Writing Your First Pattern

Everything — Well, Many Things — You Ever Wanted to Know About Pattern Writing but Were Afraid to Ask

Kevlin Henney

kevlin@curbralan.com

Allan Kelly

allan@allankelly.net

Motivation

- Anyone can write a pattern, but it can be daunting if you've never done it before
  - Knowing where to start, what (not) to include, what form to adopt, how to write and how to get it generally accepted and communicated as a pattern
- The session is aimed primarily at people who would like write a pattern but have never tried
  - Or have tried and found it difficult
  - And it also for the generally interested, i.e. those who would simply like to know what's involved

Objectives

- In this session we're going to look at how patterns get written
  - The motivation and starting point for writing a pattern
  - What should be included in a pattern
  - What should be easily visible in a pattern
  - How to combine multiple patterns in a presentation
  - How the pattern community is important in improving patterns and developing pattern authors

Why Write a Pattern?

- Pattern writers are cool dudes, so to write patterns must be a cool thing to do
  - It helps to have a more concrete motivation... as well as something to write
- OK, what do you think?

Why Write a Pattern (Revisited)?

- You've seen or had to explain something multiple times...
  - A high-level design, a C++ practice, a process, etc
- And you want to document or understand it
  - You want to share it and discuss it
  - The process of writing helps to clarify to you and others what makes the pattern tick

Have You Really Found a Pattern?

- Have you seen it elsewhere?
  - Some recurrence is considered a prerequisite — hence the use of the word *pattern*
  - E.g. two other example implementations in addition to your own
- Has it been documented before?
  - Was that version comprehensive and satisfactory?
  - Is what you have in mind just an implementation variation or does it add something deeper?

Some Things Are not Patterns

- Laws, theories and conjectures are not patterns
  - Although they may be relevant to a pattern and its presentation
- A neat, one-off and novel design lacks the recurrence of a pattern
  - Although it may well worth be capturing
- Not all idioms are patterns
  - And in fact, not all idioms are idioms — some are merely techniques that lack an idiom's mindshare

Have You Found a Good Pattern?

- This is a harder question to answer, but an important one nonetheless
- Something that recurs is not necessarily good
  - It may be dysfunctional, i.e. it causes problems
  - It may not solve an actual problem effectively, i.e. it may require a lot of workarounds
  - It may be a solution in search of a problem, i.e. it may look neat and compelling, but lack substance
  - It may just be a convention that is, of itself, neither good nor bad, but does not solve a problem

Some Things Are not Good Patterns

- Not everything that re-occurs is necessarily a good pattern...
  - Self-aware class hierarchies where the root refers to its children (cyclic dependencies)
  - Global variables (arbitrary dependencies)
  - Non-*virtual* interfaces in C++ (better solutions are available)
  - The JavaBeans *get* and *set* naming convention (an idiom, but not a pattern)

Can You Find Help?

- Many of the questions you may have are best answered by others
  - Patterns are about communication at many levels
- Would a co-author help or hinder?
  - Co-authoring can help to bring the ideas out, but it can take practice or may not be what is needed
- Do you know a friendly shepherd?
- How about colleagues to give you feedback?
  - Preferably people who have seen the pattern

A Matter of Form

- The pattern form is the written template and style through which you present the pattern
  - Pattern forms can be free and narrative or tightly structured with many standard sections
- The form needs to target the audience and be appropriate for kind of pattern being written
  - So you need to keep the audience in mind
  - Make sure that code doesn't swamp a narrative form or that essential concepts don't get lost and scattered across sections

A Starter Form

- In writing a first pattern, there is a need for structure and guidance
  - Alexandrian and similar forms are typically too difficult for a first pattern
  - However, don't get choked by structure, e.g. the Gang-of-Four form
- A few labelled sections that highlight the essential ingredients
  - Coplien and similar forms

Essential Ingredients

- The problem...
  - *Context*: where does the problem occur?
  - *Problem statement*: a summary of the problem
  - *Forces*: what makes the problem a problem?
- The solution...
  - *Solution statement*: a brief summary of the solution
  - *Solution details*: more detail, perhaps including a diagram, implementation details, etc
  - *Consequences*: the potential benefits and liabilities of applying the solution

