# SASプログラムの品質向上に向けた試み
## ～医薬品開発における プログラムバリデーションを念頭に～

益田　隆史、本多　隆之
株式会社ACRONET　データサイエンス本部　生物統計部

# Approaches to improve quality of SAS program

Takashi Masuda and Takayuki Honda
Biostatistics Department, ACRONET Corporation

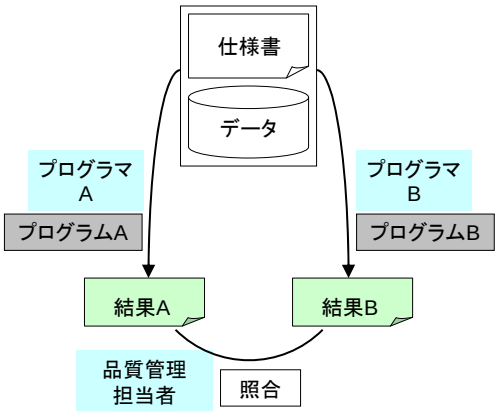
SAS ユーザー総会 アカデミア/テクノロジー&ソリューション セッション 2012

## 要旨:

医薬品開発においてプログラムバリデーションを実践するためには、プログラムの品質向上とプログラムレビューに耐え得る読みやすい記述が必須である。本稿ではこれに関するプロジェクトでの議論と成果を報告する。

キーワード:医薬品開発、バリデーション、ソフトウェア
コンストラクション、プログラミング作法

## ダブルプログラミング

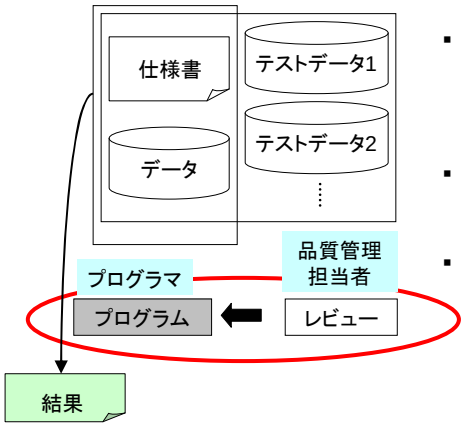



- 独立2系統で結果を出力、照合
  - 一致していれば結果を信用
  - プログラムの正しさは問えない
  - 目の前のデータに対してのみ品質を確保
- 使い捨てのプログラムに向いている





## バリデーション(狭義)




- プログラムの動作を保証
  - プログラムレビュー
  - テストデータによるテスト
- 医薬品開発以外ではこちらが主流
- 使いまわすプログラムでは必須
  - 想定するどのようなデータでも動作する必要がある

## 前回の発表[1)]の結論

- プログラムバリデーション(狭義)による品質管理を行った
- ダブルプログラミングとは大きく異なる特性を持つことが明らかとなった
- 使い回しをするプログラム・データが想定できるプログラムには非常に有効と考えられる
- ダブルプログラミングとの適切な使い分けにより、品質の向上と業務の効率化の可能性がある

1) 医薬品開発におけるプログラムバリデーションによる統計解析の品質管理、益田隆史、SASユーザー総会2011





## バリデーション(狭義)の実施方法

- 事前に計画を策定し、バリデーションの過程を記録する
- 具体的な手順
  - 事前に想定されるいくつものパターンのテストデータによる動作確認
  - プログラムのレビュー

## バリデーション(狭義)を実現するためには……

- 高品質なプログラム
  - 実はダブルプログラミングであっても必須である。条件は同じ
- 第三者のレビューに耐え得る読みやすいプログラム
  - 読みやすいプログラムとは？





## 多くの組織で抱える問題

- 標準化
  - 標準化するのであれば、適切な方向に向かって標準化したい
  - 何が適切な方向なのか分からない
- 使いまわしのプログラム
  - 例)製造販売後調査の安全性定期報告
    - 1年前のプログラムを変更したいが、内容を忘れている
    - 他の担当者が作成したプログラムを変更したいが、読みづらい
    - あまつさえ、自分が過去に作成したプログラムですら読みづらい

## プログラミングの作法を学ぶ必要性

- ソフトウェアコンストラクションの名著「Code Complete」[2)]
  - 高品質で読みやすいプログラムを作成する技術
  - 名著である、しかしながら……
    - C++やJava等、SAS以外の言語で書かれている
    - SASに応用しづらい項目も多数(クラス・オブジェクト指向等)
    - 情報処理の専門教育を受けていない、統計が専門のスタッフには敷居が高い

2) Steve McConnell (2004) Code Complete 2nd Edition : Microsoft Press.((株)クイープ訳『Code Complete第2版』日経BPソフトプレス、2005年。)





## Code CompleteをSASに適用してはどうだろう

- 社内プロジェクトを立ち上げ
- 将来的な標準化の可能性
  - このプロジェクトでは、皆が正しい方向を知るのが目的、標準化は次のステップ
- メンバーの経験を踏まえ、ディスカッション
- この領域では系統的な議論があまりされていない
- 成果物:50ページの文書

