B-013

定理証明技術を用いたソースコードの補完手法
A Code Completion Method by Using Automated Theorem Proving

小池 遼平　小野 康一　深澤 良彰
Ryohei Koike　Kouichi Ono　Yoshiaki Fukazawa

1. はじめに

ソフトウェアの高機能化・多機能化に伴いソフトウェアの規模や複雑度は年々増加している．一方でソフトウェアの開発にかけられる期間は短くなりつつあり，近年のソフトウェア開発では全てを新規開発するケースは稀である．このような背景から，開発者は効率よく且つ正確にソフトウェアを開発することが求められている．

ソフトウェア開発をより効率的に行うための技術の 1 つに，プログラムのコーディングを効率化させるコード補完がある．コード補完とは途中まで書いたソースコードをもとに，不足部分を自動的に補完する技術であり，現在多くの開発環境において標準機能として搭載されている．また近年では，実装途中のコードと類似している既存のソースコードを探索することで，補完候補として複数の命令文からなるコード断片を推薦するような技術がいくつか存在する[5][6][7][8][9]．しかし，これらの研究では，推薦されたコードが実装途中のコードに対して妥当なものであることを保証することができない．このため，推薦されたコードを補完したことによって完成したコードが開発者の意図通りに動作しなかった場合，開発者はコードに修正を加えるか，補完された部分を再び候補の中から選択し直すという手間を強いられることになる．

開発規模の大きさによらず，多くの場合はコーディングを行う前の段階でどのような機能を実装するか決定されているはずである．記述途中のソースコードのあるブロックにおいて，考えている機能を実現させるための具体的な実装方法や記述方法がわからない場合でも，「このような振る舞いをするはずである」といった仕様レベルでの理解を持っていることが多い．補完候補となるコード断片を探索する際に仕様レベルでの情報を与える事により，対象となるコード断片の候補はより限定的に絞り込むことができると考えられる．

そこで本研究では，Java によって記述されたソースコードに対し，プログラム検証手法の 1 つである定理証明技術を組み込んだコード補完手法を提案する．提案する手法は実装途中のソースコードに加え，実装しているメソッドの事前条件/事後条件を考慮するため，開発者が求めている仕様を充足するコードに限定して推薦を行うことができる．事前条件/事後条件の記述には JML を用いた形式仕様記述を用いることとする．

本研究の貢献は以下のとおりである。

・実装中のソースコードに対するコード断片補完技術の新しいアプローチを提案する．

・大規模なコードベースからメソッド呼び出しのシーケンスを抽出し，N-gram による統計的手法を用いることで，コード断片の補完のための最適なデータベース構築手法を提案する．

・形式手法である定理証明技術を用いることで，推薦されたコード断片が適切なものであることを保証することができる．

以下，第 2 節では本研究が対象とするコード断片の補完について，具体的なコード例と先行研究における問題点を述べる．第 3 節では提案手法の概要と，その背景技術の説明を行う．第 4 節では提案手法における各処理の詳細を述べる．最後に第 5 節ではまとめと今後の課題について述べる．

```
public class SimpleOutlineView extends ViewPart {
  ...
  private void widgetSelected(SelectionEvent e) {
    IWorkbench iwb= PlatformUI.getWorkbench();
    IWorkbenchWindow iwbw =
      iwb.getActiveWorkbenchWindow();
    IWorkbenchPage iwbp = iwbw.getActivePage();
    IEditorPart iep = iwbp.getActiveEditor();

    /***
    * Code for getting an ICompilationUnit (icu) object
    ***/

    try {
      IJavaElement elements[] = icu.getChildren();
    ...
    }
    ...
  }
}
```

図 1　記述中コードの例

2. 研究の動機

本節では，例を用いて解きたい問題と従来研究における問題点について説明を行う．

2.1 実装途中のソースコードの例

開発者によって途中まで実装されたソースコードの例を図 1 に示す．この例は Eclipse[1]を用いたプラグインの実装の様子を示しており，Eclipse 上のエディタに記述されたコードの情報を取得して，アウトラインビューに必要な情報を表示するためのプラグインを作成しようとしている．Eclipse ではソースコードに対応するインタフェースとして ICompilationUnit が提供されており，エディタ上のコードに対応した ICompilationUnit オブジェクトを取得することで情報を得ることができる．そのためには以下のインタフェースにアクセスする必要がある．

