

Guided Checklists を使ったインスペクション

本文書は、Guided Checklists を使ったインスペクションの方法を説明したものです。

準備

ある要件を満たすためのインスペクションの方法が記述された Guided Checklists(GC)を使用することで、成果物（要件定義書、ソースコード）がその要件を満たしているかどうかを評価できます。本ハンズオンでは、以下のドキュメントが配布されます。本文書も配布されます。

- 2つの Guided Checklists
- インスペクション対象物（要件定義書、ソースコード）
- 参考文書（設計書）
- 空の欠陥リスト
- アンケート

ハンズオンの目的： チェックリスト(Guided Checklists)の指示に従い、セキュリティ要件が満たされているかどうか評価すること

本文書では、Guided Checklist を用いてソースコードと設計書をインスペクションする方法について具体例をまじえて説明します。Checklists は要件に応じて作成します。本文書で提示する Guided Checklist の例は、保守性が十分であることを確認するためのものです。ハンズオン当日には、セキュリティ要件を満たすかどうかを確認するための Guided Checklist を配布します。

設計書とソースコードへの Guided Checklist の適用例

インスペクタである Bob さんには、以下の二つのシステムの間成果物が与えられています。

- UML 図（設計書）（図 1）
- ソースコード（図 2）

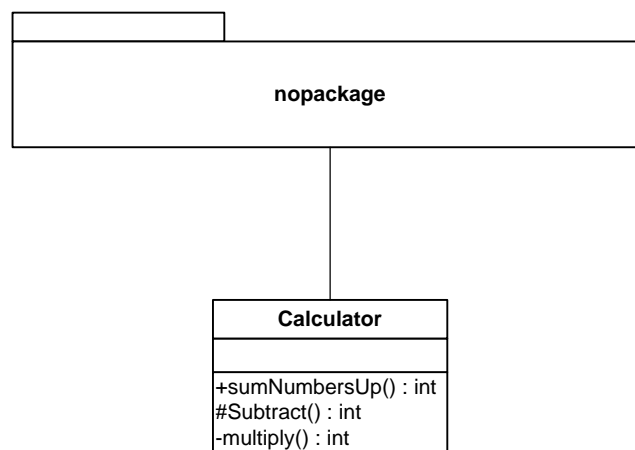


図 1 Calculator クラスの UML 図（設計書）

```

1. package nopackage;
2. /**
3.  *This clas represents a simple Calculator,
4.  *which performs some actions right after its start.
5.  *@author Charlie
6.  */
7. public class Calculator {
8.  /**
9.  *This is the standard way to start the Calculator.
10.  *@param argv
11.  */
12. public static void main(String argv[]){
13.     Calculator calc = new Calculator();
14.     System.out.println("Welcome to Calculator 2000");
15.     StringBuffer o = new StringBuffer();
16.     // hi Bob, how's the inspection?
17.     o.append("2+2*(5-1)="+calc.sumNumbersUp(2,
18.         calc.multiply(2, calc.Subtract(5, 1))););
19.     // calculation of 5-4*(3+3) and adding
20.     // the result to the StringBuffer of results
21.     o.append("5-4*(3+3)="+calc.Subtract(5,
22.         calc.multiply(4, calc.sumNumbersUp(3, 3))););
23.     // calculation of 1+2+3+4 and adding the result
24.     // to the StringBuffer of results
25.     o.append("1+2+3+4="+calc.sumNumbersUp(1,
26.         calc.sumNumbersUp(2, calc.sumNumbersUp(3, 4))););
27.     System.out.println(o.toString());
28. }
29. public int sumNumbersUp(int a, int b){return a+b;}
30. protected int Subtract(int a, int b){return a-b;}
31. private int multiply(int a, int b){return a*b;}
32. }

```

図2 Calculatorクラスのソースコード

Bob さんには、上記の2つの中間成果物に加えて、図3に示される Guided Checklists も与えられています。

図3の Guided Checklists でチェックできる要件は『保守性』です。チェックリストは、導入説明と複数の質問から構成されています。最上段にはチェックリストの目的、つまり、要件に関するすべての問題を発見するためにチェックすべきことが書かれています。ヒントには、質問に対するより詳細な説明やインスペクタが何を探せばいいのかが書かれています。