## プログラミングの作法、一目瞭然な例

```
data A;merge DEMOG(in=A) FLAG;by
SUBJID;if A;if AGE >= 20 then
B=1;/*成人フラグ作成*/;run;proc
print data=A;var SUBJID AGE
B;run ;
```

```
data DEMOGwithFLAG ;
  merge  DEMOG(in=DEMOGExist)
         FLAG ;
  by SUBJID ;
  if DEMOGExist ;
  if AGE >= 20 then
     AdultFlag = 1 ;
run ;

proc print data = DEMOGwithFLAG ;
  var SUBJID AGE AdultFlag ;
run ;
```





## ディスカッションポイントのエッセンス

1)レイアウト
2)命名規則
3)コメントと自己解説コード
4)制御構造
以降、その議論の一部を示す

## レイアウト

- 可読性・メンテナンス性に大きく影響
- プログラムの論理構造を明らかにする事が重要
  - プロシジャ・データステップの内部をインデントする
  - if文・do文等の制御構造の処理対象範囲をインデントする
  - ステートメントが複数行にわたる場合は、その事が明確に分かるようにする
  - 役割・意味が異なるブロックは、空白行により分離する
- マルチステートメントはプログラムを読む視点の上から下への流れを妨げる





## レイアウトの例

```
data ■■■■ ;
  ■■■■■■■■ ;          データステップの内部をインデント
  if ■■■■■ then do ;
    ■■■■■■■■■ ;       ifの処理対象範囲をインデント
    ■■■ ;
  end ;
  ■■■■■ ;
  ■■■■■■■■ ;
  do ■■■■■■■■■ ;
    ■■■ ;              doの処理対象範囲をインデント
    ■■■■■■ ;
  end ;
  ■■■■■■ ;
  ■■■ ;
run ;
                       異なる性格のブロックを空白行で分離する
proc ■■■■■■ ;
  ■■■■■■■■ ;         プロシジャの内部をインデント
  ■■■■■ ;
run ;
```

## 命名規則(1)

- 医薬品開発では、オリジナルのデータベースでの命名は、標準化が進んでいる
- ここでは、プログラマが自由に命名出来る一時的な変数名・データセット名やマクロ名等の識別子について議論
- 命名ルールに関しては、歴史的に数々の研究がなされてきた
  - 識別子が表すものを完全に、正確に表していること
  - 他と混同しにくく、覚えやすいこと
  - アルファベット一文字などの意味を持たない短い名前は、ループ変数のみに用いること





## 命名規則(2)

- 例;現在の日付
  - ○:currentDate、todaysDate
  - ×:cd、current、c、x1、x2、date
- 特にマクロ名は、動詞＋修飾語＋目的語の形式をとる事により、内容が明確になる

## コメントと自己解説コード

- 過剰なコメントはプログラムを煩雑にする
  - そもそもコメントが必要なほど複雑なプログラムでは、より簡便なプログラムを検討する方が重要
- 自己解説コード：ソースコードを突き詰めて考えれば、それ自身が一つの詳細なドキュメントであるという考え方
  - 良い構造、良い命名規則、分かりやすいレイアウト等、可読性が高いコードは、そのコード自身を適切に解説する
- コメントは主役ではなく、コード自身が主役である





## 制御構造(1)

- ループや条件分岐はプログラムの上から下への視点の流れを妨げる性質のものであり、一段の注意が必要
- 処理が複雑な場合は、よりシンプルな条件に置き換えられないかを検討する
- ひとつの処理を複数の目的に用いない
- ループの終了条件は明確に記述する

## 制御構造(2)

- ループや条件分岐でネストが深い場合(例えば4レベル以上)は、可読性が著しく低下するため、アルゴリズムを再考する

```
do GROUP = 1 to 2 ;
  do PERIOD = 1 to 2 ;
    do VISIT = 1 to 5 ;
      do TERM = 1 to 3 ;
        ..... ;
      end ;
      ..... ;
      ..... ;
    end ;
  end ;
end ;
```





## ディスカッション(1)

- 品質・可読性・メンテナンス性の向上が目的
- 全てのプログラムで重要
  品質の向上
- 狭義のバリデーションと相性が良い
  可読性の向上
- 使いまわすプログラムと特に相性が良い
  可読性の向上　メンテナンス性の向上

## ディスカッション(2)

- ソフトウェアコンストラクションの技術を考慮したプログラムによって

タイプ文字数増による
プログラミング時間の増加

<<

可読性向上による
確認時間の短縮

品質向上による
デバッグ時間の短縮

メンテナンス性向上による
仕様変更対応時間の短縮

- 個々の技術はもとより、可読性・メンテナンス性・品質の向上に意識が向く事自体が重要





## 今後の課題

- 今回のプロジェクトはあくまでも皆が正しい方向を知るのが目的
- ルール化・標準化とその周知徹底等、社内での統一的な記述は今後の課題
  - まずは汎用マクロ等から適用？

## まとめ

- ソフトウェアコンストラクションの技術を考慮したプログラムにより、プログラム品質を向上させる事で、プログラム開発時間の短縮も見込まれる
- 全てのプログラムにおいてこれらメリットを享受出来るが、特に狭義のバリデーションを行う場合や、プログラムの使い回しを行う場合に相性が良いと言える