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
A:  IEditorInput editorInput;
    IWorkingCopyManager manager;
    ICompilationUnit icu = manager.getWorkingCopy(editorInput);

B:  IFile file;
    ICompilationUnit icu = JavaCore.createCompilationUnitFrom(file);

C:  CompilationUnit fASTRoot;
    ICompilationUnit icu = (ICompilationUnit) fASTRoot.getTypeRoot();

D:  IType type;
    ICompilationUnit icu = type.getCompilationUnit();

E:  JavaEditor fEditor;
    ICompilationUnit icu = SelectionConverter.getInputAsCompilationUnit(fEditor);

F:  IJavaElement javaElement;
    ICompilationUnit icu = (ICompilationUnit) javaElement.getAncestor(IJavaElement.COMPILATION_UNIT);
```

図 3　ICompilationUnit を取得するコード断片の例

```
public class SimpleOutlineView extends ViewPart {
  ...
  private void widgetSelected(SelectionEvent e) {
    IWorkbench iwb = PlatformUI.getWorkbench();
    IWorkbenchWindow iwbw =
      iwb.getActiveWorkbenchWindow();
    IWorkbenchPage iwbp = iwbw.getActivePage();
    IEditorPart iep = iwbp.getActiveEditor();

    IJavaElement javaElement =
      JavaUI.getEditorInputJavaElement
        (iep.getEditorInput());
    ICompilationUnit icu =
      (ICompilationUnit)javaElement.getAncestor
        (IJavaElement.COMPILATION_UNIT);

