INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 05 MELBOURNE, AUGUST 15 – 18, 2005

USING PATTERN LANGUAGES IN DESIGN ENGINEERING

Filippo A. Salustri

Abstract

Pattern languages may be a beneficial yet unexplored way to capture emergent know-how in design engineering. A pattern is a natural-language, context-dependent description of a solution to a class of problems, that is both generative and descriptive. This paper introduces patterns and discusses how they can capture know-how in design. A key feature of patterns is their form: the style of their composition and presentation. There is neither evidence to suggest the superiority of one pattern form over any other, nor reason to believe that any existent pattern form will be suitable for design engineering. The author therefore introduces a new pattern form that is extremely adaptable while remaining true to the intent of the pattern approach. Three patterns are given in their entirety to demonstrate the author's form and suggest how patterns can capture and help communicate know-how. Proper evaluation can only be undertaken once enough patterns have been written and used that empirical data can be developed; this has not yet been done. However, early indications are that the pattern approach has good potential in design engineering.

Keywords: pattern language, know-how, emergent knowledge, body of knowledge, design methods

1 Introduction

Engineering may be considered an exercise in communication. Many agents and stakeholders must collaborate to make a good product because each has a unique and relevant perspective. Expertise and *know-how* are important elements that engineers bring to the product development table, but expertise that cannot be shared with other agents in some sensible way is of relatively little use.

In modern engineering settings, a tremendous amount of effort goes to capture, record, and communicate *know-how*, but no one has yet identified a single overarching approach or framework that reliably works in this role. In the author's experience, current methods typically used in industry settings tend to become quickly disused (see Section 2 for more on this matter). The potential improvement in corporate performance if this problem could be addressed would be substantial.

One approach that has not yet been tried in engineering is the method of *pattern languages*. Invented by the architect Christopher Alexander in the 1970s [1], pattern languages are informal, structured collections of natural language text that capture solution methods for classes of problems in defined contexts. Pattern languages have been successfully applied in architecture, software engineering, process engineering, and management science. They have not yet been applied in "conventional" (mechanical, civil, electrical…) engineering design. The reasons for this are not clear. The author suspects, however, that this may have to do with the relatively "unscientific" way in which patterns have been described in the literature. For example, Alexander himself writes of the *three-fold path* to use patterns. Such language is quite atypical in engineering, and it may be that engineers find it difficult to grasp concepts presented that way.

In any event, it may be that design engineers have overlooked this potentially useful technique. The author is carrying out research to study pattern languages, how they might be deployed to support the recording and communication of design *know-how*, and whether the design community would be receptive to pattern-based approaches. This paper will explain how patterns might be used in design engineering.

2 What are patterns?

The best definition of patterns comes from Alexander himself [1]: *"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant."*

In other words, a pattern has two aspects: a real aspect as a phenomenon or artefact, and a descriptive aspect as a generative explanation of how to design the artefact and why it works well. A well-written pattern makes both these aspects evident. The author summarises this in language more suitable to engineering as: *a pattern models a relationship between a context, a system of drivers that occurs repeatedly in the context and that is in an undesirable state, and an entity that allows these drivers to be resolved, bringing about a more preferred state*.

The use of the word *resolve* is particularly interesting because it can mean either *to solve* or *to break down into smaller units*. Both these connotations play a role in the pattern methodology because, as will be seen below, patterns are at their best when grouped into so-called *pattern languages*.

A pattern is, then, a (re)solution specification for a class of problems, written in natural language. The *application* of a pattern (i.e. a designer following the pattern to resolve a problem) results in a design that accounts for contextual issues explicitly. They are written following one of several available *forms*. Each form was developed for the sake of particular application domains. Nonetheless, all forms share certain key features, described below.

1. A Descriptive Name. A pattern is named by the kind of artefact that results from its application. A good pattern name captures not only the nature of the resulting artefact but also its purpose in the context.
2. A Problem Statement. Usually restricted to only one or two sentences, the problem statement is usually written as a description of an undesirable (or unbalanced) situation that must be resolved – in other words, a *need* to be addressed. By describing the problem situationally, one acknowledges the importance of context explicitly.
3. A List of Problem Drivers. In the pattern literature, the drivers are usually called *forces* that are unresolved in the problem. Since the term *force* has many other connotations in engineering, the author prefers the term *driver*. The drivers are usually written as a point-form list of features that must be balanced in a good solution. (The notion of *balance* is explained in detail in Section 3.1.)

