ソフトウェア工学III

プロダクトデータの解析II
——複雑さメトリクス——

ソフトウェア工学講座
門田暁人
akito-m@is.naist.jp
B303室，内線5311

プログラム解析技術

- プログラムスライシング
  - プログラム内の任意の文の着目する変数に影響を与える一部のコードのみを抽出する技術
  - デバッグ，テスト，リファクタリングなどに用いられる．
- コードクローン解析
  - プログラム中の重複するコード対を特定する技術
  - デバッグ，テスト，リファクタリングなどに用いられる．
- 複雑さメトリクス(complexity metrics)計測
  - プログラムの特定の要素に着目し，複雑さを計測する．
    - 例：ループの深さ，変数の個数，継承の深さなど
  - Fault-proneモジュール判別やリファクタリングに用いられる．

代表的なComplexity Metrics

- McCabeのCyclomatic Compexity
  - Thomas J. McCabeが1976年に提案した.
  - プログラムの制御構造に着目した尺度
- HalsteadのSoftware Science
  - Maurice H. Halsteadが1970年代初頭に提案した.
  - 語彙の多さに着目した尺度
- C&K Metrics
  - ChidamberとKemererが1994年に提案した.
  - オブジェクト指向ソフトウェアに対応した尺度
- 構造メトリクス
  - モジュール凝集度・結合度，Fan-in/out

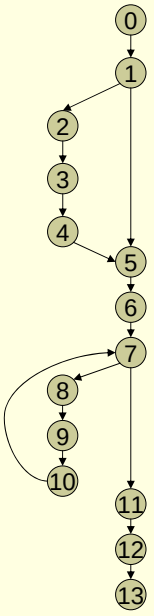
McCabeのCyclomatic Complexity

- サイクロマティック数（サイクロマティック複雑度）
- 制御構造の複雑さを数値化する尺度
- 分岐が多いほど値が大きくなる.
  - 分岐が多い→全てのパスを網羅するためのテストケースが多くなる→テスト漏れが生じやすい→バグが残存しやすい
- 派生するいくつかの尺度
  - サイクロマティック密度，設計複雑度，結合複雑度など

サイクロマティック数の定義

- 定義1．制御フローグラフ
  - mを与えられたモジュール（関数やサブルーチン）とする．
  - mの制御フローグラフは，有向グラフ$\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$で定義される．
    - $\boldsymbol{V}$はノードの集合．ノード＝文，式，または制御合流点
    - $\boldsymbol{E}$（$\subseteq \boldsymbol{V}\times\boldsymbol{V}$）はノード間の遷移を表す有向辺の集合
      ノードn_1の次にノードn_2が実行される場合，n_1からn_2に有向辺を引く
- 定義2．サイクロマティック数
  - モジュールmのサイクロマティック数$v(\boldsymbol{G})$
    - $v(\boldsymbol{G}) = |\boldsymbol{E}| - |\boldsymbol{V}| + 2$

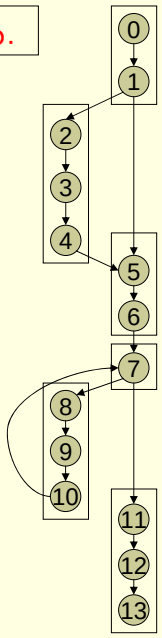
```
N0     int euclid(int m, int n) {
         /* m がnより大きく，かつ両者が0より大きいと仮定し，
            m, n の最大公約数を求める．*/
         int r;
N1       if (n > m) { /*mとnの入替*/
N2         r = m;
N3         m = n;
N4         n = r;
N5       }
N6       r = m % n; /*r は nの剰余*/
N7       while (r != 0) {
N8         m = n;
N9         n = r;
N10        r = m % n;
N11      }
N12      return n;
N13    }
```



- 有向辺の数 $|\boldsymbol{E}| = 15$
- ノードの数$|\boldsymbol{V}| = 14$
- サイクロマティック数 $v(\boldsymbol{G}) = |\boldsymbol{E}| - |\boldsymbol{V}| + 2 = 3$