    try {
      IJavaElement elements[] = icu.getChildren();
      ...
    }
    ...
  }
}
```

図 2　コード断片埋込後の例

① Eclipse のパースペクティブやエディタ，ビューの土台となる IWorkbench
② Eclipse の画面を表す IWorkbenchWindow
③ Eclipse の画面に表示されているエディタおよびビューの群（ページ）を表す IWorkBenchPage
④ Eclipse の画面に表示されているパーツのうちのエディタを表す IEditorPart

図 1 のコードは，これらのインタフェースへ順にアクセスを行うことで IEditorPart オブジェクトを取得するためのコードまでは実装したものの，ICompilationUnit オブジェクトを取得するためのコードが記述できていない状態を示している．

図 2 は Eclipse 本体のソースコードから抽出したコード断片（の最小単位）の一例である．ICompilationUnit オブジェクトを取得するためのメソッド呼び出しは複数存在し，またそのメソッドを呼び出すために必要となる引数やレシーバーオブジェクトの種類がすべて異なることをこの例は示している．このように，目的のオブジェクトを取得するための手順が 1 通りに定まらない場合，複数のコード断片候補の中からいずれかの候補を開発者が選択する必要がある．さらにこれらのコード断片を実装中のコードに適用させるためには，例えば F のコード例の場合であれば図 3 の例が示すように，IJavaElement オブジェクトを取得するためのコード記述を別に行う必要がある．

2.2 既存のコード断片補完技術とその問題点

型情報を用いたコード断片補完手法が幾つか提案されている[5][6][7]．これらの手法は，特定の型のインスタンスを取得するためのメソッド呼び出し記述の推薦に特化しており，2 つの型名の組を入力として与えることで，一方の型のインスタンスからもう一方の型のインスタンスを得るためのコード断片記述を取得することができる

Nguyen ら[8][9]は編集中のソースコードに適する，複数のメソッド呼び出し文データベースに API の使用パターンを構築し，これらをもとにして記述中のコードを補完する手法を提案した．この手法では，実装中のコードから使用できる変数を入力として抽出し，データベースの中から最も類似している API の使用パターンを取得して実装中のコードを補完する．また，候補に対して行うスコアリングの方法は，主にコード行数，データベース内での同一コードの出現頻度，記述途中のコードとの類似度によって決定さ

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

れる．コード行数は短い方が，出現頻度は高い方が，類似度は高い方が，スコアはより高くなる．

これらの手法では実装中のコードに対する妥当性に基づいた探索を行っていないため，推薦されたコードが正しいものであることを保証することができない．しかし，コード補完における保証を支援するための技術はこれまでに存在していない．なぜならば，保証のための検証技術は実行時に多くの計算コストを必要とする重い手法であるからである．このため，推薦されたコードを 1 つずつ順に実装中コードに埋め込みプログラムが正しく動くか検証するという単純な方法では，大幅な実行コストが掛かることを考慮すると実用的な手段ではないと言える．

3. 形式手法

本研究では Java 言語で書かれたプログラムを対象に，開発者が定義した事前条件・事後条件などの仕様を入力として，その仕様を充足するコード断片の候補を開発者に提示することで，開発者によるコーディングの作業を支援するための補完手法を提案する．2 節で述べたような問題を解決するためには，補完候補となるコード断片を含めたプログラム全体の正当性を検証する必要がある．プログラムの検証手法にはソフトウェアテストや形式手法などが存在するが，本研究では形式手法の 1 つである定理証明技術を適用する．検証に定理証明技術を用いる理由は以下の通りである．

・プログラムを実行することなく，静的に正当性検証を行うことができる．

・ソフトウェアテストでは用意されたテストケースに基づき検証を行うため，欠陥が存在することを示すことは可能であるが，欠陥が存在しないことを証明することができない．このため，潜在的な欠陥が含まれる可能性がある．

本節では，研究の背景となる形式手法について，その諸技術について簡単に述べる．

3.1 Design by Contract

Design by Contract（以下，DbC）とは，安全性を向上させるためのソフトウェア開発手法の 1 つである[11]．DbC の主要な考え方は，クラスとそのクライアントが互いに契約を結んでいるというものである．クライアントは，クラスによって定義されたメソッドを呼び出す際に特定の条件を保証していなければならない．一方でクラスは，呼び出し後に保持される特定のプロパティを保証している必要がある．これらの条件はそれぞれ事前条件，事後条件と呼ばれ，DbC においてはこれらの条件を用いることで事前にソフトウェアの仕様を特定する設計手法が用いられる．

本研究では DbC に基づき，契約を満たすようなソースコードを「正しいコード」であると定義する．

3.2 Java Modeling Language

Java Modeling Language（以下，JML）[2]とは， Java プログラム中に注釈（アノテーション）として，DbC に基づき事前条件・事後条件・不変条件などの契約を仕様として記述することができる[3]．

