

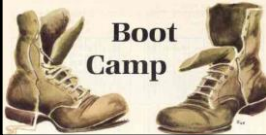
# PLoP® Pattern Writing Bootcamp

Held at Allerton Park, Monticello Illinois



Joseph (Joe) Yoder [joe@refactory.com](mailto:joe@refactory.com)

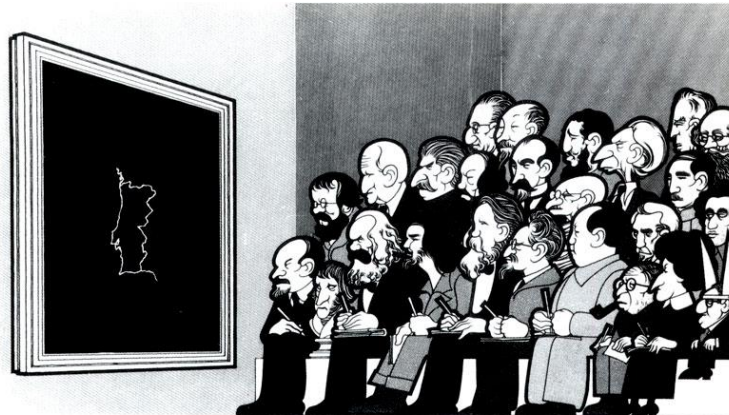
Rebecca Wirfs-Brock [rebecca@wirfs-brock.com](mailto:rebecca@wirfs-brock.com)



PLoP is a registered Trademark of The Hillside Group.

© The Refactory & Joseph W. Yoder  
Wirfs-Brock Associates  
Rebecca Wirfs-Brock  
All Rights Reserved

**A pattern usually focuses on a non-trivial problem in which several things need to be considered**





## Pattern Elements

### **Context**

In which situations  
can I use this pattern?

### **Problem**

What does it try to solve?  
What question does it answer?

### **Solution**

What can I do  
that usually works?