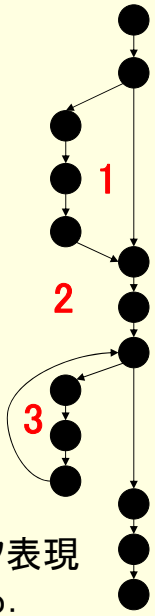
（補足）基本ブロックを1ノードとみなしても同じ値となる．

```
            /* m がnより大きく，かつ両者が0より大きいと仮定し，
               m, n の最大公約数を求める．*/
            int r;
N1          if (n > m) { /*mとnの入替*/
N2            r = m;
N3            m = n;
N4            n = r;
N5          }
N6          r = m % n;  /*r は nの剰余*/
N7          while (r != 0) {
N8            m = n;
N9            n = r;
N10           r = m % n;
N11         }
N12         return n;
N13       }
```



- 有向辺の数 $|E| = 7$
- ノードの数 $|V| = 6$
- サイクロマティック数 $v(G) = |E| - |V| + 2 = 3$

```
N0      int euclid(int m, int n) {
          /* m がnより大きく，かつ両者が0より大きいと仮定し，
             m, n の最大公約数を求める．*/
          int r;
N1        if (n > m) { /*mとnの入替*/
N2          r = m;
N3          m = n;
N4          n = r;
N5        }
N6        r = m % n;  /*r は nの剰余*/
N7        while (r != 0) {
N8          m = n;
N9          n = r;
N10         r = m % n;
N11       }
N12       return n;
N13     }
```



- サイクロマティック数は，プログラムをグラフ表現したときの，平面上の分割領域数に一致する．

  $v(G) = 3$

サイクロマティック数の実用上の目安

- モジュールの単体テスト（ホワイトボックステスト）のテストケース数に対応する．単体テストのしやすさの目安となる．
  - 10以下が望ましい [1].
  - 30を超えるとバグが増える[2].
    - IBMの事例では，32を超えるとバグを含んでいる確率が高かった．
  - 50を超えるとテストが困難となる[2].
  - 主に10～100の間に分布している[3].
    - 良いといわれるコードの複雑度は低い
    - 慌てて作ると複雑度は高くなる．

[1] Watson, A.H. and McCabe, T. J., “Structured Testing: A Teting Methodology Using the Cyclomatic Complexity Metric”, NIST Special Publication, Vol.500-235, Aug. 1996.
[2] 山浦恒央, “ソフトウェアメトリクス盛衰記”, 第12回S-openホットセッション講演資料, Dec. 2004.
[3] 二上貴夫, “メトリック・なんでも測っちゃおう”,第12回S-openホットセッション講演資料, Dec. 2004.

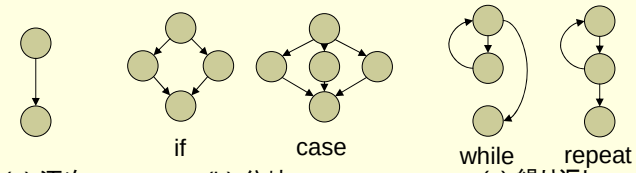
サイクロマティック密度(Cyclimatic Density) *vd*(***G***)

- 一般に，コード行数が増えると分岐が増え，サイクロマティック数も増える．
  - 「サイクロマティック数が大きい」といっても，少ない行に多くの分岐が密に定義されたことによるものか，単に行数が多いのか区別できない．
- サイクロマティック密度 vd(G)
  - 単位行数あたりのサイクロマティック数
  - サイクロマティック数を（空行やコメントを除いた）行数で割ったもの．
- vd(G)が大きい≒保守が困難
  - NASA IV&V Facility MDP(Metrics Data Program)[1]における試み

[1] http://mdp.ivv.nasa.gov/

本質的複雑度（Essential Complexity） *ev*(*G*)

- プログラム中の非構造的なロジックを定量化したもの
- 構造化プログラミングでは，プログラムの制御を



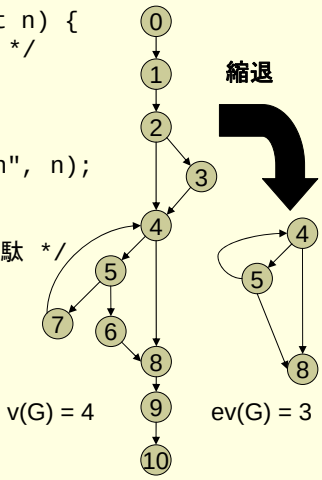
(a) 逐次　(b) 分岐　(c) 繰り返し

の三つに制限することを提唱している．

- 非構造的ロジック
  - 繰り返しの途中へジャンプする．繰り返しの途中から外へジャンプする．
- 定義3．縮退
  - 制御フローグラフGが，上図のグラフを部分グラフとして含むとき，その部分グラフを一つのノードにまとめる操作
- 定義4．本質的複雑度
  - モジュールmの制御フローグラフGが与えられたとき，Gに対して可能な限り縮退を繰り返して得られるグラフをGrとする．このとき，mの本質的複雑度ev(G)は，ev(G) = v(Gr)で与えられる．

