翻訳検証テストのテストカバレージに関する理論式

加藤 直孝† 松下 和憲† 有澤 誠††
日本アイ・ビー・エム株式会社†
慶應義塾大学 環境情報学部大学院 政策・メディア研究科††

概要：ソフトウェアの GUI（Graphical User Interface）上に出現する文字列を PII（Program Integrated Information）文字列と呼ぶ．ソフトウェアを国際化するためにはこの PII 文字列を翻訳する必要がある．この PII 文字列の翻訳検証テストのテストカバレージの定量的な評価方法は存在せず，エンジニアは経験と勘でそのテスト量や範囲を決定していた．本研究の目的はソフトウェアの翻訳検証テストのテストカバレージに関連する評価を定量的に行うことである．本論文では，その評価をするための基礎となるモデルと理論式を提示し，最後に実際のソフトウェアに理論式を適用した数値例を示す．この理論式からテストカバレージだけでなく，テスト後の残留要修正 PII 文字列数やテストのストップ基準を算出することができる．

キーワード：翻訳，PII，Program Integrated Information，翻訳検証テスト，テストカバレージ，ストップ基準，修正率

1. はじめに

ソフトウェアの GUI（Graphical User Interface）上に出現する文字列を PII（Program Integrated Information）文字列と呼ぶ．本研究の目的はソフトウェアの翻訳検証テストのテストカバレージに関連する評価を定量的に行うことである．本論文では，そのための基礎となるモデルと理論式を提示する．この理論式から，テストカバレージだけでなく，テストのストップ基準や残留要修正 PII 数を算出できる．

プログラムは PII 文字列を介してユーザー（人間）と意思疎通する．ソフトウェア開発時には，PII 文字列そのものはプログラム中には含めない．プログラム内には key を記述し，テキストリソースファイル内に key と PII 文字列を記述する．そのテキストリソースファイルを PII ファイルと呼ぶ[1, 2]．たとえば，PII ファイル内には「key=文字列」の行が繰り返す．1 行で key とそれに対応する文字列を定義している．本研究では，PII ファイル内の行の左辺を「PII の key」と呼び，右辺を「PII 文字列」と呼ぶ．両辺全体を指す場合は単に「PII」と呼ぶ．

PII ファイルは最初に英語の PII ファイルを準備する．そして，そのファイルを各言語に翻訳することで，ソフトウェアの国際化を実現する．

翻訳者は英語の PII 文字列を，開発中のソフトウェアを使うことなく PII ファイル内で翻訳する．PII ファイル中にソフトウェアのコンテクストはないため，翻訳者は PII 文字列の全てを適切に翻訳することはできない．そのため，翻訳検証テストのテスターは開発の最終段階で実際のソフトウェアを使用して GUI の操作の流れの中で PII 文字列を検証する．この検証を翻訳検証テスト(Translation Verification Test，以後略して TVT)と呼ぶ．

翻訳者が PII ファイル上で PII 文字列を翻訳できない場合，翻訳者は暫定的な翻訳を行い，TVT テスターがテストシナリオ上で翻訳を完了する必要がある．そのため，TVT のテストカバレージが翻訳の完了と未完了を分けることになり，テストカバレージが翻訳の品質を左右する．

ソフトウェアのプログラムコードに関するテストカバレージの議論は古くから行われている[3]．しかし，PII のテストカバレージに関する議論はほとんど行われていない．ソフトウェアのテストは，そのほとんどがバージョンアップにともなうテストであるため，TVT ではバージョン間の英語と日本語の変化を考慮する必要があり，単純に TVT のテストカバレージの測定をすることができないからである．

そこで，我々はバージョン間の日本語と英語の変化を基に PII を分類することで，意味のあるテストカバレージの測定を可能にした．さらに，定量的な評価を行うためにモデルを定義し，そのモデルの変数に対する理論式を開発した．この理論式は，本研究で発見した PII 文字列の翻訳の連鎖修正という概念に基づいている．研究では，さらに，TVT 中の PII を分類別に計数する手法を開発し，実際の製品の TVT で開発した理論式と手法が有効であることを確認した．