4. A Context Description. While the first three items above describe a situation in general terms, the context description allows more detail to be provided. Different solutions can arise from the same problem occurring in different contexts. Contextual information is essential to choose an appropriate pattern. The context description is grounded in the listing of drivers; that is, it explains why the drivers matter. It should be general enough to indicate clearly the breadth of applicability of the pattern, yet not so broad as to be merely a general goal.
5. A Procedural Solution. An explanation of how the drivers are achieved, typically written as steps to be followed by the pattern user to generate an artefact that balances the drivers. References to other patterns are common in this description.
6. Consequences of Application. Implementing a solution to a problem changes the context of the problem. This means that applying patterns affects the situation that drove the pattern's use to begin with. These consequences are not always positive ones. It is important that a pattern include a description of *all* the possible consequences of a pattern's application, both good and bad, so the pattern user can prepare better for changes in the situation where the problem occurred.

While nearly all well written patterns provide all this information at least to some extent, the actual *form* of the pattern description (the composition and structure of the natural language text that defines the pattern) can vary significantly. The matter of pattern form is addressed below.

Not all patterns are widely adopted within their target disciplines. Adoption is generally considered the ultimate mark of a successful pattern. Over the years, researchers have studied successful patterns to identify characteristics that might help pattern developers know when they have found or written a good pattern [2,3]. Having surveyed this work, the author summarises below some of these characteristics.

Generativity. A good pattern teaches us how to build their manifestations. More than an instruction set (or *cookbook*, or algorithm), a good pattern indicates the dynamic nature of its implementation, its adaptability, flexibility, and robustness. Because of their instructional nature, it should be possible to visually depict the kind of structure that results from a pattern application, at least in general or schematic terms.

Descriptive. A good pattern gives a sense of what its manifestation might look like. Telling how to resolve a problem is not enough; the pattern must also describe the nature ("structure") of the solution, at least in a general way.

Explicative. A good pattern gives a sense of the reasons why the solution is appropriate (as opposed to alternative solutions). Users can judge if the reasons hold for their own context.

Recurrence. A pattern must be a recurring phenomenon. Typically, three existing instances should be identified of a proposed pattern successfully resolving a problem. Patterns must exhibit a record of success. Successes should come from throughout the range indicated by the context of its use. It is also relatively common practice to at least mention some cases in which the pattern is known *not* to work; these cases, usually called *contraindications*, can be very helpful to users deciding whether or not a particular pattern will work well for them.

Non-definitive. Alternative solutions may exist to any pattern. Just because a pattern exists for some class of problem does not mean it is the only way to (re)solve the problem. Many pattern formats include a specific section enumerating other patterns that apply to the same or similar problems, expressly to ensure that the pattern's user recognises its non-definitive nature.

Context-sensitivity. Although it may be quite general, a pattern only works in, and should be described with respect to, a single context. That context must be described sufficiently well to let potential users decide if the pattern will work in their context. While broad contexts are acceptable, universally applicable solutions (i.e. solutions that are context-independent) are generally considered inappropriate.

Relational. Patterns must address the relationships between context, drivers, and solution; they must describe how and why their manifestations interact within a context, and the interactions between sub-patterns, super-patterns, and co-patterns.

Assistive (to humans). A user of a pattern must find the pattern useful to (re)solve the problem drivers. Patterns that users find difficult to use or obtuse are not considered good ones. Useful patterns will help human users develop better actual solutions to specific problems.

Evolving. Patterns can change over time, as users "tweak" its specification in response to reports of its usage. In many cases, patterns are developed collaboratively in groups [4] and are revisited and updated regularly. As artefacts themselves, pattern descriptions can be thought of as *living documents* and as such can not only improve with time but also devolve into uselessness.

A good pattern is, therefore, a descriptive guide and a mechanism for knowledge transfer between individuals and groups. Although computer support for pattern management and use is possible, patterns neither encode knowledge in a computable form nor automate design. Instead, patterns are intended to *stimulate thinking and learning* by users.

A group of tightly integrated patterns covering entirely a domain of interest is a *pattern language* (e.g. [1,5]). Pattern languages can represent *bodies of knowledge* on particular subjects. For example, Alexander's pattern language [1] ostensibly describes solutions for every kind of problem faced by architects ranging from locating urban/suburban regions to the positioning of photographs and other decorations on the wall of rooms.

