2023年度
ソフトウェア工学
ソフトウェアメトリクス

2024年 1月15日

渥美 紀寿 (情報環境機構)

京都大学

参考図書

- R. S. Pressman著，古沢聡子，正木めぐみ，関口 梢 訳:
  実践ソフトウェアエンジニアリング
- T. DeMarco著，渡辺純一 訳:
  品質と生産性を重視したソフトウェア開発プロジェクト技法

ソフトウェアメトリクスの用語

- 属性
  - 定量的または定性的に識別可能な特徴
- 測定
  - 属性に対し，数値や記号を対応付ける操作
- 測定量
  - 測定の結果
- 特性
  - 1つまたは複数の属性をある観点から整理し，分類
- 尺度
  - 属性と測定値を対応付ける基準
- 測定方法
  - ある尺度に対して属性の定量化に使用する一連の操作
- メトリクス
  - 尺度と測定方法を組み合わせた概念

ソフトウェアメトリクスの目的

- ソフトウェア開発プロジェクトを成功させる
- ソフトウェアの品質を向上させる
- ソフトウェア開発の生産性を向上させる

品質の評価

- ワインのテイスティングコンテスト
- タレントのオーディション

同じ条件で定められた基準によって比較

⇒ 主観的になるため，エキスパートが評価

- ソフトウェアの品質

⇒ 主観的では意味が無い

⇒ 定量的に評価することが必要

ソフトウェアの品質特性 (ISO/IEC 9126-1)

- 機能性
  - ソフトウェアがユーザ要求を満たしている程度
- 信頼性
  - ソフトウェアの使用可能時間 (障害の許容性，回復性)
- 使用性
  - ソフトウェアの使い勝手の良さ
- 効率性
  - ソフトウェアがシステム資源・時間を最適に利用している程度
- 保守性
  - ソフトウェアの変更容易性，安定性，試験性
- 移植性
  - ソフトウェアの移植容易性

測定の原則

- 定式化
  - 対象ソフトウェアを適切に表す尺度やメトリクスの導出
- 収集
  - 定式化プロセスで導出されたメトリクスを得るために必要なデータの蓄積および機構の整備
- 解析
  - メトリクスの算出および数学的なツールの適用
- 解釈
  - メトリクスの評価および品質に与える影響への考察
- フィードバック
  - メトリクスの理解にもとづいた，開発チームへのアドバイス

測定方法

- 直接測定可能なメトリクス
  - コードの行数
  - 開発期間
  - 機能数
  - クラス/メソッド数
- 間接的なメトリクス
  - 生産性: コード行数/開発期間
  - 欠陥率: 欠陥数/コード行数

尺度と測定結果の比較

- メトリクスごとに尺度が定まっており，尺度ごとに比較方法が違う
- 名義尺度 (Nominal Scale)
  - 値間に区別があるだけで大小関係はない
  - 例：性別 = 男 or 女
- 順序尺度 (Ordinal Scale)
  - 順序にのみ意味がある
  - 例：競技の順位 (1位と2位の差は3位と4位の差は同じとは言えない
- 間隔尺度 (Interval Scale)
  - 間隔の大きさには意味があるが，何倍とかは言えない
  - 例：摂氏温度 20℃は10℃より2倍暑いわけではない
- 比率尺度 (Ratio Scale)
  - 間隔の大きさにも，比率にも意味がある
  - 例：絶対温度 20K は 10K の 2倍の熱量

GQMパラダイム

- Basili らによって提案
- ソフトウェア測定・評価のための総合的な枠組み
  1. 測定の目標(Goal)を 明確に
  2. 目標の評価方法または達成方法を，質問(Question) の形式で記述
  3. 質問に定量的に答えるための測定量(Metric)を明確化
- 目標，質問および測定量の階層構造をGQMモデルとして表すことにより，ソフトウェア測定および評価の目的を明確化

測定の目標 (Goal)

下記について明確にしたもの

- 測定対象
  - プロダクト
    - 要求仕様書，設計書，プログラム，テストデータ...
  - プロセス
    - 仕様作成，設計，コーディング，テスト...
  - 資源
    - 作業者，ハードウェア，ソフトウェア，作業空間...
- 測定理由
- 品質モデル
- 視点
- 環境

目標の評価方法 (Question)

- 個々の目標を評価，あるいは，達成する方法を明確にしたもの
- 測定対象 (プロダクト，プロセス，資源) の特性を品質の観点から明らかにすることができる

測定量 (Metric)

- 目的の評価方法に対して定量的に答えるためのデータ集合
- 主観的データ
  - 例：ドキュメントの信頼性，顧客満足度のレベル...
- 客観的データ
  - 例：ドキュメントのバージョン数，作業工数，プログラムサイズ...

ソフトウェアの不確実性

- 論理的構造物ゆえの不確実性
  - 不可視性 (目に見えない)
  - 変更容易性 (簡単に変更可能)
  - 任意性 (自然法則から自由)
- 知的生産物がゆえの不確実性
  - 人間依存性 (開発者の能力に依存)

メトリクスの用途

- ソフトウェア開発の状況確認
- 開発コスト，リリース日，品質などの特性を予測
- ユーザの要求や目標を定量的に提示
- ソフトウェア開発の利害特質を分析
- プロジェクトの工数，品質の見積り
- プロジェクトの再計画
- リソース割り当ての計画
- プロジェクトの改善評価

現実

- 資源(人，物，金，時間) の制約に機能，性能，品質を合わせる
  - 工数 = プロジェクトチームの人数 x リリース日までの期間
- ハードウェアの出荷日にソフトウェアの完成日
- ベテランプログラマの経験に基づく開発規模と生産性の見積り

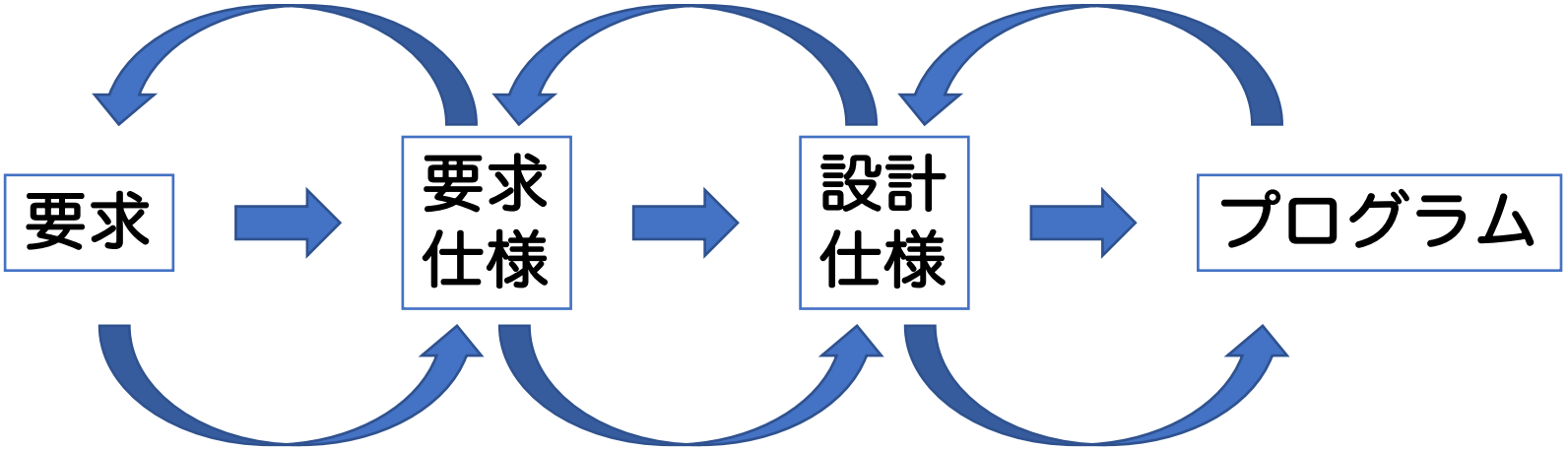
ソフトウェア開発とソフトウェア検査

- ソフトウェア開発
  - 要求に基づいてプログラムを実装
    (要求分析，設計，コーディング)

- ソフトウェア検査
  - ソフトウェア検査 (レビュー，テスト)
    - 各工程間，プログラムと仕様間の整合性を確認

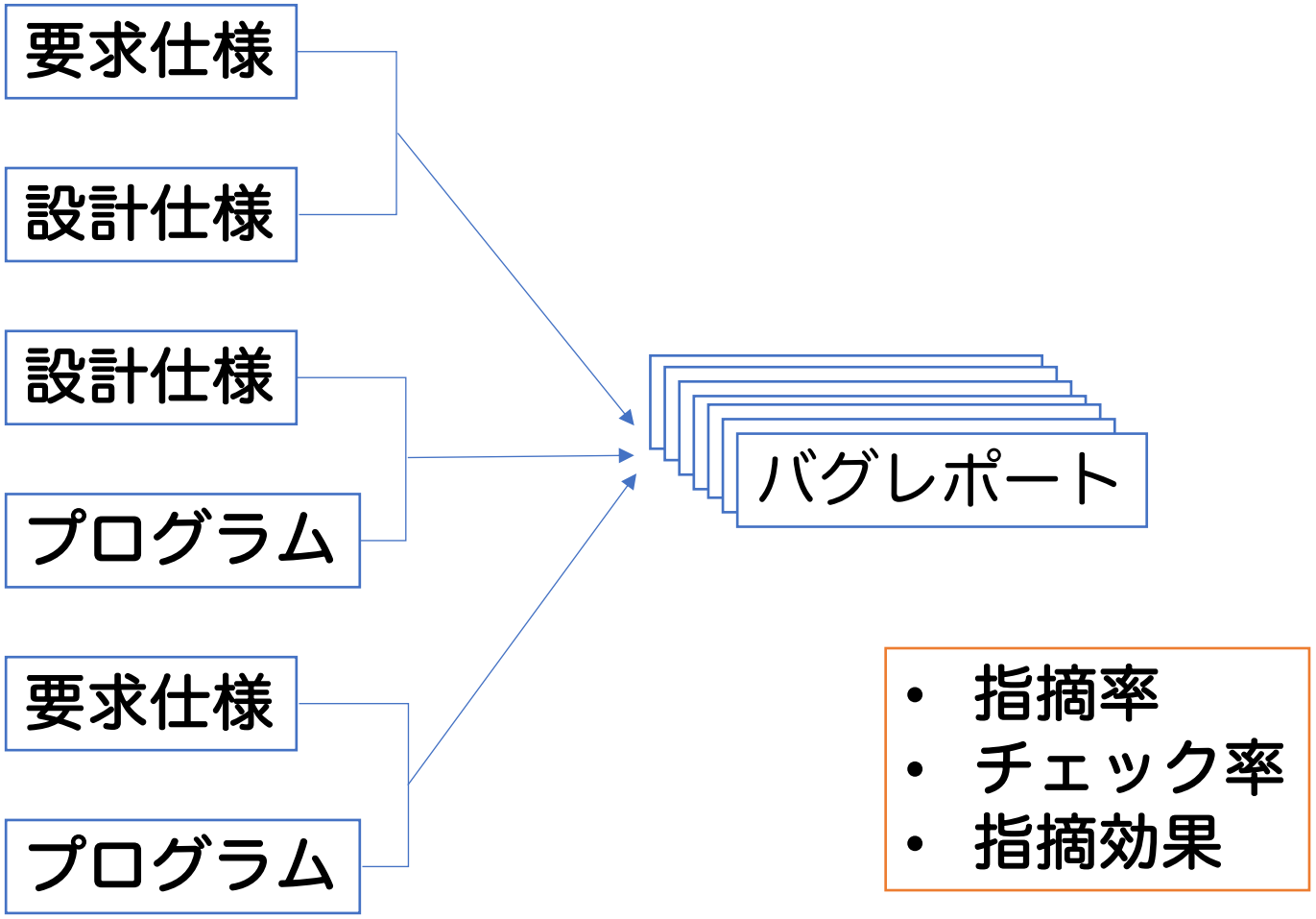
開発工程間における評価

- 正しく変換されているか
- 漏れなく変換されているか



- 変換に必要に情報が記載されているか
- わかりやすいか

ソフトウェア検査における評価



ソフトウェアメトリクスの例

- 要求メトリクス
  - ファンクションポイント
  - ユースケースポイント
- プロセスメトリクス
  - 作業時間/期間
  - 工数
- 設計メトリクス
  - 凝集度 (強度)
  - 結合度
  - C&Kメトリクス
- ソースコードメトリクス
  - McCabe のサイクロマチック数
  - Halstead のソフトウェアサイエンス
  - SLOC (SourceLines of Codes)

