言語Ｃ処理系“知紙”の実現におけるソフトウェアの再利用

7Ｊ－6

中村浩之，田中広幸，森永智之，早川栄一，並木美太郎，高橋延匡

（東京農工大学　工学研究科　電子情報工学専攻）

１．はじめに

OS/omicron 第４版[1]用の言語Ｃ処理系“知紙（KAZUSHI）”を実現した．“知紙”の実現では，我々の研究室で実現し利用している言語Ｃ処理系 CAT [2]の再利用を行った．CAT はプロセッサ依存部とプロセッサ独立部分，言語依存部と言語独立部とを明確に分離して作られている．このため，例えばプロセッサ依存部分を新たに作り直すことで，別プロセッサ用の言語Ｃ処理系を実現することが可能である．

本稿では，“知紙”の実現において行ったソフトウェアの再利用の効果について，定量的に評価する．

２．コードジェネレータの実現

コードジェネレータは，CAT の言語Ｃパーザの出力する中間コードを読み，アセンブリ言語を出力するフェーズである．コードジェネレータはプロセッサ依存部分なので，ここは新たに実現する必要がある．しかし，CAT のコードジェネレータの内部モジュールでもプロセッサ独立の部分が存在する．そこで，処理系の早期実現と信頼性の確保を目的として，CAT のコードジェネレータの再利用を一部について行った．

“知紙”のコードジェネレータの実現は，次の二段階で行った．

（１）再利用部の変更と動作確認

まず，再利用する各モジュールに処理変更や追加，削除を加えた後に，この部分だけで実行テストを行った（ここまでの段階を第一段階と呼ぶ）．

（２）新規作成部の実現

新規に作成するモジュールを第一段階で実現したモジュールに追加し，コードジェネレータを完成させる（この段階を第二段階と呼ぶ）．

完成したコードジェネレータのソース総行数は 22179 行であった．このうち，再利用した部分のソース行数は 7908 行となった．約 35.7 %を再利用したソースで構成できた．

３．再利用の評価

各段階で，実行時に発生したバグの数や関数呼出しインタフェースのチェックを行うツールを使用した結果を収集した．これらを比較して，ソースプログラムの再利用の効果を評価する．また，第二段階でのデバッグで発見したバグについては，その原因別に数を集計した．この結果から，言語Ｃのプログラムで起こしやすい誤りについて述べる．

（１）検出したバグの個数

第一段階，第二段階で検出したバグの数を表１に示す．各段階では全ファイルのコンパイル終了後に，引数や関数のサイズの整合性を確認するツールを使って一度チェックを行った．それによって検出できたバグの数も同時に示す．

表１：デバッグしたバグの個数

	第一段階	第二段階
発見したバグの数	8(988.5)	211(105.1)
ツール検出結果	27(292.9)	178(124.6)

()内はソース行数との比

ツールの検出したバグの数や実行時に発生したバグの数について，各段階でのソース行数を検出したバグ数で割った比で比較を行った．その結果では，ツールの検出結果や，デバッグで検出したバグ数ともに第一段階のほうが第二段階よりも少なかった．この結果から，ソースプログラムを再利用して作ったモジュールは，新しく作ったモジュールよりも信頼性が高いと言える．

Software Reuse in the Implementation of KAZUSHI , a C Compiler

Hiroyuki Nakamura , Hiroyuki TANAKA , Tomoyuki Morinaga , Eiichi Hayakawa , Mitarou NAMIKI and Nobumasa TAKAHASHI

Tokyo University of Agriculture and Technology

（２）デバッグ時間

第二段階でのデバッグでは，そのデバッグ作業について，作業時間を測定した．測定は OS/omicron ファイルシステムの，追記型光ディスクを用いた世代管理機能 [3] を用いて行った．このデバッグ時間とデバッグしたバグの数の関係について図 1 に示す．

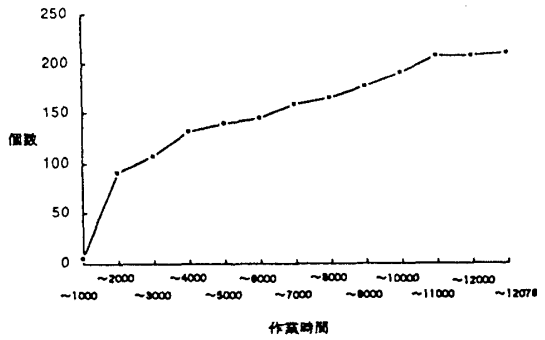

図 1：デバッグ時間とバグ数

測定の結果，211 個のバグを取るのにかかった時間は 201 時間 16 分であった．単純に平均を取ると，一つのバグを取るのに 58 分を要した．一方，第一段階のソース行数と同じ量のソースを新規に書いた場合に発生するバグの数を，第二段階のソース行数とバグ数の比から推測すると，約 75 個となる．実際に第一段階で発生したバグ数は 8 個なので，その差は 67 個である．67 個のバグを取るのに必要な時間を推測すると，64 時間 46 分となる．これだけのデバッグ時間を短縮できたと言える．

（３）バグの傾向

実現の第二段階でのデバッグで発見したバグについて，その発生原因別に集計した．集計結果を図 2 に示す．これらのバグはすべて，第二段階で新たに追加したモジュールで発生しているので，新しくソフトウェアを記述した場合と同じ考えてよい．

この結果では，言語Ｃでバグの原因となる最も多い原因は，引数に関係する間違いが最も多い．これと同種のバグで，（１）で述べたツールが検出した数は 56 個であった．これらの合計はツールとデバッグによって検出したバグの 36.3 % を占めている．この結果から，言語Ｃでよく発生するバグは関数引数の型やサイズのずれであることが言える．

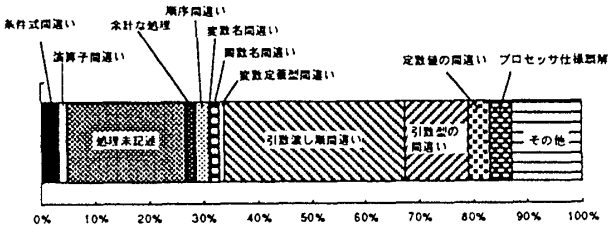

図 2：発生原因別のバグの集計（第二段階）

４．おわりに

本項では言語Ｃ処理系“知紙”のコードジェネレータの実現において行ったソフトウェアの再利用について，その実現における効果の定量的評価を述べた．また，言語Ｃのプログラムにおいて発生しやすいバグの傾向について述べた．今後は，“知紙”を実際のソフトウェア開発に利用し，その際に発生するバグを記録し，再利用を行って実現したソフトウェアの信頼性についても評価を行う．

参考文献：

[1]早川，並木，高橋：手書きインタフェースを支援する OS OS/omicron 第４版の構成，情報処理学会シンポジウム論文集 Vol.92, No.7 pp.35-42

[2]並木他６：OS/omicron 用システム記述言語Ｃ処理系 Cat のソフトウェア工学的見地からの方式設計，電子情報通信学会論文誌 D Vol.J71-D No.4 pp.652-660, 1988-4

[3]横関，並木，中川，高橋：追記型光ディスクの仮想的な書換えと世代管理機能の実現，電子情報通信学会論文誌 D-I vol.J72-D-I No.6 pp.414-422 1989-6