



設計意図発掘ツール



AtScope

操作マニュアル

# 目次

---




第1章 はじめに.....	4
1 動作環境.....	5
2 免責事項.....	5
3 制限事項／留意事項.....	6
解析対象言語.....	6
マクロの扱い.....	6
文字コード.....	7
不具合の特定.....	8
解析規模.....	8
表記法.....	8
アーキテクチャのビュー.....	8
第2章 AtScope が目指すもの.....	9
1 AtScope の機能.....	10
アーキテクチャ解析：ソフトウェアの全体像を図解する.....	10
ファイル解析：設計改善の箇所を見える化する.....	11
モジュール解析：グローバルデータをカプセル化する.....	12
劣化検知：構造の劣化を図解する.....	13
劣化検知：構造スコアの低下を検知する.....	13
2 AtScope の活用方法.....	18
開発段階ごとの活用.....	18
AtScope の活用ワークフロー.....	20
第3章 AtScope でソフトウェアを解析する.....	21
1 AtScope を起動して解析対象を選択する.....	22
AtScope を起動する.....	22
解析対象を選択する.....	24
2 構造解析し、スコアの計測を実行する.....	25
アーキテクチャ解析を実行する.....	25
ファイル解析を実行する.....	31
モジュール解析を実行する.....	35
リファクタリングスコアを計測する.....	38
3 AtScope を終了する.....	40

第4章 AtScope の設定、その他.....	41
1 AtScope の各種設定.....	42
解析設定 .....	42
表示設定 .....	44
テスト支援.....	46
2 ソースコード診断サービスについて .....	48
3 お問い合わせ先 .....	48

## このマニュアルの表記について

### 本文中の記号について

本文中に記載されている記号には、次のような意味があります。

記号	意味
 <b>注意</b>	お使いになるときに注意していただきたいことや、してはいけないことを記述しています。
 <b>POINT</b>	操作に関連することを記述しております。必要に応じてお読みください。
 <b>参照</b>	参照先を記述しています。

### 画面例について

表記されている画面は一例です。お使いの状況により、画面が異なることがあります。

# 第1章 はじめに

設計意図発掘ツール AtScope のご使用の前にご確認いただきたい事項として、  
AtScope の動作環境と免責事項および制限事項／留意事項を説明します。

# 1 動作環境

---

AtScope は、EnterpriseArchitect 上で動作するアドインプログラムです。Enterprise Architect Version16 で動作することを確認済みです。



Enterprise Architect について詳しくは、「スパークスシステムズ ジャパン株式会社」のホームページをご覧ください。

<http://www.sparxsystems.jp/>

# 2 免責事項

---

AtScope はソースコードを厳密に解析するツールではありません。

以下の項目をご了承していただき、設計構造を構築し保守するツールとしてお使いください。

1. ソースコードの厳密な解析は保証いたしません
2. マクロは解析対象外です
3. AtScope の誤解析により被害が生じたとしても、その保証はいたしかねます

誤解析やマクロに関しては、お客様サイトで解析対象のソースコードを修正していただき、AtScope を快適に使いこなしてください。

# 3 制限事項／留意事項

## 解析対象言語

C 言語です。

マルチ言語版は、C++／Java／C#に対応しています。

## マクロの扱い

マクロは展開しません。

AtScope は、ユーザーがソースコードを目視で読む行為と同等の解析をします。

- **#define(置換)で定義した定数や関数は、解析対象になりません。**


```
#define MAX 10           ←無視します
#define print(dt) printf("%d\n", dt) ←無視します
void main(void)
{
    print(MAX);          ←解析対象になりません
}
```

- #if(条件コンパイル)が原因で、関数終端検出に誤りが発生した場合、誤りの発生以降のコードが解析できないことがあります。

```
void main(void)
{
    if( flg )
    {
        #ifdef AAA                ←無視します
            data1 = 1;
        }
        #else                      ←無視します
            data1 = 2;
        }                          ←誤りの発生によりコードを解析できません
    #endif                        ←無視します
}
```

マクロを展開したソースコードを解析する場合は、コンパイラなどでマクロを展開した後、AtScope を実行してください。GCC(GNU Compiler Collection)の-E オプションでマクロを展開する例を示します。

- before.c のマクロを展開し、そのソースコードを標準出力する場合  
% gcc -E before.c
- before.c のマクロを展開し、そのソースコードを after.c へ出力する場合  
% gcc -E before.c -o after.c

 **POINT** コンパイラの使い方について詳しくは、発行元にお問い合わせください。

## 文字コード

Shift JIS に対応しています。

解析対象のソースコードが Shift JIS 以外の場合は、文字コードを Shift JIS に変換後、AtScope を実行してください。



## 不具合の特定

AtScope は、設計構造の把握などを行うツールであり、不具合を直接検出するツールではありません。

- 設計構造の把握
- リファクタリングスコアによる評価（効果計測）
- 危険な変数の特定

## 解析規模

解析可能なコードの規模は、動作 PC のメモリ容量に依存します。メモリが不足した時点で実行時エラーとなります。

## 表記法

ブロックとインターフェースに関しては、UML2 のコンポジット構造図、および SysML の内部ブロック図を使用して表記しています。

関数やファイルに関しては、UML や SysML では規定されていない表記を使用している場合があります。

## アーキテクチャのビュー

アーキテクチャ設計の考え方は、弊社のアーキテクト育成コースに準じています。詳しくは弊社ホームページをご覧ください。

<http://www.bslash.co.jp/training/>

# 第2章

## AtScope が目指すもの

ソフトウェア開発は劣化との戦いです。

アーキテクトの段階以降、開発を進めるごとに何かしらの不具合が作りこまれる可能性を含んでいます。順調に開発が進んでいても、仕様の変更やバグフィックスなど、さまざまな要因によってソフトウェアは劣化します。劣化を防ぐには、アーキテクトの業務や日々の開発作業も含めて、いかに改善に取り組むかが重要です。

AtScope は、ソフトウェアの劣化を検知して評価し、開発における「改善スパイラル」を描くための情報が得られるツールです。開発進捗のマイルストーンで AtScope を実行し、ソースコードを解析・評価することで、全体構造やファイル、関数、変数の関連などの情報を得られます。得られた情報に基づいてソフトウェアを見直すことで、ソフトウェアの劣化を防ぎ、改善することができます。

本章では、ソフトウェアの劣化を防いで改善するための AtScope の機能、および活用方法を説明します。

# 1 AtScope の機能

AtScope は、3つの粒度でそれぞれ構造図を作成するツールです。また、設計構造の劣化を検知することにも活用できます。

- **アーキテクチャ解析** ⇒フォルダ単位の解析
- **ファイル解析** ⇒ファイル単位の解析
- **モジュール解析** ⇒関数と変数単位の解析
- **劣化検知** ⇒配置を保存する機能、及び、構造をスコア化する機能

これら4つの機能を説明します。

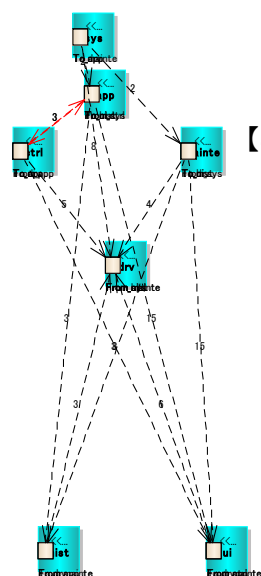
## アーキテクチャ解析:ソフトウェアの全体像を図解する

アーキテクチャは、ソフトウェアの基本構造です。設計コンセプトを反映したアーキテクチャをすることで、破綻のないソースコードが構築されます。しかし、開発が進むに連れ、アーキテクチャに影響を与えるソースコードが発生します。よって、開発初期に設計したアーキテクチャを維持するためには、常にアーキテクチャの構造を把握することが大事です。

アーキテクチャ解析はコンポーネント(フォルダー単位)でソースコードを解析し、コンポーネント構造図を出力します。コンポーネント構造図では、コンポーネントを手動で再配置して整えることができます。

- **コンポーネントを再配置することで、設計意図や課題を確認できます。**

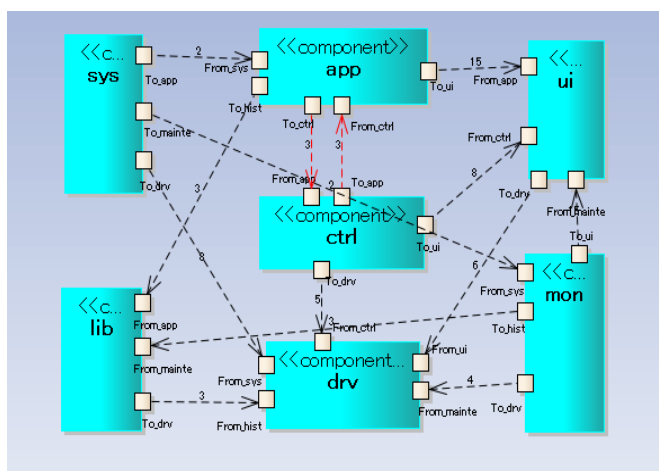
【解析結果】



【再配置】  
(手動)



【再配置結果】



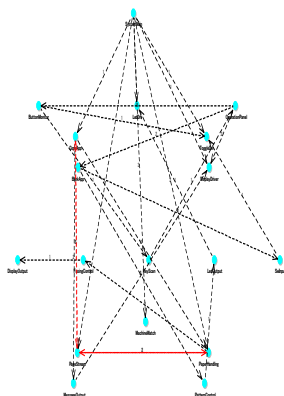
## ファイル解析: 設計改善の箇所を見える化する

ファイル解析によるファイル見取図は、全体のファイル構造を見ることができ、改善点の発見に役立ちます。また、ファイル構造図では、ファイル間の設計意図を確認できます。

ファイル単位でソースコードを解析し、ファイル見取図を出力します。ファイル見取図では、ファイルを手動で再配置して整えることができます。

- 設計ルール違反が強調色で表示されます。
- ファイルを再配置することで、設計意図や課題を確認できます。

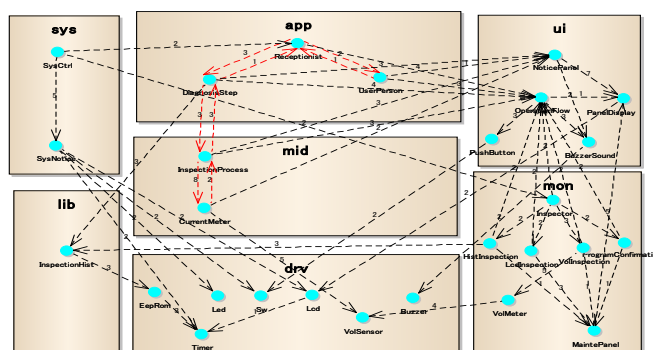
【解析結果】



【再配置】  
(手動)