チェックリストを1つずつ確認する前に、要件『保守性』を大まかに把握するために全てのチェックリストに一通り目を通します。また、システムの理解を深めるために、設計書にも簡単に目を通します。（ここで紹介する例では、インスペクションの対象は設計書とソースコードです）

インスペクタは一つずつチェックリストの質問に答え、発見した問題点を全て記録します。Guided Checklistでの制約は『与えられた順番でチェックリストを読んでいくこと』だけです。

はじめに:要件である『保守性』を満たす必要があります。可能な限り、全ての質問に対して上から順番に答えてください。まず、質問を読んでください。次に、指示に従って、記載事項を確認してください。次に何番のチェックリストに進むべきか明確な説明がない場合は、1つ下の質問に進んでください。さらに、説明に記載されているような問題点を見つけた場合は、全ての問題点を欠陥指摘リストに記入してください。ヒントには、問題点を探すための追加情報があります。

目的:全ての問題点の発見

目的

1	<p>メソッドの可視性が一貫しているかどうか、設計書とソースコードを調べてください。</p> <p>メソッドの可視性に一貫性がない場合は、そのことを欠陥リストに記録してください。</p>	<p>類似したメソッド間では、可視性(public/protected/private)は一貫していることで保守性が向上します。</p>
2	<p>メソッドの名前が一貫しているかどうか、設計書を調べてください。</p> <p>設計書の中から、これに該当する部分をマークしてください。</p> <p>一つ以上のメソッド名がある場合は、そのことを欠陥リストに記録してください。</p>	
3	<p>1行のソースコードの長さ(ソースコードの横幅)は短いかどうか、ソースコードを調べてください。</p> <p>1行のソースコードの長さが適した長さであれば、マークしてください。</p> <p>1行のソースコードの長さが長すぎる場合は、そのことを欠陥リストに記録してください。</p>	
4	<p>ソースコードの説明がされているかどうか、ソースコードを調べてください。</p> <p>ソースコード中の説明されていると思われる箇所をマークしてください。</p> <p>ソースコードの説明がない場合や非常に大雑把に記述されている場合は、そのことを欠陥リストに記録し、質問8に進んでください。</p>	
5	<p>コメントには適切な表現が用いられているかどうか、ソースコード中のコメントを調べてください。</p> <p>コメントに適切な表現が用いられていない場合は、そのことを欠陥リストに記録してください。</p>	<p>コメントが口語体になりすぎていないか、コメントにコードの説明以外のものが書かれていないか？</p>
6	<p>Javadocが使われているかどうか、ソースコード中のコメントを調べてください。</p> <p>Javadocが使われている部分をマークしてください。</p> <p>Javadocが使われていない場合は、そのことを欠陥リストに記録してください。</p>	
7	<p>Javadoc以外の説明書きが使われているかどうか、ソースコード中のコメントを調べてください。</p> <p>ソースコードが説明されている部分をマークしてください。</p> <p>説明様式が使われていない場合は、そのことを欠陥リストに記録してください。</p>	
8	<p>1文字の変数名はカウンター変数だけであるかどうか、ソースコードを調べてください。</p> <p>ソースコード中の1文字からなるカウンター変数をマークしてください。</p> <p>1文字の変数名がカウンター変数以外にもある場合は、そのことを欠陥リストに記録してください。</p>	

図3 要件『保守性』のGCの例

以下に、2つの異なる構成のチェックリスト1,2がありますが、1は最上段が横縞で2段から構成されること、2は最上段が横縞で3段から構成されている点で異なります。

1.	<p>メソッドの可視性が一貫しているかどうか、設計書とソースコードを調べてください。</p> <p>メソッドの可視性が不統一である場合は、そのことをエラーリストに記録してください。</p>
----	--

チェックリスト1の上段には、どの文書を参照するか、文書の中から何を発見するかということが記述されています。下段には、欠陥を発見した場合に、何を記録するのかが記述されています。

