CHECKS

情報の一貫性に関するパターン言語

著：Ward Cunningham
訳：友野 康子

ユーザ入力を受けつけるプログラムは、正しい入力と誤った入力を判別し、誤りのある情報を記録しないようにする必要がある。CHECKS パターン言語は、プログラムを整然と保ち、また、将来への柔軟性について妥協することなく、そのようなチェックを行う方法を述べたものである。

本パターン言語は、11 パターンからなり、それらを 3 章に分けて提示する。第 1 章では、ユーザインタフェースから獲得され、ドメインモデル内で使用される値について述べる。第 2 章および第 3 章では、データ入力中、提出または公開後における誤りの検出と修正を主題とする。本パターン言語は、筆者が金融関連のシステムを Smalltalk 上で開発した経験から生まれた。より広範囲に及ぶパターン言語の一部として記述したため、概略的で不完全な印象を与えるかもしれない。本論文は、パターンの選択および関連づけの実験であると同時に、実践的な知識の伝達についての試みでもある。

第 1 章

まず、ドメインモデルで使用される量（quantity）について検討する。ドメインコードは、ビジネス「ロジック」の（ときとして非論理的な）詳細を、微に入り細を穿ち表現する。コードに記述されるすべての文節は、実際のビジネスにおける事実に動機付けられたものでなければならない。他の関心事は、本章で述べる特化した値や、後に述べるオブジェクトにより表現される。本章では、次のパターンについて述べる。

1. **完結した値**（Whole Value）
2. **例外値**（Exceptional Value）
3. **意味のないふるまい**（Meaningless Behavior）

1. 完結した値（Whole Value）

プログラム言語に備わった少数のリテラル値（数、文字、真-偽など）、および、通常の値として使われるごく少数のオブジェクト（日付、時刻、点など）に加え、本パターンでは、ビジネス領域において意味のある量を表すオブジェクトを新たに作成し利用する。こ

うした値（通貨、カレンダー期間、電話番号など）に、完結した有用な情報のチャンクをユーザインタフェースからドメインモデルに運ばせる。

ビジネス（ドメイン）モデルをパラメタ化したり、定量化したりするとき、計算の最も基本的な構成単位で表現したいという圧倒的な誘惑にかられる。（抽象概念が備わっていないか、または弱い言語では当然の手法だったが）現在では、そうする必要性がないばかりか、プログラム同士の間およびプログラムとユーザとの間における円滑で的確なコミュニケーションを阻害する。ビット、文字列、数値、といったものは、ほぼどんなものでも表現するのに使えるため、単独では何の意味も示さないためである。

そこで：

特化した値を構築し、ドメインモデルを定量化せよ。また、こうした値をメッセージの引数や入力と出力の単位として使用せよ。このオブジェクトは、完結した量を捉えており、その意味するところは、単なる大きさ以上のものである。ただし、特定のドメインからは独立させよ。（「値」という語から、このようなオブジェクトの識別が重要でないことが示唆される。）ユーザインタフェース（フィールドやセルならなおよい）に、このオブジェクトを入力から作成したり印字出力したりするための、正確で信頼性の高いフォーマットコンバータを用意せよ。ドメインモデルは、この情報について、文字列や数値による表現形式では扱わないものとする。次のメッセージと応答を検討せよ。

```
メッセージ: contractDurationInDays
答え: 21

メッセージ: contractDurationInDaysAsString
答え: '21'
```

どちらのメッセージもまわりくどく、メッセージの戻り値からは意味が欠落している。次のメッセージは、より単純であり単独で意味の分かる**完結した値**を返す。

```
メッセージ: contractDuration
答え: Weeks(3)
```

こうしたオブジェクトに、ドメインモデルの不規則さや（よくあることだが）曖昧さを取り込ませようと思うかもしれない。そんなときは、個々のクラスをクラス階層にせよ。完結した値は、現実的でない例外的な量を含むよう拡張してはならない。これらは**例外値**(2)で適切に表現できる。また、不適切な値の組み合わせについて過度に理由付けすることは避けよ。後に**意味のないのふるまい**(3)でこれについて述べる。