**PROCESS**

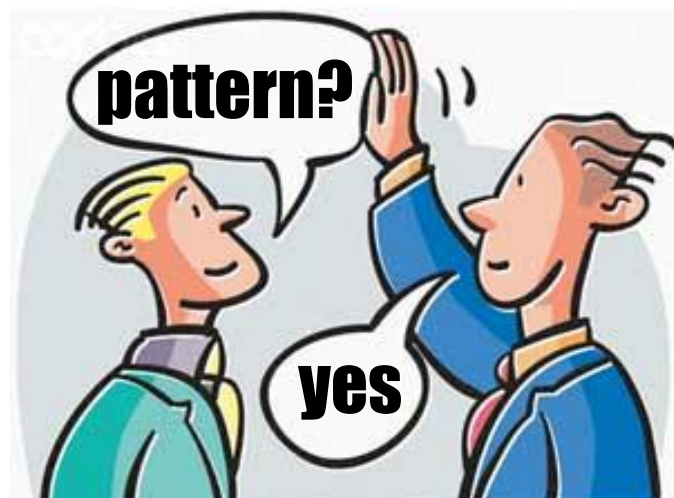
+

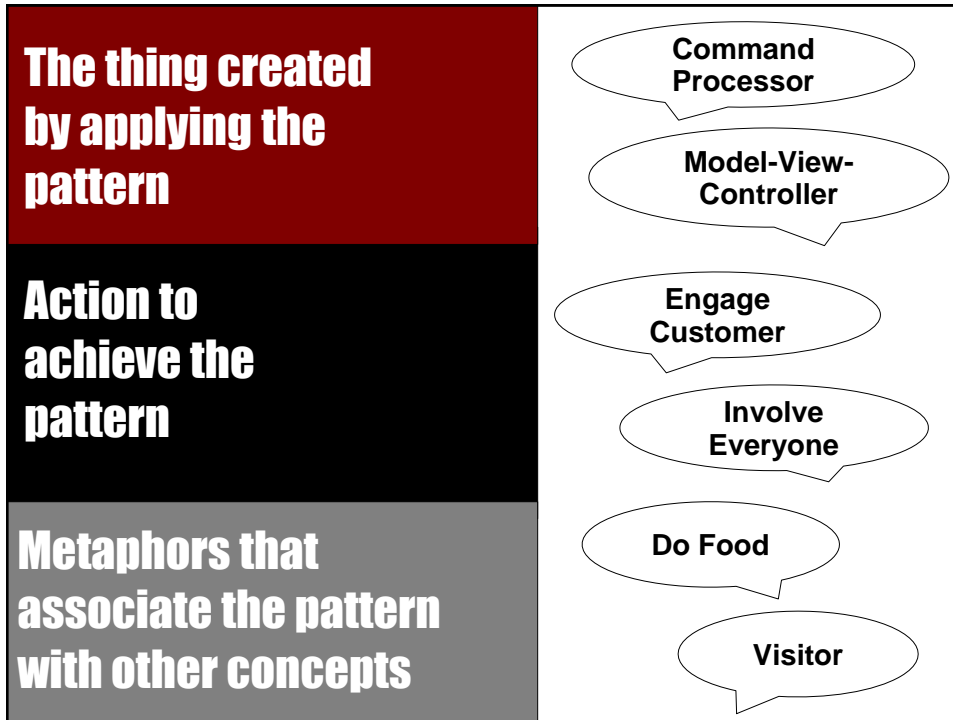


**THING**

The solution describes the final result and ways to achieve it

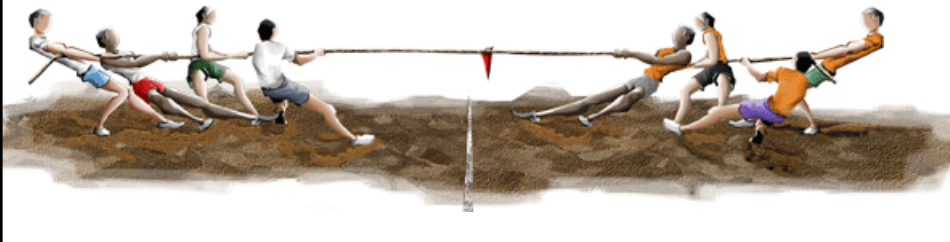
The name of the pattern is very important because it will be part of the user's vocabulary!





**Pattern forces are  
contradictory and pull  
in different directions...**

This **tension** between forces is  
what makes the problem complex!



# Some Forces in Patterns for Sustaining Architectures

- Ongoing Development:* How can you safely rework and evolve the architecture with minimal impact?
- Limited Options for Refactoring:* Why refactor something that is currently released and working? Should you evolve parts of the system and leave code alone?
- Limited Value to Refactoring:* It isn't obvious that refactoring some code will make programming any less tedious or error prone.

## Patterns for Sustaining Architectures

REBECCA WIEFFS-BROCK, *Senior Back-End Architect, IBM*  
 JOSEPH W. YODER, *IBM Software Inc.*

When creating patterns in a field as well-worn as architecture, it's tempting to focus on the most obvious requirements that can be addressed by patterns and describe the pattern accordingly. However, in the case of creating patterns for patterns, it's important to consider the patterns themselves as well as the patterns that can be created by the patterns themselves. This paper presents two patterns for creating patterns in the field of creating patterns for patterns. The first pattern is for creating patterns for patterns and the second pattern is for creating patterns for patterns.

**1.1. Introduction**  
 Software design decisions and architectural structures accumulate. Some parts of a complex software system may have been built and evolved and continue to hold up well after many modifications. However, in most complex systems some parts don't hold up so well. They often, over a great distance, can be categorized by their architectural structure. In this paper, we will make that the architecture that actually provides the structure in the IBM, IBM, or IBM (IBM, or IBM) (IBM, or IBM).

**1.2. Introduction**  
 Software design decisions and architectural structures accumulate. Some parts of a complex software system may have been built and evolved and continue to hold up well after many modifications. However, in most complex systems some parts don't hold up so well. They often, over a great distance, can be categorized by their architectural structure. In this paper, we will make that the architecture that actually provides the structure in the IBM, IBM, or IBM (IBM, or IBM).

**1.3. Introduction**  
 Software design decisions and architectural structures accumulate. Some parts of a complex software system may have been built and evolved and continue to hold up well after many modifications. However, in most complex systems some parts don't hold up so well. They often, over a great distance, can be categorized by their architectural structure. In this paper, we will make that the architecture that actually provides the structure in the IBM, IBM, or IBM (IBM, or IBM).



