

# Giving “The Quality” a Name<sup>1,2</sup>

## Precise Specification of Design Patterns: A Second Look at the Manuscripts

Amnon H. Eden<sup>3,4</sup>

### Abstract

*We discuss the prospects of precise specification of design patterns. We analyze the manuscript of the GoF patterns catalog, and prove that the essence of many design patterns can be preserved through the translation to some formal language.*

### Contemporary Means of Specification

There is widespread interest in design patterns [GoF 95; Coplien & Schmidt 95; Vlissides, Coplien & Kerth 96; Buschmann et. al 96; Pree 94; Martin, Riehle & Buschmann 97], and in the “pattern” form as means of conveying expertise in software design. In particular, the universal applicability of the constructs listed in the GoF [95] catalog took over the industry’s attention and grew to serve as building blocks in the documentation, reasoning, and construction of object oriented software.

Design patterns convey regularities, plans, aspects, or abstractions of programs rather than concrete instances. Therefore most design patterns have each an unbounded number of implementations, usually in various programming languages and numerous application domains.

In lack of a dedicated specification language, design patterns are invariably communicated through a list of prototypical instances, source code of simplified implementations, and class (or object) diagrams thereof. As specific instances of design patterns cannot account for the generalizations intended, specification manuscripts consist mostly of natural language narrative. As a result, “clients” of design patterns are compelled to project from the given examples or to interpret the designation of the verbal description that should apply to their context of interest.

To summarize, the prevalent means of their specification are not sufficiently rigorous to allow unambiguous interpretation.

---

<sup>1</sup> [http://www.math.tau.ac.il/~eden/bibliography.html#giving\\_the\\_quality\\_a\\_name](http://www.math.tau.ac.il/~eden/bibliography.html#giving_the_quality_a_name)

<sup>2</sup> Guest column, June 1998, *Journal of Object Oriented Programming*. SIGS Publications.

<sup>3</sup> Supported in part by a grant from the German-Israeli Foundation for Scientific Research and Development (GIF) and the Israel Ministry of Science and Arts.

<sup>4</sup> Department of Computer Science, Schriber School of Mathematics, Tel Aviv University.

## The Need for Precise Specification

One who follows postings to the pattern discussion mailing lists will find that even attentive readers of the GoF's [95] highly detailed descriptions cannot tell, in more than one case<sup>5</sup>, which design pattern (if any) portrays a certain aspect of a particular Java (C++, Smalltalk, etc.) program. This is not surprising, as verbal descriptions are inherently ambiguous, and induction (namely, generalizations based on a subset of concrete instances) leads to infinite directions.

If, however, the underlying abstractions in pattern catalogs are to become fundamental elements of software construction they must mature at least to have a definite and unambiguous specification.

Furthermore, in the absence of a suitable, dedicated pattern specification language, design patterns were never entirely codified, and no attempt towards putting order into their interactions has ever gone beyond "this pattern *uses* that pattern", or "these two patterns often *occur simultaneously*". Results of such inquiries take, at most, the form of rudimentary diagrams which shed little light on the associations between patterns ("Organizing the Catalog" [GoF 95 pp. 9-11]; "Design Patterns Relationship" [GoF 95 pp. 11-13]), or crude and fuzzy narratives ("Relationships Between Design Patterns" [Zimmer 95]; "A System of Patterns" [Buschmann & Meunier 95]; "Relationships Between Patterns" [Buschmann et. al 96 pp. 16-19]).

In contrast, much benefit can be gained if formal means of reasoning on design patterns are provided and relations between were conclusively established (such as  $P_1$  is a special case of  $P_2$ ), and so forth. See for instance the confusion resulted from the inability to conclusively pin down the difference between the patterns *MULTICAST* and *OBSERVER* [Vlissides 97a, 97b]. Compare that debate, that is made by verbal means, to the formal definition of *refinement* as logic implication and the overlapping between the respective LePUS diagrams [Eden, Hirshfeld & Yehudai 98c].