2. 例外値（Exceptional Value）

通常、ビジネスモデルは、基本的な事例や抽象的な部分から構成される。この抽象的な部分は、ビジネス現場の多様性を実現するために特化および精緻化される。しかし、ビジネスにおけるすべてのありうることをクラス階層に盛り込むと、多くの場合、猥雑、難解、または不適切なものになる。そのため、属性の領域を**完結した値**(1)以外のものも含むように拡張することがあるだろう。

世論調査では、「そう思う」、「非常にそう思う」といった返答を集計する。「判読不能」、「回答拒否」といった定量化できない返答については、曖昧になってしまうとしても、この値域外として表現しておく方がよい。しかし、そのためには、ドメインモデルの構造に、この種の不明データのための余地を残しておかなければならない。あとから、こうしたデータが現れるかもしれないからである。実際には、極めて小さなドメインモデルでもない限り、オブジェクト生成（データ入力）中に不明データが現れることは避けられない。

そこで：

例外的な状況を表現する、少数の特例的な値を使用せよ。例外値は、`isNil` や `printOn:` といった識別用のプロトコルを除き、あらゆるメッセージを受けつけ、大半のメッセージに対し他の例外値を返答するか、（`does not understand` を返し）あらゆるメッセージを拒絶する。インタフェースウィジェットは、ブランクが入力されたときは `nil` を生成し、`nil` に対する出力をブランクにする。ドメインモデルは、少なくとも一時的に、`nil` や他の例外値を正規の入力として受け入れるべきである。Smalltalk では `UndefinedObject` オブジェクトを洗練し、これに説明を運ばせることができる。こうすると、`aValue == nil` は、もはや `aValue isNil` と異なる意味になる。

```
purchaseDate
  ^(buys := self trades select: [:each | each isPurchases])
  size <<= 1
    ifTrue: [buys first settleDate]
    ifFalse: [ExceptionalValue reporting: ‘various’]
```

例外値はドメインモデル階層およびメソッドの構造を単純化する。明示的に各メソッドの例外値をテストしなくともよい。例外型は、メッセージを吸収するか、**意味のないふるまい**(3)を生じさせるからである。それに必要な細かい例外値のハンドリングは、ユーザインタフェース周辺に集中する。例えば、報告書作成で正確に加重平均を計算するためには例外値を検出する必要がある。`WeightAverageColumn` オブジェクトが適切にオブジェクトとして切り出され、ドメインオブジェクトに代わり計算を実行することで関心が分離

される。**仮説的公開**(7)ではテストを完全に行わないが、かわりに、**後回しの検証**(6)がテストの完全性に責任をもつ。

3. 意味のないふるまい（Meaningless Behavior）

完結した値(1)によりビジネスロジックを定量化したところ、ふるまいが少しずつ様変わりし、計算のところどころで例外値が現れるとする。その場合、メソッドのコードは予想しない事態に遭遇する可能性がある。
ビジネスのルールは選択的に適用されるにすぎないことを心に留めよ。ビジネスプラクティスは、「適用すべき」ルールの周辺を揺らぎながら進化するものである。ドメインモデルにおいて、ビジネスロジックを表現するのに、元の考え方や現在の表現より複雑になることは許されない。

そこで：

失敗の可能性を無視してメソッドを書け。失敗から回復し処理を続行するための計算は、入力・出力ウィジェットに任せる。出力は、ブランクのままにせよ。さもないと、**意味のないふるまい**に意味を付加することになってしまう。ユーザは予想外のブランクを、入力拒否や無効な出力と解釈するだろう。