Explain both positive and negative consequences of applying the pattern



**Benefits**



**Drawbacks**



## Pattern Languages and Generativity

### Generativity

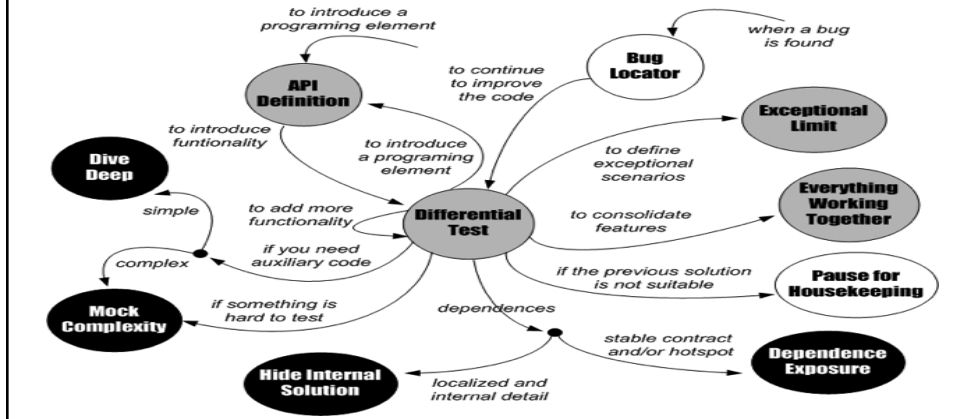
- **Definition:** Having the ability to originate, produce, or procreate (make something new)
- **Wikipedia:**
  - Generativity in essence describes a self-contained system from which its user draws an independent ability to create, generate, or produce new content unique to that system without additional help or input from the system's original creators



# Pattern Languages

Related patterns  
documenting solutions  
for a given domain

“Each pattern then, depends both on the smaller patterns it contains, and on the larger patterns within which it is contained”  
Alexander, TTWOB, p 312



## Collections vs. Languages

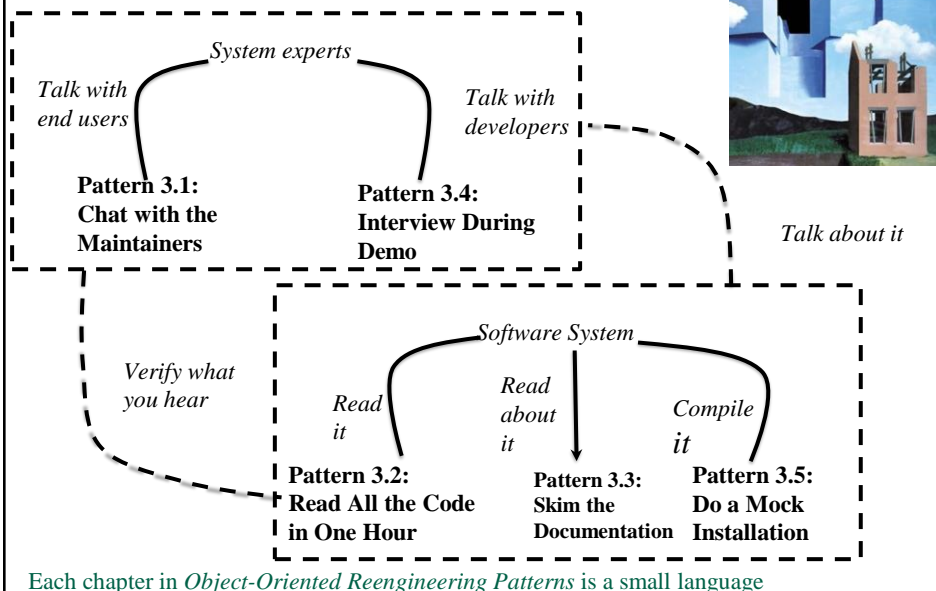
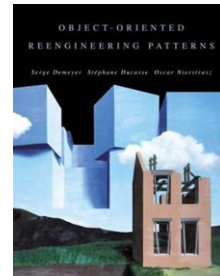
- **Pattern collection**
  - no claim of completeness or deep connections between patterns
  - patterns often are used in isolation
- **Pattern language**
  - include higher level patterns that guide you to more specific, lower-level patterns
  - discusses all the things you need to consider to build something
- **Both have value, languages more complete**

## Parts of a Pattern Language

- **Intent:** A description of the language's intent
- **Maps:** Diagrams showing how patterns build on and relate
- **Context:** An explanation of situations where the pattern language is complete enough to make something significant
- The **patterns** that make up the language
- **Sequences:** Stories/descriptions of how to use several patterns in the language to solve a larger problem

## Example Pattern Languages

### Chapter 3: Initial Contact





## Sequences...



94 patterns within 2 pattern languages  
 Product Organization Pattern Language  
 & Value Stream Pattern Language



Described at ScrumPLoP as the sequence of one of Jeff's most successful teams doing Scrum...