本論文では，モデルとその理論式の部分を説明する．2 章で PII の分類を簡単に述べ，3章でモデルと理論式を詳細に説明する．4章で理論式の適用結果と分析を述べる．

2. テストカバレージを求めるための PII の分類

次の3章で述べる理論式は PII の分類ごとに適用する．その分類は，PII の key の存在の有無や PII の新旧バージョンの日本語と英語の文字列の変化を基に行う．

PII文字列の新旧日英の変化に対する分類は文献[4]で行っている．理論式は文献[4]の分類番号（ChgID）ごとに適用できる．本論文では，ChgID[1]を次の（1）から（3）の3つのグループに分け理論式を適用した．

[1]文献[4]のChgIDは全部で 22 個ある．3 つのグループに含まれないChgIDは翻訳対象外のkeyなど例外的なkeyでテストカバレージと関係のないChgIDである．

(1) 旧版[2]で翻訳がすでに存在しているグループ. これを既存部分と呼び, ChgID0からChgID5までを加えたグループである.

(2) PIIが新規に生まれかつ翻訳対象である新規PII翻訳グループ. これを新規部分と呼び, 分類はChgID8のみである.

(3) 旧版で英語が存在し翻訳範囲が拡大した翻訳グループ. これを翻訳拡大部分と呼び, ChgID6とChgID7を加えたグループである.

旧版と新版の GM(Golden Master)中の PII ファイルで ChgID の分類は行う. TVT 中の修正は TVT の直前と直後の PII ファイルを比較することで知ることができる.

3. 翻訳検証テストを定量化するための理論

3.1 連鎖修正

TVT における PII 文字列の修正には 2 種類の修正がある. 1 つはテストシナリオを流したときに, GUI 上で検証した PII 文字列に対する修正である. この修正を GUI 検証修正と呼ぶ. もう 1 つは, その修正に関連して, GUI 上で検証することなく, PII ファイル上での確認のみで行う修正である. この修正を連鎖修正と呼ぶ. 我々は連鎖修正がテストカバレージと深く関係することを初めて発見した.

通常の翻訳でも, 文章全体の一貫性や統一性を図るために, 連鎖修正は行われている. 通常の翻訳では文章中でその修正が適切かどうか確認できる. しかし, PII の場合は GUI 上での検証を必要とする. そのため, PII の翻訳では1つの修正を行ったときに, 関連する修正がPIIファイル上にあるかどうかを確認し, TVT テスターの洞察力により PII ファイル上で連鎖修正すべきかすべきでないかの判断をする. 効率よく修正を行うためには, GUI 上で検証を行わない連鎖修正が必要である.

たとえば, ある PII の英語文字列「Aspect Variant」の翻訳を GUI 上で検証し,「縦横比」から「外観」に修正したとする. このとき, PIIファイル上に他のPIIの英語文字列にも「Aspect Variant」があると, PII ファイル上の確認のみで GUI 上で検証することなく, 翻訳を「縦横比」から「外観」に修正する. この修正が連鎖修正である. 連鎖修正はこのような直接的な訳語の修正だけではなく, 関連する英語のグループや反対語などに対しても行う.

3.2 翻訳検証テストの理論式

3.2.1 変数の定義

次の(1)から(4)で TVT における主要な変数を定義する. ここで, 大文字にしている変数は TVT 中に PII の個数として計測できる変数で, TVT カウント変数と呼ぶ. それ以外の変数を小文字で示し, TVT 理論変数と呼ぶ. 定義する変数の中で, TVT カウント変数 N,V,F,A と TVT 理論変数 u に関する PII のテストカバレージの関係を図1に示す.