Moreover, defining precisely software design patterns is a prerequisite for allowing tool support in their implementation. To summarize, coherent specifications of patterns are essential to improve their comprehension, to allow formal reasoning about their properties and relationship, and to support and automate their application.

## Precise Specification Languages

A number of publications attempted towards more precise means of design patterns' specifications, such as dedicated programming languages or extensions to existing OOP languages [Bosch 96; Alencar, Cowan, & Lucena 96], or by mandating constraints on their implementation [Helm, Holland, & Gangopadhyay 90; Hedin 97; Pal 95; Klarlund, Koistinen, & Schwartzbach 96]. Other research focused on tool support [Budinsky, Finnie, Vlissides, & Yu 96; Quintessoft 97; Florijn, Meijers, & van Winsen 97; Eden, Gil, & Yehudai 97c].

We should note that the research in supporting tools and specification languages focuses invariably in the solution segment of design patterns; i.e., they aim to capture the abstractions behind all possible conforming *implementations*. For instance, specification languages that evolved from the investigation of the GoF [95] catalog directed at restating the information included in the *structure*, *participants*, and *collaborations* sections of each article.

In order to distinguish these underlying abstractions of the solutions from the remaining designations of the term "design patterns" we denoted them as *lattices*. Detailed justifications to the universal focus in lattices rather than in the remaining elements of the description appear in [Eden &

---

<sup>5</sup> Notably, neither their authors agree among themselves.

Yehudai 97b]. For practical reasons we hereby shall hold on to this approach and our analysis applies to the verbal specification of the lattices of design patterns.

### **A “Quality Without A Name”?**

In contrast with the search for a precise specification language, trends within the patterns community do not see favorably the attempts to subject design patterns to scientific analysis. This school of thought holds a reactionary view of patterns which adheres to the *pattern* concept as originally conceived by Alexander [77; 79]. By this view, a pattern is an idea, an element of a language, and a quasicorporeal concept whose essence is intangible, elusive, and hence beyond the scope of a literal expression. A “good” pattern departs from mere micro-architectural prescription by some immaterial quality that cannot be explicitly expressed, a “quality without a name”<sup>6</sup>, and therefore cannot be interpreted outside its context or taken apart. Thus, “going meta”, as the reflexive treatment of design patterns which follows the analytic tradition’s course of is connoted, is meaningless<sup>7</sup>.

Others do not treat patterns as sacred cows. The rational school considers the actual and potential contribution of design patterns too valuable for them to be referred to as heavenly bodies. Much benefit can be gained from an analytic approach, namely, by “dissecting” the proposed solutions and reducing them to building blocks of higher resolution, thereby allowing systematic reconstruction of new design patterns, formal treatment of their relationship, and tools that support their application.

### **Second Look at the Manuscripts**

Experimental specification languages and tools for design patterns form hypotheses that, by their very nature, achieve varying degrees of success. As experiments, their results need all be compared to a common, conventional benchmark. Yet it is the very lack of a fixed interpretation that motivated this research in the first place.

In our own “experiments” in precise specification languages [Eden, Gil, & Yehudai 97c; Eden, Hirshfeld, & Yehudai 98a], we studied the manuscripts of “Design Patterns: Elements of Reusable Object Oriented Software” [GoF 95], the book which forms a *de facto* canon of the genre. The GoF means of specifications include verbal descriptions mixed with technical jargon, source code clips, and OMT [Rumbaugh et. al 91] class and object diagrams. Evidently the GoF made a consistent effort to render their descriptions as definite and conclusive as possible without losing generality.

We concluded, however, that despite the systematic effort made to render the description accurate, the results in this respect are inconsistent. The problem is, one may deduce, with the specification language, or rather, the lack of a suitable one.

Here we take a step back and focus on a more basic issue, and ask: To what extent can such specifications be understood precisely? Does a definite interpretation of the text exist at all? And how much of the verbal specifications indeed have unambiguous interpretation?

As the first step towards common, fixed norms, which can serve as mutual yardstick in comparing different specification languages, we hereby deliver the results of an analysis of the patterns’ specification text.