【再配置結果】

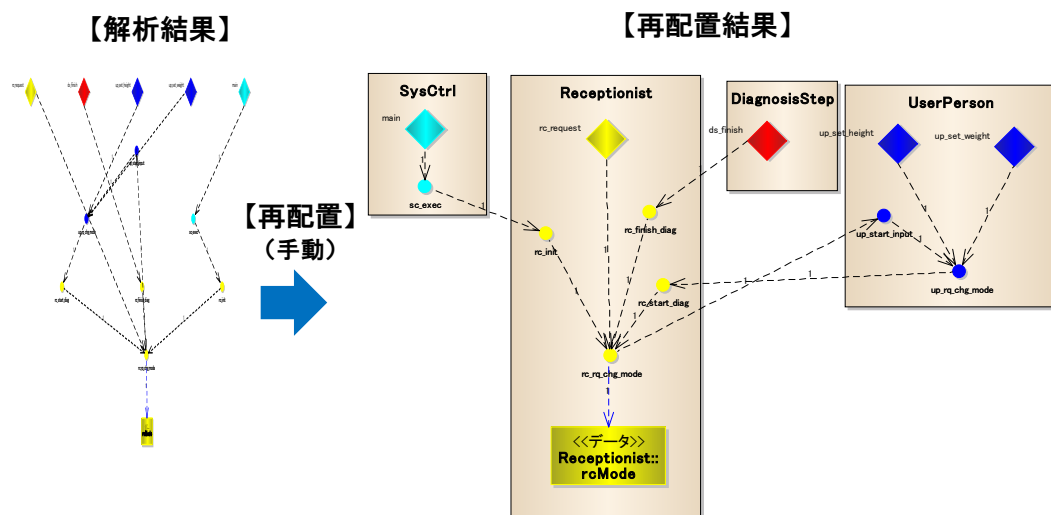


## モジュール解析:グローバルデータをカプセル化する

モジュール解析による関数構造図(変数起点)は、変数を指定することで変数へのアクセスをさかのぼります。これにより、変数のカプセル化状況や異なるスレッドからの衝突を確認し、グローバルデータを検出できます。検出したグローバルデータをカプセル化することで、ソースコードが改善されます。

関数や変数単位でソースコードを解析し、関数構造図を出力します。関数構造図では、関数や変数を手動で再配置して整えることができます。

- 指定したデータから関数コールをさかのぼることができます。
- 変数のカプセル化の状況、および異なるスレッドからの衝突を確認できます。

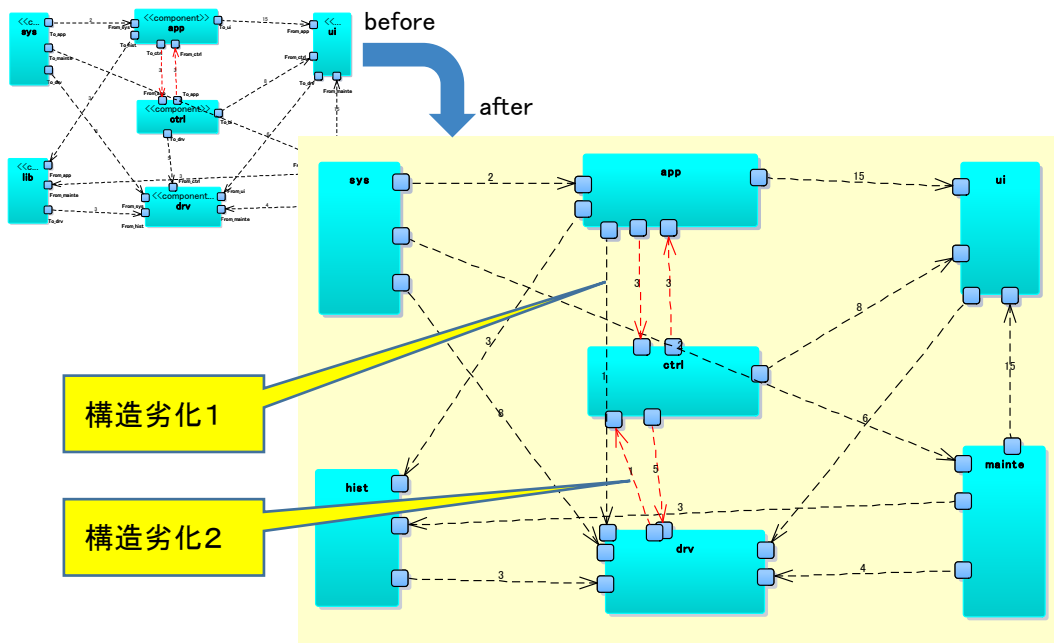


## 劣化検知: 構造の劣化を図解する

アーキテクチャ構造に従って再配置した図面を表示して、その配置を保存して、再度ソースコードを解析することができます。その図面を見ることで、設計ルール違反を検出することができます。

下の例では、アプリ層からドライバ層への飛び越えた依存と、ドライバ層からミドル層への逆方向の依存が増えています。

- 配置を保存することで、設計構造の崩れを検知できます。



## 劣化検知: 構造スコアの低下を検知する

ソフトウェアの開発が進むに連れ、不具合が作り込まれる可能性が大きくなります。しかし、開発が進めば進むほどソースコードの規模は大きくなり、grep 検索や目視では劣化の具合がはっきりつかめません。リファクタリングスコアを計測することで、ソフトウェア品質を点数評価で確認できます。

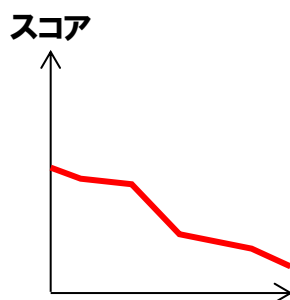
### ■ リファクタリングスコアの特徴

- 設計改善に直結し、ソフトウェア資産価値が高まります。  
「要素点」が低い場合は、リファクタリングが効果的です。  
「構造点」が低い場合は、リバース設計が効果的です。
- 組織設計力を把握できます。  
徐々にスコアが下がる場合、設計力が低い組織です。  
設計図でピアレビューできる組織づくりを目指せます。

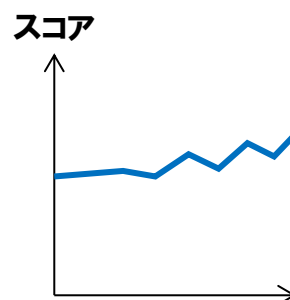
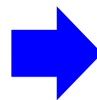
## ■ 構成要素と設計構造をスコア化

AtScope は、ソフトウェアの劣化を評価し、点数表示します。ソースコードを構造的な視点で解析することで点数化を実現しています。点数には「要素点」と「構造点」があります。

- 満点は「100 点」です。劣化原因ごとにマイナス評価される減点方式です。  
「0 点」以下の評価(つまりマイナス点)もあります。
- 場当たりのなコード変更を続けると、点数が下がり続ける可能性があります。
- アーキテクチャ解析、ファイル解析、およびモジュール解析の結果に基づいて「リファクタリング」や「変更時の構造設計」を実施することで、点数を維持／向上します。これが「改善スパイラル」です。
- 開発のマイルストーンごとにリファクタリングスコアを実施し、点数が減少している場合はソフトウェア品質が「劣化スパイラル」を描いていることとなります。各種解析結果に基づいて点数を増加させるような「改善スパイラル」を描かなければなりません。



【場当たりのなコード変更】  
劣化原因の作りこみにより減点される



【リファクタリングや変更時の構造設計の実施】  
改善スパイラルによって点数が増加

## ■ 要素点と構造点の算定項目

### ● 要素点の算定項目

項目	正常値
ファイル長さ	1000 ステップ未満
ファイル内の関数の数	20 未満
関数の長さ	30 ステップ未満
フィールド変数の数	7 個いない

**POINT** フィールド変数の数は、解析設定で「変数アクセスを含める」を選択したときのみ算出します。

### ● 構造点の算定項目

項目	正常値
相互依存	0
ファンアウト数	10 個未満(関数コール)
データスコープ	0(ファイル外アクセス)

**POINT** データスコープは、解析設定で「変数アクセスを含める」を選択したときのみ算出します。

## ■ 要素点と構造点による設計力の強化施策と戦略的な設計改善

### ● 設計力の強化施策

要素点と構造点のスコアによって、下表に示した設計力の強化施策を検討します。

要素点 \ 構造点	高い	低い
	高い	ピアレビュー (さらなる設計力向上)
低い	リファクタリング	設計技法の見直し 構造化設計の学習



- **戦略的な設計改善**

リファクタリングスコアの計測で出力される「要素の警告リスト」や「構造の警告リスト」を基に、トップ 5 リストから実施する項目を決めて、設計改善します。また、改善しなくても良い項目を決定します。