## Understanding Sequences...

Let  $a_1, a_2, a_3, \dots, a_n$  be a sequence of numbers where  $a_1 = -2, a_2 = 4$ , and for  $n \geq 3$ ,  $a_n = \frac{a_{n-1}}{a_{n-2}}$ . What is the sum of the first 99 terms?

(A) -16  
 (B) -12  
 (C) -8  
 (D) -4  
 (E) 0

Handwritten calculations show the sequence terms:  $a_3 = \frac{a_2}{a_1} = \frac{4}{-2} = -2$ ,  $a_4 = \frac{a_3}{a_2} = \frac{-2}{4} = -\frac{1}{2}$ ,  $a_5 = \frac{a_4}{a_3} = \frac{-\frac{1}{2}}{-2} = \frac{1}{4}$ ,  $a_6 = \frac{a_5}{a_4} = \frac{\frac{1}{4}}{-\frac{1}{2}} = -\frac{1}{2}$ ,  $a_7 = \frac{a_6}{a_5} = \frac{-\frac{1}{2}}{\frac{1}{4}} = -2$ ,  $a_8 = \frac{a_7}{a_6} = \frac{-2}{-\frac{1}{2}} = 4$ . The sequence repeats every 6 terms:  $-2, 4, -2, -\frac{1}{2}, \frac{1}{4}, -\frac{1}{2}$ .

Simpler than this

Name \_\_\_\_\_ Date \_\_\_\_\_

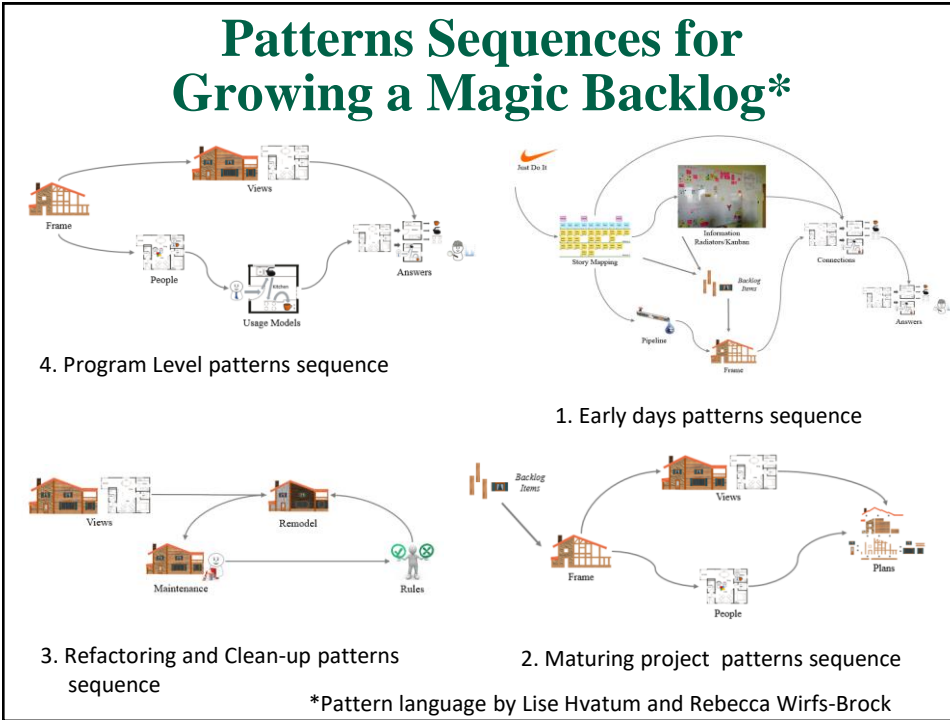
### Sequence Order

Write down the events that took place in the correct order.