本質的複雑度の計算例

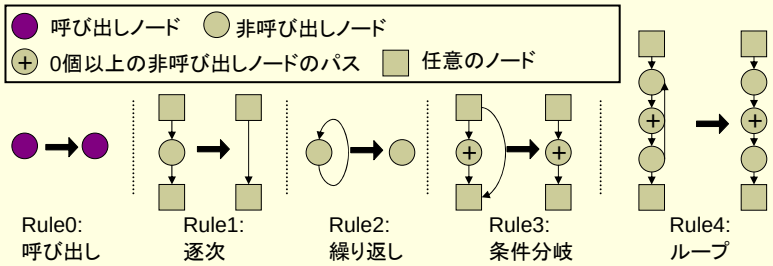
```
N0  int array_compare(int A[], int B[], int n) {
    /* 配列を比べる．両方の内容が全く同じなら0を返す．*/
    int i, cond;
N1    i = 0;
N2    if (n<0)
N3      fprintf(stderr, "Warning: Size %d¥n", n);
N4    while(i<n) {
N5      if ((cond = A[i] - B[i])!=0)
N6        break; /*一旦異なれば，以降比較しても無駄 */
N7      i++;
N8    }
N9    return cond;
N10 }
```



- 本質的複雑度が大きい → リファクタリングの候補となり得る

設計複雑度(Design Complexity) *iv*(*G*)

- モジュール間の呼び出しに必要な制御構造のみを抽出してサイクロマティック数を計算した値
  - 結合テストで必要なテストケース数を見積もる際の目安となる.
  - モジュール呼び出しに関係のあるパスを重点的にテストすべきという考え方
- 制御縮退ルール



- 定義5. 設計複雑度
  - モジュールmの制御フローグラフGが与えられたとき，Gに対して可能な限り制御縮退を繰り返して得られるグラフをG_{dr}とする. このとき，mの本質的複雑度iv(G)は，
    $iv(G) = v(G_{dr})$ で与えられる.

その他のMcCabeメトリクス

- 結合複雑度
  - 結合されるモジュール群の設計複雑度の合計値－モジュール数+1
  - 結合テストにおけるテストケース数と対応する.
- 設計密度
  - 設計複雑度をサイクロマティック数で割った値
  - モジュール呼び出しの複雑さを表す.
- 大域データ複雑度
  - グローバル変数を参照するノードのみを抽出してサイクロマティック数を計算した値
- 大域データ密度
  - 大域データ複雑度をサイクロマティック数で割った値
- 保守困難さ(Maintenance Severity)
  - 本質的複雑度をサイクロマティック数で割った値

McCabeメトリクスの特徴

- 利点
  - サイクロマティック複雑度は，計算が比較的容易である．
  - サイクロマティック複雑度≒必要なテストケース数，という考え方が人間の直感に合う．
  - プログラミング言語に依存しにくい．構造化言語であれば全て測定可能．オブジェクト指向言語では，メソッドレベルで測定可能．
- 限界
  - データの複雑さは計測対象外

HalsteadのSoftware Science

- 語彙の多さに着目したメトリクスの集合
  - Vocabulary, Length, Volume，Effort, Difficulty, Level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．
  - 一方，新聞は語彙が多いので読みにくい．
  - プログラムも文書の一種であるから同様のことが言えないか？
- プログラミングの難しさの推定や，プログラミングを作成するのに必要な労力の見積もりに使えないかと期待された．