## ■ リファクタリングスコアの計測結果の説明

リファクタリングスコアの計測結果の各項目について説明します。

- **スコア**

「総合」は、「要素」点と「構造」点の平均値です。満点は 100 点です。劣化や不具合ごとに減点され、0 点以下（つまりマイナス点）もあります。

「要素」は、ソースコード内の要素に関する計測結果です。満点は 100 点です。劣化や不具合ごとに減点され、0 点以下（つまりマイナス点）もあります。

「構造」は、ソースコード内の構造に関する計測結果です。満点は 100 点です。劣化や不具合ごとに減点され、0 点以下（つまりマイナス点）もあります。

- **要素の警告リスト**

要素点の減点対象となった不具合や劣化原因が表示されます。

- **構造の警告リスト**

構造点の減点対象となった不具合や劣化原因が表示されます。

- **日付**

リファクタリングスコアの計測を実施した日時、およびリファクタリングスコアを計測した対象のソースコードが保存されているフォルダーのパスが「計測対象」に表示されます。

## ■リファクタリングスコアの目安

一人で担当する範囲で計測した場合のリファクタリングスコアの目安を示します。

**POINT** 絶対値は参考としてご使用ください。  
相対的な使い方を推奨します。

スコア	評価	コメント
80点～100点	優	他の人でも変更可能なソフトウェア
60点～79点	良	引き継ぎ説明が少して済むソフトウェア
0点～59点	可	引き継ぎ期間が必要なソフトウェア
マイナス点	不可	熟知していないと修正することが大変なソフトウェア

局所的な修正を繰り返していると、スコアは徐々に低下していきます。

マネージャは、スコアの推移を管理して、スコアを徐々に上げていくための体制や仕組みを作ることで、ソースコード劣化を未然防止することができます。

また、受発注における納品条件(検収条件)の目安として使うこともできます。

## 2 AtScope の活用方法

AtScope による解析や劣化検知を適切なタイミングで実行することで、ソフトウェアは改善されます。本章では、開発進捗の段階ごとに AtScope を活用する方法を説明します。

### 開発段階ごとの活用

#### ■ 開発初期

アーキテクチャ構造を作成し、アーキテクチャ構造を順守しながら設計／実装する段階です。AtScope で「アーキテクチャ解析」をしながら「リファクタリングスコア」で評価することで、試行錯誤や手戻りが減少し、革新的なソフトウェア開発ができます。

#### ■ 引き継ぎ時

開発担当を引き継ぐ際には、実装レベルで理解するのではなく、設計構造を理解する、つまり「見える化」が必要です。

grep 検索に必要な情報をひとつひとつ確認するのではなく、AtScope で「ファイル解析」や「モジュール解析」を実施し、ファイル単位／関数・変数単位でのソースコードの構造を明示することで、ソースコードを理解するためのリードタイムが大幅に減少します。

#### ■ デイリービルド

日々の開発作業による設計構造の劣化をコードコミット時にチェックします。

AtScope の「アーキテクチャ解析」と「リファクタリングスコア」を実施し、得られた結果と評価を確認・検討することで、開発者が主導するモチベーションの高い開発組織作りができます。また、ソフトウェア開発をアジャイル開発化することで、高品質なソースコードを早く作成できます。

#### ■ レビュー時

コードレビューやドキュメントレビューではなく、AtScope で出力した構造図によるレビューの実施が推奨されます。

AtScope の構造図は抽象度が高く、厳密なため、レビューの効率と効果が向上します。これにより、上流工程での品質の作りこみを可能にします。

## ■ 定期検証(毎週／毎月)

定期的な検証は、開発プロセスの改善に役立ちます。

デイリービルドと同じく、AtScope の「アーキテクチャ解析」と「リファクタリングスコア」を実施し、設計ルール違反を抽出して設計スコアを計測することで、担当者の指導とプロセスの改善ができます。このような定期検証を繰り返し実施することで、ソースコードの資産価値を維持するマネジメントができます。また、検証結果によって設計違反を発見し、担当者を指導することで、組織力の向上が期待できます。

## ■ 振り返り

設計を改善するため、開発の振り返りは重要です。ソースコードそのものを改善するのではなく、設計構造を改善し、アーキテクチャ構造を洗練化することができます。

AtScope で「ファイル解析」や「モジュール解析」を実施し、ファイル単位／関数・変数単位でのソースコードの構造を明示することで、開発進捗によって変更された箇所が明確化され、影響範囲を限定できます。これにより、機能追加や不具合対応の生産性が向上します。

## ■ 納品／検収

納品／検収時に AtScope を活用することで、設計品質が維持できていることを確認できます。

納品／検収時の AtScope の活用は、実は開発初期の AtScope の活用と同じです。受け入れテストでの水際作戦で対応するのではなく、開発初期に AtScope で設計したアーキテクチャが納品／検収まで保たれていることが重要です。

リファレンスモデルを提示することで、ソースコードの乱れを事前に検出し、リファクタリングスコアで評価を実施します。「開発初期」で説明したことを実施することで、契約時に設計品質を取り決めたこととなります。納品／検収時は、その設計品質を維持したかどうか重要です。以下の手順で設計品質が維持されていることを確認します。

- 1.「アーキテクチャ解析」で全体構造を確認します。
- 2.「リファクタリングスコア」で劣化検知を実施します。
- 3.開発初期の品質が保たれていることを確認します。

## AtScope の活用ワークフロー

「開発段階ごとの活用」で説明した内容をワークフローで示します。

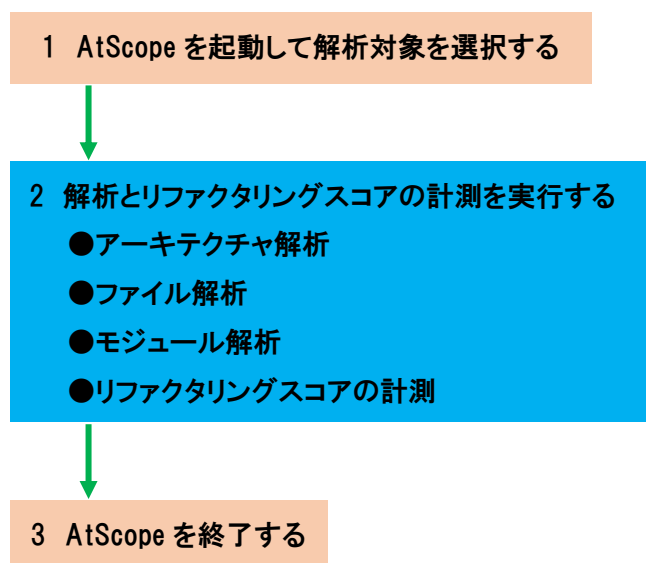
活用時期	初回	2回目以降	狙い
開発初期 納品・検収時  <b>アーキテクト</b>	①コンポーネント解析 → 再配置 → ④リファクタリングスコア	配置を保存 → ①コンポーネント解析 → ④リファクタリングスコア	全体を俯瞰し、構造の変化を検知  構造劣化をスコアで確認
引継ぎ時 レビュー時 ふりかえり時  <b>エンジニア</b>	②ファイル解析 → 再配置 → ③関数解析	配置を保存 → ②ファイル解析 → ③関数解析	構造的な課題を検出し、すぐに改善する  グローバル変数を検出しカプセル化する
デイリービルド 定期的に (毎週、毎月)  <b>マネージャ 品質管理者</b>	①コンポーネント解析 → 再配置 → ④リファクタリングスコア	配置を保存 → ①コンポーネント解析 → ④リファクタリングスコア	構造の変化を検出し、担当者で改善する  要素点が低ければリファクタリングし、構造点が低ければ構造をレビューする

## 第3章

# AtScope でソフトウェアを解析する

AtScope でソフトウェアを解析する操作について説明します。

AtScope の操作の流れは以下のとおりです。



# 1 AtScope を起動して解析対象を選択する

AtScope を起動し、解析対象を選択する方法を説明します。

## AtScope を起動する

- 1 「Enterprise Architect」アイコンをダブルクリックします。



Enterprise Architect が起動します。



Enterprise Architect、「スパークスジャパン株式会社」製のモデリングツールです。

- 2 Enterprise Architect のプロジェクトを開きます。

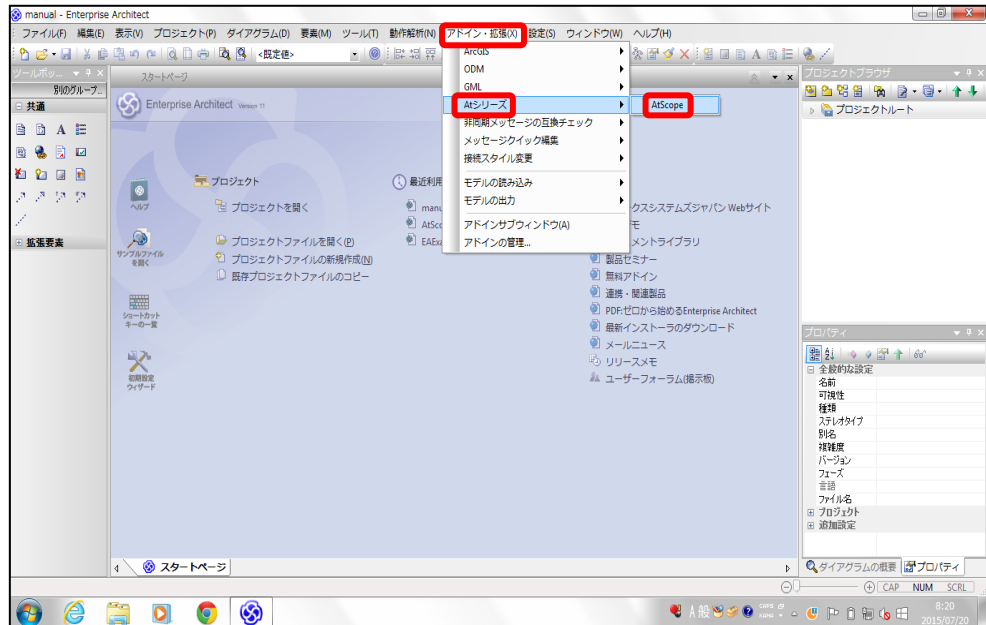
**注意** プロジェクトを開かないで次の手順を実行すると、AtScope が起動しません。解析対象のソースコードを含むプロジェクト、または任意のプロジェクトファイルを開いてください。AtScope で解析するソースコードは、AtScope 起動後に選択できます。