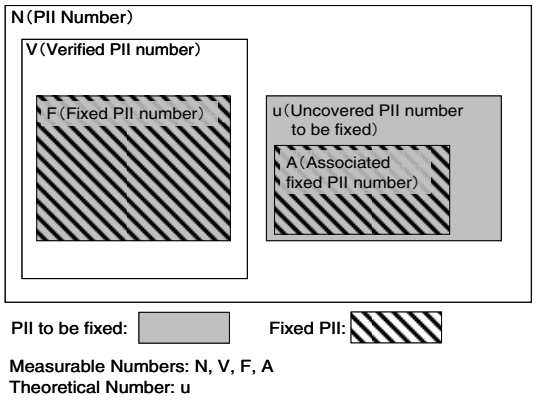

図 1 PII のテストカバレージの関係

(1) テストカバレージ

ソフトウェアのすべての PII ファイル中の PII 総数(すなわち PII の key の総数)を N(pii Number)とし, GUI 上で検証した PII ファイル中の PII の個数を V(Verified pii number)とする. N を PII 総数, V を GUI 検証数と呼ぶ. このときテストカバレージ c(Coverage)を次のように定義する.

$$c = V/N \quad (1)$$

(2) GUI 検証領域修正率

GUI 上で検証した PII に対して, 修正を行った PII の個数を F(Fixed pii number)とする. F を GUI 検証領域修正数と呼び, GUI 検証領域修正率 f(Fixed pii ratio)を次のように定義する.

$$f = F/V \quad (2)$$

(3) 非検証領域要修正率

TVTのGUI上で検証できなかったPIIのうちで修正を必要とするPIIの個数をu(Uncovered pii number to be fixed)とする. uを非GUI検証領域要修正数と呼び, 非GUI検証領域要修正率 f_u (Uncovered pii fixed ratio)を次のように定義する.

$$f_u = u/(N-V) \quad (3)$$

(4) 連鎖修正率

GUI 上で検証しかつ修正した PII の個数 F に対して, その修正した PII の文字列に関連して, 最終的に GUI 上で検証を行っていない PII の文字列に対して修正を行った PII の個数を A(Associated fixed pii number)とする. A を連鎖修正数と呼び, 連鎖修正率 a(Associated fixed pii ratio)を次のように定義する.

$$a = A/F \quad (4)$$

3.2.2 連鎖修正率とテストカバレージの関係

本項では連鎖修正率 a とテストカバレージ c の間の関係式を示す. c の値が変化したときに, a が c とどのような関係にあるかを正確に求めるためには, a と c の統計的な情報を必要とする. しかし, 近似的な関係であれば, 連鎖修正

[2]ここでは旧バージョンを旧版, 新バージョンを新版と呼ぶ.

率 a をテストカバレージ c の 1 次式として示すことができる．次にその近似式を説明する．

テストカバレージ c が 0 に近づくとき，連鎖修正率 a が g に近づくとする．この g（associated pii Group rate）を連鎖修正グループ率と呼ぶ．また，テストカバレージ c が 1 に近づくときは，連鎖修正率 a は 0 に近づく．そこで，横軸を c，縦軸を a として，a は c の 1 次式で，点(0,g)と点(1,0)を通る直線とする．このとき，次の関係式が成り立つ．

$$a = g(1-c) \quad (5)$$

c はテストカバレージであるから，c の範囲は[0,1]である．この式(5)は実際の TVT で a と c を求めることができれば，g を決定できることを示している．

3．2．3 残留要修正 PII 数

本項では，TVT 後に修正されずに残った PII の個数 r（Residual pii number to be fixed，残留要修正 PII 数）を求める式を示す．図 1 を用いて説明する．図 1 の灰色の部分は修正を必要とする PII を示し，斜線部分は TVT で実際に修正を入れた部分を示す．図 1 の非 GUI 検証領域要修正数 u から連鎖修正数 A の斜線部分を引いた部分の個数が残留要修正 PII 数 r である．式で示すと，

$$r = u-A \quad (6)$$

となる．

ここで，GUI検証領域修正率fと非GUI検証領域要修正率f_uは等しいと仮定[3]すると，$f = f_u$および式(3)より

$$f = u/(N-V) \quad (7)$$

となる．

式(1)より V を消去して

$$u = Nf(1-c) \quad (8)$$

となる．

式(1)，(2)，(4)，(6)，(8)より，

$$r = Nf\{1-c(1+a)\} \quad (9)$$

$$r = (NF/V)\{1-(V/N)(1+A/F)\} \quad (10)$$

となる．

実際に TVT を実施することで大文字の N，V，F，A を測定できるので，式(10)より残留要修正 PII 数が求まる．また，式(9)に式(5)を代入すると，

$$r = Nf[1-c\{1+g(1-c)\}]=Nf(gc-1)(c-1) \quad (11)$$

となる．残留要修正PII率をf_r(Residual pii ratio to be fixed)とすると，f_r=r/Nであるので，

$$f_r = f[1-c\{1+g(1-c)\}]=f(gc-1)(c-1) \quad (12)$$

となる．

テストのストップ基準（Stop criterion）を残留要修正PII率を基に決め，その値をf_sとすると，f_rがf_s以下になったときテストをストップすればよい．そのときのテストカバレージの値をc_sとすると式(13)ができる．c_sで解くと式(14)になる．