意味を詰め込むと：

```
weightedAverageCost
    |total weight|
    total := self weghtedTotalCost.
    total isCurrency ifFalse:
        [^ExceptionalValue reporting: ‘N/A’].
    weight := self totalWeight.
    (weight isNumber and: [weight isZero not]) ifFalse:
        [^ExceptionalValue reporting: ‘Empty’].
    ^total / weight
```

意味がないことを許すと：

```
weightedAverageCost
    ^self weightedTotalCost / self totalWeight
```

メモ：このパターンをエラーハンドリングの記述として理解した読者もいるかもしれない。しかし、そうではない。論点は多様な様相を示すドメインメソッドの記述にある。ここでは、瑣末なエラーハンドリングの問題は入・出力システムに配置されるものとしているが、必ずしもそうではない。**意味のないふるまい**は**例外値**(2)の代替実装と捉えられ、多くの場合ユーザからその差異はみえない。ドメインにおける意味の条件が予想できない場合、**意味のないふるまい**を選択せよ。（ドメインにおける意味と対照的なのは、単なる操作上の意味である。たとえば欄が埋まっていないといったことである。）ところで、プログラム内部に非常に大きな問題があるかもしれない。その場合、手がかりを与えることが肝心である。**エコーバック** (4)は、ブランクを出力することにより失敗を警告する。重大なトラブルが発生したときは、**含意の表示**(5)を報告する入力画面を同様にブランクにする。不正な記録が残ってしまうことを避けられない場合、**後回しの検証**(6)にて意味のある振る舞いを求める。

第 2 章

人は、プログラムのインタフェースを通じてしかドメインモデルを操作できない。このインタフェース自身も１つのプログラム（インタフェースモデルとグラフィック機構）であり、その目的は、できるだけ透過的にドメインモデルを操作できるようにすることである。ユーザインタフェースは、ユーザの心にシステムを制御しているという感覚を生み出すようにプログラムされる。それには、ドメインモデルの状態についての十分な情報を提供する必要がある。そして、そのための理に適った操作を備えなければならない。以下のパターン群は必要なフィードバックを提供する。

4. **エコーバック**（Echo Back）
5. **含意の表示**（Visible Implication）
6. **後回しの検証**（Deferred Validation）
7. **筋書の素描**（Instant Projection）
8. **仮説的公開**（Hypothetical Publication）

4. エコーバック（Echo Back）

フィールドおよびセルウィジェットが**完結した値**(1)を構成し、それをドメインモデルに渡すものとする。通常、オブジェクト指向プログラミングにおいて、提示された情報を抽出、解釈、拒否する選択はドメインモデルに委ねられている。**エコーバック**パターンは、ドメインモデルにその選択を説明するという最低の義務を負わせるものである。

ユーザは、ひとしきり値を入力した後、タイプミスを確認するために見直すものだ。ユーザは、これを繰り返すが、その区切れ目は一定でない。いつ見直しを行うか全くわから

ない（それは一瞥ですんでしまう）。そのため、各フィールド又はセルが処理され次第、値の入力がうまく行ったかどうかをユーザに伝えなければならない。また、問題が起きたとしても、その情報を作業の妨げとならないように提示しなければならない。

そこで：

ドメインモデルに書き込まれたすべての情報をそのまま読み出しなおしたものを提示せよ。フィールドおよびセルの値は、入力されるごとにすぐドメインモデルから読み出しなおされエコーされたものとせよ。元の情報から単純に導かれるプロトコルを使って、再構築された値を返答せよ。メモ：通常の setter/getter の慣習（つまり、`attribute` とか `attribute:`）はこれを満たすが、属性以外のものも対象となる。常に入力値がユーザにエコーされることで、ユーザには入力値についてとられる選択や解釈がわかる。

ドメインモデルが境界値や不正な値の解釈までも通知を介して説明するものとしてはいけない。モデル側がこの役割を担うのは適切でない。漸次的に行われる入力を妨げてしまうからである。例として、次の支払い日の入力を検討しよう。