Enterprise Architect ドキュメントライブラリをご覧ください。

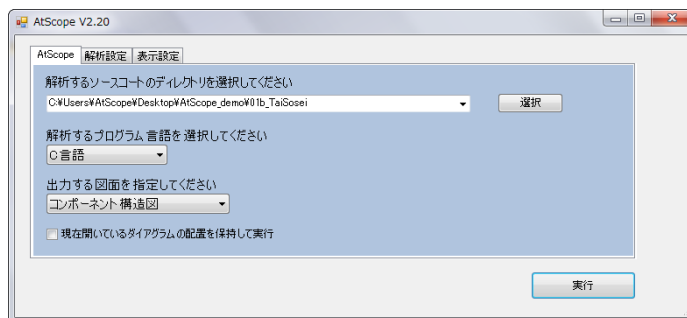
[http://www.sparxsystems.jp/products/EA/ea\\_documents.htm](http://www.sparxsystems.jp/products/EA/ea_documents.htm)

### 3 「アドイン・拡張」→「At シリーズ」→「AtScope」を選択します。



「AtScope Vx.xx」画面の「AtScope」タブが表示されます。

**POINT** 本書では、「AtScope V2.20」の画面を使用して説明しています。

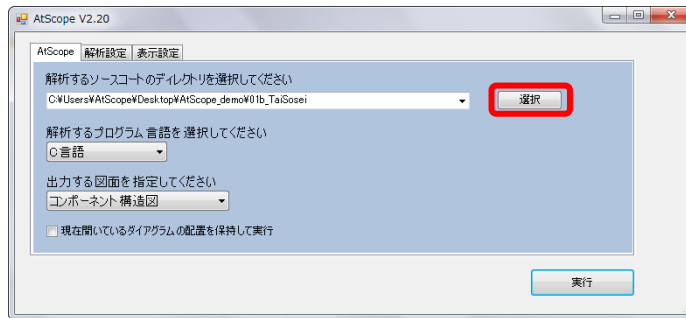




## 解析対象を選択する

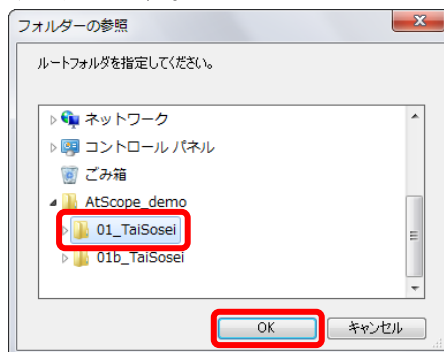
解析対象のソースコードを選択する方法を説明します。

- 1 「AtScope Vx.xx」画面の「AtScope」タブで、「解析するソースコードのディレクトリを選択してください」の「選択」ボタンをクリックします。

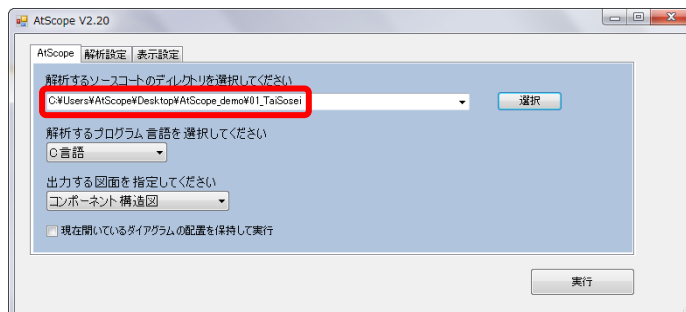


「フォルダーの参照」画面が表示されます。

- 2 解析するソースコードが保存されているフォルダーを選択し、「OK」をクリックします。



選択したフォルダーのパスが「解析するソースコードのディレクトリを選択してください」のドロップダウンリストに表示されます。



## 2 構造解析し、スコアの計測を実行する

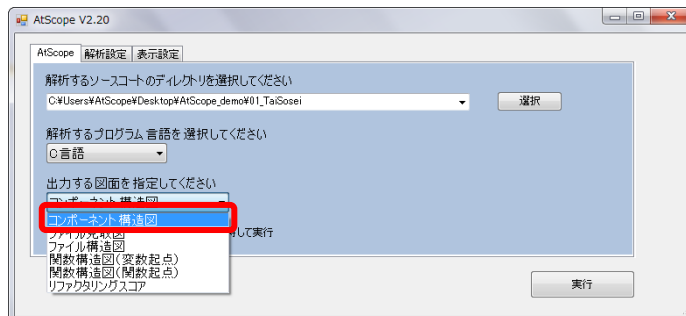
アーキテクチャ解析、ファイル解析、モジュール解析、およびリファクタリングスコアの計測の実行方法を説明します。

### アーキテクチャ解析を実行する

AtScope でアーキテクチャを解析し、コンポーネント構造図を出力する方法を説明します。

**POINT** 次の手順で記載している各機能の操作については、AtScope を起動し、「AtScope Vx.xx」画面で解析対象のソースコードが選択されていることを前提に説明しています。

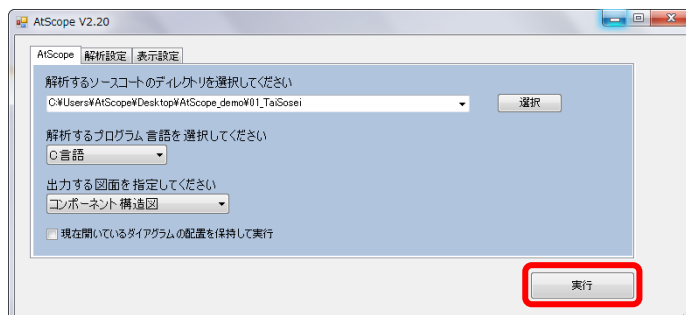
- 1 「出力する図面を指定してください」ドロップダウンリストで「コンポーネント構造図」を選択します。



---

## 2 「実行」をクリックします。

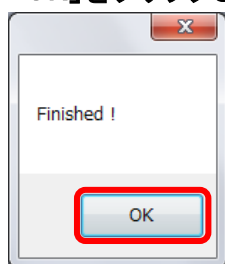
**POINT** 「現在開いているダイアグラムの配置を保持して実行」チェックボックスにチェックを入れると、再配置したコンポーネントを保持してアーキテクチャ解析を実行します。