```
/*@ requires x >= 0;
  @ ensures
  @  \result >= 0 && \result * \result <= x
  @   && x < (\result + 1) * (\result + 1);
  @*/
public static int sqrt(int x) {
   int count = 0, sum = 1;
   /*@ loop_invariant
     @   count >= 0 && x >= count*count &&
     @   sum == (count+1)*(count+1);
     @*/
   while (sum <= x) {
      count++;
      sum = sum + 2*count+1;
   }
   return count;
}
```

図 4　sqrt メソッドにおける JML 記述の例

図 4 は sqrt メソッドに求める機能振舞いを表現した JML 仕様記述である[1]．これらの契約は，以下のキーワードを用いて実現される．

requires メソッドの事前条件を記述する．実行時に呼び出し側が満たしているべき契約を表す.

ensures メソッドの事後条件を記述する．呼び出しメソッドの実行後に満たしているべき契約を表す.

invariant クラス不変式を記述する．このクラスのオブジェクトが生存中に各インスタンス変数が満たすべき契約を表す.

3.3 定理証明

定理証明とは形式的検証手法の一つであり，あるプログラムが与えられた仕様を満たすことを証明するための手法である．検証したい性質を定理の形で記述を行い，証明系と呼ばれる定理証明ツールを用いて定理が正しいかどうか確認することができる．実装したプログラムが仕様どおりであることを個別のテストケースを用いて確認するのではなく，いかなる場合でも仕様どおりであることを論理的に保証することができる．

定理証明による検証手法は Floyd-Hoare 論理[16]に基づき，事前条件 P を満たす状態においてプログラム C を実行した場合に事後条件 Q を必ず満たすことを証明することで行われる．

自動定理証明を行うためのツールの 1 つに，ソフトウェア検証プラットフォームの Why3[4]がある．主な用途として，アノテーションが付与された C または Java で書かれたプログラムの検証に用いられる．プログラムを読み取り外部の定理証明ツールの入力となる検証条件を自動で生成することにより，契約として与えられた仕様記述とプログラムが一致しているかどうかを自動検証することができる．

4. 定理証明を用いたコード断片補完

提案手法はデータベース構築，コード断片候補作成，定理証明の 3 つの機能によって構成されている．本手法の概

[1] http://krakatoa.lri.fr/krakatoa.html

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

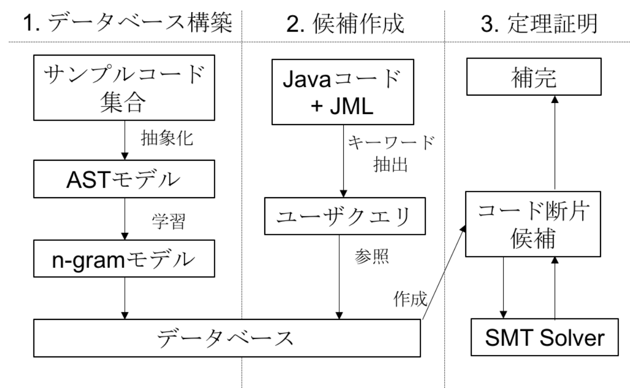

図 5 提案手法の概要図

要図を図 5 に示す．開発者が本手法を実装したコード補完システムを実行すると，以下の処理が行われる．
1. 開発者によって与えられた実装中のコードおよび JML 記述をもとにユーザクエリが作成される．
2. ユーザクエリをもとにデータベースの探索が行われ，複数のコード断片候補が作成される．
3. 2 の処理で得られたコード断片のうちの 1 つを実装中コードに埋め込み，定理証明を実行する．
4. 3 の処理を，コード断片候補の数だけ実行する．
5. 3 の処理で検証に成功したコード断片の候補を，開発者に提示する．

以降では，各処理における詳細を説明する．

4.1 データベースの構築

本研究で提案する手法では，補完候補となるコード断片を探索するためのデータベースが必要となる．データベース構築処理は，開発者がコード補完のリクエストを送る前に完了している必要があるため，システム構築の際に 1 度だけ実行される．

補完候補データベースには以下の情報が必要となる．

・補完候補となるコード断片
・同一データの出現頻度
・データベース探索に用いるメタデータ記述
・N-gram モデルに基づいた確率分布データ

データベースとして利用するコード断片には，変数名のみが異なるものや，依存関係を持たない文の順番が異なるものなどが存在する可能性がある．これらのコード断片を同一のものとみなすために，本研究ではコード断片の抽象化を行い，グラフベースの言語モデルとして表現する．

メタデータ記述は，抽象化されたそれぞれのコード断片に付与されるものであり，データベース探索のキーワードとして用いられる．キーワードにはクラス名，スーパークラス名，インタフェース名，宣言されている変数の型名，呼び出しメソッド名などの情報を記述する．

4.1.1 サンプルコードの抽象化

まず過去に作成されたソフトウェアから Java ソースコードの集合を抽出し，これらのソースコードをメソッド単位に分割する．次にソースコードの抽象化を行うために，得られたメソッド集合を AST(Abstract Syntax Tree)に変換する．本研究では Java のソースコードを AST に変換するためのツールとして，JDT の AST Parser[17]を用いる．

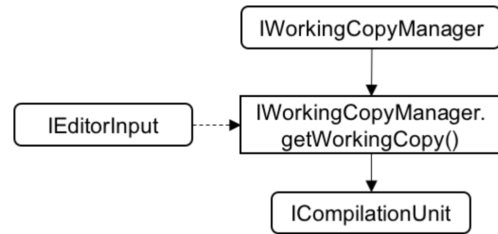

図 6 コード断片のグラフ構造変換の例

次に AST を有向グラフのデータ構造に変換する．有向グラフにするメリットは，変数のデータ依存や制御依存の関係が明確になることである．メソッド呼び出し文や制御構文をノードで表し，データ依存や制御依存の関係はエッジを用いることで表現する．この時，for 文や while 文などのループ構造を含むプログラムは，内部の処理を 1 回実行したものとみなし作成する．本研究では Groum[10]を用いたグラフ構造を利用する． 図 6 は，図 2 の A のコード断片をグラフ構造へ変換した例である．実線の矢印は制御依存関係を，点線の矢印はデータ依存関係をそれぞれ示している．また変数名はソースコード抽象化の過程を経て，対応するクラスの simple name として表現されている．

以上の工程によって得られた有向グラフモデルは構文テンプレートとしてデータベースに保持される．後のデータベース探索の処理において検索されるように，それぞれの構文テンプレートについてメタデータ記述を付与する．

4.1.2 確率分布データベースの作成

本研究ではデータベースの構築手法として，N-gram を用いる．N-gram とは，直前 N-1 個のメソッド呼び出しの並び方から次のメソッド呼び出しを推測する統計的手法である．本研究では 4.1.1 節で作成されたグラフ構造のうち，メソッド呼び出しに対応するノード 1 つを 1 言語単位と定義する．

本手法で作成する確率分布データベースは図 7 のように構成される．ID タグは，グラフ構造のデータから抽出されたメソッド呼び出しのセットに順に番号を割り当てたものである．Freq タグは同一データの出現頻度を表したものである．本手法ではこの値が大きいほど，多くのソースコードで使用される確率が高いメソッド呼び出しセットであると考える．

4.2 コード断片候補の作成

4.1 節によって作成されたデータベースを用いて，途中まで実装されたコードの続きを補完するようなコード断片候補の生成を行う．この処理は，開発者がコード補完のリクエストを送るたびに実行される．

4.2.1 ユーザクエリの作成

本研究で提案するコード補完システムを利用するためには，実装中のメソッドに対する実装中のコードおよび仕様記述の 2 つが開発者によって与えられる必要がある．本手法では，実装中のコードに出現するメソッド呼び出しの並びを抽出し，ユーザクエリとして作成する．このユーザクエリは，データベース探索のためのキーワードとして用いられる．

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
ID: 1
Freq: 4
<IWorkbenchWindow.getActivePage>,
<IWorkbenchPage.getActiveEditor>,
<JavaCore.createCompilationUnitFrom>

ID:2
Freq: 8
<IWorkbenchWindow.getActivePage>,
<IWorkbenchPage.getActiveEditor>,
<IType.getCompilationUnit>
...
```