ユーザ入力: `5/8/94`
エコーバック: `05/08/94` (1994 年 5 月 8 日にあたる完結した値の標準形式)

ユーザ入力: `5/5/94`
エコーバック: `05/08/94` (モデルが選択した直近の給与支払日、通常は金曜日)

元の 6 文字をモデルが日付の標準的な出力表現に置き換えたため、どちらも先頭は 0 になっている。2 番目の例では実際の数字が異なっている。モデルが、不適切な値の入力に替えて、一番近い適切な値を保存するよう選択したためである。エコーバックによりこの選択が明示される。モデルが不適切な値を黙認すること、または、（**例外値**として）無条件に受け入れることといった代替案も同等に正しい。ここでの論点は、どちらの方法が優れているかではなく、ユーザの作業の流れを妨げることなく下された決定をユーザに示すことである。これに従えば、意味のない setter は不適切な値を黙認し、意味のない getter はブランクを印字すべきだろう。

本パターンは、トラブルが発生したときに、まずビープを鳴らしたり、画面をちかちかさせたりすることでユーザに知らせるという、通常取られるやり方とは一線を画している。不正な値であることを訴えかける機会は、**後回しの検証**(6)で十分に与えられる。

5. 可視な含意（Visible Implication）

エコーバック(4)の仕組みと属性を計算するメソッドを組み合わせることで、入力を簡単にし、視覚的レビューの効率を向上する。

事物を定量化するには、さまざまな方法がある。こうした測定値には、冗長なものもあり、ある事物の特徴はその一部で完全に捉えることができる。ある値は、他の（導出される）値の基底となっている場合もある。また、同じものを違った視点から見ただけのものである場合もある。

そこで：

既に入力されている量が含意する、導出可能または冗長な量を算出せよ。変更される値に加え、算出した値もフィールドまたはセルに表示する。可能なら、導出される量を入力可能にし、基底の測定値を逆算する。どちらの測定値を基底とするかは曖昧だが、よりアプリケーションに非依存で変数が多いものを基底として選べ。

入力された量が: 12
単価が:6.50
ならば合計は: 78.00

入力された量が: 12
合計が: 72
ならば単価は: 6.00

入力された単価が: 7.00
合計が: 77.00
ならば量は: 11

ドメインモデルのアクセスメソッドでこのロジックを実現せよ。getter は他の入力から不明な値を算出するようにせよ。

```
unitPrice ^unitPrice: notNil
    ifTrue: [unitPrice]
ifFalse: [total / quantity]
```

setter は冗長な量を選択した基底の測定値に変換せよ。

```
total: aValue
    unitPrice isNil
        ifTrue: [unitPrimce := total / quantity]
        ifFalse: [quantity := total / unitPrice]
```

ドメインモデルの含意を算出するロジックは簡素に保とう。また、ユーザインタフェースのフィールド間の依存関係として埋め込んではいけない。そのかわり、一つでもフィールドに変更があれば全フィールドを更新するようにする。
事物の指定が不完全なときは、**意味のないふるまい**(3)に任せよ。これがブランクを表示し、不完全であることをユーザに印象づけるだろう。含意の計算はほんの一瞬で終わることを確認すること。長時間かかる計算や、不十分な指定の下では安全に実行できない計算の場合、**筋書の素描** (7)に任せるとよい。

6. 後回しの検証（Deferred Validation）

ドメインモデルを定量化する**完結した値**(1)は、条件を満たす値であるかチェックされ、ときにはドメインモデルが望む形に修正され、ユーザへ**エコーバック**(4)される。このようなチェックは登録に際して即時に行われる。けれど、可能な限り先の時点まで実行を延期すべきチェックもある。

ユーザは、一連の入力を完了すると、コンピュータによるさらに発展した動作を望む。それは、例えば「いかがでしたか」という単なる問いかけであったり、「明日までにやっておきます」という作業の中断であったり、「ここから取得してください」といった責任者の変更であったりするかもしれない。一貫性として要求されることは、正確には、コンピュータの動作が要求されるまではわからない。