$$f_s = f(gc_s-1)(c_s-1) \quad (13)$$

$$c_s = [(g+1)-sqrt\{(g+1)^2-4g(1- f_s/f)\}]/2g \quad (14)$$

sqrtは平方根（square root）を示す．f_sの値を決めるとテストのストップ基準となるテストカバレージが決まる．

3．2．4 ミニマルテストカバレージと既存レベルへのテストカバレージ

前項の3.2.3項の式(13)，(14)から，f_sの値を決めることで，ストップ基準となるテストカバレージが求まる．本項では次の（1）（2）で，ミニマルテストカバレージc_m（Minimal test coverage，最低限必要なテストカバレージ）と既存レベルへのテストカバレージc_o(Old level test coverage，新規部分の残留要修正率を既存部分のそれと同じにするテストカバレージ）を求める．

（1）ミニマルテストカバレージ

式(13)のf_sの値を 0 とすることで，最低限必要なテストカバレージc_mを求めることができる．このテストカバレージc_mをミニマルテストカバレージと呼ぶ．

$$0 = f\{(gc_s-1)(c_s-1)\} \quad (15)$$

より

$c_s = 1/g$ または 1 のときf_s=0 となる．したがって，

g が 1 より大きい場合，

$$c_m = 1/g \quad (16)$$

がミニマルテストカバレージである．

g が 1 以下の場合，

$$c_m = 1 \quad (17)$$

がミニマルテストカバレージで 100%のテストカバレージを必要とする．

（2）既存レベルへのテストカバレージ

新規部分を中心にテストシナリオを作成する場合，新規部分が既存部分と同等の要修正率になるまでテストすることが 1 つのテストのストップ基準になる．コスト的に$f_r = 0$ の達成が困難な場合，この基準は重要である．

TVT後の新規部分の残留要修正率を既存部分の残留要修正率と同等にする新規部分のテストカバレージを既存レベルへのテストカバレージと呼び，c_oとする．

c_oを求めるために，新規部分と既存部分の分類ごとに残留要修正率f_rを求めることで，新規部分の残留要修正率を既存部分の残留要修正率と同じにするテストカバレージを算出できる．残留要修正率f_rの添え字を変更し，添え字nr(New residual ratio)を新規部分の値，添え字or(Old residual ratio)を既存部分の値とすると，新規部分のテストカバレージのストップ基準c_oを求めるために式(14)のfをf_{nr}に，f_sをf_{or}に，c_sをc_oに置き換えればよい．

$$c_o = [(g+1)-sqrt\{(g+1)^2-4g(1- f_{or}/f_{nr})\}]/2g \quad (18)$$

新版部分を対象としてテストシナリオを作成する場合，このテストカバレージ値c_o までテストを行えば，新規翻訳部分

[3] 4 章の実際のソフトウェアでテストシナリオを偶数奇数番目で分けて修正率を調べた．その結果，偶数番目と奇数番目の修正率はほぼ同じ修正率になった．

の残留要修正率が既存翻訳部分の残留要修正率と同じになる.

4. 実際のソフトウェアに対する測定結果と分析

4.1 測定対象ソフトウェアと測定結果

実際に TVT カウント変数を計測した. 対象としたソフトウェアは CAD/CAM システムソフトウェアで, PII の総数は 21 万個を超える. テスト量は 2 人で 2 週間のテストである. 2 章で説明した 3 つのグループに対する測定結果と理論式の適用結果を表 1 に示す.