ソフトウェアメトリクスの例

- 要求メトリクス
  - ファンクションポイント
  - ユースケースポイント
- プロセスメトリクス
  - 作業時間/期間
  - 工数
- 設計メトリクス
  - 凝集度 (強度)
  - 結合度
  - C&Kメトリクス
- ソースコードメトリクス
  - McCabe のサイクロマチック数
  - Halstead のソフトウェアサイエンス
  - SLOC (SourceLines of Codes)

要求メトリクス

- ファンクションポイント
- ユースケースポイント

ファンクションポイント (FP)

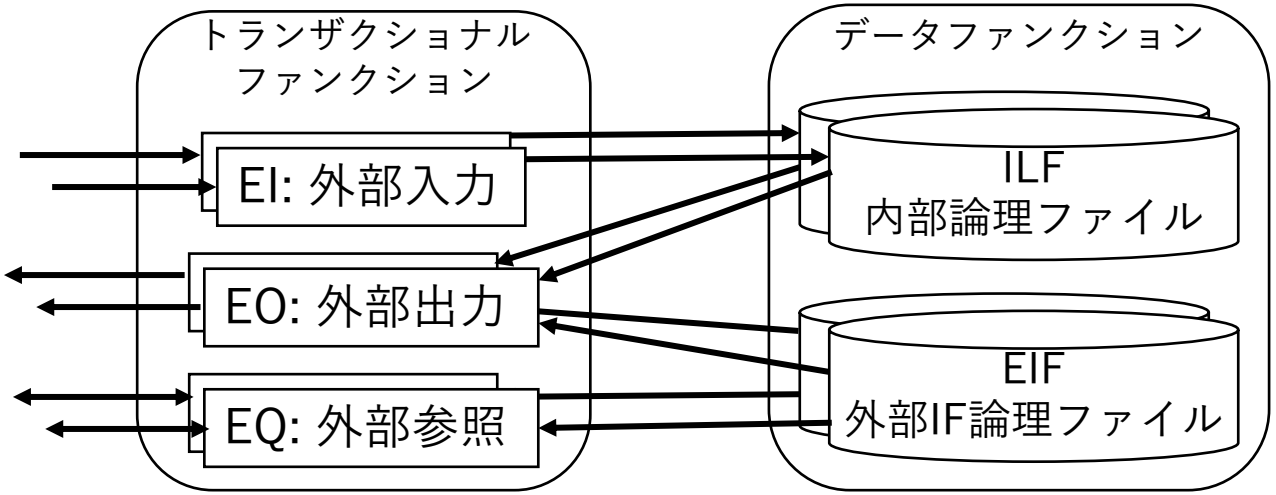
- Function Point
- ソフトウェアがどのようにできているか(行数等)ではなく，何をしてくれるかに基づき，ソフトウェアの規模を測るメトリクス
- 要求や仕様を入力として，対象ソフトウェアがどの程度大きいかが分かる
- 大きいソフトウェアなら，開発に時間や費用がかかる

FPの基本

- 仕様書を見て，システムの入力，出力，インタフェース，データベース等を見つける
  - これを構成要素とする
- 構成要素ごとに容易，普通，複雑の点数をつける
- つけた点数の合計が仕様に基づくシステムの規模
- システム全体の複雑さ等を考慮した係数をかける場合が多い
- IFPUG法，NESMA法，COSMIC-FFP法など様々な評価方法が提案されている

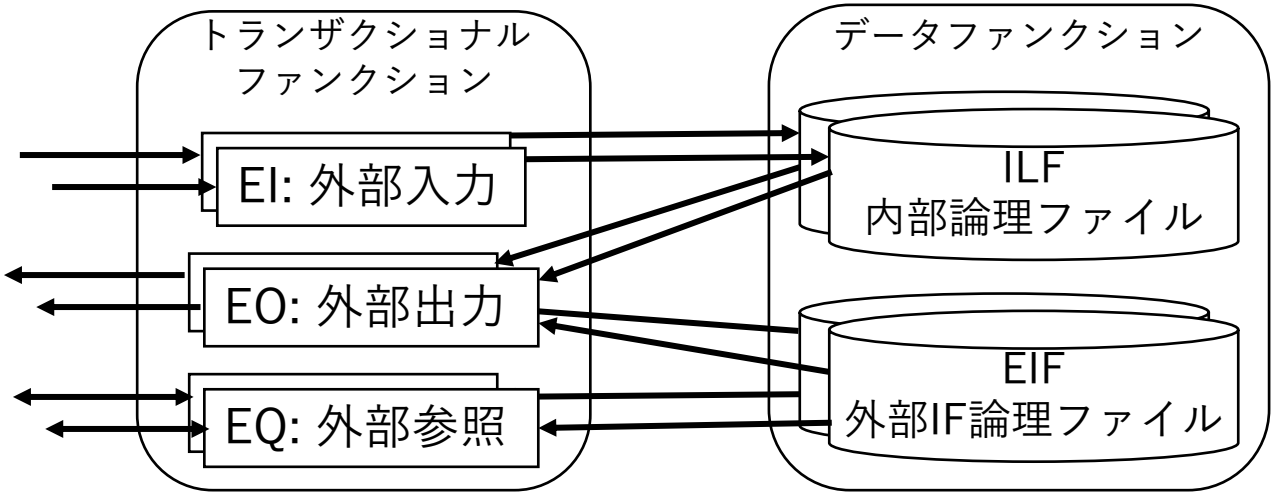
FPの基本要素

- トランザクショナルファンクション
  - 外部入力: (EI: External Input)
  - 外部出力: (EO: External Output)
  - 外部参照: (EQ: External inQuiry)
- データファンクション
  - 内部論理ファイル: (ILF: Internal Logical File)
  - 外部インタフェースファイル: (EIF: External Interface File)



IFPUG法によるFPの計算

- $FP = \Sigma E_i$
- E_i: EI, EO, EQ, ILF, EIF の個々の要素の複雑度
- 複雑度を定義した表が IPPUG で定義されている
  - データファンクション：レコード種類数（RET：Record Element Type）とデータ項目数（DET：DataElement Type）で決まる
  - トランザクションファンクション：関連ファイル数（FTR：File Type Referenced）とデータ項目数で決まる



FP試算法 (NESMA試算法)

- FP試算値＝35×ILFの個数＋15×EIFの個数
- 35，15という係数は，ILF，EIFのそれぞれに平均的に付加されるトランザクションファンクションの数を仮定
- 要求分析の早期段階で測定可能

FP概算法 (NESMA概算法)

- 各機能の複雑度は評定せずに下記仮定の元計測
  - データファンクション
    複雑度: 低
  - トランザクショナルファンクション
    複雑度: 中
- FP試算法より開発が進まないと評価できない

ユースケースポイント (UCP)

- Use Case Points
- FP の考え方をユースケース図，ユースケース記述に適用して，システムの規模を測定
- 基本的には，それぞれのアクター，ユースケースを単純，普通，複雑に分類し，点数付けして合計を計測する
  - 未調整 UCP (UUCP) と呼ばれる
- 技術や環境の要因を考慮して UUCP に係数をかける

ソフトウェアメトリクスの例

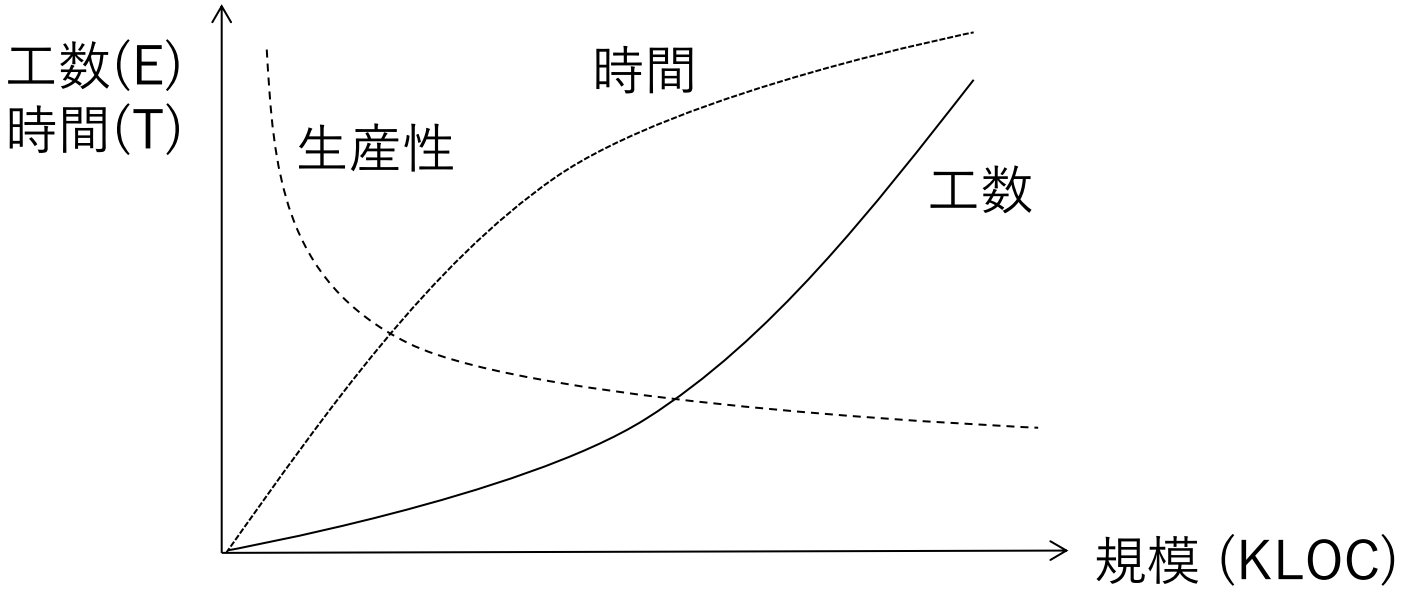
- 要求メトリクス
  - ファンクションポイント
  - ユースケースポイント
- プロセスメトリクス
  - 作業時間/期間
  - 工数
- 設計メトリクス
  - 凝集度 (強度)
  - 結合度
  - C&Kメトリクス
- ソースコードメトリクス
  - McCabe のサイクロマチック数
  - Halstead のソフトウェアサイエンス
  - SLOC (SourceLines of Codes)

プロセスメトリクス

- 工数の見積り
- 作業時間/期間の見積り

工数・時間と規模の相関

- COCOMOモデル[1]
  - ソフトウェア規模を基礎とする経験的モデル
- 工数: $E = a \times (KLOC)^b$ [$a > 0$, $b > 1$]
- 時間: $T = c \times (E)^d$ [$c > 0$, $0 < d < 1$]



[1]B. W. Boehm: Software Engineering Economics, Prentice-Hall (1981)

Basic COCOMO による工数見積り

- 工数は規模に対する複雑度の増大の割合に比例
  - 独立型：比較的小規模で複雑度の低いソフトウェア
  - 組合わせ型：規模，複雑度が中程度のソフトウェア
  - 組込み型：組込みソフトウェア，設計上の制約が厳しい

対象ドメイン	工数見積り	期間見積り
独立型	$E=2.4(KLOC)^{1.05}$	$T=2.5(E)^{0.38}$
組合わせ型	$E=3.0(KLOC)^{1.12}$	$T=2.5(E)^{0.35}$
組込み型	$E=3.6(KLOC)^{1.29}$	$T=2.5(E)^{0.32}$

ソフトウェアメトリクスの例

- 要求メトリクス
  - ファンクションポイント
  - ユースケースポイント
- プロセスメトリクス
  - 作業時間/期間
  - 工数
- 設計メトリクス
  - 凝集度 (強度)
  - 結合度
  - C&Kメトリクス
- ソースコードメトリクス
  - McCabe のサイクロマチック数
  - Halstead のソフトウェアサイエンス
  - SLOC (SourceLines of Codes)