そこで：

ドメインモデルによる検証の詳細を、動作が要求されるまで延期せよ。特定の動作に検証の範囲を絞れ。未完成の作業を私的な場所に保存する際には、完成した作業を公の場所に提出するときほどの検証を要さない。（必ずではないが）複雑な動作についての検証チェックは、単純な動作についてのすべてのチェックにいくつかチェックを追加したものとなる。ドメインモデルにメソッドを記述し、想定される使い方をコード化せよ。各メソッドは、独自のチェックの前に、より単純な検証へと委譲する。本質的な問題をいち早く知らせるためにも、この順序を守らなければならない。また、要求された量(quantity)と他との一貫性を調べる前に、その量があるかを確認すること。

```
validateForPublication: notificationHandler
    self validateForSale: notificationhandler
    self validateForComputation: notificationHandler.
    self validateVariablesForPublication: notificationHandler
    self validateRelationsForPublication: notificationHandler
```

検証の各メソッドは複雑になり、日常的に変更が加えられるものと想定せよ。これ自身は、ビジネスルールや制限の検証に特化したシステムとなるだろう。

後回しの検証(6)は、ドメインオブジェクトがシステムのより公開された部分に出るときに通らなければならない関門である。こうしたシステムでは、そうしたオブジェクトを公開する前に問題点についてユーザの注意を促すため、検証することを必要とする。しかし、どんなに精密に検証したとしても発見できない潜在的問題をユーザが意識している場合もある。**筋書の素描**(7)と**仮説的公開**(8)では、システムへの入力をさらに評価するための、2つの関連しあう戦略を展開する。

7. 筋書の素描（Instant Projection）

先々利用することを考慮して情報が集められ、その入力についての**含意の表示**(5)がなされることで、不整合が発見されることが期待される。さて、ユーザの関心は入力から公開へと移る。それにあわせ、公開のもたらす影響の予知に話を進めよう。

後に利用できるようと情報をいろいろ集めるが、実際に「どのように」使われることになるのかは、予想もできない未来の事象にかかっている。とはいえ、多分、こんなことが起こるだろうという流れは想定され、事物がどのような結果となるかを予想できる。

そこで：

実際に公開する前に、公開することによってどんなことが起こるかの筋書を提示せよ。さらに前提となることを入力させたり、予測技法の選択肢を提供してもよい。筋書の演出には骨の折れるものである。いろいろな表や要約を含む「筋書ウィンドウ」を開くことを検討せよ。古めかしい表もパラメタの影響を観察するのに役立つだろう。

筋書には、公開間近のドメインモデルに関する問合せが想定される。単なる問合せであれば、システムがレポートを返すという以上の影響を残さない。問合せが複数の公開に関与する状況では、扱いは面倒だが万能な、**仮説的公開**(8)が筋書の代わりとなる。

8. 仮説的公開（Hypothetical Publication）

複雑なドメインモデルが公開のための**後回しの検証**(6)をすべて通ったとしても、まだ公開のリスクが高い恐れがある。こうしたリスクは、既に行われた動作の記述の誤りよりも、モデルに記述される動作を行うことによるものである。そのために、間違いを検出するためのかなり巧緻な仕組みを検討する。

ある情報を公開（提出、または、入力を完了）すると、その情報は、さまざまなところに届けられる。文脈や他のやはり疑わしいデータから非依存に、情報が与えるすべての影響を捉えることは難しい。

そこで：

ユーザは、ある制約の元に、**仮説的公開**を好きなだけ作成し、システム内に公開してもよいこととする。その配布範囲は、ユーザ本人のワークステーション上のサブシステムに制限する。または、他のユーザのために仮のものであることを明示する。仮説を一つひとつまたは一まとめに棄却するための仕組みを用意する。