Pattern languages are often developed collaboratively. The collaborative authoring group is usually a subset of the community most likely to use the language, typically each with some degree of expertise in the area. Each contributor to such a project can get a very significant return on the investment of their time, because the contributor has access to the entire language in exchange for developing and maintaining (usually) just a few patterns. Measured in this way, pattern languages can yield a substantial *return on investment*. This is especially important in corporate settings, where employee efficiency is a key corporate success factor.

Since a pattern must be "proven" in that its application must have had a number of demonstrably successful cases, one may argue that their use stifles innovation by preferring existent solutions. However, this is not necessarily the case. Firstly, patterns are only best practices *within identified contexts*. In different contexts not covered by one pattern, other patterns might be better suited. Contexts can become obsolete. For example, as globalisation becomes more important in engineering, patterns that applied in strictly local settings become inappropriate. By including contextual information, a user can make an informed decision about when *not* to use a pattern. In this case, an opportunity for innovation is identified (i.e. the existent way of doing things – the existent pattern – is not good enough).

Secondly, pattern languages are neither prescriptive nor universal. They are only guidelines and as such can be used as a springboard to thinking about different possibly more innovative solutions. The pattern approach can also be used to write innovation solution methods; the pattern author would simply not report on the pattern's record of success. This will immediately inform a potential user that the pattern is only tentative. In time, successes of its use may be added to the pattern – indeed, the pattern itself may be refined to account for the experiences of its user community.

Thirdly, patterns can be used as "jumping off points" for reasoning about innovative solutions entirely different from the pattern itself, exactly because sufficient contextual and procedural information is given in a good pattern to stimulate such thinking (assuming the user has the interest and training to carry out such a thinking exercise).

Finally, innovation can be related often to some insight derived from having a broad and diverse range of experience (e.g. design by analogy). Patterns provide a mechanism for a designer to scour available information for reports of experiences that the designer himself might not have. Thus, "browsing" pattern languages can help identify opportunities for innovation.

Therefore, the author believes that patterns do not necessarily stifle creativity and, in fact, may encourage innovation if they are not abused.

Pattern languages impact directly the recording of *know-how* in industry. Practising engineers have often recounted how "procedures" that were carefully documented at significant expense end up in dusty, unused manuals. When asked, these engineers explain that only experts who already understand the procedures can understand the documents. This is not necessarily a function of the complexity of the knowledge, but rather often a result of how the material is presented. Pattern languages are meant to promote better transfer of knowledge through written text, so it is likely that good patterns will help in these cases. Their success in software engineering and architecture are a testament to their capacity in this regard. Unfortunately, the *know-how* typically captured in engineering "procedures" is highly proprietary; it has been difficult to gain the confidence of industry needed to allow that knowledge to be shared with us for the sake of writing patterns for them. Nonetheless, we continue to promote the pattern approach and are confident that a few small successes will lead to a gradual adoption of the method in time.

3 A three-part framework for pattern usage

Alexander developed the *Three-Fold Path* to summarise his philosophy of pattern usage. The current author has adapted Alexander's work for engineering by translating it into a form more consistent with design engineering terminology. This leads to a *three-part framework* for pattern use in design engineering, described below.

The goal of the framework is to present in broad strokes the overarching situation in which pattern languages are best employed. The author has kept the framework presented here no more or less general than Alexander's *Three-Fold Path*.

3.1 Part 1: balanced design

Some designs satisfy not only their technical requirements, but are also suited to a market, represent a good business tactic, and satisfy a societal need. They address issues of aesthetics, usability, etc. as well as fitting into a global context. They have a certain appeal at every level that makes them greater than the sum of their parts or functions, by finding a dynamic harmony among the competing problem drivers. The author call such designs *balanced*. Examples of balanced designs include: the DC-3 and the Boeing 747, the (original) VW Beetle and the Studebaker, the first Palm Pilot and the Apple iMac, the BIC ballpoint pen and the Mont Blanc fountain pen. *Balance* is a way of thinking about the match between a context (which changes in time) and a design. Balanced designs hit a "sweet-spot" within their particular contexts. If they had been brought to market too early (e.g. the Apple Liza) or too late (imagine trying to sell the original 747 in today's market), then they would have seemed *unbalanced*. Striking the right balance is like hitting a moving target.

There are no known methods for assuring that a particular design is well-balanced, yet such designs remain the ideal of all engineers, and are quite readily identified where they exist. Well-written patterns can help achieve balanced designs because they address the act of balancing a design directly.