表 1 測定結果と理論式適用結果

	既存部分 ChgID0-9	新規部分 ChgID8	翻訳拡大部分 ChgID6-7
N	173,946	7,455	4,605
V	3,766	738	42
F	84	56	4
A	149	97	2
c	0.0217	0.0990	0.00912
f	0.0223	0.0759	0.0952
a	1.773	1.732	0.500
u	3796	510	435
r	3647	413	433
f_u	0.0223	0.0759	0.0952
f_r	0.0210	0.0554	0.0939
g	1.813	1.922	0.505
c_m	0.552	0.520	1
c_o	-	0.301	-

4.2 結果分析とその意味

(1) 既存部分のテストカバレージ c(0.0217)は新規部分の c(0.0990)の 1/4 から 1/5 である. このテストシナリオが新規部分に焦点を当てたシナリオであることが分かる. 焦点の程度が数値で分かる意味は大きい.

(2) 既存部分の GUI 検証領域修正率 f(0.0223)は新規部分の GUI 検証領域修正率 f(0.0759)の 1/3 以下になっている. これは既存部分がより枯れた翻訳になっていることを示している. このソフトウェアは以前のバージョンの TVT も同程度のテスト量だったので, 既存部分の翻訳品質は製品出荷後の別の PII の翻訳検証テストで品質を向上した結果であることが分かる.

(3) TVT後の新規部分の残留要修正率f_r(0.0554)は既存部分の残留要修正率f_r(0.0210)よりかなり大きな値である. 新規部分に焦点を当てたシナリオではあるが, テストが不十分であることが分かる. 不十分である度合いが数値で示された意義は大きい.

新規部分について, 既存レベルへのテストカバレージc_o(0.301)と実際のテストカバレージc(0.0990)を比較すると, 新規部分を既存部分の残留要修正率にするためには, 測定したTVTの約 3 倍のテストカバレージが必要であることがわかる. 分析からどの程度テストカバレージを増せば既存部分と同等の品質になるかが数値で分かる意味は大きい.

(4) 連鎖修正率 a は新規部分(1.73)と既存部分(1.77)に大差はない. 連鎖修正が一様であることを裏付けている.

(5) ミニマルカバレージc_mも新規部分(0.520)と既存部分(0.551)に大差はない.

(6) 翻訳拡大部分(ChgID6,7)はサンプル数が少ないが, テストカバレージ c は既存部分(0.0217)よりさらに低い値(0.00912)になっている.

(7) 翻訳拡大部分の GUI 検証領域修正率 f(0.0952)は新規部分(0.0759)よりやや悪い値である. 連鎖修正率 a(0.500)は新規部分より低い. ただし, サンプル数が少なく有効な比較とは言えない.

理論式を実際の TVT に適用した場合に得られるこれらの結果は, 理論式作成時に導入した仮定と矛盾する結果は出現しなかった.

分析結果は TVT 後や次回のバージョンの対策に活用できる. また, 短い理論データ収集のための TVT を実施すれば, その結果を本番の TVT のテスト量の決定に活用できる.

5. おわりに

本論文ではソフトウェアの翻訳検証テスト(TVT)のテストカバレージに関連する評価を定量的に行うための基礎となるモデルと理論式を提示した. この理論式により, テストのストップ基準や残留要修正 PII 数を算出できる. テストシナリオがカバーしている部分も新規部分なのか旧版部分なのか, 数値でより具体的に把握できる. PII の翻訳品質向上を確実に行える意義は大きい. 今後の課題は, この理論式が示した TVT カウント変数をソフトウェア開発環境が自動的に計数する仕組みを開発することである.

参考文献

[1] Andrew Deitsch and David Czarnecki, *Java Internationalization*, O'REILLY, 2001

[2] Nadine Kano, *Developing International Software for Windows95 and Windows NT*, Microsoft Press, 1995

[3] Cem Kaner, Jack Falk, Hung Q. Nguyen, *Testing Computer Software*, Wiley, 2nd Ed., 1999

[4] Naotaka Kato, Kazunori Matsushita, and Makoto Arisawa, *Utility for Showing Program Integrated Information Changes between Versions in a Translation Verification Test*, 2008 Second International Symposium on Universal Communication, pp. 330-337, 2008