アーキテクチャ解析の進捗状況が表示されます。アーキテクチャの解析が完了すると、メッセージが表示されます。

---

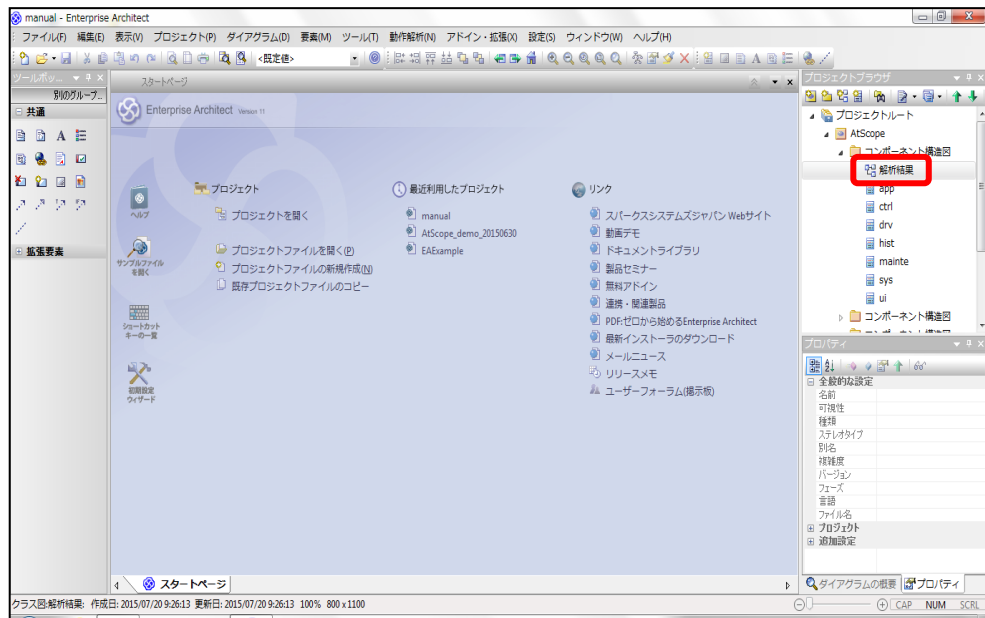
## 3 「OK」をクリックします。



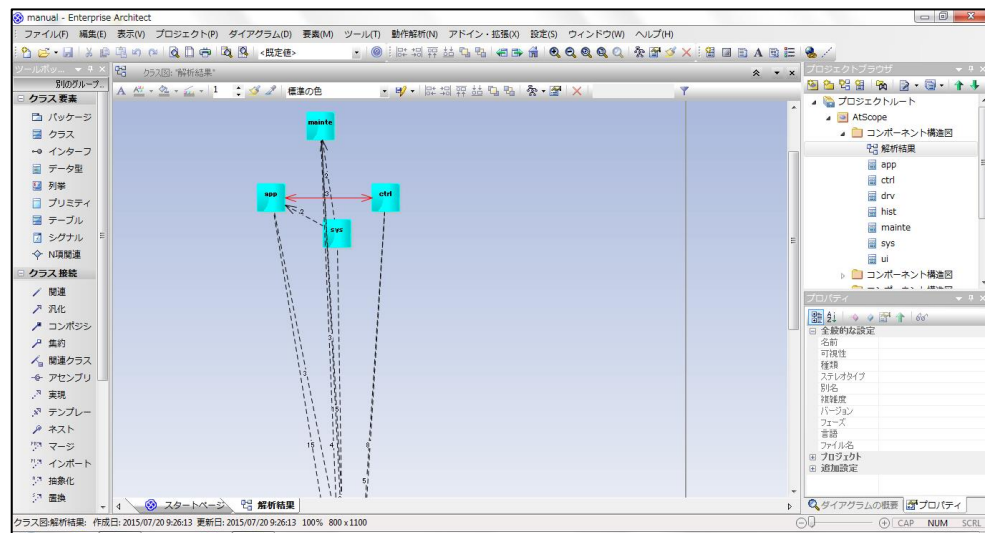
メッセージ画面が閉じます。

---

4 「プロジェクトブラウザ」に表示されている「プロジェクトルート」→「AtScope」→「コンポーネント構造図」直下の「解析結果」をダブルクリックします。



アーキテクチャの解析結果が表示されます。



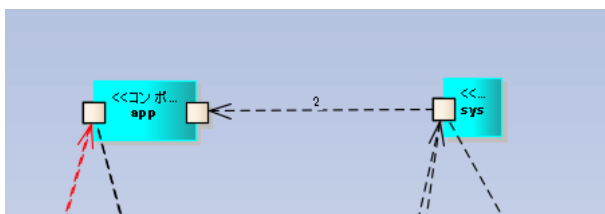
## ■コンポーネントの色分け

コンポーネント内のファイル数に応じて、コンポーネントの色が変わります。

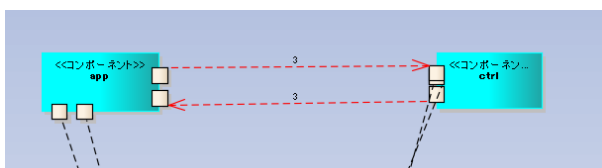
コンポーネントの色	複雑度の判定	フォルダー内のファイル数(個)
青	正常	40 未満
緑	注意	100 未満
オレンジ	警戒	160 未満
赤	危険	160 以上