HalsteadのSoftware Scienceの定義

- 基本となる尺度
  - n1: プログラム中のオペレータ種類数
  - n2: プログラム中のオペランド種類数
  - N1: プログラム中のオペレータ総出現回数
  - N2: プログラム中のオペランド総出現回数
    - オペレータとは，計算機の動作を規定するトークン
      - 算出演算子（+, -, *, /)やread, while, gotoなどの命令
    - オペランドは，変数や定数などのデータを表すトークン
- 語彙 Vocabulary n = n1 + n2
- プログラムの長さ Length N = N1 + N2
- プログラムの大きさ Volume V = N $\log_2$ n
  - 全トークンNのうちn個がユニークであるようなプログラムを記述するのに必要なビット数を意味する.
- プログラムの抽象化レベル Abstract Level L = (2*n2) / (n1*N2)
  - 最も効率よくプログラムを組んだ場合に1（最大値）をとる.
  - プログラムが冗長になるほど値が小さくなる.
- プログラミング労力 Effort E = V / L
  - 人月と相関があると仮定されている.
  - 最も脚光を浴びた尺度

HalsteadのSoftware Scienceのこれまでの知見

- 次の尺度は互いに強い相関がある.
  - プログラムの長さ N
  - プログラムの大きさ V
  - プログラミング労力 E
- これらの尺度は，プログラムのサイズを表す他のメトリクスとも強い相関がある（ソースコード行数など）
- プログラムの開発工数は，主に，プログラムのサイズの関数である.

→ トークンの数を計測するよりも，コード行数を計測する方が遥かに楽であり，しかも，ほぼ同様の効果が得られるので，Halstead尺度は廃れていった.

HalsteadのSoftware Scienceの現状

- 古典的尺度として，ほとんどのメトリクス計測ツールでサポートされている．
  - ソースコード行数を測るよりは面倒だが，字句解析だけで算出できるため，計測は容易な部類である．
- 他のメトリクスと併用される．
  - バグ数の予測，バグの多いモジュールの判別などに用いられる．
- Halstead尺度だけに頼るのは危険であるが，知識として知っておくべきである．

C&K Metrics (CK Metrics)

- C = Chidamber, K = Kemerer
- オブジェクト指向設計，および，プログラムに対する複雑さメトリクスの集まり
- クラス内部，継承，結合の点から評価する．
- 保守性に関する6つのメトリクス
  - WMC, DIT, NOC, CBO, RFC, LCO

C&K Metrics ー 6つのメトリクス

- WMC (Weighted Methods per Class)
  - あるクラスに含まれるメソッド群を，おのおのの複雑さに基づいて重み付けして加算した値
  - メソッドの重み付けには，メソッドの行数，サイクロマティック数などを用いるが，単純に重み＝1として算出することも多い．
  - WMCが大きいほど保守性が低い．
- DIT (Depth of Inheritance Tree)
  - 継承ツリーの深さ
    - クラス階層中のあるノードからルートまでの最大深さ
  - DITが大きいほど継承した変数やメソッドが多いこととなり，保守性は下がる．
  - DITがゼロのクラスが多いと継承が活かされていない．

C&K Metrics ー 6つのメトリクス

- NOC (Number Of Children of a class)
  - あるクラスの直下のサブクラスの数
  - NOCが大きいほどサブクラスへの影響が大きいので保守性が下がる．
  - NOCがゼロのクラスが多いと継承が活かされていない．
- CBO (Coupling Between Object classes)
  - あるクラスに結合しているクラスの数
    - オブジェクトクラスは，他のクラスのメソッドやインスタンス変数を参照・実行する場合，そのクラスと結合する．
  - CBOが大きいほど他のクラスに依存していることとなり保守性が下がる．

C&K Metrics ー 6つのメトリクス

- RFC (Response For a Class)
  - クラスの応答の規模
    - あるクラスのオブジェクトが受け取るメッセージに応答して実行されるメソッドの数
    - ローカルメソッド数と，ローカルなメソッドから呼び出されるメソッド数の合計
  - RFCが大きいほど保守性が下がる.
- LCOM (Lack of Cohesion in Methods)
  - あるクラスの凝集性の欠如の度合い
    - ローカルメソッドがそのクラスのローカルインスタンス変数にどのくらい密接に関連しているかを数値化したもの
  - LCOMが大きい＝凝集性が低い＝クラス分割の余地あり

LCOM (Lack of Cohesion in Methods)