---

<sup>6</sup> Jim Coplien’s “The Column Without a Name” published in *C++ Report* is named after this concept.

<sup>7</sup> In accordance with the elusive nature of design patterns, the very sentiment of the irrational school were never made in a form that is more explicit than the PLoP’s conference’s policy, and therefore cannot be recited as necessary.

## Degrees of Precision

Studying the [GoF 95], we identified six categories of verbal descriptions with respect to precision and formality. The results of our analysis are summarized in Table 1.

The first three categories comprise the relatively “precise” statements made throughout the verbal descriptions. One may note by the examples that “precision” does not necessarily indicate a singular implementation in every conceivable O-O programming language. Detailed implementation instructions are relatively infrequent and accomplished almost exclusively by quoting the implementation itself in one language of choice.

A verbal specification is considered “precise” if it has a ‘compact’ *set* of interpretations in sev-

<i>Interpretation Category</i>	<i>Examples</i> (extracts from [GoF 95])
1. Precise, singular	<ul style="list-style-type: none"> <li>• (<i>DECORATOR</i>) “maintains a reference to a Component object”</li> <li>• (<i>VISITOR</i>’s ConcreteElement) “implements an Accept operation that takes a visitor as an argument”</li> </ul>
2. Enumerated alternatives	<ul style="list-style-type: none"> <li>• (<i>FACTORY METHOD</i>’s Creator) “may call the factory method to create a Product object”</li> <li>• (<i>STRATEGY</i>, Collaborations) “Alternatively, the context can pass itself as an argument to Strategy operations”</li> <li>• (<i>DECORATOR</i>, Collaborations) “It may optionally perform additional operations before and after forwarding the operations.”</li> </ul>
3. Precise generalization	<ul style="list-style-type: none"> <li>• (<i>VISITOR</i>) “declares a Visit operation for each class of ConcreteElement in the object structure”</li> <li>• (<i>DECORATOR</i>) “defines an interface that conforms to Component’s interface”</li> </ul>
4. Technical terms, yet open to various interpretations	<ul style="list-style-type: none"> <li>• (<i>PROXY</i>) “Virtual proxies ... <u>cache</u> additional information about the real subject so they can postpone accessing to it”</li> <li>• (<i>PROTOTYPE</i>) “implements an operation for <u>cloning</u> itself”</li> <li>• (<i>MEMENTO</i>) “stores <u>internal state</u> of the Originator object.”</li> </ul>
5. Fuzzy, informal, or teleologic description	<ul style="list-style-type: none"> <li>• (<i>ADAPTOR</i>’s Adaptee) “defines an existing interface that <u>needs adapting</u>”</li> <li>• (<i>COMPOSITE</i>’s Component) “implements default behavior for the interface common to all classes, <u>as appropriate</u>”</li> <li>• (<i>OBSERVER</i>’s Collaborations) “ConcreteObserver uses this information to <u>reconcile</u> its state with that of the subject.”</li> </ul>
6. Deliberate omission of detail	<ul style="list-style-type: none"> <li>• (<i>STATE</i>) “The State pattern does not specify which participant defines the criteria for state transitions”</li> </ul>

Table 1: Categories in the GoF specification of lattices. Quotations are taken from [GoF 95]

eral ‘conventional’ O-O programming languages. This “definition” is illustrated in Table 2 using the examples of Table 1.

<i>Verbal Description</i>	<i>Conforming Implementations</i>
<i>pass itself as an argument</i>	<ul style="list-style-type: none"> <li>• <code>aStrategy.algorithm(*this)</code></li> <li>• <code>aStrategy algorithm: self</code></li> </ul>
<i>forwarding the operation</i>	<ul style="list-style-type: none"> <li>• <code>operation(arg: ARGUMENT): RETURN is</code>  <code>do</code>  <code>    Result := the_component.operation(arg);</code>  <code>    Current.register;</code>  <code>end</code></li> </ul>
<i>a conforming interface</i>	<ul style="list-style-type: none"> <li>• <code>Base &amp; Component::operation(Argument) const;</code>  <code>Derived &amp; Decorator::operation(Argument) const;</code></li> <li>• <code>Component&gt;&gt;operation:</code>  <code>Decorator&gt;&gt;operation:</code></li> </ul>

Table 2: Descriptions vs. conforming implementations

## Empirical Results

Our analysis indicated the following. In the majority of GoF patterns, large parts of the specifications of the lattice of each are “precise”, that is, descriptions which fall under categories 1, 2, and 3 of Table 1.