## ■依存線の色分け

- 黒い破線は、「単方向依存」を表しています。

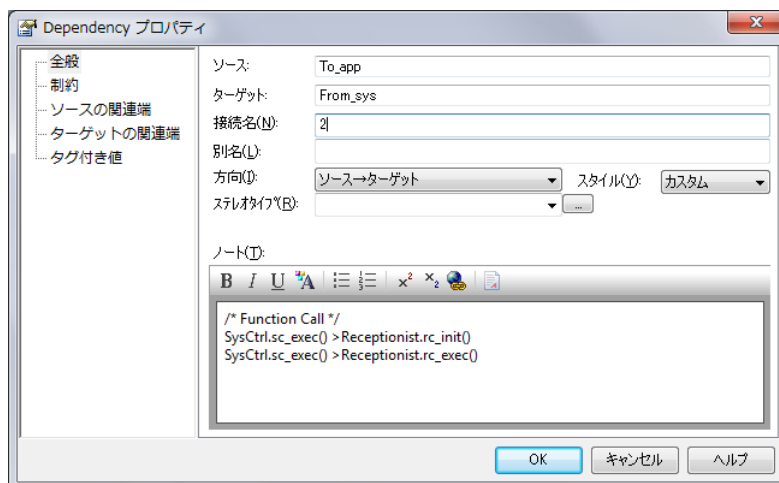


- 赤い破線は、「双方向依存」を表しています。



**注意** 出力直後は線が重なっていますので、EA 上で線を移動してください。

- 依存線をダブルクリック(右クリックでプロパティも同じ)し、依存プロパティで依存の詳細を表示します。





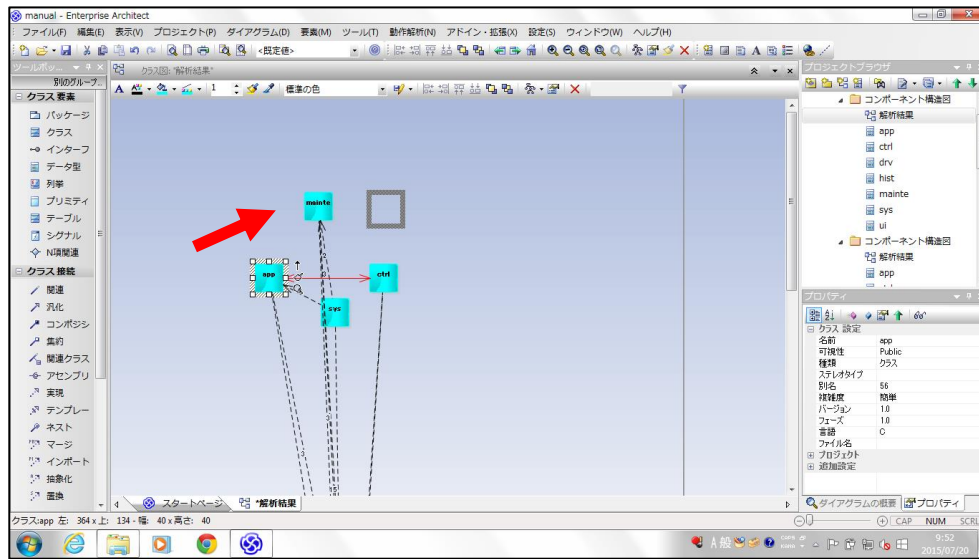
依存プロパティ上のノート(T)に、以下のように表示されます。

`/* Function Call */`

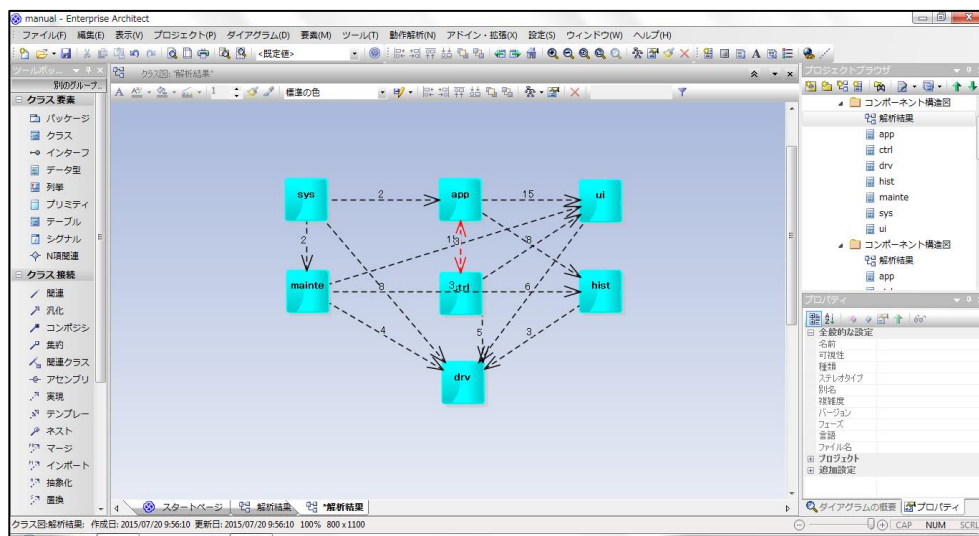
「呼び出し元ファイル名.関数名 > 呼び出し先ファイル名.関数名」

## ■ 手動によるコンポーネントの再配置

解析結果に表示されているボックスがコンポーネントです。クリックしてドラッグすることで、手動で位置を変更できます。

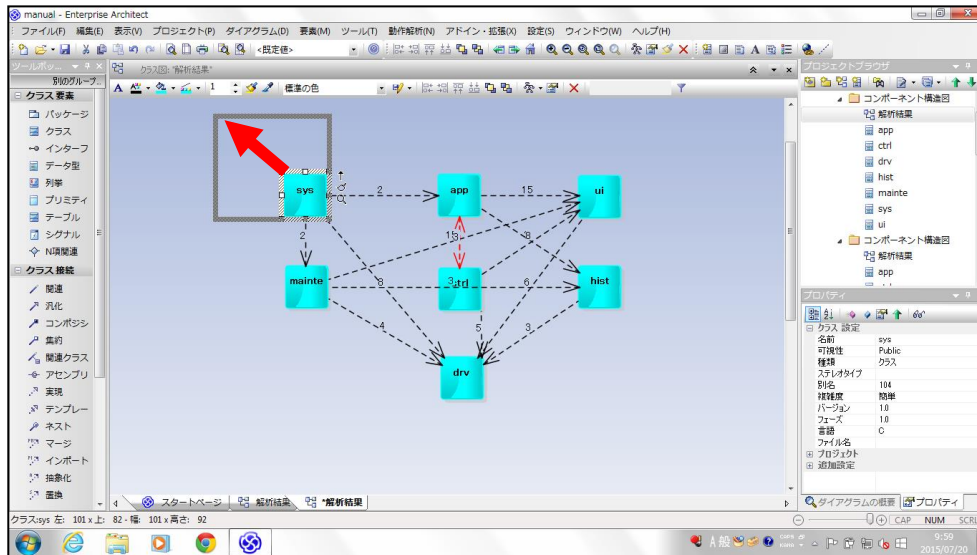


再配置例を示します。

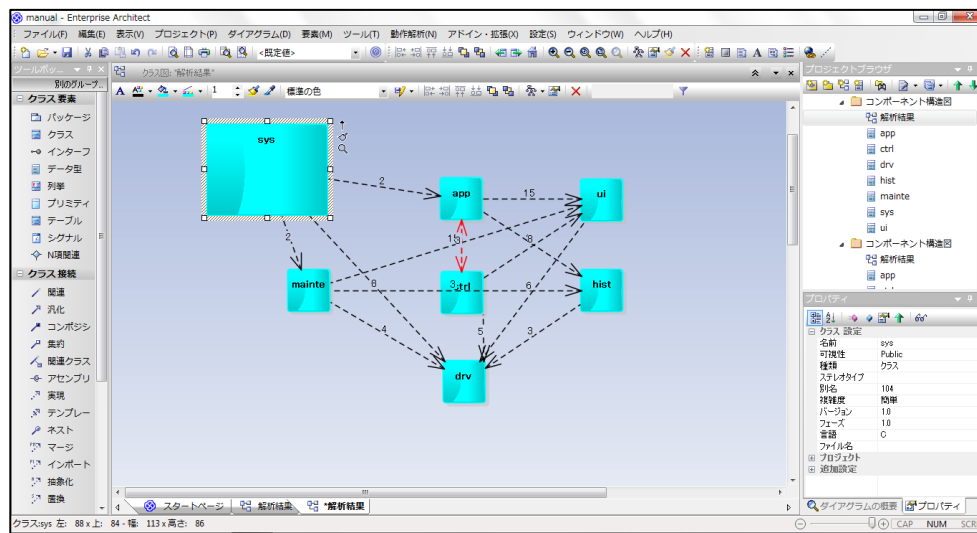


## ■コンポーネントの大きさの変更

クリックして表示されるアンカーポイントをドラッグすることで、コンポーネントの大きさを変更できます。



大きさを変更した例を示します。

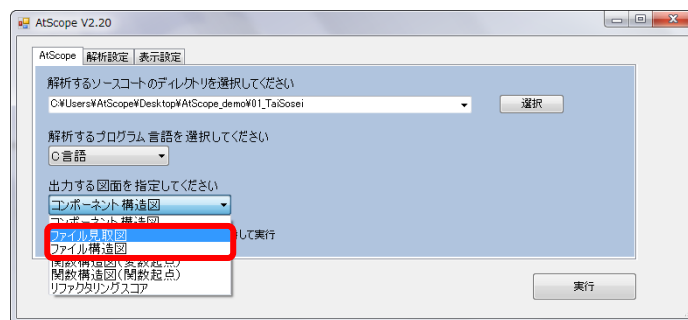


## ファイル解析を実行する

AtScope でファイル解析を実行し、ファイル見取図とファイル構造図を出力する方法を説明します。

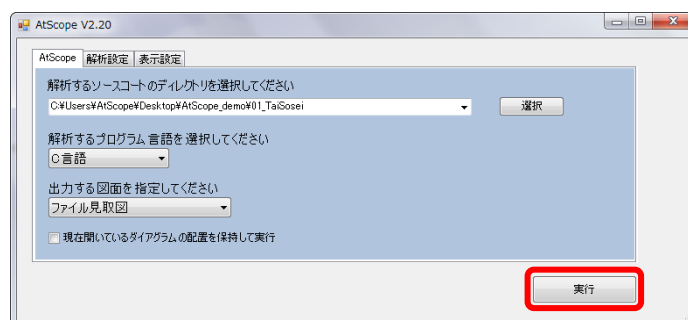
**POINT** 次の手順で記載している各機能の操作については、AtScope を起動し、「AtScope Vx.xx」画面で解析対象のソースコードが選択されていることを前提に説明しています。

- 1 「出力する図面を指定してください」ドロップダウンリストで「ファイル見取図」または「ファイル構造図」を選択します。



- 2 「実行」をクリックします。

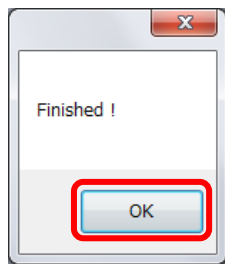
**POINT** 「現在開いているダイアグラムの配置を保持して実行」チェックボックスにチェックを入れると、再配置したファイルを保持してファイル解析を実行します。



ファイル解析の進捗状況が表示されます。ファイル解析が完了すると、メッセージが表示されます。

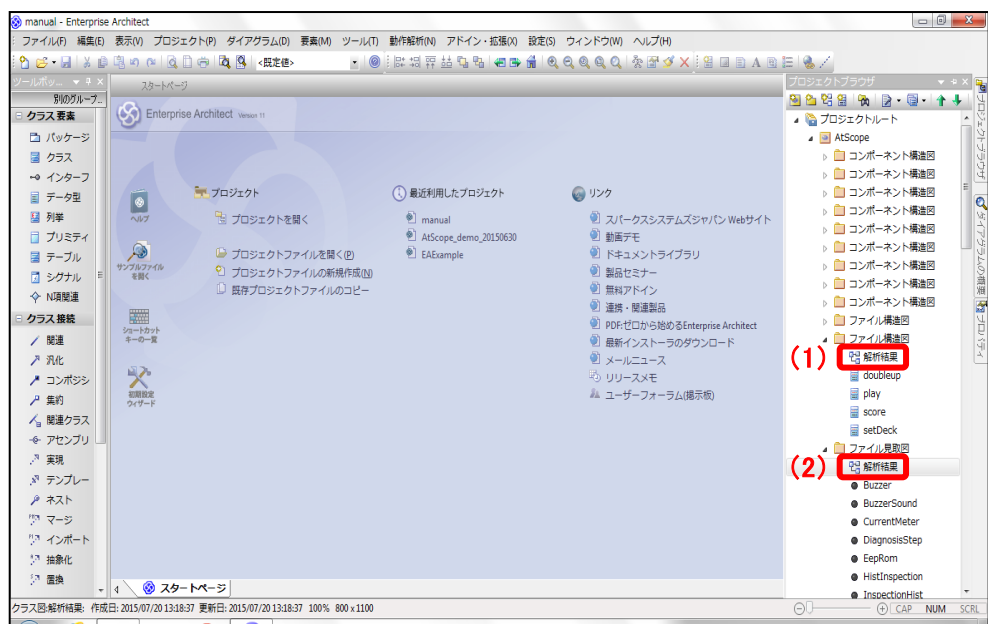


### 3 「OK」をクリックします。



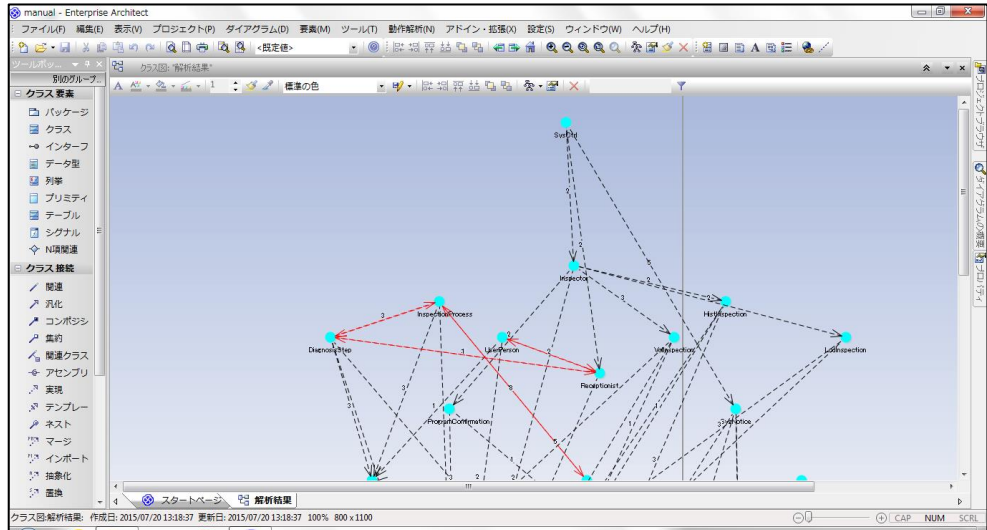
メッセージ画面が閉じます。

- ### 4
- (1) ファイル構造図を表示する場合、「プロジェクトブラウザ」に表示されている「プロジェクトルート」→「AtScope」→「ファイル構造図」直下の「解析結果」をダブルクリックします。
  - (2) ファイル見取図を表示する場合、「プロジェクトブラウザ」に表示されている「プロジェクトルート」→「AtScope」→「ファイル見取図」直下の「解析結果」をダブルクリックします。

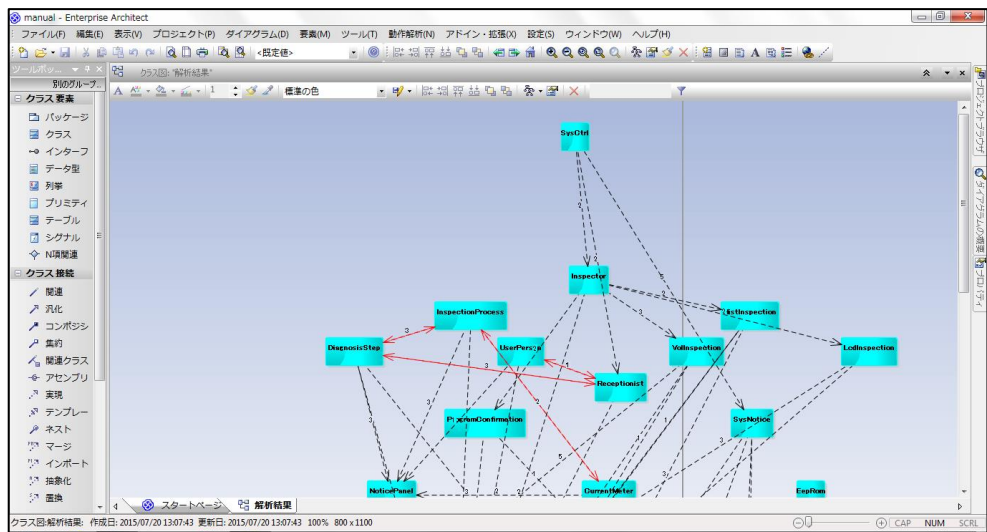


ファイル解析の結果が表示されます。

### 【ファイル見取図】



### 【ファイル構造図】



## ■ファイルの色分け

ファイル内の行数と関数数に応じて、ファイルの色が変わります。

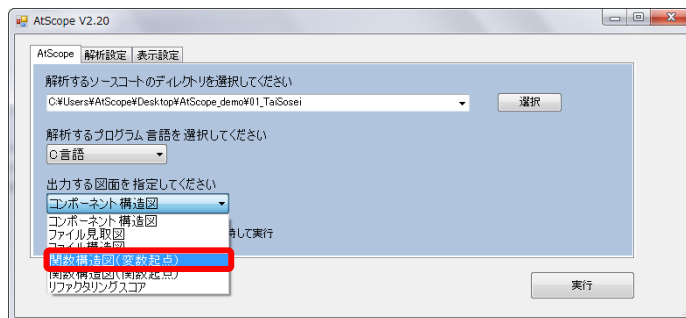
ファイルの色	複雑度の判定	1ファイル内のコード行数(行)	1ファイル内の関数数(個)
青	正常	1,000 未満	20 未満
緑	注意	3,000 未満	60 未満
オレンジ	警戒	5,000 未満	100 未満
赤	危険	5,000 以上	100 以上