The Problem and the Solution

- Context and problem may be better merged
  - The identified context may seem trivial or may restate much of what is in the problem statement
- Consider posing the problem as a question that the solution statement answers
  - They should be readable together
  - Both the problem and solution statements should be short, sweet and sufficient
  - The solution statement should start by saying what the solution is, not what it's about

The Forces and the Consequences

- Each of the forces and consequences can be enumerated as bullets
  - Ensure that there is more than one force and that forces have some conflict with one another — an absence of tension suggests an absence of problem
  - The consequences should answer the forces
  - And the consequences need to deliver more than just good news — if it looks too good to be true, it probably is: design is about trade-offs, so these trade-offs should be presented

An Example (or Two)

- Concrete examples help to motivate a pattern
  - May otherwise appear too abstract, especially if code-centric, but remember the example is not the pattern — all sections must carry their own weight
  - A worked example can bring out attempted solutions that don't work, solution variations, etc
  - Example must be simple but sufficiently realistic
- More than one worked example can help to illustrate variations
  - And encourage readers to see the pattern's breadth

Code and Diagrams

- Code can be used in the context of existing sections if the fragments are brief
  - But additional sections for motivating examples and detailed implementation advice are more appropriate for longer, more fully worked code
- Can you draw something that represents the solution or even the problem?
  - A picture can help to anchor and illustrate the pattern in the reader's mind
  - The picture need not be a UML class diagram

Other Considerations

- There are other features that may play a role in presenting a pattern
  - Noting variations in the solution
  - Discussing alternative and related approaches
  - Mentioning known uses (not particularly useful for common patterns)
- Detailed discussion can be useful if you have more questions than you started with!
  - But it should not dominate or carry the weight of the pattern

A Writing Process

- Get your ideas down
- The problem statement can be tough
  - So consider writing the solution statement first
- Forces are tougher
  - So consider working them after listing the consequences
- Do the sections flow and make sense?
  - Move, split, merge, add and remove as necessary
- Review and revise continually

Ask Yourself...

- Are there forces hiding in other sections?
  - E.g. the problem statement, the consequences, the detailed solution advice
- Am I squeezing too much in here?
  - Is it interesting but outside the pattern's scope?
  - Does this stuff belong elsewhere?
- Are there any other patterns hiding in here?
  - Is this pattern actually the start and beginnings of a pattern language?
  - If so, is that too ambitious for a first pattern?

Beyond the Single Pattern

- Are there more related patterns, recognised or not, written or unwritten?
  - External references, thumbnails, etc, to hint at patterns to look at or patterns to come
- If the implementation advice becomes its own odyssey, consider expressing it with patterns
  - If each practice and consideration within the solution is identifiable, valid and recurring, that is the basis for using the original pattern as the entry point into of a structured community of patterns

On Pattern Communities

- There are many forms of pattern community that a pattern may find itself a part of
  - Pattern catalogues organised by theme or intent
  - Pattern compounds that capture and name common, fairly fixed pattern co-ops
  - Pattern sequences that describe common pattern orderings to achieve a specific end
  - Pattern languages with richer interconnections
- Consider illustrating a pattern community using pattern stories that employ the patterns

On the Patterns Community

- What to do with a pattern once it has been written or when you are writing it?
  - Shepherding is an iterative process of review and revision led by (for first timers) a more experienced pattern author
  - A writers' workshop offers onamous peer review
  - The PLoP family of conferences offer a vehicle and venue for shepherding and workshopping patterns
  - Consider publishing a reviewed pattern (whether online or treeline) to solicit more feedback

In Conclusion

- The good news...
  - In principle, anyone that can describe a problem and recognise a solution can write a pattern
  - In practice, helps to have form, focus and feedback
- The bad news...
  - A pattern is never finished (it's a work in progress)
- The good news (revisited)...
  - A pattern is never finished (it's a work in progress)
  - And you might get someone to continue your work