The "living," evolutionary nature of patterns helps keep them current. Individual patterns can be updated regularly and in a structured, predefined way, or in a more informal way, depending on the situation and the preferences of the pattern developers. Since patterns are usually developed by people likely to benefit from their use, they are often a very reliable source of *know-how*.

3.2 Part 2: tempered creativity

A pattern language gives an architecture for human cognitive design activities within a domain to temper creativity with reason, and to provide coarse guides to direct the creative energies of designers. A pattern language is a tool that augments designers' innate and learned skills by suggesting potentially successful routes and providing warnings to pitfalls and problems. Patterns therefore provide "checks and balances" to help control the possibility of making significant design errors.

The notion of "tempered creativity" is very important in engineering design. Innovation is often described as a corporate requirement, but innovation implies risk. Corporations need to balance the benefits of innovation against the risks. Too much innovation is just as dangerous as is not enough. Being able to temper the creativity required for innovation is one way to achieve this.

Patterns help temper creativity by describing both the good and bad characteristics of a particular solution, and by indicating descriptively albeit generally the amount of work needed to bring a design into being. They promote considered thought of the advantages and disadvantages of a solution within a context and help create a corporate mindset that is more likely to understand the inherent benefits and risks of a design in the broadest sense.

3.3 Part 3: top-down systems design

Within the pattern-based approach, the "best" processes are top-down and systems-based. They start by considering the role of the product in larger operational contexts – such as determining first the user requirements for a nondescript product – and then detailing the product to satisfy those requirements.

Requirements specifications often emerge in parallel with the design solutions – this is sometimes called *co-evolution*. Patterns would be used after a phase of requirements specification has occurred, as a means of matching requirements to possible solutions. The pattern approach therefore fits quite well with typical design processes. Pattern languages can be laid out to correspond to the different levels of detail in a product development process to help organise and manage know-how.

3.4 Summary

We see then an emerging general framework for the development and use of patterns. Patterns are inherently systems oriented if for no other reason that they always include the product being designed, operating environment, and other agents (human or otherwise) that will interact with the product (i.e. the context). Patterns work best in top-down development processes because these ensure the primacy of the context of use. They promote clear thinking without removing the opportunity for innovation, thus facilitating efficient and effective design work. Finally, they address directly the multi-objective nature of any design problem by facilitating balanced designs.

This framework is presented to roughly the same level of detail as Alexander's original. The intention was to recast Alexander's notions for engineers without making statements that might misrepresent the original work. In a sense, it is contrary to the pattern approach to "give away all the answers" by detailing the framework to a significant degree. Patterns intended after all to promote curiosity and thinking, to stimulate new ideas. Detailing their use pre-empts a user's own need to struggle with the notion and achieve one's own personal understanding – to learn *deeply* – what patterns are.

4 A form for engineering design patterns

In the patterns community, the term *form* is used to describe the structure and layout of a pattern description. Various forms have been used by other practitioners of the approach. The differences between forms usually arise from the needs of the form developer to address particular features of the domain of interest. They are most pronounced between different disciplines (e.g. software development versus architecture). Since patterns have not yet been used in conventional engineering, it is important to examine the notion of pattern form, to ensure a form for engineering design patterns is well suited to the discipline.

The simplest pattern form is called *therefore-but*. There are three sections to patterns defined with this form: a problem statement, a description of how the problem is (re)solved, and a discussion of consequences arising from the pattern's application. The three parts are literally separated by the text *therefore* (between the problem and solution) and *but* (between the solution and the consequences). While this form is extremely simple and therefore flexible, it can lead to disorganised and inconsistent descriptions of patterns which makes them difficult to understand and apply. It is, however, especially useful for capturing short patterns that encapsulate some kinds of *best practices* or general principles. For example, refer to Figure 1.

Pattern: Involve Users Actively.

You need to ensure that the needs of all users are addressed during product design.

Therefore: Include representatives of the user community in the design team from the outset, and charge them to liaise with other user groups and the design team.

But: Beware that user representatives may selectively filter or change the information passed to them by other users in accordance with their own biases and preferences.

Figure 1: A sample pattern using the *therefore-but* pattern form.