図 7　確率分布データベースの内容の例

4.2.2 コード断片候補の探索

コード断片候補の探索は，データベース内の構文テンプレートに付与されたメタデータをもとに行う．

4.3 定理証明によるコード断片の検証

この処理では，4.2.2 節で出力されたそれぞれのコード断片の候補に対して，開発者によって与えられた事前条件・事後条件の仕様を満たすかどうかを，定理証明を用いた検証を行うことで調べる．本研究では，検証プラットフォームとして Why3[4]を利用し，Why3 の構成ツールとして Krakatoa[15]と SMT Solver の 1 つである Yices[12]をそれぞれ用いる．コード断片の検証手順の概要図を図 8 に示す．

まず，4.2.2 節で作成されたコード断片の候補のうちの 1 つを実装中のソースコードと組み合わせ，実装が完了したコードとみなす．これらのコードは JML アノテーション付き Java プログラムとして，Krakatoa に入力として与えられる．Krakatoa は Why3 が読み込み可能な Hoare 検証文を出力し，さらに SMT Solver が読み込み可能な述語論理に変換される．SMT Solver は JML アノテーション付き Java プログラムに対応した述語論理を定理として受け取ることで，与えられたコード断片が仕様を満たすか自動で検証を行い，妥当性の判断を行う．このようにして妥当であるとみなされたコード断片は最終的に補完候補となり，開発者に優先的に提示するようにする．