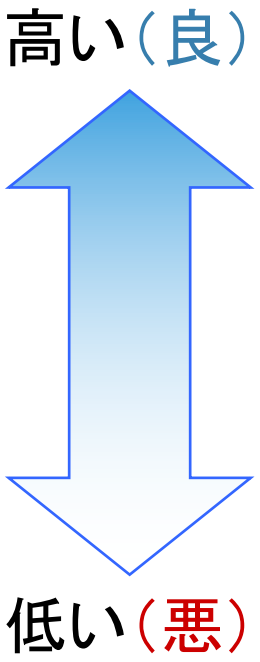
設計メトリクス

- （モジュール内）凝集度 (強度)
- （モジュール間）結合度
- システム複雑度
- C&Kメトリクス
  - オブジェクト指向設計に対する複雑さメトリクス
  - クラス内部，継承，結合の点から評価する

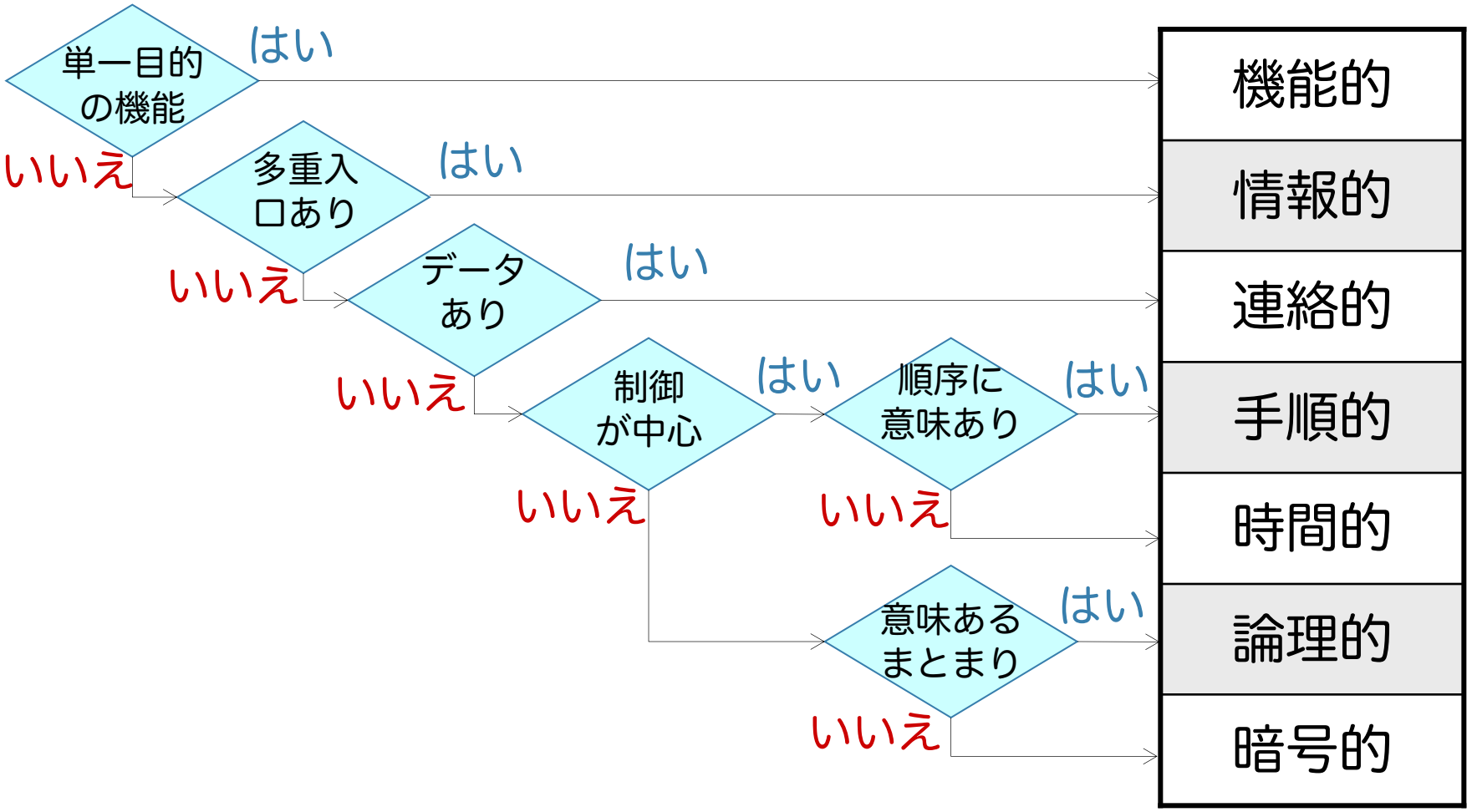
凝集度 (強度)

- モジュールが単一の目的のみを有しているか

1. 機能的
2. 情報的
3. 連絡的
4. 手順的
5. 時間的
6. 論理的
7. 暗号的

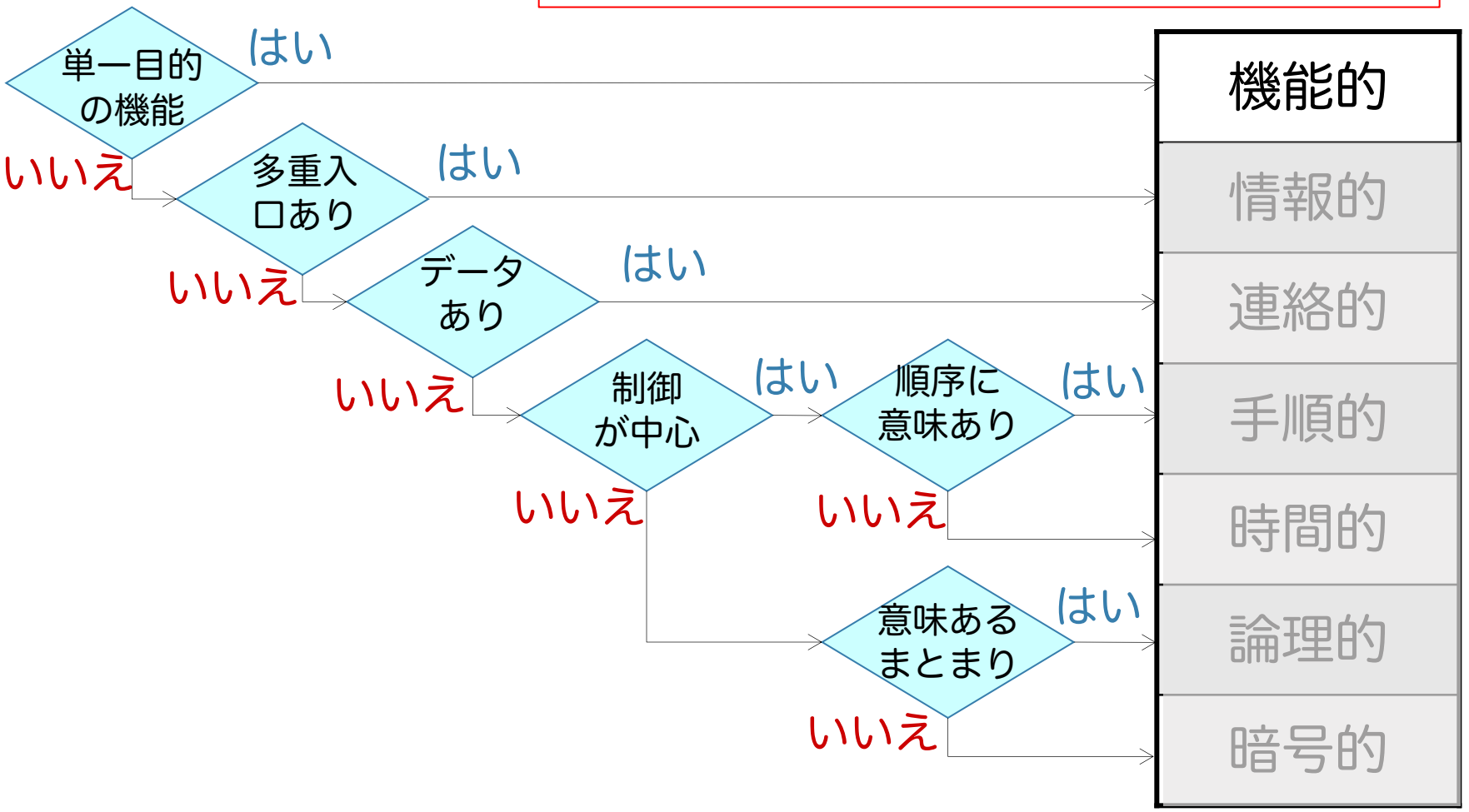


凝集度の識別


単一目的の機能
はい
機能的
いいえ
多重入口あり
はい
情報的
いいえ
データあり
はい
連絡的
いいえ
制御が中心
はい
順序に意味あり
はい
手順的
いいえ
いいえ
時間的
意味あるまとまり
はい
論理的
いいえ
暗号的