公開されるモデルに対して予測手段があれば、**筋書の素描**(7)が**仮説的公開**(8)に代わりうる。大量の履歴情報を入力およびチェックしなければならないときにも、仮説は有効である。

第 3 章

続いて、長期的な視点における情報の一貫性のための仕組みを検討する。2 つのパターンが疑わしい情報の 1、2 の形態に取り組む。これらは、ビジネスにおける多くの量が曖昧に指定されたり、いかようにも解釈できたりするものであることを認識している。これらのパターンは、一貫性の保証へとつながっていくのだが、残念ながら、このパターン言語では省略した。

9. **予測承認**（Forecast Confirmation）
10. **診断用照会**（Diagnostic Query）

9. 予測承認（Forecast Confirmation）

通常、現実の事象はコンピュータシステム上にモデルとして生成されるより先に起こる。事象が予想できるとき、機械的にそのモデルを生成し、公に利用できるよう公開されるだろう。しかし、コンピュータシステムが現実に追いついたとき、正確に現実をモデル化しているかが重要である。

そこで：

機械的に公開された事象に対し、その値を調整し承認する仕組みを提供せよ。次の例の流れを検討せよ。

火曜日：　金曜日の自動的な預金は 187,655.47 と予想。
金曜日：　機械的に 187,655.47 を預金勘定に記帳。
月曜日：　銀行記録は金曜日の預金額を 187,655.50 と記すので、この通りに調整。
後日：　当月記録は例外的な動作を含み、終了。

現実化するまで、その情報を公開することについて厳密には誰も責任を負わないのだが、

どの時点においても最善の情報が生きていることが重要である。一貫性の保証という視点からは、**予測承認**は入力される情報とあまり変わらない。**予測承認**は機械的に生成されたモデルだけに適用される。承認された後、その値は一貫性の保証に委ねられる。

10. 診断用照会（Diagnostic Query）

完結した値(1) と**例外値** (2) により有用な情報が集められるが、これらは、情報の丸めやその他の単純化のため、必ずしもユーザインタフェースから見えない。値が表示されるところから、元の情報（またはそれ以上のもの）を取り出せるべきである。

ビジネス活動のさまざまな記録を多面的に検証したところ、矛盾点が発見されたとしよう。このとき、エラーの原因が見極められるまで、すでに公開された情報の多くに疑いがかかる。記録エラーを追跡するのは、通常のビジネスにおける操作を観察するのと全く異なる。しかし、全く新しい観察の仕組みに移行してしまうと、なじみのあることがらと決別し、長時間の作業を強いられることとなる。

そこで：

あらゆる値について診断のために追跡する仕組みをシステムに組み込め。丸めや集計を伴うすべての表示に、調査用の処理前の値を提示させよ。

通常の表示: 67%
診断用の表示: 66.6454329

通常の表示: 652 MM USD EQV
診断用の表示: 622,456,325.07 USD + 3,624,878,450 JPY + 23,549,54 FRF

同様に、ルールや公式が適用されるところには、これ自身を取り出せるようにし、変数名と特定の計算において当てはめられた値を含めて整形せよ。追跡していくと、最後に入力値に辿り着くだろう。そこで、この情報源の日付、時刻、識別子を提示する。

通常の表示: 22%
診断用の表示: ROR = 22% = internal rate of return (P0, P1, Cfi)
P0 = 32,454,55 = market value (1/1/93)
P1 = 36,537,39 = market value (12/31/93)
CF1 = 354.00 = cash flow (3/15/93, #1000324)
CF2 = -400.000 = cash flow (7/31/93, #1000378)
CF3 = -100.00 = cash flow (8/30/93, #1000412)

入力ミスの修正は、診断情報にとって別の情報源になる。全情報源の以前の値、時刻、識別子も診断に供されるべきである。

Ward Cunningham **への連絡は：**
Cunningham & Cunningham, Inc., 7830 S.W. 40th Avenue, Portland, OR 97219
ward@cz.com