First	Then	A little while later.
Next	In the middle of the story	Towards the end.
Then	Afterwards	Finally

Not as linear as this

# Patterns Sequences for Growing a Magic Backlog\*



## Strangler Sequences



### Sequence of Steps for Preparation:

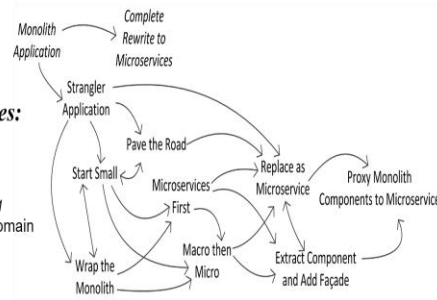
- Decide whether to *Wrap the Monolith* (if so apply standard wrapper patterns)
- Start applying the *Strangler Application*
  - Start Small
  - Pave the Road
  - Microservices First
- Fine tune as needed

### Sequence of Steps for Writing your First Microservices:

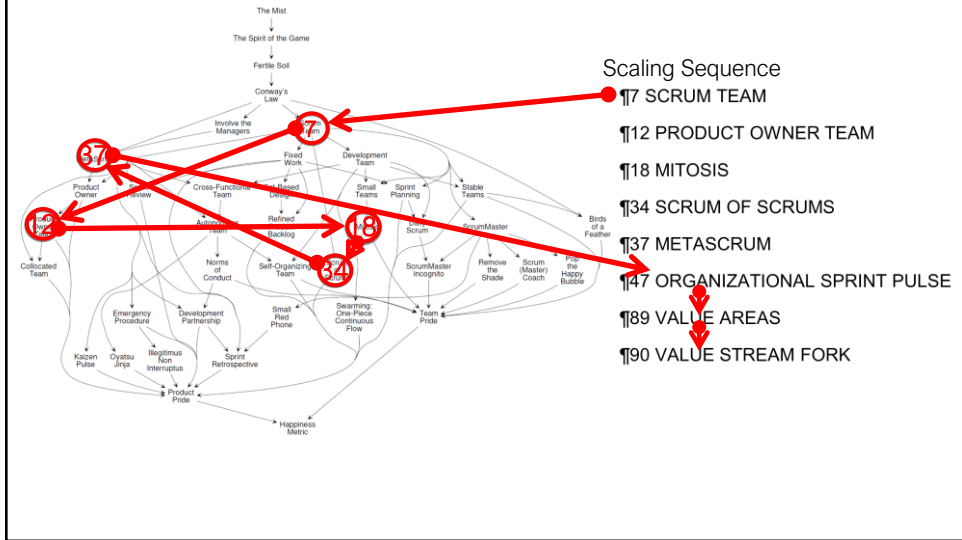
- Start Small (1-2 teams) which may do any of the following
  - Pave the Road (building up infrastructure if needed)
  - Microservices First (write new code as microservices)
- As you write your Microservices, consider any of the following
  - Macro then Micro as you learn microservices and the domain
  - Where possible, *Extract Component and Add Façade*
- Proxy Monolith Components to Microservices if needed
- Repeat (as many times as needed)

### Sequence of Steps for Extracting Functionality to Microservices:

- Decide to rewrite or extract
  - If you can extract, then *Extract Component and Add Façade*
  - For larger components, extract *Macro then Micro*
  - If you have to rewrite, then *Replace as Microservice*
- Proxy Monolith Components to Microservices as needed
- Repeat (as many times as needed)



## Pattern Languages vs Sequences



## Group Exercise



**Let's help Mary Lynn find sequences for the Fearless Change Patterns**



## Pattern Forms and Styles

## Contrasting Poetic Styles

- **SONNET**—a poem of fourteen lines, usually in iambic pentameter that has one of two regular rhyme schemes

From fairest creatures we desire increase,  
 That thereby beauty's rose might never die.  
 But as the ripener should by time decease,  
 His tender heir might bear his memory:  
 But thou, contracted to thine own bright eyes,  
 Feed'st thy light's flame with self-substantial fuel,  
 Making a famine where abundance lies,  
 Thyself thy foe, to thy sweet self too cruel.  
 Thou that art now the world's fresh ornament  
 And only herald to the gaudy spring,  
 Within thine own bud buriest thy content  
 And, tender churl, mak'st waste in niggarding.  
 Pity the world, or else this glutton be,  
 To eat the world's due, by the grave and thee.

-William Shakespeare

## Contrasting Poetic Styles

- **Haiku**—three lines, with the first and last line having 5 moras, and the middle line having 7. A mora is a sound unit, much like a syllable, but is not identical to it. Since the moras do not translate well into English, in English it has been adapted and syllables are used as moras.