- Ii=メソッドmiによって使用されるインスタンス変数の集合
- $P = \{(Ii, Ij) \mid Ii \cap Ij = \phi\}$
- $Q = \{(Ii, Ij) \mid Ii \cap Ij \neq \phi\}$
- LCOM = |P| - |Q| if |P| > |Q|
  = 0 otherwise

```
Class test{
  private int a, b, c, d;
  public void m1(){a=0;b=0;}
  public void m2(){a=1;b=1;}
  public void m3(){c=0;d=0;}
}
```

I1={a,b}
I2={a,b}
I3={c,d}
P ={(I1,I3),(I2,I3)}
Q={(I1,I2)}
LCOM = |P|-|Q|=2-1=1

この例では，クラスを2つに分割すべきである.

C&K Metricsおよび関連メトリクスの活用

- Rosenberg, Stapko, Galloの提案[1]
  - 以下の基準に2つ以上該当するクラスは要注意である.
    - クラスの応答数 RFC > 100
    - RFC > クラスのメソッド数×5
    - オブジェクトクラス間結合度 CBO > 5
    - メソッド数 > 40
- Lorenzの経験則[2]
  - クラスあたりの平均メソッド数 ＜20
  - クラスあたりの平均インスタンス変数 ＜6
  - 継承ツリーの深さ DIT < 6

[1] Rosenberg, L., Stapko, R., and Gallo, A., “Applying Object-Oriented Metrics”, 6th Int’l Symposium on Software Metrics, Nov. 1999.
[2] Lorenz, M., “Object-Oriented Software Development: A Practical Guide”, Englewood Cliffs, N. J.: PTR Prentice Hall, 1993.

C&K Metricsの現状

- よく使われているもの
  - WMC (Weighted Methods per Class)
  - DIT (Depth of Inheritance Tree)
  - NOC (Number Of Children of a class)
- 批判されたり新たな定義が提案されているもの
  - CBO (Coupling Between Object classes)
  - RFC (Response For a Class)
  - LCOM (Lack of Cohesion in Methods)

C&K Metricsの利点

- 設計書の計測が可能（UMLドキュメント等）
  - UMLモデリングツールから自動計測可能
  - コーディングに入るまえに設計の見直しが可能

構造メトリクス

- YourdonとConstantine(1979)，Myers(1978)の提案
  - Fan-in: あるモジュールを呼び出すモジュールの総数
  - Fan-out: あるモジュールが呼び出すモジュールの総数
- HenryとKafuraの提案
  - D-INFO = (Fan-in * Fan-out)^2
- 一般に，fan-inが大きいモジュールは，サイズが小さい．
- 一方，大きく複雑なモジュールは，fan-outが大きく，fan-inが小さくなりがちである．
- Fan-in, fan-outともに大きなモジュールは，再設計の候補となる．

複雑さメトリクスの計測結果の例

- KC1_product_metrics.csv
  - NASA IV&V Facility MDPの一部
  - C++で記述された1つのプログラム
    - 2107個のモジュール
    - うち327個はバグを含んでいた.
  - 欠損値あり
- clone.csv
  - オープンソースプロジェクトを計測したもの
  - C言語で記述された115個のプログラム
  - コードクローン関連のメトリクスが多い.
  - 欠損値なし

演習レポート課題

- レポート課題
  - プロダクトデータ(KC1_product_metrics.csv, clone.csvのいずれか)を分析して，何らかの知見を2つ以上示せ.
  - 例:
    - データの可視化，および，検定
    - バグのあるモジュール(ERROR_COUNTが1以上)とないモジュールの複雑さメトリクスの比較
    - 必要に応じて，Weka，DAVIS，および，青木繁伸先生のWebページにある統計ツールなどを使うこと.
  - 分析結果だけでなく，分析の過程の説明，結果の考察も含めてレポートにまとめよ．書式は問わない.

演習課題 ― 提出期限

- レポート提出期限
  - 2006年1月19日（金）2限
  - 希望者はレポート課題の内容を発表すること
    - レポートの内容について説明する.
    - レポート用紙をスクリーンに映す，もしくは，パワーポイント等のプレゼン資料を用いる.
  - 時間が余れば，提出されたレポートの中からいくつかを講義中に（門田が）紹介します.
- 連絡先
  - 門田暁人 akito-m@is.naist.jp
  - B303室，内線5311