メソッドの名前が一貫しているかどうか、設計書を調べてください。
設計書の中から、これに該当する部分をマークしてください。
一つ以上のメソッド名がある場合は、そのことをエラーリストに記録してください。

チェックリスト2の上段には、どの文書を参照するか、文書の中から何を発見するかということが記述されています。中段には、上段の質問部分とは異なり、文書の中でマークする必要があるものが書かれています。これは欠陥や問題点を発見するのではなく、設計書やソースコードの中から該当する記述を発見するということです。該当する部分を調べることにより、正しく表記、実装されているかどうかを確認できるとお考えください。下段には、正しく表記、実装されていない部分（欠陥）を発見した場合に、記録する必要のある項目を示しています。

では、具体的に Guided Checklists を用いたインスペクションを例を使いながら説明していきます。

Bob さんはインスペクションを開始します。

1. まず、チェックリスト1にある指示文『メソッドの可視性が一貫しているかどうか、設計書とソースコードを調べてください』を読んでインスペクションを開始します。

インスペクションでは、利用できる文書毎にチェックリストに記述された問題点を発見することが重要です。文書を見る一方で、Bob さんはメソッドの可視性が一貫しているかどうかを調べなければいけません。この場合では、ヒントが提示されています。1番目の質問の横の箱には、質問内容に対しての追加の説明やヒントを表しています。ヒントを読んだ後、インスペクタは、インスペクション対象の設計書である UML 図（図1）に目を通します。そこで、Calculator クラスが、足し算（sumNumbersUp）、引き算（Subtract）、掛け算（multiply）の3つのメソッドから構成されていることがわかります。

3つのメソッドは全て異なる可視性（+ public、# protected、- private）を持ちます。明らかにメソッドの可視性に一貫性がないため、これは欠陥です。『メソッドの可視性に一貫性がない場合は、そのことを欠陥リストに記録してください』とチェックリスト1の3段目に書かれているので、Bob さんはこの問題点を欠陥リストに以下のように記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): <u>チェックリスト1</u>
Java のクラス名、行 / ページ: <u>UML 図、Calculator クラス</u>
指摘内容: <u>メソッドの可視性が矛盾している！ Calculator クラスの全てのメソッドが異なる可視性を持っている。</u>

UML 図を確認した後、次はソースコードを見ます。3つのメソッドがあり、メソッドは異なる可視性を持つという、先ほどと同じ問題があることを確認できます。そのため、Bob さんは欠陥リストにこの問題を記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): <u>チェックリスト1</u>
Java のクラス名、行 / ページ: <u>Calculator クラス、29-31 行目</u>
指摘内容: <u>メソッドの可視性が矛盾している！ Calculator クラスの全てのメソッドが</u>

異なる可視性を持っている。

チェックリストに全て答え、チェックリストの確認が完了したら、次の質問に取りかかります。

- 次に、チェックリスト2『メソッドの名前が一貫しているかどうか、設計書を調べてください』を読んで確認します。

Bobさんは指示されたとおりに設計書に目を通し、確認できたメソッドにマークをします。実際に調べると、この例ではメソッドの名前には一貫性がありません。UML図より、メソッドの名前がsumNumbersUP、Subtract、multiplyであることがわかります。メソッドは1単語から成るわけでも、小文字/大文字で一貫しているわけでもないなので、メソッドの名前には一貫性がないと判断しました。したがって、Bobさんは指示に従い、この問題点を欠陥リストに記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): チェックリスト2

Java のクラス名、行 / ページ: UML図、Calculator クラス

指摘内容: Calculator クラスの全てのメソッドの名前は一貫していない。

- 続いて、チェックリスト3『1行のソースコードの長さは短いかどうか、ソースコードを調べてください』を読んで確認します。

このチェックリストでは、「ソースコードを調べてください」とありますので、ソースコードを対象とします。Bobさんはソースコードに目を通し、短くする必要のある3つの長い行(17-18、21-22、25-26行目)があることを確認します。したがって、Bobさんは指示に従い、この問題点を欠陥リストに記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): チェックリスト3