Furuike ya  
Kawazu tobikomu  
Mizu no oto

-Basho Matsuo

The old pond,  
A frog jumps in,  
PLoP!

-Translated by: Allan Watts

## Contrasting Poetic Styles

- **Limerick**—a five-line joke of a poem — witty, usually involving place names and puns, and most often bawdy, sometimes unprintable...

There was a small boy of Quebec  
Who was buried in snow to his neck  
When they said, “Are you friz?”  
He replied, “Yes, I is —  
But we don't call this cold in Quebec”

-Rudyard Kipling


# Pattern Styles

- There isn't one "right" way to write a pattern
- Patterns can be written in various forms and forms can be adjusted to your audience
- Major forms:
  - Christopher Alexander style
  - SW Pattern styles:
    - GOF (Design Patterns) style, POSA style, software pattern paper styles
  - Narrative styles (good for non-software patterns)
    - Story-based variation of Alexander's style, Mary Lynn Manns and Linda Rising, *Fearless Change*
    - Pattern 3.0 format, Takashi Iba
    - Rebecca and Joe style

# Pattern vs. Pattern Gists

## Full pattern description

**Find Essential Qualities**  
 "The ability to quickly assess or eliminate the unnecessary or that the necessary may exist." - Steve Yelland



Other other essential system qualities are overlooked or simplified until late in the development process. This can cause delays due to extensive refactoring and rework of the software design in order to correct quality flaws. To avoid extensive rework it is important that agile teams identify these fundamental qualities early in the development process in order to correct quality flaws. To avoid extensive rework it is important that agile teams identify these fundamental qualities early in the development process in order to correct quality flaws.

**How can agile teams understand essential qualities for an evolving system?**

Not having or fundamental qualities early enough can cause significant problems. Early and correct identification of essential qualities can be accomplished by using a workshop or a quality attribute workshop. This workshop should include key members such as the product owner, developers, architects, quality assurance, and the customer. Whenever there are major changes to the roadmap or new system qualities become apparent, the team can choose to hold another quality workshop.

On one hand, it would be good if you could identify and address most of all system qualities in a workshop or a new system quality workshop. However, for example, having something like quality assurance or a quality attribute workshop is not a frequent thing to do in a workshop. It is important to have a workshop or a quality attribute workshop in order to correct quality flaws.

**Workshops** have been meeting or gathering ideas with important stakeholders in order to understand the system and to understand the system. In the past, it was common to have a workshop or a quality attribute workshop in order to understand the system. In the past, it was common to have a workshop or a quality attribute workshop in order to understand the system.

Workshops have been meeting or gathering ideas with important stakeholders in order to understand the system and to understand the system. In the past, it was common to have a workshop or a quality attribute workshop in order to understand the system.

## Pattern gist

**Pattern: Find Essential Qualities**

Quite often essential system qualities are overlooked or simplified until late in the development process. This can cause delays due to extensive refactoring and rework of the software design in order to correct quality flaws. To avoid extensive rework it is important that agile teams identify these fundamental qualities and make those qualities visible to the team in a timely manner.

**How can agile teams understand essential qualities for an evolving system?**

At the start of a project it is important to identify essential qualities critical to the success of the project. This can be done via an agile quality attribute workshop where you agree on essential qualities, and make sure they are visible to team. These workshops should include key members such as the product owner, developers, architects, quality assurance, and the customer. Whenever there are major changes to the roadmap or new system qualities become apparent, the team can choose to hold another quality workshop.

During a quality workshop, which might last an hour or two, simple collaborative techniques can be used to identify and characterize system qualities. People can identify a concern and write it on a sticky note that is associated with a specific system quality (such as performance or reliability). The team can vote on what they consider most important and urgent and then write Agile Quality Scenarios for those.

