Patterns of Simplicity

Giovanni Asproni
Kevlin Henney
Peter Sommerlad

Agenda

- Kick-off
- Simplicity
- Examples
- Patterns
- Workshop
- Wrap-up

Defining Simplicity

Simplicity is being simple. It is a property, condition, or quality which things can be judged to have. It usually relates to the burden which a thing puts on someone trying to explain or understand it. Something which is easy to understand or explain is simple, in contrast to something complicated. In some uses, simplicity can be used to imply beauty, purity or clarity. Simplicity may also be used in a negative connotation to denote a deficit or insufficiency of nuance or complexity of a thing, relative to what is supposed to be required.

http://en.wikipedia.org/wiki/Simplicity

Thoughts on Simplicity

The price of reliability is the pursuit of the utmost simplicity.

C A R Hoare

The approach to style is by way of plainness, simplicity, orderliness, sincerity.

William Strunk and E B White

Simplicity before understanding is simplistic; simplicity after understanding is simple.

Edward de Bono

Favour Logic over Control Flow

```
if(isOldest)
{
    if(timeLastAccessed < cutOffTime)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return false;
}
```

Simplifies to...

```
return isOldest && timeLastAccessed < cutOffTime;
```

Replace Flags with Conditions

```
boolean failed = false;
if(message != null)
{
    if(!enqueue(message))
        failed = true;
}
else
    failed = true;
if(failed)
    throw new MessageEnqueueException();
```

Simplifies to...

```
if(message == null || !enqueue(message))
    throw new MessageEnqueueException();
```

Consolidate Duplicate Code

```
if(!file.exists() ||
   (file.exists() && !file.hasWritePermission()))
    return false;
```

```
if(!file.exists())
    return false;
else
    if(!file.hasWritePermission())
        return false;
```

Eliminates duplicates, but long-winded and clumsy rather than simple

Both simplify to...

```
if(!file.exists() || !file.hasWritePermission())
    return false;
```

Behavioural Test Cases

```
void testIsLeapYear()
{
    // years not divisible by 4 are not leap years
    assert !isLeapYear(1906)
    assert !isLeapYear(2009)

    // years divisible by 4 but not by 100 are leap years
    assert isLeapYear(1984)
    assert isLeapYear(2008)

    // years divisible by 100 but not by 400 are not leap years
    assert !isLeapYear(1900)
    assert !isLeapYear(2100)

    // years divisible by 400 are leap years
    assert isLeapYear(2000)
    assert isLeapYear(2400)
}
```

More clearly expressed as...

```
void testThatYearsNotDivisibleBy4AreNotLeapYears()
{
    assert !isLeapYear(1906)
    assert !isLeapYear(2009)
}
void testThatYearsDivisibleBy4ButNotBy100AreLeapYears()
{
    assert isLeapYear(1984)
    assert isLeapYear(2008)
}
void testThatYearsDivisibleBy100ButNotBy400AreNotLeapYears()
{
    assert !isLeapYear(1900)
    assert !isLeapYear(2100)
}
void testThatYearsDivisibleBy400AreLeapYears()
{
    assert isLeapYear(2000)
    assert isLeapYear(2400)
}
```

Simplicity through Practice

- Many examples of practices can be captured as considerations and heuristics
  - E.g., be suspicious of lots of Boolean literals
  - E.g., refactor against tests
  - E.g., care about the little details, because complexity starts in the small
- Others are more specific
  - E.g., apply De Morgan's Laws
  - E.g., name interfaces after roles

Patterns

- Patterns allow us to name, deconstruct and communicate proven practice
  - Recurrence (ideally *good*) is key
  - Patterns can refer to general design ideas, code detail, day-to-day development practice, broader development process, etc.
- Patterns are of interest as a means of communicating and reflecting on simplicity and the practices that foster it

Naming and Context

- Patterns are named
  - Typically, but not necessarily, noun phrases
  - Nouns work well for describing structure, but verbs can be effective for process advice
- Patterns apply in a context
  - They are not universal principles: a good practice in one context can be poor in another
  - E.g., programming language, framework, tool chain, project type, organisational culture

From Problem to Solution

- Patterns are motivated by a problem
  - The problem is often characterised by forces and a simple summary of the problem
- Patterns propose a solution
  - A simple summary of a solution is often accompanied by more detail and discussion
- Patterns have consequences
  - Both benefits and liabilities, actual and potential, should be considered

Patterns for Simplicity

- What patterns can we find that help with the act of simplifying a software system?
  - What patterns of design, from the fine grained to the large scale, do we see in code that we consider to be simple?
  - What patterns of practice do we know that help us to keep things simple?
  - In what ways are these different to the patterns or combinations of patterns we find that lead to complexity and complications?

Patterns for Complexity

- Not all that recurs is necessarily good
  - Originally referred to simply as *bad patterns*, many terms now cover this idea: smells, dysfunctional patterns and anti-patterns
- Accidental complexity arises from groupthink and inappropriate pattern use
  - E.g., most applications of Singleton
  - E.g., copy and paste coding
  - E.g., a setter per getter, a getter per field

Pattern Mining

- Workshop to identify and detail some patterns that promote simplicity
  - And also note some that lead to complexity
- Workshop approach:
  - Brainstorm pattern names
  - Timeboxed pattern writing
  - Group patterns together

Pattern Names

- Brainstorming
  - No need to filter or edit
- Choose descriptive names
  - You should be able to tell what the idea of the pattern is by its name
  - You can include existing known pattern names, practices and refactorings

Pattern Elaboration

- Name
  - Directly from brainstorming or refined
  - Also mark the confidence or quality next to the name (select from *Good*, *OK*, *Neutral*, *Ungood*)
- Context
  - Where and when pattern applies
- Problem
  - What are the issues?
- Solution
  - The practice/technique/etc. and its consequences?

Grouping

- Group similar and related patterns together
  - Similar because they are the same in terms of outcome, overall intent or actual practice
  - Related because they relate to the same kind of thing or at the same level of scale

Done

http://wiki.hsr.ch/SimpleCode/