Most other forms have many very specific sections, such as: problem statement, background information, drivers, symptoms, renderings of typical results, solution procedure, solution structure, contraindications, examples, and lists of related patterns, and other reference materials. While this more fine-grained structure can help organise pattern descriptions (making them easier to search, for example) it can be distracting to pattern authors. Sometimes, a pattern author will include text in a particular section simply to have something there, regardless of whether it really contributes to understanding the pattern – authors can feel *obliged* to write material to fill in form sections that detracts from the clarity of the presentation in the long run. Furthermore, pattern authors may find it necessary to repeat information in multiple sections because the pattern form is not well suited to that particular pattern or that particular pattern author's writing style. That is, authors may feel constrained by the form to break the natural flow of exposition of a particular pattern. It is important to write the pattern so the presentation of material flows easily. This greatly facilitates users' comprehension of the pattern. As a general rule of thumb, a pattern is well written if one can read it from beginning to end without having to skip forward or reread previous material to understand the pattern well. Sometimes, having too many sections virtually forces a pattern writer to violate this rule of thumb.

The present author has decided to develop a new form that is a variant of *therefore-but*. The reason for this is that pattern languages have not yet been used in conventional (e.g. mechanical, electrical, civil, etc.) engineering domains, and it is not clear what level of granularity is appropriate. Instead, the author's form contains the three major sections of the *therefore-but* form, but makes additional *recommendations* on the nature of the contents of each section. The recommendations are reminiscent of the more detailed pattern forms, but are largely optional in nature. One hopes that this will encourage pattern writers to adapt the form to the needs of the particular pattern they are writing, while giving them a base well rooted in the existent literature.

While "therefore" seemed like the most obvious choice to link the problem and solution, the author did consider using some different word instead of "but" (e.g. "consequences" or "caveats"). These other words, however, seemed to have more limited connotations. "Consequences" implies a causal (and possible exclusive) relation and "caveats" suggests conditions or constraints. "But," on the other hand, seemed to carry all these connotations and more. Because we do not want to bias unnecessarily a pattern writer's thinking by having particular connotations of these guidewords, "but" was retained in the author's pattern form.

The form is outlined below. It was developed to address all the characteristics of good patterns identified in Section 2.

Descriptive Pattern Name

Author(s), modification date

Problem: *A one-sentence statement of the problem, expressed as a need to be addressed.*

Paragraphs describing the context of the problem.

Indicate as appropriate any known contraindications to the use of this pattern.

The key drivers are:

- a point-form bulleted list of problem drivers.

Therefore: A short instructive phrase indicating the nature of the problem solution, expressed as a directive to the pattern user.

Paragraphs describing the tasks to be carried out to (re)solve the problem, leading to at least a conceptual description of the resulting artefact.

Indicate the relationship of the described solution to other patterns, solutions, or methods.

Include a description of how the artefact resulting from the pattern's application is used, initialised, and maintained. Refer to other patterns as appropriate.

But:

Paragraphs describing the consequences of using this pattern, paying particular attention to changes of the context resulting from the pattern's application. Ensure that any known consequences that are known to be detrimental are explicated.

Specific examples may be provided of the successful application of the pattern. These may be kept in separate documents and only referenced here. Ideally, there should be three distinct examples, each as different as possible from the others.

If possible, examples of cases where the pattern does *not* work should be provided.

See Also: (optional)

- Another Pattern Name – brief description of its relevance to the current pattern (must not be a pattern already cited in the body of this pattern).

Figure 2: General structure of the author's pattern form.

5 Three sample patterns

In this section, three patterns are presented to further explain the author's pattern form. The formatting of these patterns has been slightly modified to suit the conference format. Underlining is used to indicate the name of another pattern. Note that the third example is taken from the documentation for a software tool being developed by the author to support the use and development of pattern languages. The tool, called Xiki, is briefly described in Section 6.

5.1 Sample pattern #1: variable fluid mixer

Problem: *design a system that mixes fluid (including powder) ingredients, controlling production rate and mix ratios.*

Sometimes, the same equipment may be used to mix different ingredients at different mix ratios to produce different products (EG: mixers for paint, fertilisers, or dry ingredients of baked foods). Although variability in mix ratio rarely affects production rate in such cases, even small mix ratio variations can lower product quality.

Modular design suggests there are advantages to duplicating the ingredient delivery system identically for each ingredient (for example, having a separately driven pump for each ingredient) to get economies of scale in the subassemblies. However, variation in the relative rates of the motors that drive the pumps can cause unacceptable variations in the mix ratio. The variation arises because the motors, though structurally separate from one another, are *functionally* coupled: the mix ratio is determined by *all* motor (and hence pump) speeds and their associated variations, which are additive.

Furthermore, many environments in which this situation can occur are "dirty;" i.e. there is an assortment of contaminants in the operating environment that can hinder relatively delicate control machinery or electronics.