As of the remaining text extracts we reached the following conclusions, ordered by their categories:

**Category 4** (technical terms): If infinite interpretations exist, the formalization effort can focus on representing well-defined subsets of interpretations, and declare each as a separate pattern.

For example, we have encountered this difficulty with tool support for the application of the *PROXY* pattern. The pattern’s specification states that “Virtual proxies ... cache additional information about the real subject”. The difficulty was with the term “cache”, as whether a particular object “caches information” about another depends on interpretation and context.

As a solution, we distinguished a particular implementation of the *PROXY* with a limited and self-defined interpretation (in our specification language) of “information caching” [Eden, Gil & Yehudai 97c]

**Category 5** (teleologic and fuzzy specifications): At this point and we consider such sentences as impossible to be rendered precise by “simple” means.

**Category 6** (deliberate omission): Details are omitted because patterns are abstractions; these are, in fact, *generalizations* of implementations of an unspecified nature, thereby similar to excerpts of category 3. We conclude that expressions in the pattern specification language should account not only for the well-defined generalizations (“an inheritance class hierarchy”), but also allow implementations to vary considerably without telling how (“which participant defines the criteria for state transitions”).

## Conclusions

The results of our analysis lead us to conclude that prospective success of research towards precise specifications that faithfully convey the specification of design patterns' lattices depend primarily on the following factors:

**Category 4** (technical terms): Successful discrimination of useful subsets of interpretations of such terms within the limits of the specification language

**Category 5** (teleologic and fuzzy specifications): The frequency of such specifications

**Category 6** (deliberate omission): The power of the specification language to deliver expressions that accurately and correctly account for such generalizations

As the actual frequency of teleologic specifications (category 5) is kept low throughout the GoF catalog, we reach the following optimistic conclusion: At least in theory, we can determine that the essence of lattices of many design patterns can be preserved through the translation to some formal language.

## The Next Step

Our analysis delineates the prerequisites of a formal specification language and the kind of generalizations it should convey. The basic abstractions in a formal specification language should treat "precise" specifications at the level defined above, namely, each standing for a "compact" set of interpretations in conventional O-O programming languages.

We took the next step and defined such a language. LePUS [Eden, Hirshfeld & Yehudai 98a, 98c] is a small subset of higher order monadic logic with natural graphic representation, which abstracts "singular" specifications (category 1) as atomic relations and entities in a logic model (also *structure*). Design patterns are translated to predicates in LePUS that restrict programs to those that conform to the specifications of the pattern.

Phrasing most of GoF patterns in LePUS has helped us to resolve the relationship between design patterns and to investigate tool support in their implementation.

## References

- Alencar P. S. C., D. D. Cowan, C. J. P. Lucena (1996). "A Formal Approach to Architectural Design Patterns". *Proceedings of the 3rd International Symposium of Formal Methods Europe*, pp. 576-594.
- Alexander, C., S. Ishikawa, M. Silverstein, M. Jacobson, I. Fixdahl-King, S. Angel (1977). *A Pattern Language*. Oxford University Press, New York.
- Alexander C. (1979). *The Timeless Way of Building*. Oxford University Press, New York.
- Bosch J. (1996). *Language Support for Design Patterns*. Proceedings TOOLS Europe '96.
- Budinsky F. J., M. A. Finnie, J. M. Vlissides, P. S. Yu (1996). "Automatic code generation from design patterns". *Object Technology* Vol. 35, No. 2.
- Buschmann F. R. Meunier. *A System of Patterns*. In: Coplien J. O., D. C. Schmidt (1995, eds.) *Pattern Languages of Program Design*. Addison Wesley.

