Pattern Languages

• How can we organize what we know about design? … So that it provides useful guidance in design work?
• Design guidelines – Abstract apple pie, hard to instantiate
• Examples of good results – Often depend on application details, hard to generalize
• Design theories?

Origins of pattern languages

• Christopher Alexander – Notes on the synthesis of form (1964) – “new design methods” - Reitman, Rittel, Jones, Simon
• A Pattern Language (1979) – names for 253 design archetypes, at various scales
• actual impact in architecture somewhat in doubt!
• 4. Guard Clause

• Indented Control Flow (2) and Guard Clause (1) produce methods where indentation communicates the gross structure of the method.

• How should you format code which shouldn’t execute if a condition holds?

• In the bad old days of Fortran programming, when it was possible to have multiple entries and exits to a single routine, tracing the flow of control was a nightmare. Which statements in a routine got executed when was impossible to determine statically. This lead to the commandment “Every routine shall have one entry and one exit.”

• Smalltalk labors under few of the same constraints of long ago Fortran, but the prohibition against multiple exits persists. When routines are only a few lines long, understanding flow of control within a routine is simple, it is the flow between routines that becomes the legitimate focus of attention. Multiple returns can simplify the formatting of code, particularly conditionals. What’s more, the multiple return version of a method is often a more direct expression of the programmer’s intent.

• Therefore, format conditionals which prevent the execution of the rest of a method with a return. Let’s say you have a method which connects a communication device only if the device isn’t already connected. The single exit version of the method might be:

``` Smalltalk
connect
  self isConnected
        ifFalse: [self connectConnection]
```

• You can read this as “If I am not already connected, connect my connection.” The guard clause version of the same method is:

``` Smalltalk
connect
  self isConnected ifTrue: [^self].
  self connectConnection
```

• You can read this as “Don’t do anything if I am connected. Connect my connection.” The guard clause is more a statement of fact, or an invariant, than a path of control to be followed.

---

**Example: Street Café**

• **example:** picture, description of experience

• **relationship to larger/smaller scale patterns**
  – larger scale - Identifiable Neighborhood, Small Public Squares, Activity Nodes, ...
  – smaller scale - Opening to the Street, Place to Wait, Different Chairs, Stair Seats, Canvas Roof, ...

• **rationale** (viz the experience of the space)
  – low-key social mixing, safe to relax, nurse a beer, ...

• **implementation**
  – specializing lower-level patterns (wide Opening to the Street, make terrace a Place to Wait, …)
How you use a Pattern Language
(following Alexander)

• “Spreading activation” in a pattern network
  – select best pattern
  – identify all patterns that support that pattern, all
    patterns that support each of them, …
  – identify all patterns supported by any selected
    pattern (and patterns they support, etc.)
  – modify any pattern as appropriate

• *Not* rules to be followed
• *Not* templates to be “instantiated”

How pattern languages can help

• Capture collective wisdom
  – Combines design rationale with suggested actions
  – Can grow by specialization for contexts
• *lingua franca* for design
  – standardize terms and concepts
  – ease participation of clients in design
• Reuse and extend ideas instead of just copy
  – Organize solutions to common problems
  – “think outside the toolkit” -- slightly higher level of
    abstraction than particular widgets
• Converge on designs that *work* for people
  – “the quality without a name”
Elements of typical pattern schema

- **Examples**: of pattern in application
- **Context**: in which pattern is employed
- **Problem**: that pattern solves or addresses
- **Forces**: requirements/tradeoffs that constrain possible solutions
- **Solution**: the pattern
- **Resulting context**: having employed the pattern, what do you need to do now?
- **Notes**: any useful annotations

Choice from a small set

(from Tidwell)

- **Examples**: Set of radio buttons; Combo box (drop-down list); "Circle one: Mr / Mrs / Ms" on a paper form
- **Context**: Display/set a value out of a small set of possible values (10 or fewer).
- **Problem**: How should the artifact indicate what kind of information should be supplied?
- **Forces**:
  - User should see all the possible values, to put the actual value in context.
  - If the user needs to set the value (not just look at it), they should know what choices are available.
  - Small numbers of things can be taken stock of quickly, and don't take up much space.
**Solution:** Show all possible choices up front, show clearly which choice(s) have been made, indicate clearly whether one or several values can be chosen. Provide a choice for "Other" or "None of the above," if that will ever be an issue -- never prevent the user from providing correct information.

**Resulting Context:** Good Defaults may let the user look at the default value, judge it to be OK, and move on without even bothering to set the value. If the choices are pictorial, or are cryptic in some other way, Short Description may be needed to describe the choices further.

With physical or electronic artifacts, a single selection can be enforced by causing the previous choice to "unselect" when a choice is made. Old car radios and GUI radio boxes do this.

**Notes:** For a small number of choices, it is often pointless to hide choices, as in a combo box -- if it won't cost a huge amount of space; show all the possibilities so the user can see them without going through an extra step to reveal them.
Patterns as a design language

- concrete prototypes vs abstract principles
  - more accessible (but harder to generalize?)
- socially grounded
  - emphasize human activity and experience
- incorporate values
  - Individually-Owned Shops, Bike Paths and Racks,
- supports piecemeal adoption & use
  - don’t have to adopt/use the whole framework, inherently open-ended

Community-based Information Technology Workforce Development

- Problem:
  Digital divide in workforce development
- Solution:
  Enhance and utilize community networks viz. informal learning-by-doing
- Solve the problem of declining social capital as a side effect
Context

• Information technology is a key long-term opportunity area for jobs
  – but need technical skills to be able to participate
• People are being left behind educationally
  – women, children, the elderly, economically disadvantaged, ethnic/racial minority groups
  – the “digital divide”
• The web is world-wide
  – but people relate to it as consumers/observers, not producers/participants

Problem

• How can we help to open the information technology workforce to all persons?
  – adults in the workforce, but who lack IT skills
  – adults not in the workforce
Forces

• Learning of skills and motivation to apply skills is increased …
  √ when people are allowed to solve real problems
  √ when they are given control and responsibility
  √ when they have the opportunity to collaborate
• Adult-education tends to be managed through formal programs
  – teach information technology through individual problem-solving with generic & abstract projects

Forces (continued)

• Some information technology skill is now required in almost every job
• People not in the workforce
  – are often more engaged in their local community than the fully employed (who work outside the community)
Solution

• Leverage and enhance community networking infrastructures
  – provide leading-edge tools and environments
  – emphasize collaboration and end-user authoring

• Seed and facilitate model projects
  – exploit “interaction beats publication” and “pockets of innovation” patterns
  – Struble’s Creek, B’burg Nostalgia, teacher KM

• Work with regional workforce development agencies

members of community groups learn IT skills through work on meaningful community projects

they receive job and career counseling

they find employment in IT jobs
Resulting context

• Recast social service agencies as facilitators of local innovation rather than implementers of state/federal mandates
  – Service provision saps initiative on both sides
• Move skills and innovation downwards and outwards in the social power hierarchy
• Strengthen proximal community social structures (aka build social capital)

Notes

• This pattern does not imply that formal training programs for IT workforce development are bad
  – top-down complements bottom-up
• Identifying the “right” community groups, grassroots projects, and workforce development administrators is critical, and poorly specified