Java のクラス名、行 / ページ: Calculator クラス、17-18、21-22、25-26 行目

指摘内容: これらの行は長すぎるため、適当なメソッドの引数区切りで行を折り返し、短くすべきである。

- 続いて、質問4『ソースコードの説明がされているかどうか、ソースコードを調べてください』に移ります。

ソースコード中にあるコメントを調べることで、クラス定義の前、メソッド定義の前、メソッドの中にコメントがあることを確認し、その内容が説明として十分であると判断しました。そして、チェックリスト4に記述されているように、コメントにマークをします。

3つ目の指示に、『ソースコードの説明がない場合や非常に大雑把に記述されている場合は、そのことを欠陥リストに記録し、チェックリスト8に進んでください』と書かれています。今回の場合、ソースコードは十分に説明がされているので、この指示には従いません。もしソースコードに十分な説明がない場合は、欠陥リストに問題点を記録し、チェックリスト8に進みます。チェックリスト5,6,7は、コメントが存在する場合のみ意味をなすチェックリストだからですが、インスペクタはそのことを意識する必要はありません。

5. 続いて、Bob さんはチェックリスト 5「コメントには適切な表現が用いられているかどうか、ソースコード中のコメントを調べてください」に進みます。

この質問に答えるために、Bob さんはソースコード中にある適切でないコメントがないか探します。このチェックリストでも、右側のヒントによって質問の補足説明がされています。

Bob さんはチェックリスト 4 ですでにコメントに目を通してしているので、コメントを探すのは容易です。そこで、ソースコードの 16 行目に Bob さん宛のコメント「やあボブ、インスペクションはどうだい？」を発見します。面白いコミュニケーションですが、これはコメントとしては明らかに適切ではありません。Bob さんの友人の開発者 Charlie さんが書いたものと考えられます。Charlie さんは Bob さんがコードインスペクションすることを事前に知っていたからでしょう。Bob さんはこれを欠陥リストに記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): チェックリスト 5
Java のクラス名、行 / ページ: Calculator クラス、16 行目
指摘内容: Charlie はソースコードのコメントを通して私と連絡をとりたがる。

次に、Bob さんはチェックリスト 6「Javadoc が使われているかどうか、ソースコード中のコメントを調べてください」に進みます。

これはすぐに発見することができます。というのも、Bob さんはすでに Javadoc コメントを発見していたからです。(もしチェックリスト 4 でマークしていない場合は) ソースコード中の Javadoc が使われている箇所にマークをします。この質問では問題がないので、欠陥リストには何も書きこまず、次のチェックリストに移ります。

6. Bob さんはチェックリスト 7「Javadoc 以外の説明書きが使われているかどうか、ソースコード中のコメントを調べてください」に進みます。

この質問でも、Bob さんはすでにソースコード中のインラインコメントとして発見しているので、すぐに答えられます。(もし質問 4 でマークしていない場合は) ソースコード中にマークし、クラスに対して十分にコメントがされていると判断することができます。前問同様、欠陥リストに記入する必要はありません。

7. 最後のチェックリスト 8「1 文字の変数名はカウンター変数だけであるかどうか、ソースコードを調べてください」に進みます。

StringBuffer o という名前の変数名が 1 文字のものがあることに気づきます。o はカウンター変数ではないので、Bob さんは「1 文字の変数名がカウンター変数以外にもある場合は、そのことを欠陥リストに記録してください」に従い、欠陥リストに記入します。

指標番号 / 質問番号 (SGIT、Guided Checklist の方のみ): 質問 8
Java のクラス名、行 / ページ: Calculator クラス、15 行目
指摘内容: StringBuffer 変数 o は、カウンター変数でないにもかかわらず、変数の中身を適切に表していない。

これで、**Bob**さんは全てのチェックリストに答え、「保守性」を妨げる複数の欠陥を特定することができました。

Bobさんのインスペクションの成果物は、欠陥リストと、印が記入されたソースコードであり、次回のインスペクションミーティングの議題となります。