## Patlets or Thumbnails

Pattern Name	Brief Description
Whole Team	Involve QA early on and make QA part of the whole team.
Quality Focused Sprint	Focus on your software's non-functional qualities by devoting a sprint to measuring and improving one or more of your system's qualities.
Product Quality Champion	Include as part of your agile team a Product Quality Champion. This is someone who helps the team keep focused on important system qualities. This person is involved from the start of the project understanding the customer requirements and continues working throughout assisting the team with a quality focus.
Automate as You Go	Create an environment and use tools to automate fundamental things that add value as soon as you can. Early on the most essential things to automate are the build, integration and test environment configuration.
Agile Quality Specialist	QA provides experience to agile teams by outlining and creating specific test strategies for validating and monitoring important system qualities.
Spread the Quality Workload	Rebalance quality efforts by involving more than just those who are in QA work on quality-related tasks. Another way to spread the work on quality is to include quality-related tasks throughout the project and not just at the end of the project.
Shadow the Quality Expert	Spread expertise about how to think about system qualities or implement quality-related tests and quality-conscious code by having another person spend time working with someone who is highly skilled and knowledgeable about quality assurance on key tasks.
Pair with a Quality Advocate	Have a developer work directly with quality assurance to complete a quality related task that involves programming.

## Group Discussion



**HOW MIGHT YOU ADJUST THE FORM YOU ARE USING TO WRITE YOUR PATTERNS IF YOU HAD A DIFFERENT AUDIENCE?**



## Shepherding, Giving, and Responding to Advice

### The Role of A Shepherd: Guide, Supporter, Mentor

#### Reads, reviews, advises

- An *experienced* pattern writer
- Works with the author to improve their paper
  - Offers advice
  - Has opinions
  - Suggests improvements
  - Guides the author
- Offers encouragement!





## Shepherds and Reviewers are Advocates

- Supports the author
- Doesn't have a personal agenda
- Generous, unselfish
- Works on behalf of author to make them be the best they can be



## The Role of a Pattern Author: Effectively Share Wisdom

### **Writes, thinks, rewrites**

- Conceives of the pattern
- Writes the pattern
- Agrees to be “shepherded”
- Receives and responds to advice
- Asks for help
- Asks clarifying questions
- Rewrites



## Focus on Your Reader



## Improve Your Writing

- Share with others
- Write clearly
  - Shorten long sentences
  - Eliminate extra words
  - Summarize main points
- Practice. Revise.
  - understandable problem
  - what are the forces
  - clear tradeoffs



## How do I revise my patterns?

- What do I do with shepherds comments?
- What do I do after a writers' workshop?
- How much revision should I do?
- When should I stop?

## Giving Advice: Consider Its Impact



### Adapt a “triage” approach

- Read and note issues/ questions/concerns
- Spend time wisely:
  - Address most important things first
  - Note minor issues
  - Don't spend time on hopeless causes (if it is really, really bad)
- Be sure to mention what you like, too!


## Types of Advice

- **Valid**—There is clearly a problem I should address.  
“Your pattern is missing some forces”
- **Invalid**—The comment is not useful or valid.
- **Judgmental**—The reviewer liked or disliked something.  
“The name of your pattern doesn’t reflect the solution”
- **Complexity**—The reviewer thinks I should express my ideas more simply or that my solution is too complex.  
“Reword your solution and include a diagram to help explain”
- **Aesthetics**—A comment about the form, not the substance of my pattern. “Can you add a rating to all your patterns?”
- **Praise**—The reviewer is happy. “This is much improved.”

## Responding To Constructive Advice

- You are in control of what to do with advice.  
Here’s how you might react and respond:
  - **Valid**—You need to improve you pattern. You may need to ask some questions before making changes.
  - **Invalid**—Most likely you ignore the comment (unless the shepherd or reviewer thought it was very important).
  - **Judgmental**—Ask them why they think that.
  - **Complexity**—May want to explain more clearly...or break down a complicated solution into alternatives or...maybe there is more than one pattern here.
  - **Aesthetics**—You decide whether to fix or ignore “style” issues.
  - **Praise**—You may want to know why they are so happy.  
“What do you like about my new solution?”

## Open and Closed Questions



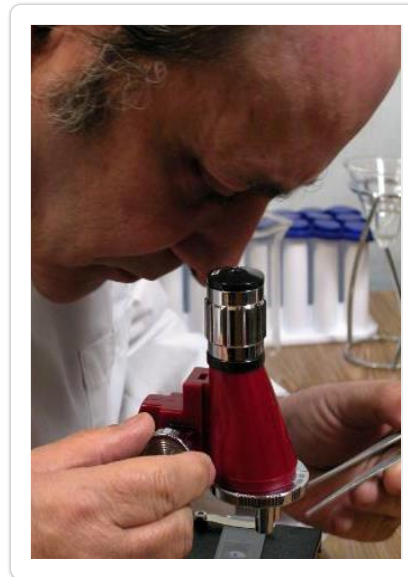
The same question can be asked differently...