5. おわりに

本研究では，形式手法である定理証明技術を用いて，実装中のソースコードに対し開発者が求めている仕様を充足するようなコード断片を推薦する手法を提案した．

今後の課題として，まず仕様抽出の自動化を行うことで，開発者のユーザビリティを向上させる必要があると考えられる．具体的には，表明を自動生成するためのツールである Daikon[14]等を組み合わせて用いることで，開発者が形式仕様記述を行う手間を省くことによる効率化などが考えられる．また今回提案した手法では，コード断片生成の処理と定理証明による検証の処理が独立したものとなっているが，これらの処理に依存関係を持たせる事により，実行時間の削減につなげることができると考える必要がある．具体的には，定理証明をコード断片の検証のみに利用するのではなく，仕様記述から定理証明に基づいたコード生成を行い，生成されたコードと類似したコード断片をデータベースの中から探索する方法などが考えられる．

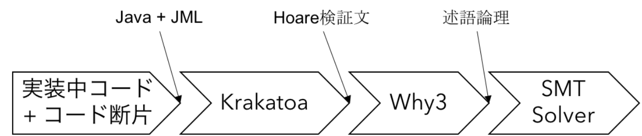

図 8　コード断片検証の概要図

参考文献

[1] Eclipse, 入手先 ⟨http://www.eclipse.org⟩ .
[2] The Java Modeling Language, 入 手 先 ⟨http://www.jmlspecs.org⟩.
[3] Gary T.Leavens, Yoonsik Cheon Design by Contract with JML, September 28, 2006.
[4] F. Bobot, J.-C. Filliâtre, Claude Marché, and Andrei Paskevich. “Why3: Shepherd your herd of provers.” In International Workshop on Intermediate Verification Languages (Boogie), 2011.
[5] D. Mandelin, L. Xu, R. Bodik, and D.Kimelman. “Jungloid mining: helping to navigate the API jungle.” Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. ACM SIGPLAN Notices 40.6 (2005): pp. 48-61.
[6] N. Sahavechaphan and K. Claypool.“XSnippet: mining for sample code.” Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. pp. 413-430, 2006.
[7] S. Thummalapenta and T. Xie. “PARSEWeb: a programmer assistant for reusing open source code on the web.” Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. ACM, 2007. pp. 204-213.
[8] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. Al-Kofahi, and T. N. Nguyen. “Graph-based pattern-oriented, context-sensitive source code completion.” Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012. pp. 69-79.
[9] A. T Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. “GraPacc: A Graph-based Pattern-oriented, Context-sensitive Code Completion Tool.” Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012. pp. 1407-1410.
[10] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, “Graph-based mining of multiple object usage patterns.” Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, (New York, NY, USA), pp. 383– 392, ACM, 2009.
[11] B. Meyer, “Applying Design by Contract.” IEEE Computer, vol. 25, no. 10, pp. 40-51, Oct. 1992.
[12] The Yices SMT Solver, 入手先 ⟨http://yices.csl.sri.com⟩ .
[13] The Alt-Ergo SMT Solver, 入手先 ⟨http://ergo.lri.fr⟩.
[14] M. D. Ernst, J. H. Perkins, P. J. Guo, S. Mccamant, C. Pacheco, M. S. Tschantz, and C. Xiao, “The daikon system for dynamic detection of likely invariants.” Science of Computer Programming, Vol. 69, No. 1-3, pp. 35–45, 2007.
[15] Krakatoa and Jessie: verification tools for Java and C programs, 入手先 ⟨ http://krakatoa.lri.fr⟩.
[16] C. A. R. Hoare, “An axiomatic basis for computer programming.” Communications of the ACM, 12(10):576–580 and 583, 1969.
[17] JDT – Java development tools,入手先 ⟨ https://projects.eclipse.org/projects/eclipse.jdt ⟩.

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.