Key drivers are:

- Fine control of mix ratio is required
- Delicate electronics should not be used because of dirty environment
- The mix ratio must be adjustable
- The mix ratio variability must be low
- Reliability must be high
- Capital, operating, and maintenance costs must be kept low

Therefore: design a *functionally* modular system.

Design the system to consist of independent *functional* modules. Assign one pump per ingredient, but use a single motor to drive all the pumps, and variable ratio gearboxes to vary the speeds of the pumps (and therefore the mix ratio). The variability arising from a single motor and a single gearbox will be transmitted proportionally to all pumps, effectively cancelling it out or at least dramatically reducing it.

This solution is consistent with Axiomatic Design [6]. The functional requirements of this product are (a) it must produce the right amount of product and (b) it must mix the ingredients correctly. If separate motors drive each pump, then the design parameters are the speed of each pump. This induces a fully coupled design, which is undesirable. Setting the speeds of the motors to achieve prescribed productivity and mix ratio values becomes an iterative process. Variation over time in the speeds of the motors requires active, dynamic control of the system.

However, if using a single motor and variable ratio gearboxes, the design parameters are the speed of the motor and the gearbox ratio; this design is decoupled, which is preferred. Setting the motor speed and gear ratios is not an iterative process and does not require active, dynamic control.

But:

Selection of gears and calculation of tolerances must be done carefully, to minimise variability of the gearboxes between gears.

Functional modularity introduces structural coupling here. One must pay attention to ensuring this coupling does not lower reliability in specific cases. Reliability Analysis and Failure Mode And Effect Analysis are recommended methods to assess this.

Structural modularity can be salvaged to a degree by designing the shafts and structural elements to facilitate replacement of gearboxes, pumps, the motor, and other major subsystems.

Furthermore, particular attention must be given to the gearbox design or specification. Backlash and other effects must be accounted for to ensure the gearbox is properly specified to prevent a quality loss similar using multiple motors.

5.2 Sample pattern #2: free body diagram

Problem: *you need to model the geometry, applied forces, and resultant forces in an object.*

When you carry out a Static Behaviour Analysis of a structure, you must understand the Kinds Of Forces and Kinds Of Constraints acting on the structure. A key element of such an analysis is to represent the geometry and loading conditions of the object. However, as the analysis is only approximate, one does not need much detail. You need to capture the minimum amount of information needed for the analysis; too much information can lead to errors and confusion in the analysis. This clarifies thinking about the problem and facilitates calculations.

Key drivers are:

- the model must be as simple as possible
- the model must include contain all required geometry, loading, and constraint information
- the model must not require significant effort to render
- the model must be a "standalone" representation of the model (so that it can be reused in other settings)

Therefore: Use a free body diagram to model the problem.

A *free body diagram* (FBD) is a simplified diagram that models an object and the loads applied to it that helps calculate other forces in the object (see What Are Forces?, What Are Moments? and Why Do Objects Resist Forces?).

Identify *clearly* the goal of the analysis (e.g. *Find the forces acting in the member X of the object*). The goal usually involves finding a particular force or moment in a member or element of the object. If the goal is to find more than one force or moment, treat each one as if it were a separate problem.

Create a Geometric Schematic of the object. Use Arrows To Represent Forces (both Applied Forces and Reaction Forces).

Indicate clearly on the diagram the Static Boundary Conditions of the problem. This will identify which Degrees Of Freedom (DOFs) are fixed and which are free. We assume a Perfectly Rigid Frame unless told otherwise. Fixed DOFs will generate Reaction Forces; free DOFs will not. (What does a free DOF generate instead of a reaction force?)

Consider the connections between the object and the frame, and identify only the *fixed* connections. In the diagram, replace the frame with arrows that represent the Reaction Forces caused by the fixed DOFs between the object and the frame.

Identify on the sketch the member that is referred to by the goal.

Use a FBD Cutting Line to Split The FBD. The cutting line must pass through the member referred to by goal. This splits the object into two disconnected "chunks."

It is important to Match The Cutting Line To The Problem.

From now on, consider only one of the two chunks that result from the cutting operation. Choose the chunk that has the fewest unknowns; disregard the other chunk.

Where the line cuts each member in the chunk, draw a vector representing a force that would have been caused by the chunk of the object that you sliced away; that is, Substitute Forces For Members in the chunk.

Now one can apply Summation Of Forces on the Force Components and Summation Of Moments on the Moment Components to develop a set of equations describing the static loading of the chunk. In the set of equations, one unknown will be the force in the member referred to by the goal of the analysis. There will be others. One can then solve this set of Simultaneous Equations that result to determine the required force.