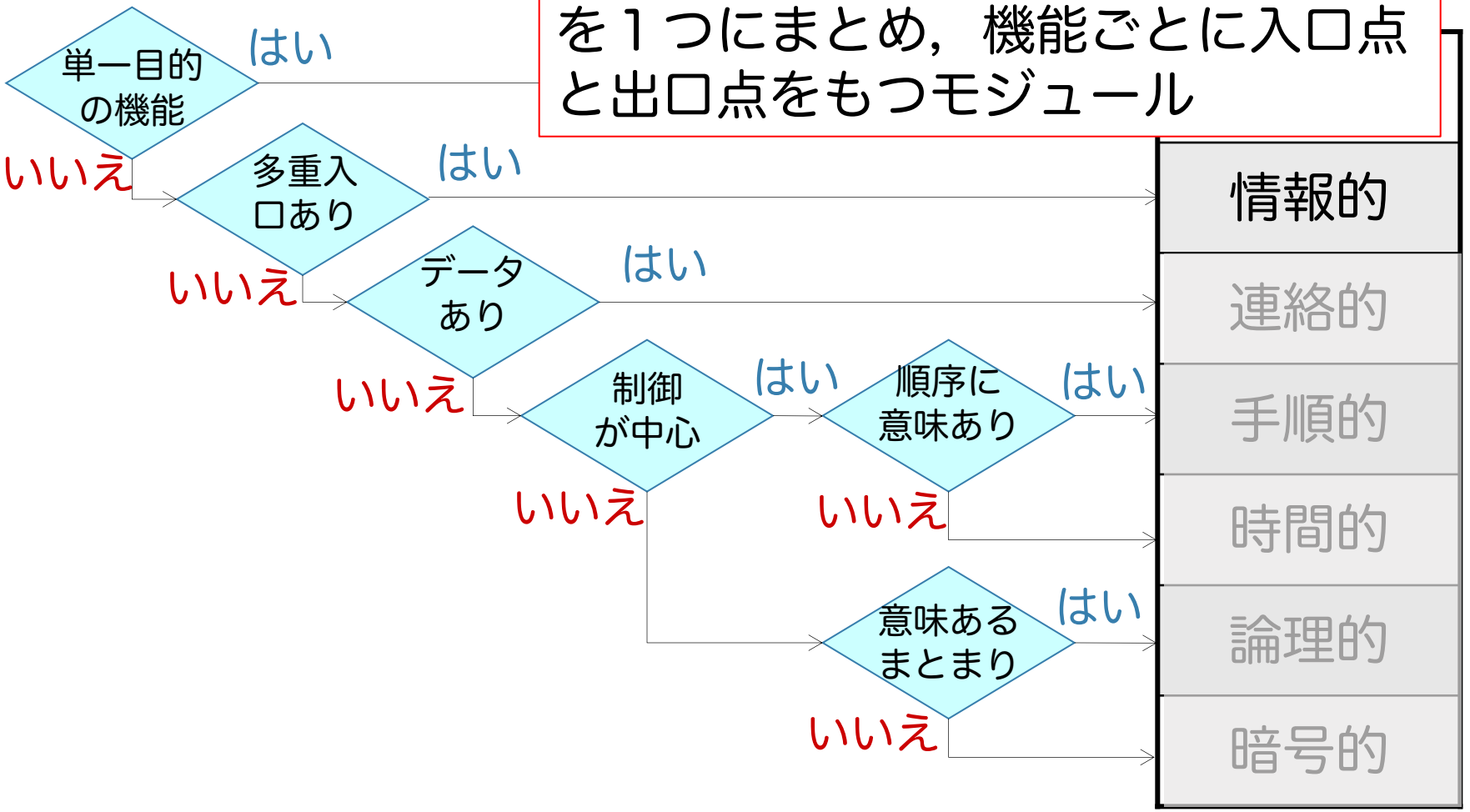
凝集度の識別

１つの機能だけからなるモジュール

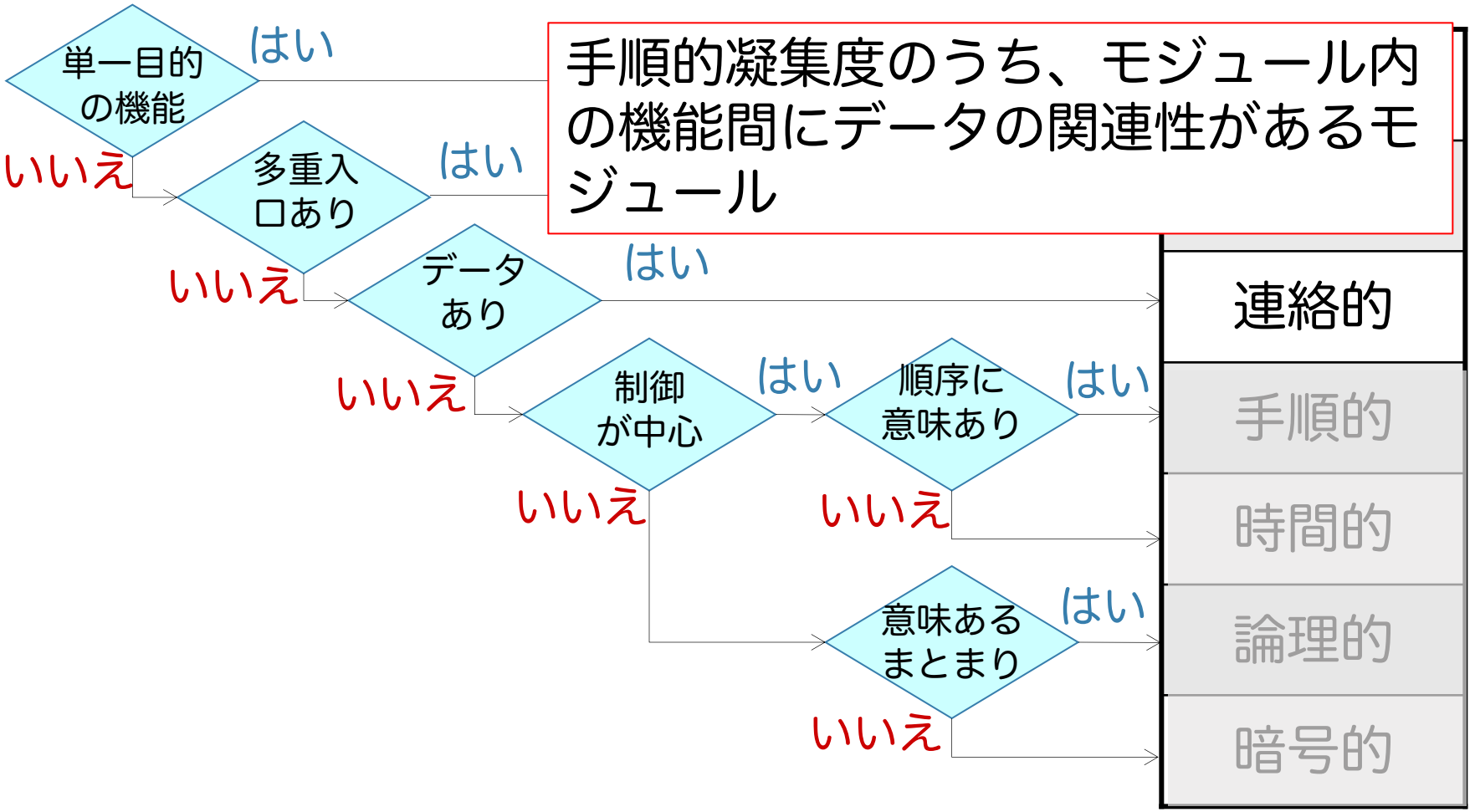


凝集度の識別

同一のデータ構造や資源を扱う機能を１つにまとめ，機能ごとに入口点と出口点をもつモジュール

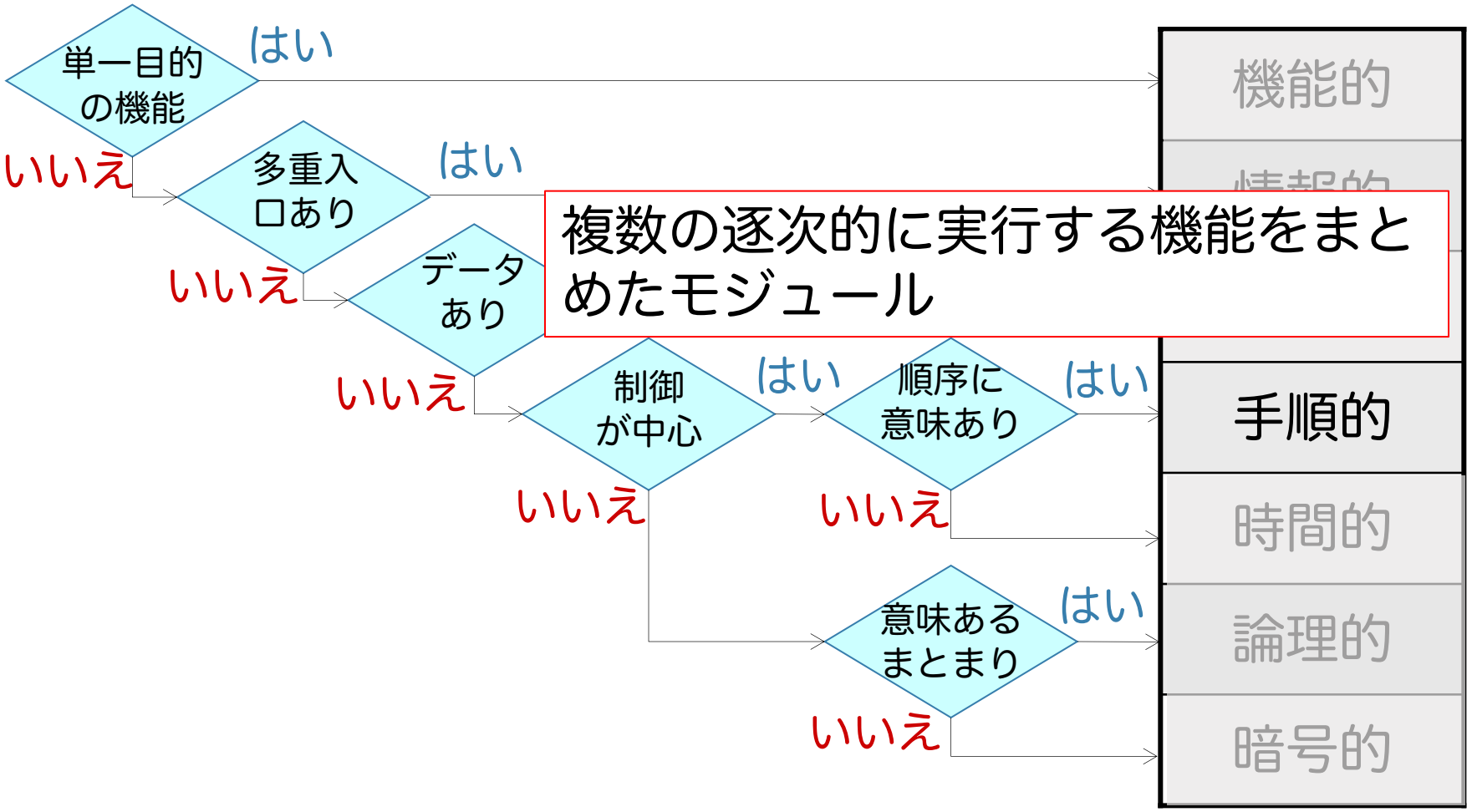


凝集度の識別

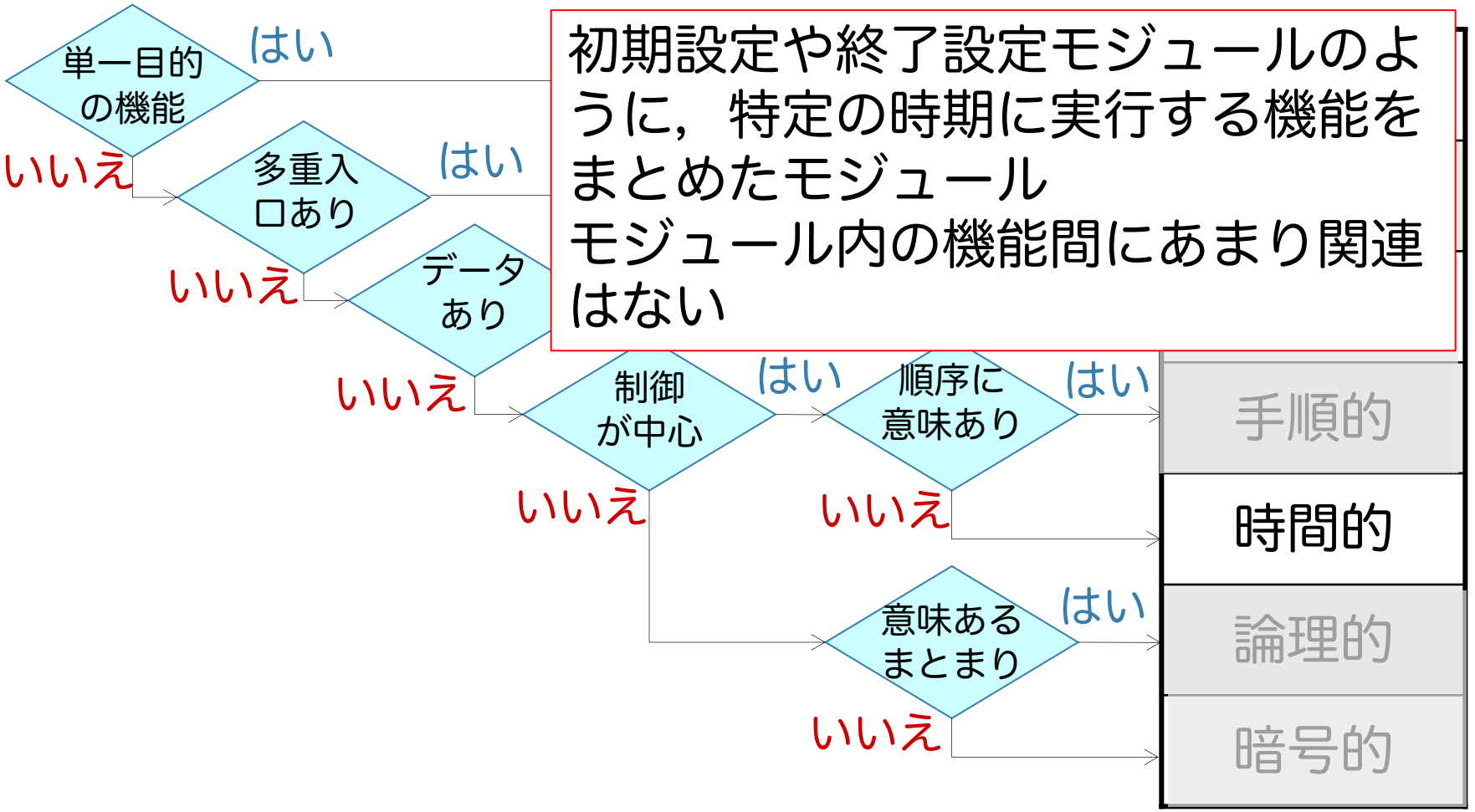


手順的凝集度のうち、モジュール内の機能間にデータの関連性があるモジュール

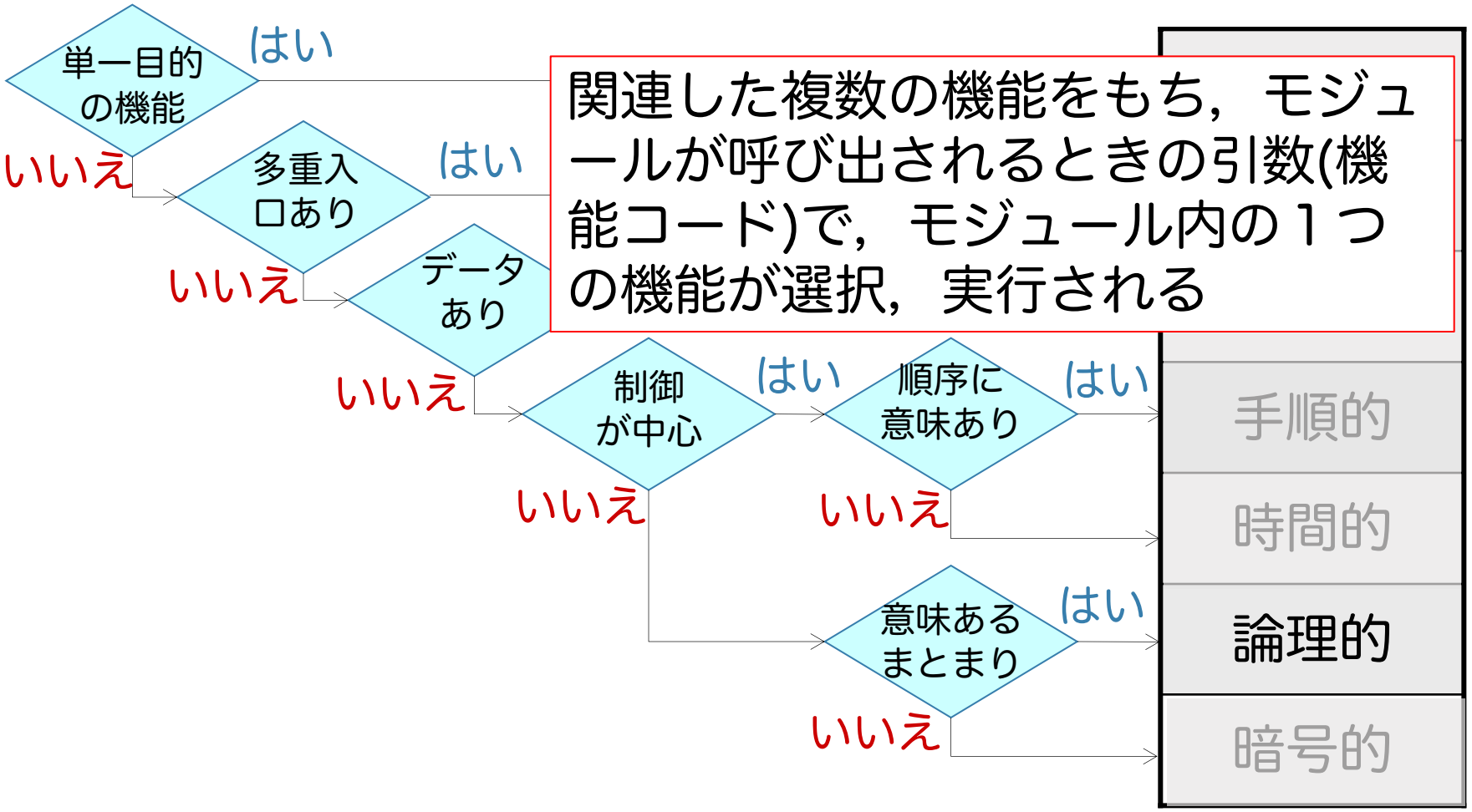
凝集度の識別


単一目的の機能
はい
機能的
いいえ
多重入口あり
はい
いいえ
データあり
複数の逐次的に実行する機能をまとめたモジュール
いいえ
制御が中心
はい
順序に意味あり
はい
手順的
いいえ
いいえ
時間的
意味あるまとまり
はい
論理的
いいえ
暗号的

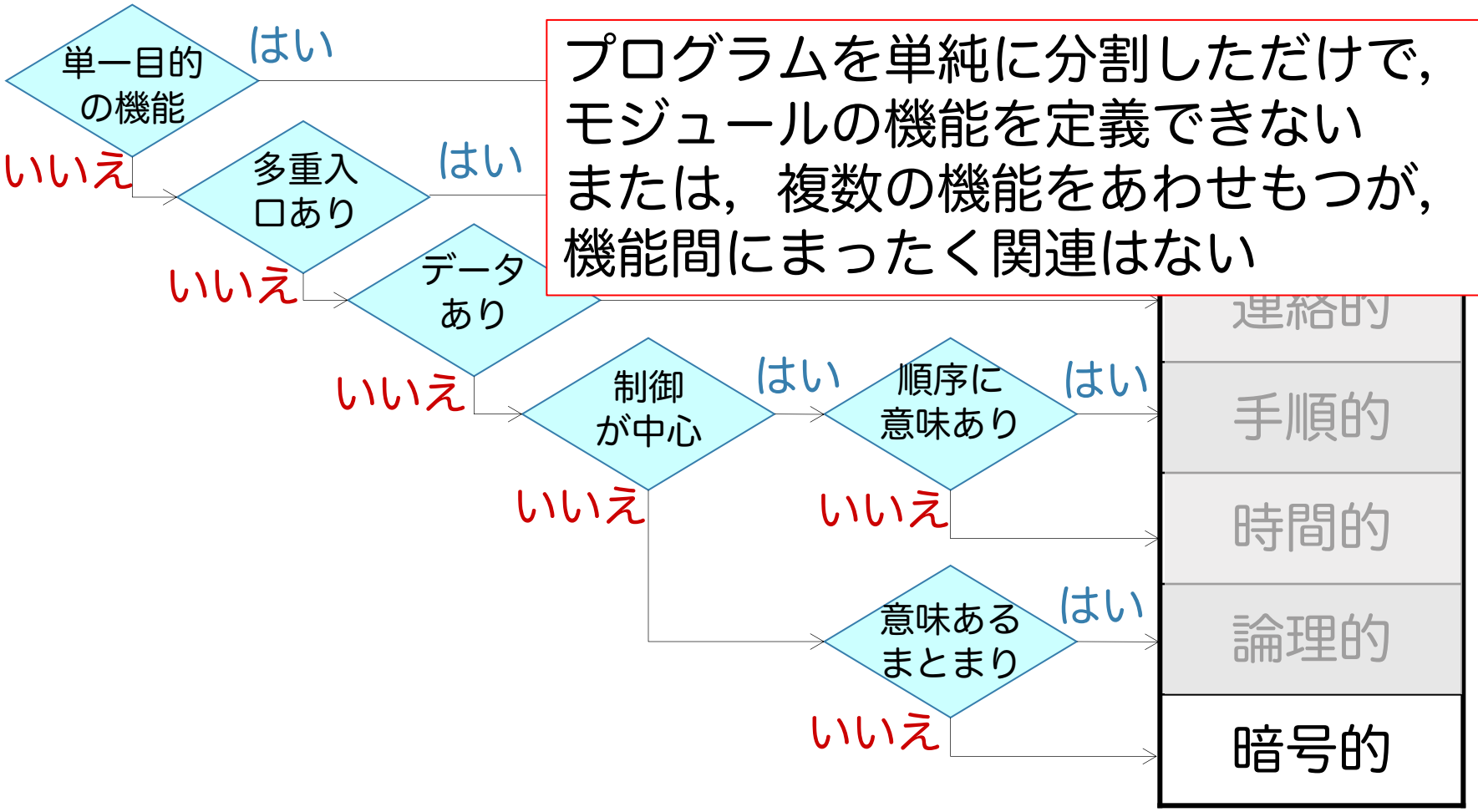
凝集度の識別


単一目的の機能
はい
いいえ
多重入口あり
