

# Position Paper for “Usability Pattern Language: Creating a Community” (Interact 1999 Workshop)

Michael J. Mahemoff  
Department of Computer Science and Software Engineering,  
The University of Melbourne,  
Parkville, 3052, Australia  
Email: m.mahemoff@csse.unimelb.edu.au  
Tel: +61 3 9344-9100, Fax: +61 3 9348-1184

## 1 Incorporating Usability into the Software Process

My interest in patterns began after studying software process issues and their relationship to usability. There are several difficulties in integrating human-computer interaction and software engineering:

1. Usability is only one of many qualities to achieve. To make matters worse, it is often in tension with other qualities, such as portability, maintainability, and reliability.
2. Usability is necessarily iterative, because (a) we are unable to accurately predict human reactions to systems and (b) the task-artefact cycle suggests that people will always suggest enhancements after working with an application for some time [2]. Software, on the other hand, tends to degrade with every new cycle, making it increasingly difficult to understand and modify.
3. HCI experts and software engineers have different perspectives and knowledge bases.

Patterns are an important step towards resolving each of these issues:

1. Patterns are about resolving forces which are in tension. Patterns can help to highlight situations when developers must make a trade-off between usability and some other factor. In addition, higher-level patterns which do not consider detailed software design can still help with decisions concerning sub-attributes of usability.
2. Alexander’s original vision for patterns was that they should support iterative development, and this vision has been carried forth into the software design pattern realm. Usability patterns can help to make incremental development a smooth process. This means less work for developers and a sense of consistency for users.
3. Patterns are an excellent way to educate software engineers. Usability principles and guidelines also have this goal, but have been less than satisfactory [3]. Principles like “Prevent user errors” are not very useful in solving specific problems, whereas the generative nature of patterns means they can be applied more directly. Furthermore, patterns also facilitate communication by providing a common vocabulary. They are also a useful medium for communicating an overall design philosophy.

## 2 Current Work in Usability Patterns

In the past few years, many people have helped to bring the promises of usability patterns closer to reality. This section describes some of the work I have been performing as part of my PhD, in conjunction with my supervisor Lorraine Johnston.

## 2.1 Domain-Specific Usability Patterns

Most usability pattern collections to date have been broadly-applicable, which can help provide patterns which are well-proven (because there are many examples to choose from) and aid practitioners, because they can reuse patterns in different situations. However, there is also a need to study usability patterns in more constrained contexts. The main advantage is that several patterns should fit together in a more cohesive manner if they are drawn from systems where the patterns are often applied in tight combination with each another. Thus, it is easier to produce collections of patterns which are more like the Alexandrian notion of a *language*, as opposed to a bunch of patterns.

In our work, two forms of constraint have been considered. Firstly, we have looked at constraining the patterns according to a particular user characteristic, namely cultural background. Thus, we have developed a high-level pattern language to help developers produce internationalised software. The language has definitive entry points and a clear path towards the final product. A central pattern, **Online Repository** is shown in the appendix.

A second constraint is domain. To this end, we have been considering usability of safety-critical systems, in conjunction with Andrew Hussey from the Software Verification Research Centre at the University of Queensland. An example is the **Conjunction** pattern, which suggests critical tasks should be performed more than once. This leads to another task organisation pattern, **Transaction**, which lets users specify a sequence of tasks before any of them are executed. A strong sense of connectedness arises from the fact that well-designed safety-critical systems usually contain similar features, and these features are combined in similar ways.

## 2.2 From Functionality and User-Interface Design to Detailed Software Design

I mentioned earlier that patterns can help to resolve the tension between usability and other software factors. In working towards this goal, I have considered how we can produce patterns of detailed software design which facilitate usability. In fact, many software design patterns already do this to some extent. The **Command Processor** [1] pattern supports undo, which certainly improves usability, and it is still sympathetic to other design issues.