Remember to report all results using the appropriate Significant Digits and Units Of Measure.

But:

If one chooses poorly the members to cut, one may have too many unknowns to solve. One then must redo the Free Body Diagram differently, choosing a different FBD Cutting Line. This wastes time. It is far more productive to invest time early on, choosing the best members to cut; this will significantly speed up finding a solution, and the solution will generally be easier to calculate.

Learning how to identify the best FBD Cutting Line is something best done by practice. It is therefore in the student's best interest to do as many problems involving free body diagrams as possible. In time, finding the best cutting line will become second nature.

If the goal involves finding the forces in many members in an object, it is likely that you will have to make different cuts to solve for different forces. This will require re-sketching the geometry, and rebuilding the Free Body Diagram.

5.3 Sample pattern #3: Uploading files in Xiki

Problem: *you want to make a particular file (say, an image or a file created by a word processor) available to others, but you do not have access to a web site to which you can post such files.*

Xiki is based on users typing source text into Xiki Topics. It is not productive to re-type material already available in, say, MS Word. Furthermore, if you cannot otherwise make images available through a web server, you need a mechanism to make those images available.

Key drivers are:

- Make non-text files available to others
- Maintain security of files and of server access
- Minimise contributor's workload to make those files accessible

Therefore: upload files using Xiki.

Many wikis allow files to be attached to topic pages. Xiki provides a more general mechanism of uploading files to be associated with a particular Xiki Web.

To upload an attachment:

1. Use the *browse* button at the bottom of a Xiki page to identify the file on your computer that you want to upload to Xiki. This will automatically fill in one of the two boxes in the Upload area.

2. Add a *brief* description of the file in the other text box (no more than 255 characters).
3. Hit Return.
4. You will see a message just above the topic name indicating whether the upload was successful.

A record of the upload is placed in the Uploaded Files topic in the current Xiki Web. Typically, the Uploaded Files topic will appear in the Web Menu if it exists – i.e. if at least one file has been uploaded.

To refer to an uploaded file in a topic, either:

A. For an uploaded file named `Fred`, refer to `\Attachments/Fred` in any topic in that Xiki Web, and it will be rendered as a link to the uploaded file. If the uploaded file is an image, then the image will be rendered directly.

B. Go to the Uploaded Files topic and copy the link (typically, right-click and select from the popup menu) of the desired file. Then use a Xiki editing session to paste the link into another topic.

But:

Uploading only works for authenticated users (i.e. *not* for Guest User). Unauthenticated users will not see the upload area at all on any page, but they *will* see links to uploaded files and they will be able to download those files or see those images.

The exact location of the upload area will vary depending on the Xiki Skin you are using. In the Basic Skin, it appears at the bottom of a page.

Uploading a file will permanently destroy any other file by the same name.

On the other hand, uploaded files are *not* deleted when the topics that refer to them are deleted. This means that the number of uploaded files can grow without bound.

5.4 Remarks regarding the examples

The first example shows how a pattern can capture a "best practice" in more detail than a simple statement like "prefer functional modularity to structural modularity." We also see both the operational and descriptive nature of the pattern. It is easy to visualise a schematic configuration of the resulting product, and it is evident the solution has disadvantageous as well as advantageous consequences. Relations are drawn to other methods (e.g. Axiomatic Design and FMEA). This pattern may not be perfect, but one would expect users of the pattern to maintain it, adjusting it over time.

The second example indicates how patterns could be used in education. The Free Body Diagram pattern is designed to be part of a pattern language for an introductory course in static analysis that is typical in mechanical engineering curricula. There are extensive references to other patterns. Each pattern represents a unit of knowledge that students should learn. One can envision a graphical index of the pattern language such that nodes are patterns and arcs are references of one pattern in another. Various paths will exist through all the patterns. The resulting "web" gives instructors many alternatives to the strictly linear presentation typically found in textbooks on this subject. Some instructors might choose to cover all the patterns used by a pattern like Free Body Diagram before covering the pattern itself. Other instructors may choose to use a "just-in-time" teaching approach. Students, when studying the material individually, can easily select to follow paths through the pattern language different from those followed by the instructor. By compartmentalising units of knowledge into patterns, it should be possible to help students remember possibly complex solution methods as discrete sequences of steps. The relative brevity of each pattern should also help retention of the materials.