はい
いいえ
データあり
いいえ
制御が中心
はい
順序に意味あり
はい
いいえ
いいえ
意味あるまとまり
はい
いいえ
初期設定や終了設定モジュールのように，特定の時期に実行する機能をまとめたモジュール
モジュール内の機能間にあまり関連はない
手順的
時間的
論理的
暗号的

凝集度の識別


単一目的の機能
はい
いいえ
多重入口あり
はい
いいえ
データあり
いいえ
制御が中心
はい
順序に意味あり
はい
いいえ
いいえ
意味あるまとまり
はい
いいえ
関連した複数の機能をもち，モジュールが呼び出されるときの引数(機能コード)で，モジュール内の１つの機能が選択，実行される
手順的
時間的
論理的
暗号的

凝集度の識別



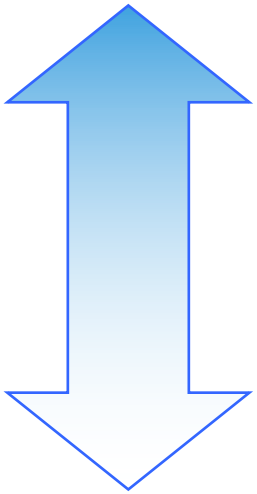
プログラムを単純に分割しただけで，
モジュールの機能を定義できない
または，複数の機能をあわせもつが，
機能間にまったく関連はない

結合度

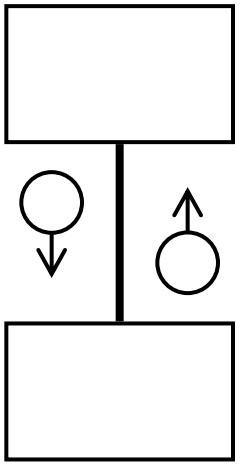
- モジュール間の依存関係が強い/弱い
  - 弱いほど良い
    - あるモジュールを変更した際に他モジュールが影響を受ける度合いを少なくする