However, software design patterns which do support usability generally do so when the feature is well-established, such as *Undo*. As usability patterns emerge, there will be more opportunity to investigate how these ideas can be effectively implemented in software. The constrained-context pattern languages mentioned above are a useful starting point for producing these kinds of patterns. We have also looked at activities which recur across different applications, with a view to capturing ways in which software designers can support those activities.

## References

- [1] F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns*. John Wiley & Sons, Chichester, 1995.
- [2] H. R. Hartson and D. Boehm-Davis. User interface development processes and methodologies. *Behaviour & Information Technology*, 12(2):98–114, March 1993.
- [3] M. J. Mahemoff and L. J. Johnston. Principles for a usability-oriented pattern language. In P. Calder and B. Thomas, editors, *OZCHI '98 Proceedings*, pages 132–139. IEEE Computer Society, Los Alamitos, CA, 1998.

## A Sample Pattern: Online Repository

This pattern is part of the high-level Planet pattern language for providing software to users from diverse cultural backgrounds. **Typewriter font** indicates references to other patterns.

**Context:** You have begun to maintain **Culture Models** according to the same **Vector Metamodel** (i.e. same factors for each culture).

**Problem:** As you start accumulating **Culture Models**, you will realise the need to organise them together. **How can a collection of culture models be organised in a manner which is useful for software projects?**

**Forces:**

- Culture Models should be organisation-wide to avoid duplication; it is feasible and desirable to transfer information learnt from one project into other projects.
- Information about cultures should be shared, but is often discovered in physically-distant locations.
- If developers can't access the models quickly and easily, the information will be ignored.
- If developers can't update the models easily, the repository will lose accuracy over time.
- Cultures have relationships with one another. We should be able to capture associations, such as one group being a sub-culture of another group.
- It is useful to look up a specific **Culture Model**, but there are many other ways people might like to access information during the culturalisation process. Depending on the task at hand, developers may wish to explore the information in unanticipated ways, e.g. comparing two cultures, considering a single factor across numerous cultures.

**Solution:** Create an online repository, accessible to the entire organisation. Compose it of **Culture Models** all based on the same **Vector Metamodel**.

The following guidelines make it easy to *access* information in the repository:

- Provide browsing facilities which present each culture and factor. Let the user select a factor (and show how each culture varies on it), a culture (and show all of its factors), or a combination of both.
- Provide facilities to search the **Culture Models**.
- Link from one model to another if it helps to demonstrate a point of similarity or difference.
- Link to the original artefacts if they are online, or identify the source if they are not.

The following guidelines make it easy to *update* the repository:

- Facilitate discussion among contributors, e.g. via a mailing list or within the repository system.
- For each **Culture Model**, make one or more people responsible for maintaining it.

Large repositories can group **Culture Models** together, e.g. according to continent. This approach can follow the **Composite** pattern [2], e.g. a model of Europe contains its own information and also contains child models (France, Italy, etc.). When a user looks up a model, information about its ancestor models are also shown.

**Rationale:** Like pattern languages, repositories of this nature help people reuse existing, tested, knowledge. Reuse of proven concepts involving human-computer interaction are particularly helpful, because people's reactions can be difficult to predict. In the case of interaction with international systems, the case for reuse is even stronger, because more work is required to obtain original knowledge (e.g. travelling overseas, establishing partnerships with foreign consultants).

**Examples:** Fernandes [1] contains some tables showing factors versus culture. However, the text stops short of exhaustively listing this information. One table might show cultures A, B, and C, and a table on other factors might show A, D, and E.

Ito and Nakakoji discussed a knowledge base for software development organised according to culture.

The authors are currently undertaking a project to build a web-based repository. The intention is that developers and users from around the world will use and contribute to the database.

**Resulting Context:** The repository enables developers to easily access a corpus of culture-specific information. You can use this information to specify **Flexible Functions**, an **Elastic User-Interface**, and **Targeted Elements**. Incorporate all culture-specific variants of each feature with **All-In-One**.

## References

- [1] T. Fernandes. *Global Interface Design*. AP Professional, Chesnut Hill, MA, 1995.
- [2] E. Gamma, R. Helm, R. Johnson, and R. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.