ファイルとは、実装ファイル(.c)のことを指しています。ヘッダファイル(.h)と実装ファイル(.c)がペア(同じ名前)の場合は、そのペアでファイルとみなします。

## モジュール解析を実行する

AtScope でモジュール解析を実行し、関数構造図(変数起点)を出力する方法を説明します。

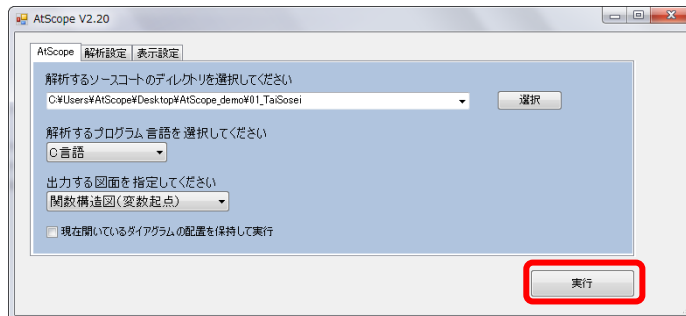
- 1 「出力する図面を指定してください」ドロップダウンリストで「関数構造図(変数起点)」を選択します。



- 2 「実行」をクリックします。



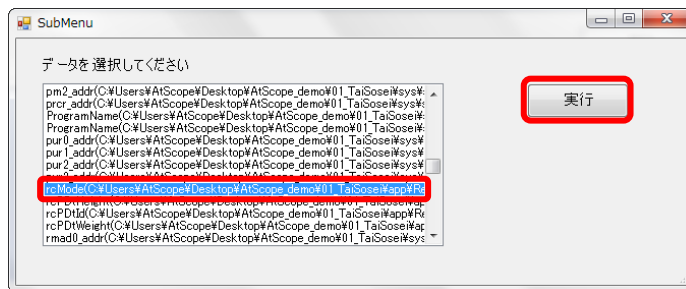
「現在開いているダイアグラムの配置を保持して実行」チェックボックスにチェックを入れると、再配置したファイルを保持してファイル解析を実行します。



「SubMenu」画面が表示されます。

---

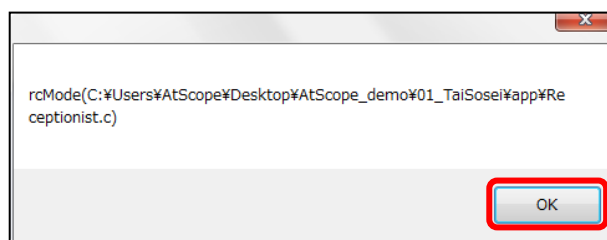
### 3 モジュール解析を実行する変数を選択し、「実行」をクリックします。



選択した変数の確認メッセージが表示されます。

---

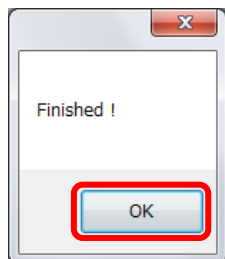
### 4 「OK」をクリックします。



モジュール解析の進捗状況が表示されます。モジュール解析が完了すると、メッセージが表示されます。

---

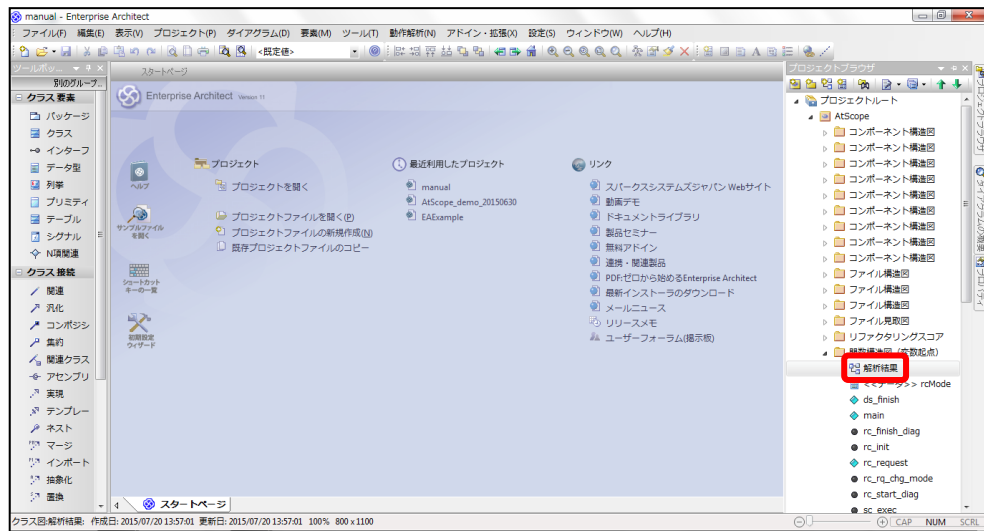
### 5 「OK」をクリックします。



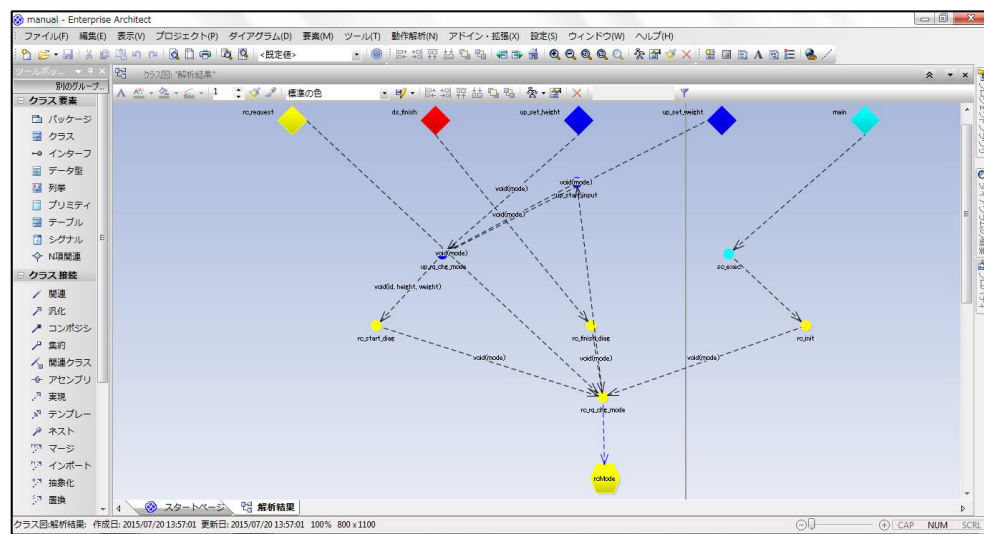
メッセージ画面が閉じます。

---

6 「プロジェクトブラウザ」に表示されている「プロジェクトルート」→「AtScope」→「関数構造図(変数起点)」直下の「解析結果」をダブルクリックします。



モジュール解析の結果が表示されます。

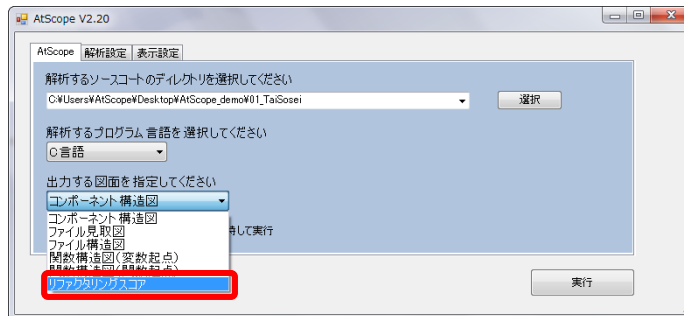


## リファクタリングスコアを計測する

リファクタリングスコアの計測方法を説明します。

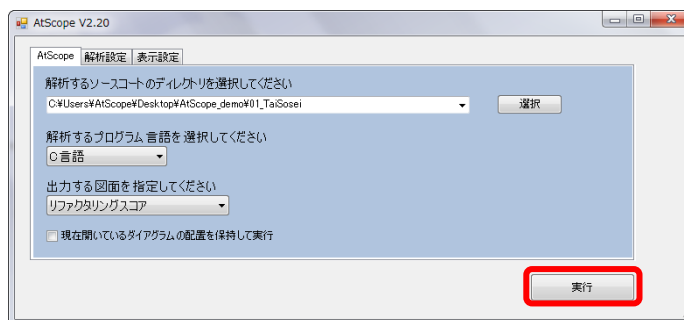
- POINT** 次の手順で記載している各機能の操作については、AtScope を起動し、「AtScope Vx.xx」画面で解析対象のソースコードが選択されていることを前提に説明しています。

### 1 「出力する図面を指定してください」ドロップダウンリストで「リファクタリングスコア」を選択します。



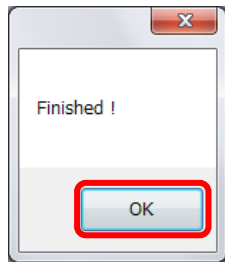
### 2 「実行」をクリックします。

- POINT** 「現在開いているダイアグラムの配置を保持して実行」チェックボックスにチェックを入れると、再配置したコンポーネントを保持してリファクタリングスコアを計測します。