1. 非直接結合
2. データ結合
3. スタンプ結合
4. 制御結合
5. 外部結合
6. 共通結合
7. 内容結合

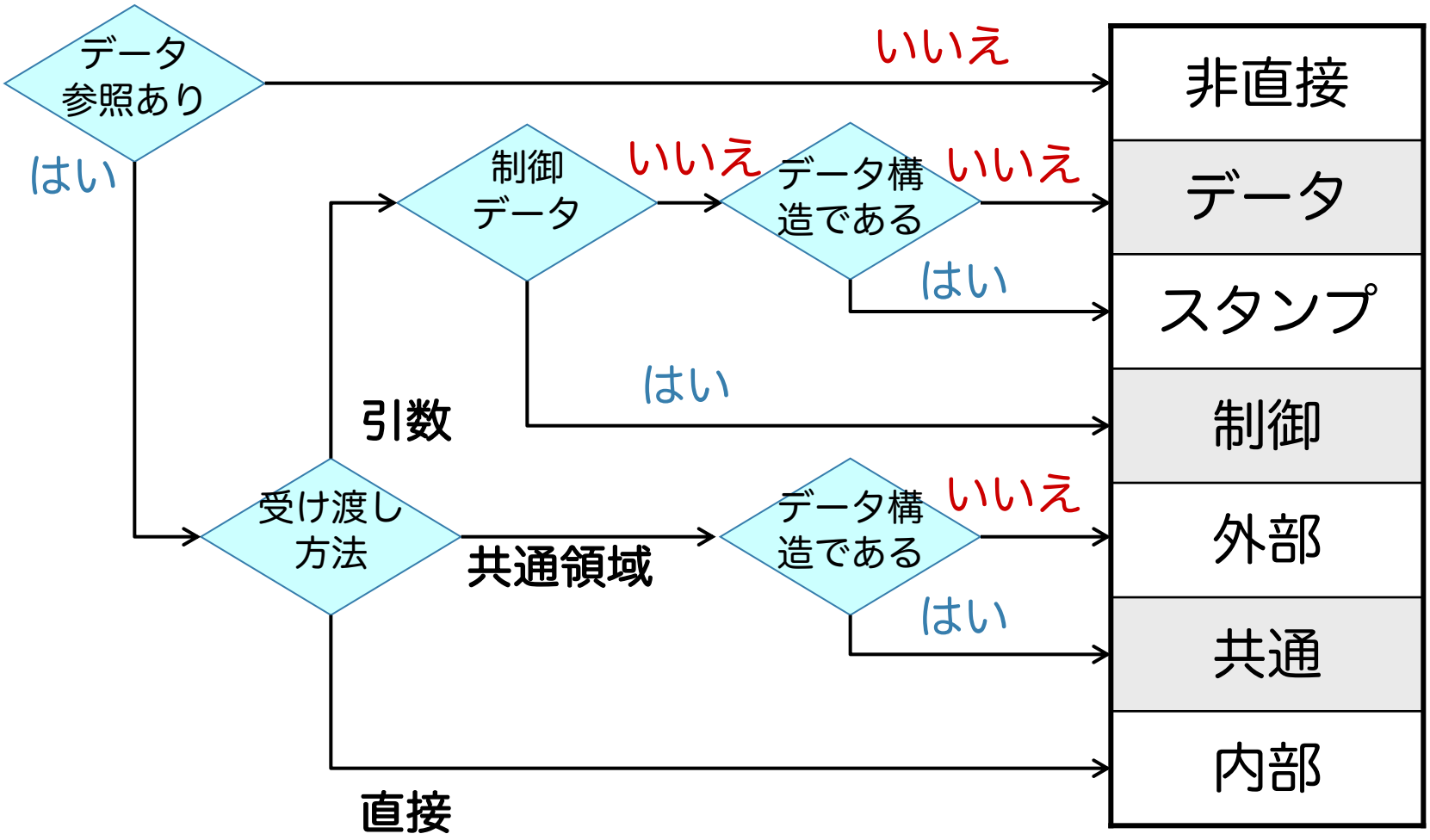
弱い（良，疎結合）



強い（悪，密結合）

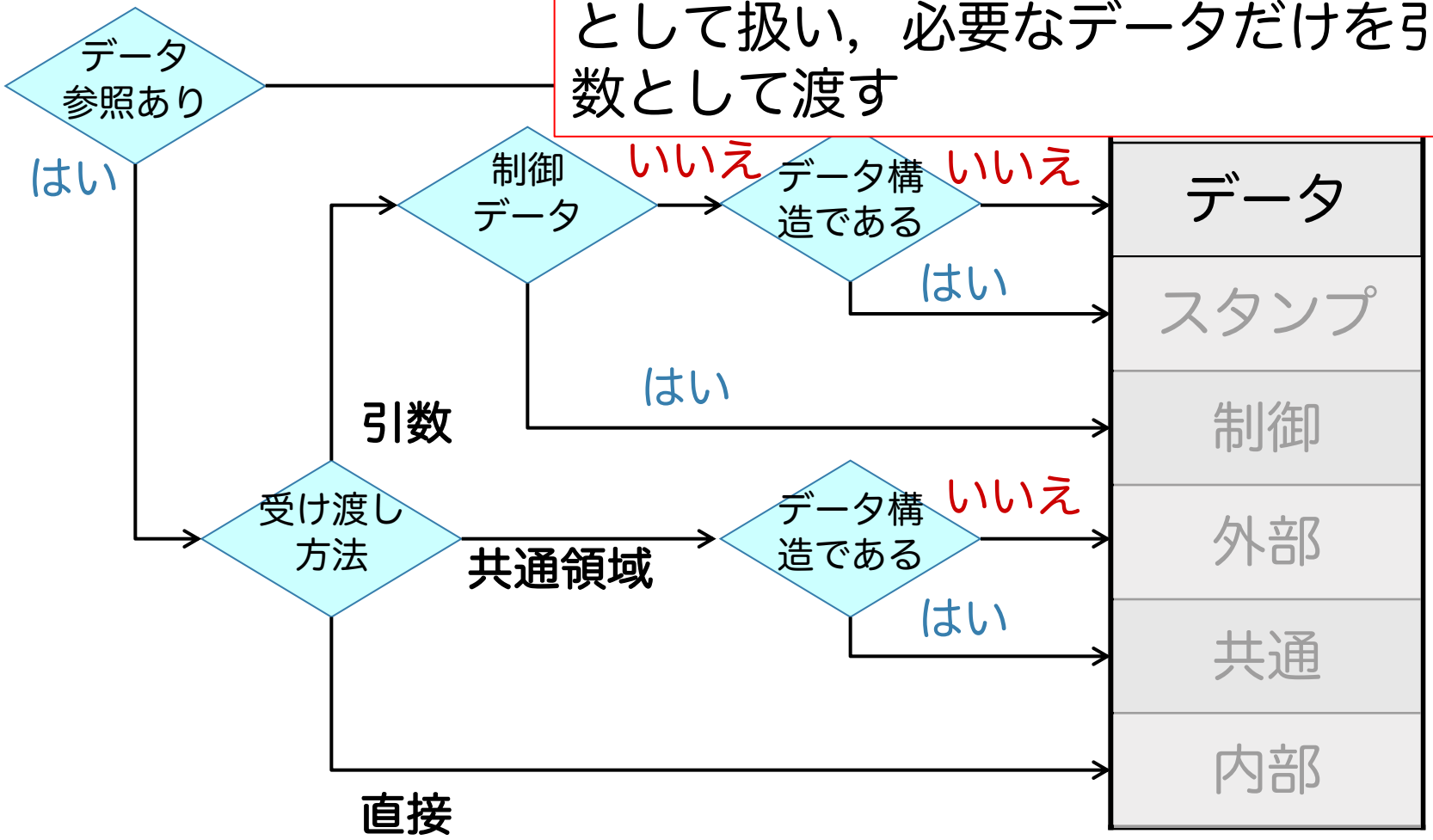


結合度の識別


データ
参照あり
いいえ
非直接
はい
制御
データ
いいえ
データ構
造である
いいえ
データ
はい
スタンプ
はい
制御
引数
受け渡し
方法
共通領域
データ構
造である
いいえ
外部
はい
共通
直接
内部