**Open**—can't be answered yes or no. It invites conversation.  
“What do you like about flying?”

**Closed**—Answered with a simple yes or no. (Most likely no). Sometimes used to intentionally shut off discussion.  
“Do you like flying?”

## Probing Questions

- Evaluation...how good do you think it will be?
- Accuracy...how did you come up with those numbers?
- Completeness... is that all...?
- Relevance...does this apply here?
- Purpose...why did you suggest that?
- Extension...tell me more..



## Clarifying Questions

- Get others to think:
  - Why do you say that?
  - What exactly do you mean?
  - Can you give me an example?
  - Are you saying ... or ... ?
  - Can you restate your concern?



### Group Exercise



**BREAK UP INTO GROUPS AND  
COME UP WITH SOME ADVICE FOR  
IMPROVING WAKEUP CALL PATTERN**

## Writers' Workshops at PLoP



### Writers' Workshops & the Work of Making Things...Richard Gabriel

A circle of interested colleagues, led by a strong, neutral moderator, that provides feedback to the author on how the pattern is understood by the group and suggestions for improvement

#### Roles:

- Author
- Moderator/Leader
- Summarizer
- Sympathetic Participants

Participants read the pattern before the workshop



**The author stands, reads a selection from the pattern, then becomes a “fly on the wall,” outside the circle. No eye contact is made. The author’s name is never mentioned; use “the author”**

<https://www.dreamsongs.com/Files/Shepherding.pdf>  
<http://www.dreamsongs.com/Files/WritersWorkshop.pdf>



## Reviewing and Shepherding are Gifts

*“If someone gives you something ... pass it on”*

- Comments and suggestions are gifts
- Giving these gifts does not mean that your author always is ready to accept it
  - Some may not want your gift because they are looking to validate their ideas, not respond to questions or comments
  - Some may misinterpret your gift because they don't understand what you are saying
  - Some may not be prepared for your gift because they are focused on other concerns
  - Some may not be able to accept it because the time is not right



## Hillside and Patterns Community



## PLoP Community Culture

“the integrated pattern of human knowledge, belief, and behavior that depends upon man’s capacity for learning and transmitting knowledge to succeeding generations”

“the customary beliefs, social forms, and material traits of a religious, or social group”

“the set of shared attitudes, values, goals, and practices that characterizes a company or corporation”

Merriam Webster Collegiate Dictionary  
on the web <http://www.m-w.com/dictionary.htm>

- Shared experiences
- Shared rituals
  - Writers’ Workshop
  - Shepherding
  - Gifting and Games







## Patterns Community Gatherings

- Pattern Conferences

- **PLoP®** since 1994 at Allerton House, Monticello, IL\*

Except: 2006: OOPSLA in Portland, OR, 2008: OOPSLA in Nashville, TN  
 2009: AGILE in Chicago, IL, 2010: Splash in Reno, NV  
 2011: Splash in Portland, OR, 2012: Tucson Arizona,  
 2015: Pittsburgh, PA, 2017 Vancouver Canada, 2018: Portland, OR

- Euro PLoP® since 1996 at Kloster Irsee, Bavaria
- SugarLoaf PLoP® since 2001-2015 Brazil, 2016-17 Argentina
- Viking PLoP® since 2002 rotating among Scandinavian countries
- AsianPloP® 2010, 2011, 2014, 2015, 2017, 2018 in Tokyo Japan, 2016 Taipei Taiwan
- ScrumPloP® 2009-2014 in Scandinavian countries, 2015-2018 Douro Valley, Portugal
- Chili PLoP® “A different Kind of PLoP”, near Phoenix, AZ 1998-2012
- ParaPloP® Parallel programming patterns, 2009, 2010, 2011
- Koala PLoP® 2000-2002 in Melbourne, Australia
- Mensore PLoP® 2001 Okinawa, Japan
- MetaPloP® 2011 in Douro Valley, Portugal
- UP 1998 Mohonk Mountain House, New Paltz, NY

For more information visit <http://hillside.net/conferences>

- **Transactions on Pattern Languages of Programming**

- The new peer reviewed pattern journal.
- Published by Springer
- More information: <http://hillside.net/tplop>

**PURPLSOC**  
 Pursuit of Pattern Languages for Societal Change

**EduPloP® 2015-16**  
 Schermerhorn, Netherlands





THE  
HILLSIDE GROUP