- Buschmann F., R. Meunier, H. Rohnert, P. Sommerlad, M. Stal (1996). *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons.
- Eden A. H., A. Yehudai (1997b). *Patterns of the Agenda*. In: Bosch J. and S. Mitchell (1997, eds.) *Object-Oriented Technology - ECOOP'97 Workshop Reader*. Lecture Notes in Computer Science no. 1357. Berlin: Springer-Verlag.  
<http://www.math.tau.ac.il/~eden/bibliography.html#ecoop97>
- Eden A. H., J. Gil, A. Yehudai (1997c). "Precise Specification and Automatic Application of Design Patterns". *The Twelfth IEEE International Automated Software Engineering Conference - ASE 1997*. IEEE Computer Society.  
<http://www.math.tau.ac.il/~eden/bibliography.html#ase>
- Eden A. H., Y. Hirshfeld, A. Yehudai (1998a). "LePUS - A Declarative Pattern Specification Language". Technical report 326/98, department of computer science, Tel Aviv University. <http://www.math.tau.ac.il/~eden/bibliography.html#lepust>
- Eden A. H., Y. Hirshfeld, A. Yehudai (1998b). "Precise Notation for Design Patterns". Submitted: *Journal of Object Oriented Programming*. SIGS Publications. [http://www.math.tau.ac.il/~eden/bibliography.html#precise\\_notation\\_for\\_design\\_patterns](http://www.math.tau.ac.il/~eden/bibliography.html#precise_notation_for_design_patterns)
- Eden A. H., Y. Hirshfeld, A. Yehudai (1998c). "Multicast - Observer  $\neq$  Typed Message". Submitted: *C++ Report*, SIGS Publications. [http://www.math.tau.ac.il/~eden/bibliography.html#multicast\\_observer\\_typed\\_message](http://www.math.tau.ac.il/~eden/bibliography.html#multicast_observer_typed_message)
- Florijn G., M. Meijers, P. van Winsen (1997). *Tool Support in Design Patterns*. In: Askit M., S. Matsuoka (1997, eds.) *Proceedings of the 11<sup>th</sup> European conference on Object Oriented Programming - ECOOP'97*. Lecture Notes in Computer Science, Vol. 1241, Berlin: Springer-Verlag.
- GoF 95: Gamma E., R. Helm, R. Johnson, J. Vlissides (1995). *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.
- Hedin G. (1997). *Language Support for Design Patterns using Attribute Extensions*. In: Bosch J., S. Mitchell (1997, eds.) *Object-Oriented Technology - ECOOP'97 Workshop Reader*. Lecture Notes in Computer Science no. 1357. Berlin: Springer-Verlag.
- Helm R., I. M. Holland, D. Gangopadhyay (1990). "Contracts: Specifying Compositions in Object Oriented Systems". *Proceedings of OOPSLA, SIGPLAN Notices*, vol.25 no.10.
- Klarlund N., J. Koistinen, M. I. Schwartzbach (1996). "Formal Design Constraints". *Proceedings OOPSLA '96 Conference on Object-Oriented Programming Systems, Languages, and Applications*, ACM SIGPLAN Notices, 31(10), ACM Press.
- Martin R., D. Riehle, F. Buschmann (1997), eds. *Pattern Languages of Program Design 3*. Addison-Wesley
- Pal P. P. (1995). "Law-Governed Support for Realizing Design Patterns". *Proceedings of TOOLS USA 17*.
- Quintessoft Engineering, Inc. (1997). *C++ Code Navigator 1.1*. <http://www.quintessoft.com>
- Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen (1991). *Object Oriented Modeling and Design*. Prentice Hall.
- Vlissides J. (1997a). "Multicast". *C++ Report*, Sep. 97. SIGS Publications.

Vlissides J. (1997b). "Multicast - Observer = Typed Message". *C++ Report*, Nov.-Dec. 97. SIGS Publications.

Zimmer W. (1995). Relationships Between Design Patterns. In: Coplien J. O., D. C. Schmidt (1995, eds.) *Pattern Languages of Program Design*. Addison Wesley.