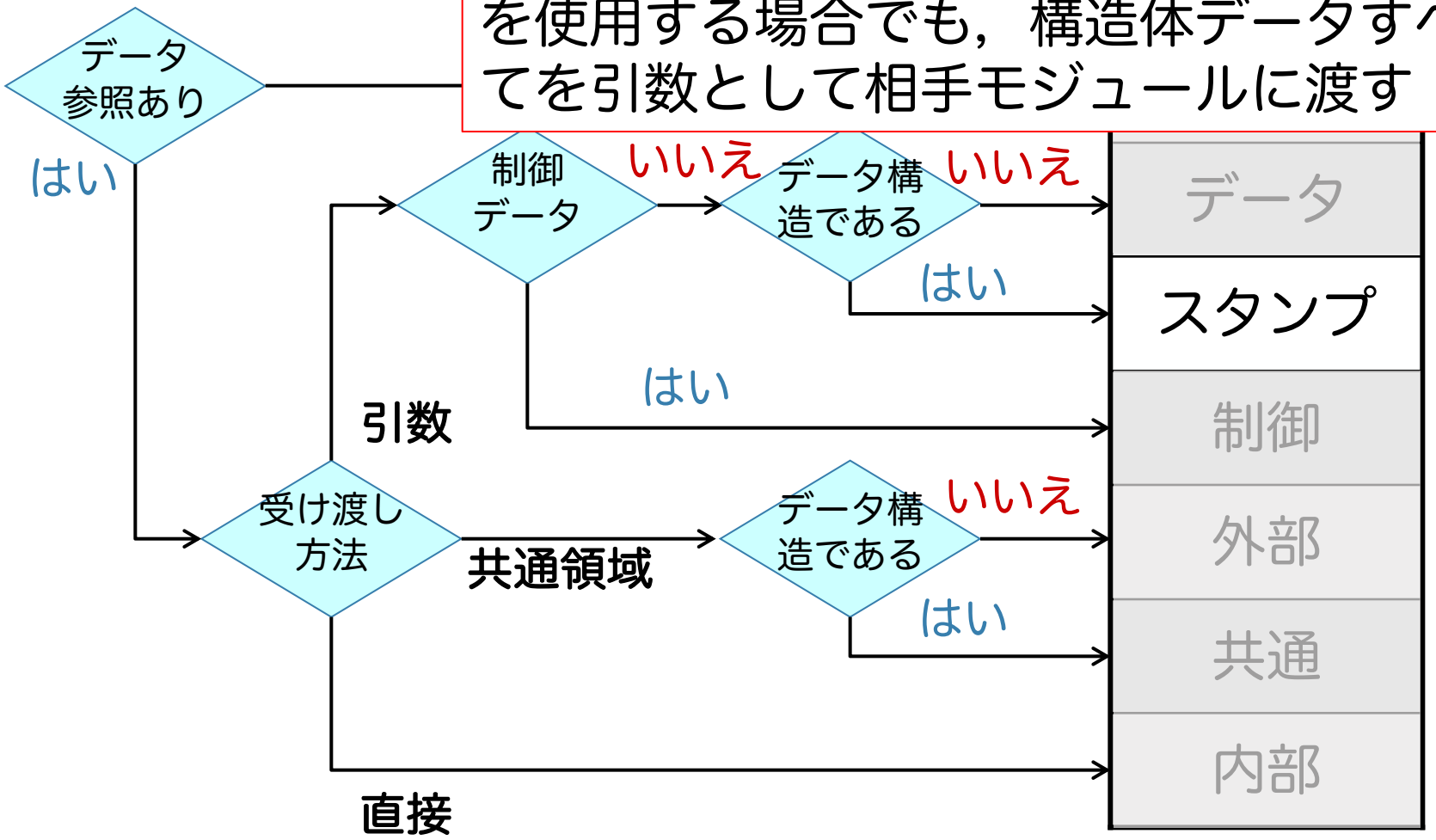
結合度の識別

相手モジュールをブラックボックスとして扱い，必要なデータだけを引数として渡す



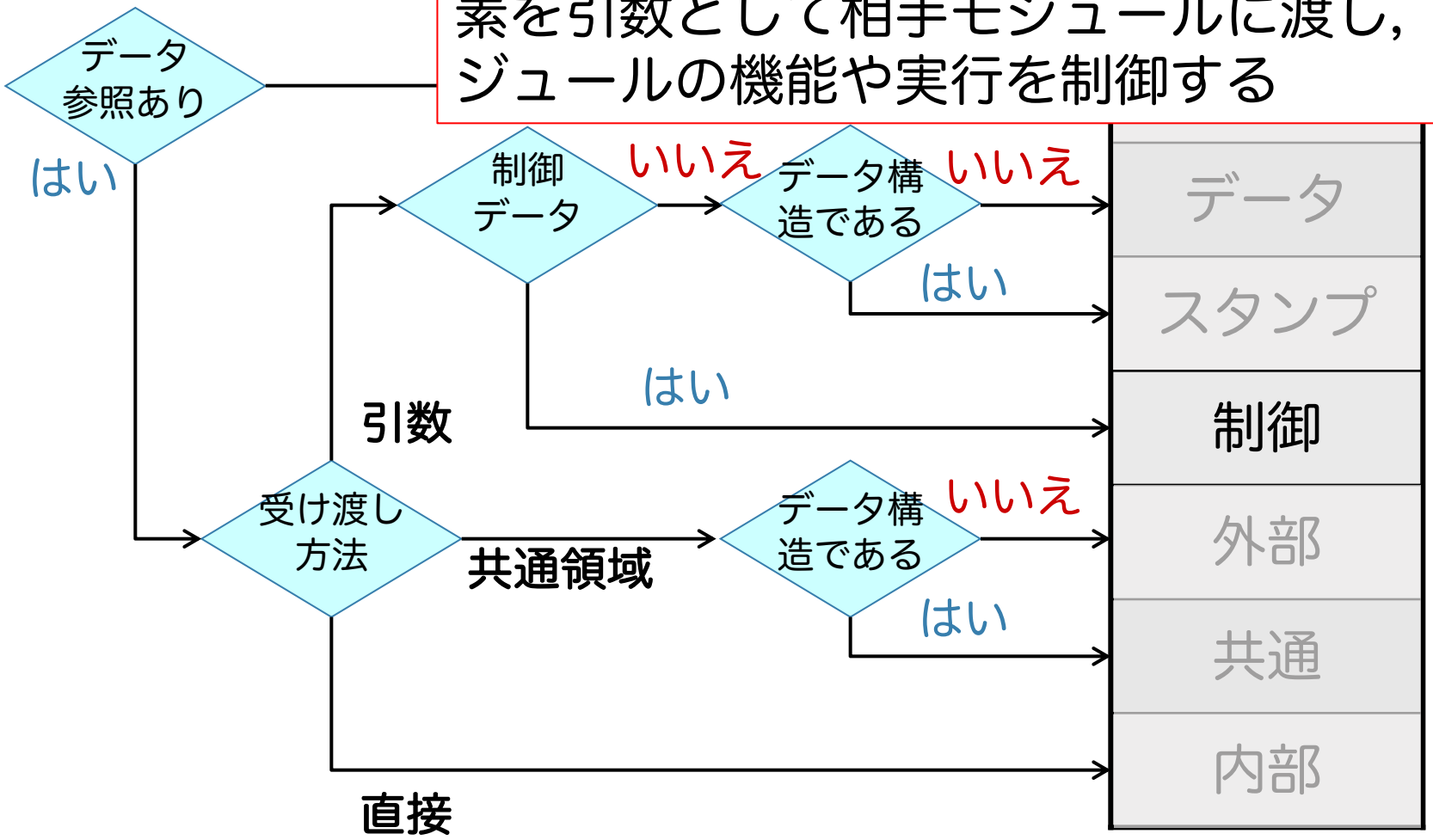
結合度の識別

相手モジュールで，構造体データの一部を使用する場合でも，構造体データすべてを引数として相手モジュールに渡す



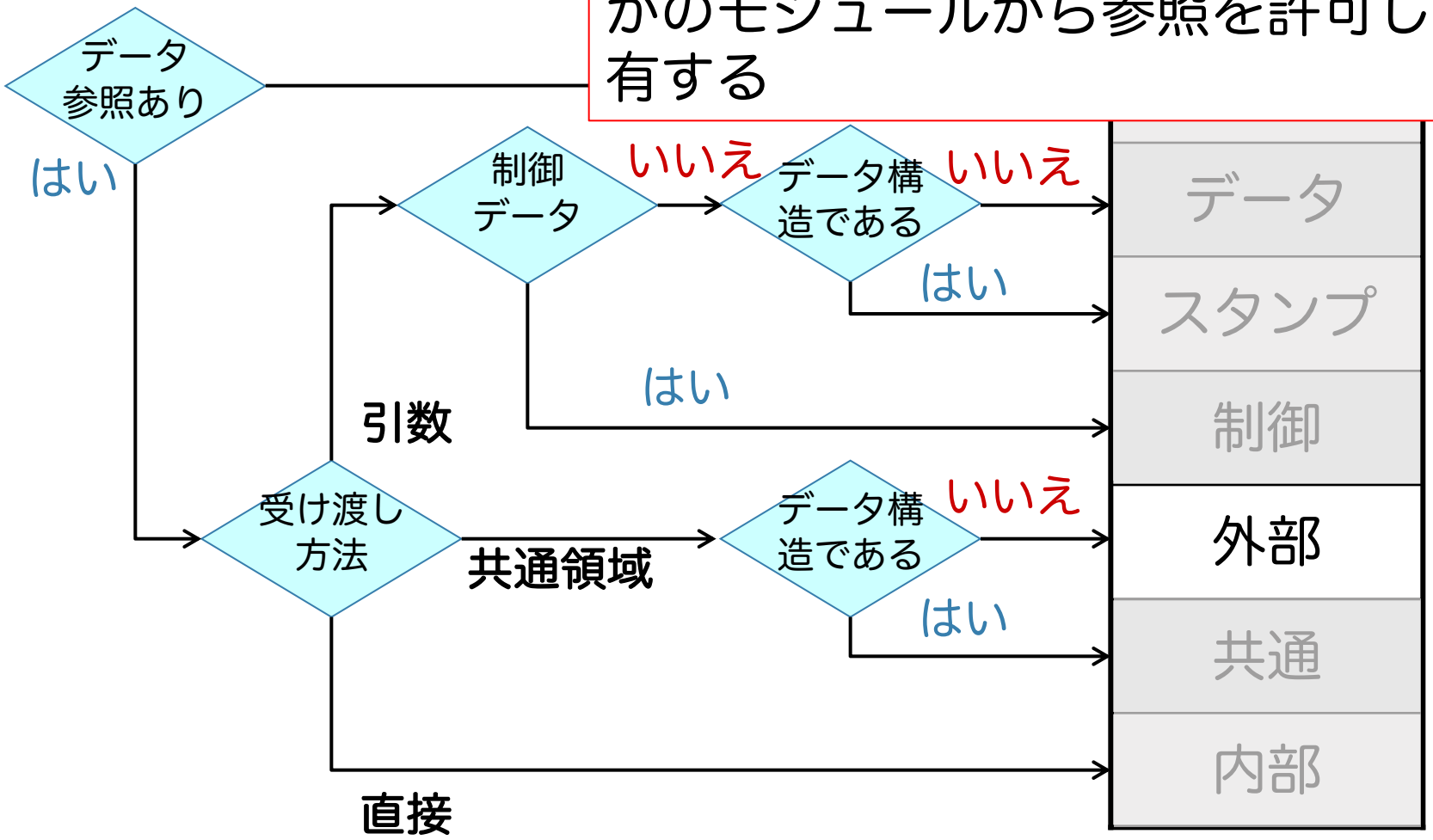
結合度の識別

機能コードなど，モジュールを制御する要素を引数として相手モジュールに渡し，モジュールの機能や実行を制御する



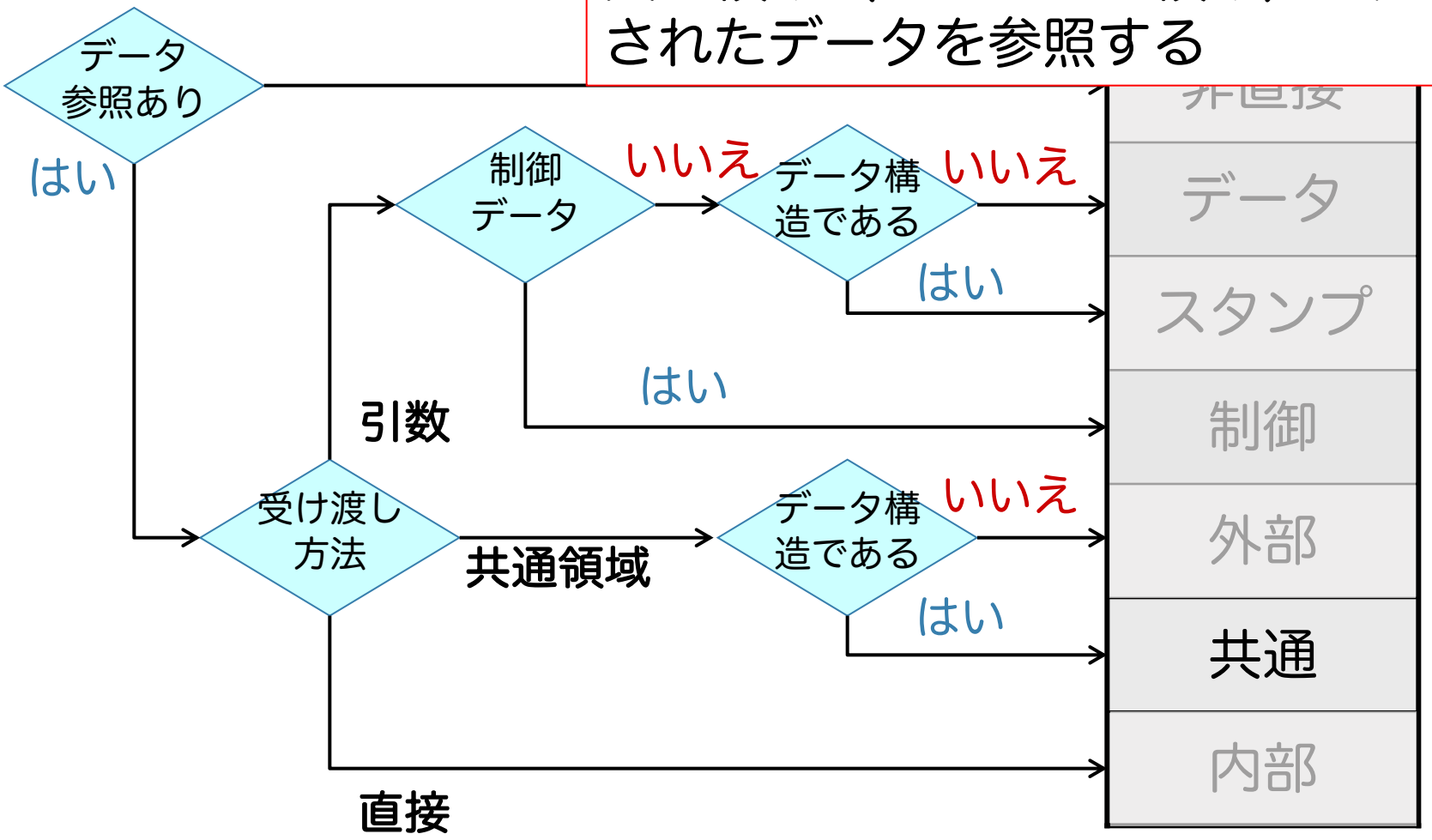
結合度の識別

必要なデータだけを外部宣言し，ほかのモジュールから参照を許可し共有する



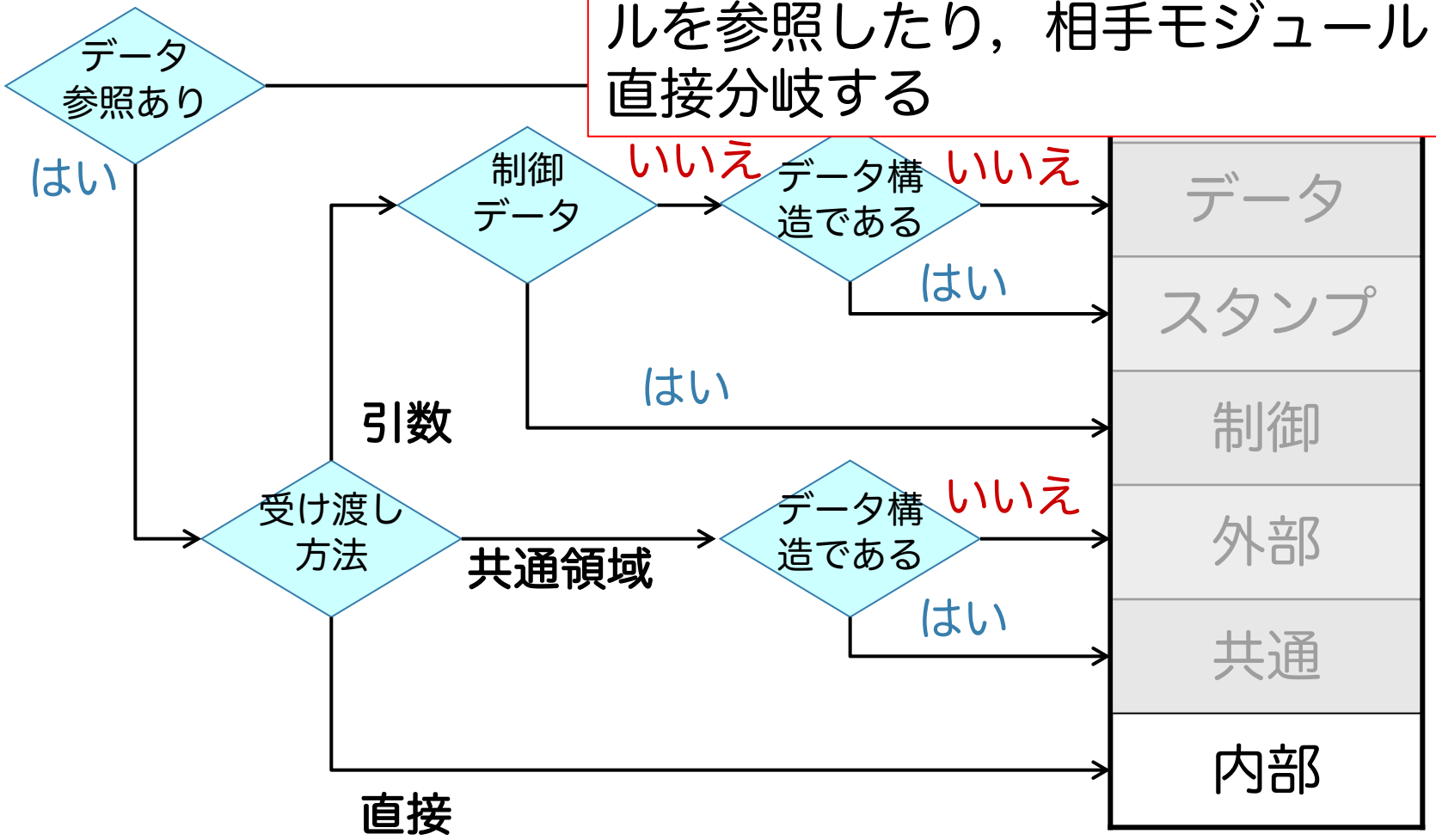
結合度の識別

共通領域（グローバル領域）に定義されたデータを参照する



結合度の識別

絶対番地を用いて直接相手モジュールを参照したり，相手モジュールに直接分岐する



C&Kメトリクス

- S. R. ChidamberとC. F. Kemerer によって提案
- オブジェクト指向ソフトウェアのためのメトリクス
  - 重み付きメソッド数
    (WMC: weighted methods per class)
  - メソッド間非凝集度
    (LCOM: lack of cohesion in methods)
  - オブジェクト間結合度
    (CBO: coupling between object classes)
  - 応答メソッド数 (RFC: response for a class)
  - 継承木深度 (DIT: depth of inheritance tree)
  - 子クラス数 (NOC: number of children)