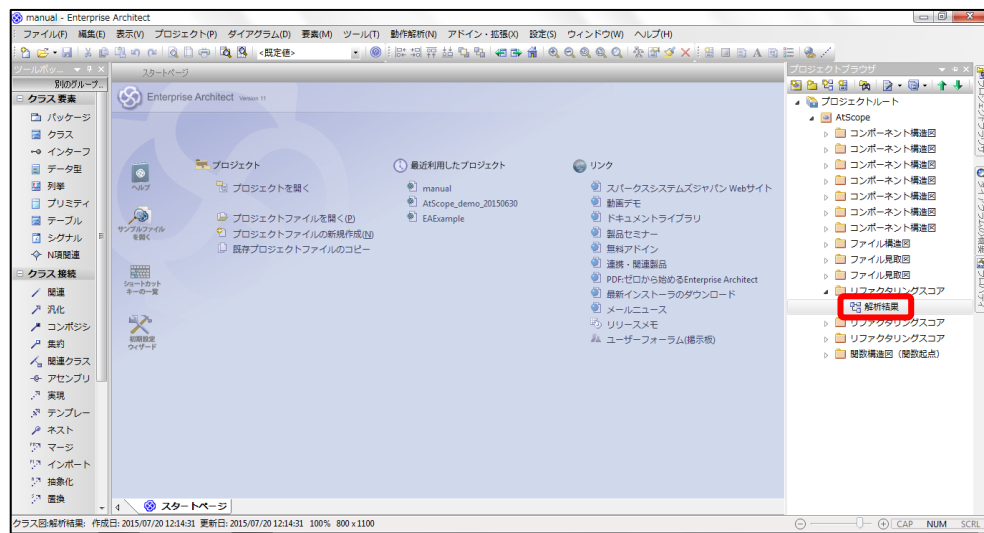
リファクタリングスコアの計測の進捗状況が表示されます。リファクタリングスコアの計測が完了すると、メッセージが表示されます。

### 3 「OK」をクリックします。

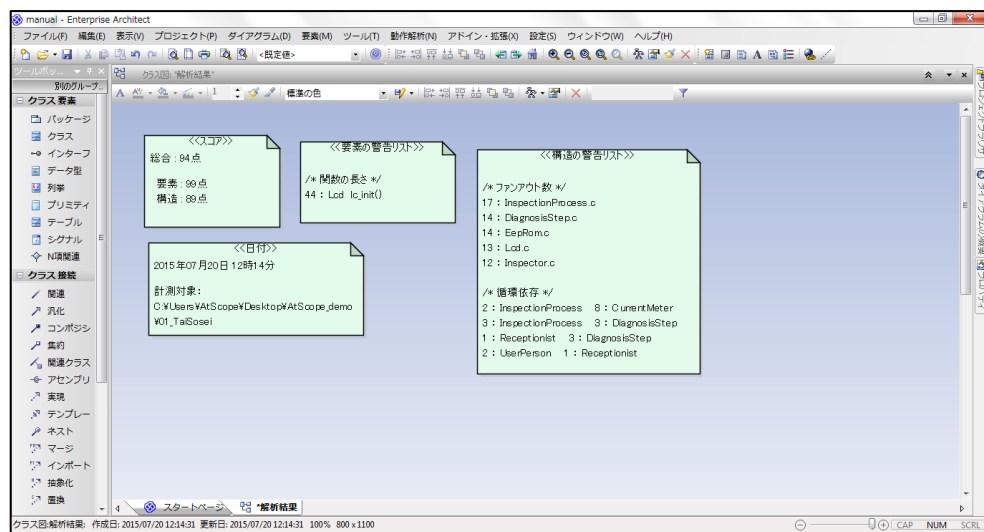


メッセージ画面が閉じます。

### 4 「プロジェクトブラウザ」に表示されている「プロジェクトルート」→「AtScope」→「リファクタリングスコア」直下の「解析結果」をダブルクリックします。



リファクタリングスコアの計測結果が表示されます。





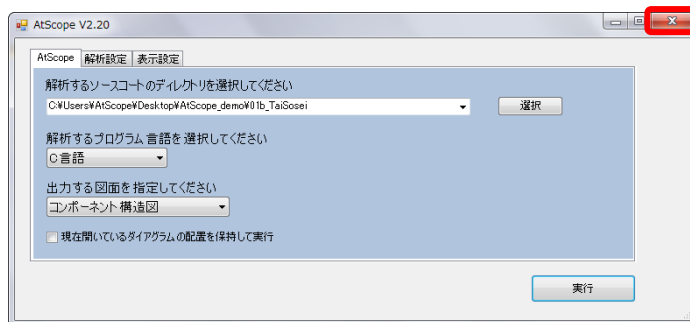
# 3 AtScopeを終了する

---

AtScope の終了方法を説明します。

---

- 1 「AtScope Vx.xx」画面右上の  をクリックします。



AtScope が終了します。

---

## 第4章

# AtScope の設定、その他

AtScope の各種設定、ソースコード診断サービスの紹介、および AtScope に関するお問い合わせ先を記載しています。

# 1 AtScope の各種設定

解析設定と表示設定について、説明します。

## 解析設定

「AtScope Vx.xx」画面の「解析設定」タブで、解析に関する設定をします。



### ■ 共通設定 > 依存線

各解析の依存線に変数アクセスを含める場合、「変数アクセスを含める」チェックボックスにチェックを入れます。

**POINT** デフォルト設定では、チェックが入っていません。

### ■ コンポーネント構造図 > 階層指定

アーキテクチャ解析のコンポーネント構造図に出力する階層を選択します。

#### ● 全て

すべての階層が出力されます。

#### ● 直下


ルート直下のコンポーネントのみ出力されます。

#### ● 2層目

2 層目のコンポーネントのみ出力されます。

- 3 層目

3 層目のコンポーネントのみ出力されます。

 デフォルト設定では、「全て」が選択されています。

## ■ 間接コール指定 > 編集

間接コールの呼出元と呼出先を指定することができます。

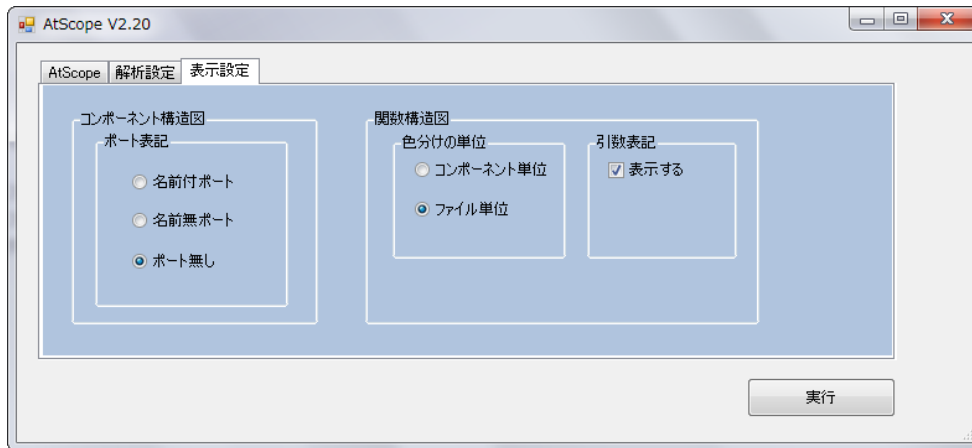
「編集」ボタンを押下すると、テキストエディタが開きます。「呼出元,呼出先」を列挙することで、呼び出し線を引くことができます。

## ■ 実行

「実行」をクリックすると、変更した設定で解析を実行します。

## 表示設定

「AtScope Vx.xx」画面の「表示設定」タブで、コンポーネント構造図と関数構造図の表示に関する設定をします。



### ■コンポーネント構造図＞ポート表記

コンポーネント構造図にポート(インターフェース)を表示するかどうかを選択します。

#### ● 名前付ポート

ポート(インターフェース)が名前付きでコンポーネント構造図に表示されます。

#### ● 名前無ポート

ポート(インターフェース)が名前なしでコンポーネント構造図に表示されます。

#### ● ポート無し

コンポーネント構造図にポート(インターフェース)は表示されません。



デフォルト設定では、「ポート無し」が選択されています。

## ■ 関数構造図 > 色分けの単位


関数構造図に表示されるオブジェクトの色分けの単位を選択します。

- コンポーネント単位

関数構造図に表示されるオブジェクトがコンポーネント単位で色分けされます。


- ファイル単位

関数構造図に表示されるオブジェクトがファイル単位で色分けされます。

 デフォルト設定では、「ファイル単位」が選択されています。

## ■ 関数構造図 > 引数表記

関数構造図に引数を表示する場合、「表示する」チェックボックスにチェックを入れます。

 デフォルト設定で、チェックが入っています。

## ■ 実行

「実行」をクリックすると、変更した設定で解析を実行します。

## テスト支援

単体テストと結合テストを支援する機能です。(本機能は保守期間中のみ有効です)



### ■ 関数複雑度

サイクロマチック複雑度が 10 を超えている関数を列挙します。要素をダブルクリックすることで、10 を超えている関数を降順に表示します。

また、図面としては 10 を超えている関数の数で色分けしています。

要素の色	複雑度の判定	単体テストのし易さ	1ファイル内のサイクロマチック複雑度 10 超の関数の数
青	正常	単体テスト容易	0 個
緑	注意	関数リファクタリングをすれば 単体テスト容易	1 個
オレンジ	警戒	やや広域な関数リファクタリングが必要	2 個～5 個
赤	危険	単体テスト困難 戦略的な設計改善が必要	6 個以上



関数リファクタリングすべき部分を図面で特定できます

## ■ RiTMUS 法

戦略的に結合テストを実施するためのメトリクス計測ファイルを出力します。

メトリクス計測ファイルの活用方法は、お問い合わせください。



## 2 ソースコード診断サービスについて

---

弊社では、ソースコードを設計品質の視点で解析して診断レポートを作成する「ソースコード診断サービス」を提供しています。  
ソースコード診断サービスについて、詳しくは以下の URL のページをご覧ください。

<http://www.bslash.co.jp/support-service>

## 3 お問い合わせ先

---

AtScope についてのお問い合わせは、メールまたは Web 上のフォームで承っております。

### ■メールでのお問い合わせ

[support@bslash.co.jp](mailto:support@bslash.co.jp) にご連絡ください。

### ■フォームでのお問い合わせ

以下の URL のフォームをご利用ください。

<http://www.bslash.co.jp/contact/>

---

発行責任 ビースラッシュ株式会社  
〒222-0033

神奈川県横浜市港北区新横浜 3-20-12  
新横浜望星ビル 8F

050-3793-0336

<http://www.bslash.co.jp/>

発行 2023 年 7 月

Ver.8.00

---