーキテクチャが悪いと役割は活きない
- アーキテクチャ設計に適した組織か
  - アーキテクチャを作り維持するコストを認めている
  - アーキテクチャを守らせる権限を認めている
- アーキテクチャの価値を認める組織か
  - 組織の文化やメンタリティ

方向性

- 目的をクリアに絞り込む
- 求めるミニマムから攻める
  - 固有技術エキスパート
  - ドメインエキスパート
  - ソフトウェアエンジニアリングエキスパート
- 個人と組織の両方の育成が必要
  - 役割を定義する
- 文化は簡単には変わらない
  - が、時間を書ければ変わる

まとめ

- アーキテクチャ設計はチャレンジング
- アーキテクチャ設計の効用は時間・組織横断的
- アーキテクトは組織の中の役割
- 組織のアーキテクトなくしてアーキテクト育たず