WMC: weighted methods per class

- クラスCにn個のメソッドが定義されていて，複雑度がそれぞれ c_1〜c_n とする

$$\mathrm{WMC} = \sum c_i$$

- 複雑度にはMcCabeのサイクロマチック数が使われることが多い

LCOM: lack of cohesion in methods

- 共通の属性を操作するメソッドの数
- LCOM が高いと
  - 属性を介して多くのメソッドが結び付いている
  - メソッドの凝集度が低い
  - クラスは高い凝集度をもっているかもしれない

CBO: coupling between object classes

- 結合しているクラスの数
  - 他のクラスのメソッド呼び出し
  - 他のクラスのインスタンス参照
- CBOが高いと
  - クラスの変更/テストが複雑になる
  - 再利用性が低くなる

RFC: response for a class

- メッセージを受け取ることによって実行されるメソッドの数
- RFCが高いと
  - テスト列が増大
  - 設計全体も複雑に

DIT: depth of the inheritance tree

- 継承木のノードからルートへの長さの最大値
- クラス階層が深い(DITが大きい)
  - 設計が複雑
  - 多くのメソッドが再利用されている

NOC: number of children

- 子クラス(直接の派生クラス)の数
- 子クラスが多いと
  - よく再利用されている
  - 親クラスのメンバとして適切でないクラスがあれば，親クラスの抽象性が薄いことに

MOODメトリクス

- R. Harrison，S. J. Counsell，R. V. Nithi によって提案

- メソッド継承要因
  (MIF: method inheritance factor)
  - メソッドや属性を継承する程度
- 結合要因 (CF: coupling factor)
  - クラス間の結合の強さ

MIF: method inheritance factor

- メソッドや属性を継承する程度

$$\mathrm{MIF} = \sum M_i(C_i) / \sum M_a(C_i)$$

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

$M_a(C_i)$：C_iによって呼び出されるメソッドの数

$M_d(C_i)$：クラスC_i内で宣言されるメソッドの数

$M_i(C_i)$：クラスC_i内で継承されるメソッド数

CF: coupling factor

- クラス間の結合の強さ

$$\mathrm{CF} = \sum_i \sum_j is\_client(C_i, C_j) / (T_c^{\,2} - T_c)$$

T_c :クラスの総数

$is\_client(C_c, C_s)$: C_cとC_sに関連性があり,

$C_c \neq C_s$のとき1, それ以外のとき0

ソフトウェアメトリクスの例

- 要求メトリクス
  - ファンクションポイント
  - ユースケースポイント
- プロセスメトリクス
  - 作業時間/期間
  - 工数
- 設計メトリクス
  - 凝集度 (強度)
  - 結合度
  - C&Kメトリクス
- ソースコードメトリクス
  - McCabe のサイクロマチック数
  - Halstead のソフトウェアサイエンス
  - SLOC (SourceLines of Codes)

ソースコードメトリクス

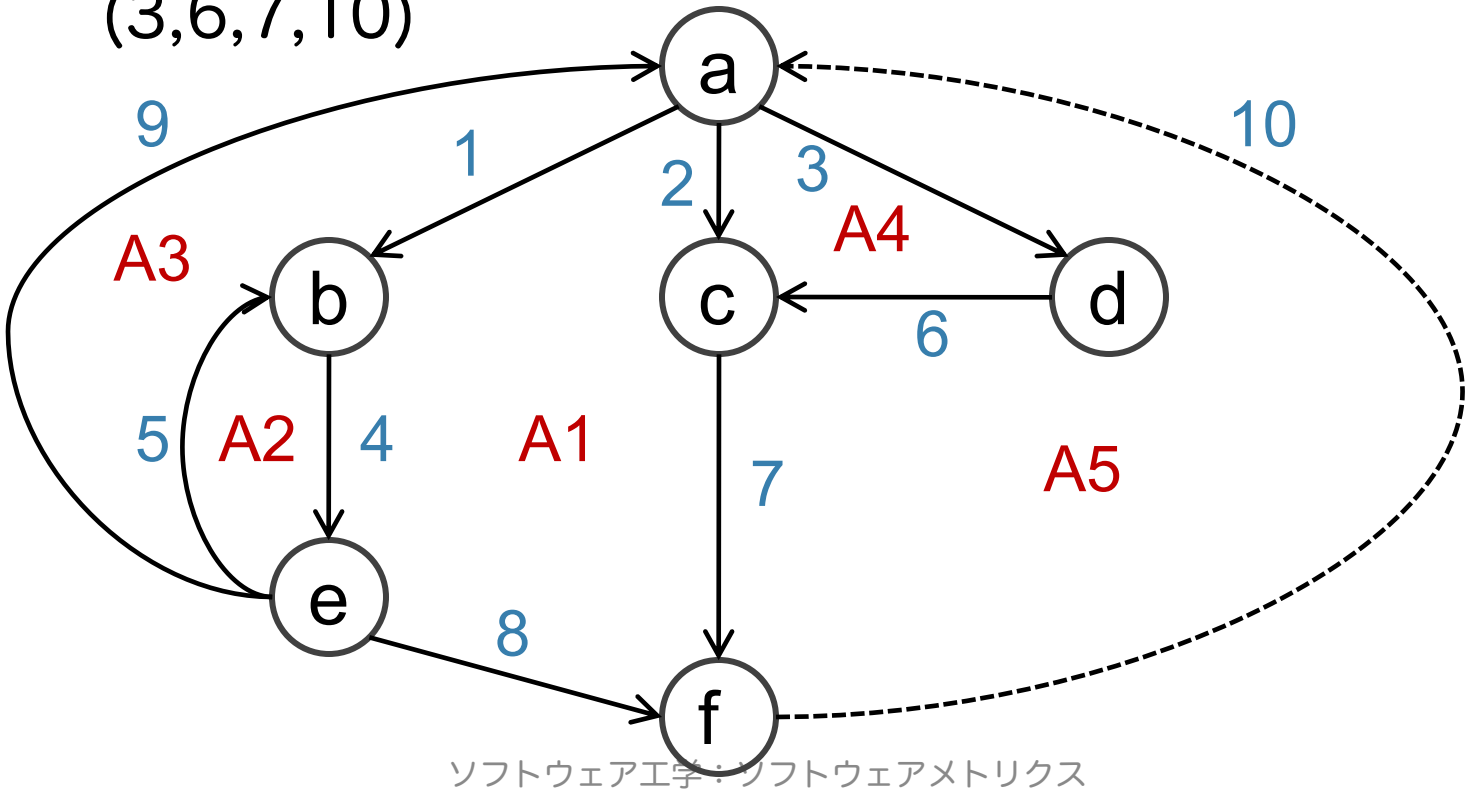
- 複雑さメトリクス
  - データ構造やデータフローの複雑さ
    - Span：同一識別子が参照される距離
  - McCabe のサイクロマチック数
    - 制御構造の複雑さ
- Halsteadのソフトウェアサイエンス
  - ソースコード中に含まれる演算子(operator)と非演算子(operand)の種類数，出現回数から規模や工数の計測
- 行数：LOC

McCabeのサイクロマチック数

- プログラム（OOのメソッド）の制御構造メトリクス
  - プログラムの制御の流れを有向グラフで表現し，性質に基づいてプログラムの複雑性を表す
  - c(m) = e-n+2
    - c(m)：モジュール m のサイクロマチック数
    - e： Control Flow の edge の数
    - n： Control Flow の node の数

McCabeのサイクロマチック数

- c(m) = e-n+2 = 9-6+2 = 5
  - 領域（A1，A2，A3，A4，A5）
  - 基本パス(1,4,8,10)，(1,4,5)，(1,4,9)，(2,7,10)，(3,6,7,10)

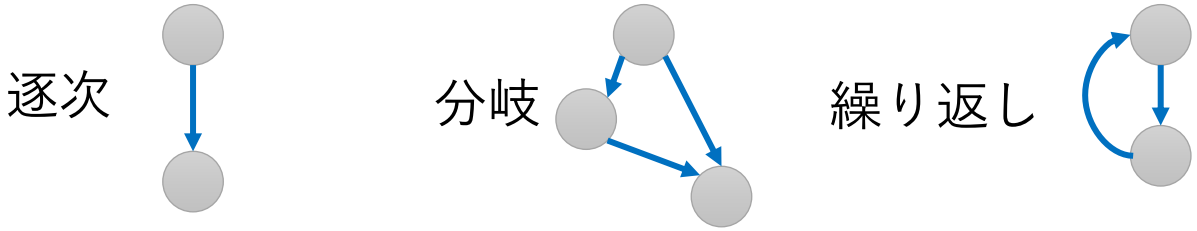


サイクロマチック密度

- コード行数が増えると分岐が増え，サイクロマチック数も増える
  - サイクロマチック数が大きい原因が，少ない行数に多数の分岐が密にあるのか，単に行数が多いのか区別できない
- サイクロマチック密度 cd(G)
  - 単位行数あたりのサイクロマチック数
  - サイクロマチック数/(空行やコメントを除いた)行数
- cd(G) が大きい ≒ 保守が困難

本質的複雑度

- プログラム中の非構造的なロジックを定量化したメトリクス
- 構造化プログラミングで制限した下記プログラムの制御に縮退する



- 非構造的ロジック
  - 繰り返しの途中にジャンプ
  - 繰り返しの途中から外へジャンプ
- 縮退
  - 制御グラフGが上記グラフを部分グラフとして含むとき，その部分グラフを1つのノードにまとめる
- 本質的複雑度
  - モジュール m の制御フローグラフ G において，可能な限り縮退を繰返して得られるグラフをGr としたとき，m の本質的複雑度 ec(G) = c(Gr)

Halstead のソフトウェアサイエンス

- 語彙の多さに着目したメトリクスの集合
  - 語彙，長さ，大きさ，労力，抽象化レベルなど
- 自然言語の文書の読みやすさに基づく
  - 幼児向けの図書は語彙が少ないので読みやすい
  - 新聞は語彙が多いので読みにくい
  - プログラムでも同様のことが言えるのでは?
- プログラムの難しさの推定や，プログラムを作成するのに必要なコストの見積りに使えないかと期待された

Halsteadのソフトウェアサイエンスの定義

- 基本尺度
  - n1: プログラム中のオペレータの種類数
  - n2: プログラム中のオペランドの種類数
  - N1: プログラム中のオペレータの総出現数
  - N2: プログラム中のオペランドの総出現数
- 語彙：n = n1 + n2
- プログラムの長さ：N = N1 + N2
- プログラムの大きさ：$V = N\log_2 n$
- 抽象化レベル: L = (2*n2)/(n1*N2)
  - 最も効率良くプログラムを組んだ場合に1をとる
  - プログラムが冗長なほど値が小さくなる
- プログラミング労力：E = V/L

Source Lines of Codes (SLOC)

- プログラムの行数
- 数え方は多様
  - そのまま数える
  - コメントを除く
  - 空行を除く
  - 命令文の数だけ数える
- 単純なメトリクスだが強力
  - 自動測定可能
  - 長い関数やメソッドは欠陥が多い傾向にある

ソフトウェアメトリクスの目的

- ソフトウェア開発プロジェクトを成功させる
- ソフトウェアの品質を向上させる
- ソフトウェア開発の生産性を向上させる

まとめ

- ソフトウェアメトリクスの目的
  - ソフトウェア開発プロジェクトの成功
  - ソフトウェアの品質の向上
  - ソフトウェア開発の生産性の向上
- メトリクス測定の原則
  - 定式化→収集→解析→解釈→フィードバック
- 測定する利点
  - 過去実績に基づく見積り，可視化し改善計画に活用
- ソフトウェアメトリクスの例
  - 要求メトリクス (FP)
  - プロセスメトリクス (工数)
  - 設計メトリクス (凝集度，結合度，複雑度)
  - ソースコードメトリクス (サイクロマチック数，LOC)