The third example is drawn directly from the documentation of Xiki, a software tool being developed by the author to support the development and use of pattern languages in design engineering. This example is intended to show how simple *know-how* and instructional help can be captured by a pattern. The particular pattern shown is for uploading attachments into a collaborative workspace. It refers to other elements of Xiki documentation, but remains a self-contained unit addressing a particular function provided by the software. Further details about Xiki are given below.

6 Discussion

While pattern languages are not inherently computer-based tools, they can benefit from computer support. It is easy to envision Web technologies being used to interconnect patterns, creating online repositories of pattern languages.

Indeed, one specific Web-based tool has been nearly universally adopted to facilitate the creation and use of pattern languages: a Wiki. The Wiki (meaning *quick* in Hawaiian) was invented by Ward Cunningham not long after the Web itself was invented [7]. Wikis use simple server-side technologies like CGI scripts to create collections of information that can be edited by anyone via a conventional browser. The author is developing a new Wiki, called Xiki, for use in engineering design settings. Details on Xiki are available in another ICED 05 paper by the current author. The key feature of Xiki that pertains to pattern languages is it can automatically create links between web pages based on *wiki words*. A wiki word is a string of capitalised words run together without spaces. For example, `FreeBodyDiagram` and `ReactionForces` are examples of wiki words. Xiki interprets such strings as references to other pages by the same name and renders the strings as links pointing to the appropriate URL. Thus, `ReactionForces` is rendered as a link to a page name *Reaction Forces*. By giving each pattern its own page in Xiki, we can trivially interconnect patterns by using wiki words. There are other features of Xiki that help support pattern languages; they are discussed in the author's other paper.

The author has used patterns in teaching settings and found that they can promote more rapid uptake of *know-how* by students. Developing course notes using patterns (especially with computer support by a wiki) *significantly* improves their development and maintenance – even compared to more powerful content management systems like Blackboard – allowing the instructor to spend more time actually teaching. Informal studies conducted by the author suggest the pattern approach can be a valuable aid in recording and transferring *know-how*, and that students are able to remember material better because of the "chunking" of units of knowledge in patterns. Of course, more rigorous studies need to be carried out, but a sufficiently broad set of patterns must be created first that have value to a user community (be they students or practicing engineers). As the author builds a broader collection of patterns, we will continue to test their usefulness in a variety of educational and industrial settings, and will be able to report on this work further in two years.

Finally, it has been a common observation about pattern languages that the act of their construction can significantly refine the level of understanding of a subject by at least the pattern development community. This is because as patterns evolve, bodies of knowledge emerge from the interconnected patterns. Pattern forms are usually structured enough to make explicit many relationships that might only otherwise be implied, while maintaining a high degree of flexibility and adaptability. The obvious application here is to furthering our understanding of design and the possibility of evolving a body of knowledge for our discipline.

7 Conclusions

The author has introduced pattern languages as a means to capture know-how in design. Though successful in other fields, the pattern method has not yet been applied to conventional engineering settings. A new pattern form for design pattern languages has been presented, with three examples that demonstrate what actual patterns can look like. Early analysis of the use of the pattern method shows promise but more analysis must be done once a more extensive collection of sample patterns has been developed. Preliminary results indicate that there are no inherent hindrances to the application of patterns in design engineering, and that well written patterns can communicate some kinds of knowledge effectively. Within the next two years, we expect to conduct more formalised evaluations of the method.

References

[1] C. Alexander, S. Ishikawa, and M. Silverstein. 1977. A pattern language: towns, buildings, construction. Oxford University Press, London.

[2] B. Appleton. 1997. Patterns and software: essential concepts and terminology. Object Magazine Online, 3(5).

[3] J. Bennedsen and O. Eriksen. 2003. Applying and developing patterns in teaching. Frontier in Education Conference, Session T4A. ASEE/IEEE.

[4] J.O. Coplien. 1999. A pattern language for writers' workshops. Patterns Languages of Program Design 4. Addison-Wesley, Mass.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. 1995. Design patterns. Addison-Wesley, Mass.

[6] N.P. Suh. 1990. The Principles of Design. Oxford University Press, New York.

[7] B. Leuf and W. Cunningham. 2001. The Wiki Way. Addison-Wesley, Boston.

Filippo A. Salustri
Department of Mechanical and Industrial Engineering
Ryerson University
350 Victoria Street
Toronto, ON, M5B 2K3, Canada
tel: +001-416-979-5000 x7749
fax +001-416-979-5265
email: salustri@ryerson.ca
http://deed.ryerson.ca/~fil