When are design patterns the problem instead of the solution? [closed]

Asked 15 years, 11 months ago Modified 5 years, 2 months ago

Viewed 9k times

58

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 7 years ago.

Improve this question

I've never worked on software where I needed to use design patterns. According to Paul Graham's Revenge of the Nerds essay, design patterns are a sign of not enough abstraction.

To quote him directly, "For example, in the OO world you hear a good deal about "patterns". I wonder if these patterns are not sometimes evidence of case (c), the human compiler, at work. When I see patterns in my programs, I consider it a sign of trouble. The shape of a program should reflect only the problem it needs to solve. Any other regularity in the code is a sign, to me at least, that I'm using abstractions that aren't powerful enough--often that I'm generating by hand the expansions of some macro that I need to write.”

I was just wondering if everyone thought design patterns are overused and are symptoms of not having enough abstraction in your code.

design-patterns

Share

Improve this question

Follow

edited Feb 12, 2009 at 8:16

brian d foy
132k ● 31 ● 211 ● 604

asked Jan 21, 2009 at 18:06

Jared
39.8k ● 29 ● 113 ● 146

26 Answers

Sorted by: Highest score (default)

42

I don't think the patterns per se are the problem, but rather the fact that developers can learn patterns and then overapply them, or apply them in ways that are wildly inappropriate.

The use of patterns is something that experienced programmers just learn naturally. You've solved some problem X many times, you know what approach works, you use that approach because your skill and experience tell you it's appropriate. That's a pattern, and it's okay.

But it's equally possible for a programmer who's less skilled to find one way to do things and try to cram every problem they come across into that mold, because they don't know any other way. That's a pattern too, and it's evil.

Share Improve this answer Follow

edited Jan 21, 2009 at 20:04

answered Jan 21, 2009 at 18:16

Adam Bellaire
110k ● 19 ● 152 ● 165

26

Come on folks, please read the whole quote, read it carefully. Or even better, read the essay.

Paul Graham criticizes, among other things, C-like languages for not providing adequate means of abstraction. Within the scope of the essay his criticism of patterns is an afterthought, or rather a case-in-point for his main argument. His reasoning goes like this:

It is logical to use common strategies to solve recurring problems. In really abstract languages it is possible to formalize those strategies and put them into a library. Whenever you need to use them, you merely #include them, instantiate them, expand them, or whatever. C-like languages in contrast do not provide the necessary means of abstraction. That is witnessed by the fact that there exists something like "patterns". A pattern is such a common strategy that cannot be expressed by library code and thus has to be expressively written every time it is applied.

Paul Graham does not think that patterns are evil by themselves. They are a symptom of languages that fall short in providing means of abstraction. In that respect he is almost certainly right. Whether we should use different languages because of that, is of course another discussion.

The original poster of the question, on the other hand, is wrong: Patterns are not "symptoms of not having not enough abstraction in your code", but are symptoms of having not enough means of abstraction in your language.

Share Improve this answer Follow

answered Jan 22, 2009 at 2:11

edgar.holleis
4,991 ● 2 ● 25 ● 27

22

Patterns are really just a way of describing how things work. It is a way of classifying them. Are there some programs that overuse them? Sure. The biggest advantage of having patterns is that by classifying something as this or that, everyone is on the same page (assuming they have the level of knowledge to know what is being talked about.). When you have a system with

10,000 of lines of code, it becomes necessary to be able to quickly determine how something is going to work.

Does this mean that you should always use patterns, no. That will lead to problems to force things into a classification, but you shouldn't shy away from them either.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:18

kemiller2002
115k ● 28 ● 199 ● 253

1 Very nice recapitulation of patterns. Unfortunately it misses the mark with respect to Paul Graham's argument: If your language provided you with the necessary means to simply #include <factory.h>, you wouldn't have to write it. Then your app would be smaller and "factory" wouldn't be a "pattern". – edgar.holleis Jan 22, 2009 at 2:21

1 It doesn't miss the mark at all. If patterns are just a way of describing code then the differences between using a pattern or some other abstraction is the same as the difference between saying "This code uses Factory" and "This code uses the Factory Pattern", which is a fairly trivial difference. – CurtainDog May 3, 2009 at 1:58

22

My problem with patterns is that there seems to be a central lie at the core of the concept: The idea that if you can somehow categorize the code experts write, then anyone can write expert code by just recognizing and mechanicaly applying the categories. That sounds great to managers, as expert software designers are relatively rare.

The problem is that it isn't true. You can't write expert-quality code with only "design patterns" any more than you can design your own professional fashion designer-quality clothing using only sewing patterns.

Share Improve this answer Follow

edited Sep 25, 2019 at 22:28
PD81
20.5k ● 3 ● 42 ● 58

answered Jan 21, 2009 at 18:50
T.E.D.
44.7k ● 10 ● 76 ● 138

19

> The shape of a program should reflect only the problem it needs to solve.

And what happens when requirements change and your module wasn't abstracted using a Facade or potentially a Mediator, thus making it overly difficult to swap out?

> design patterns are a sign of not enough abstraction

Chances are, if you've abstracted everything properly, then you have a design pattern in there somewhere.

What if there is an extremely 'heavy' object that does not have to be loaded? The Proxy pattern saves the user from waiting forever.

I could go on, but I think I have said enough. Design patterns are great tools when used correctly. The problem comes when they are used improperly, but I guess that's why misused patterns are called anti-patterns.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:16

geowa4
41.7k ● 17 ● 89 ● 107

1 One could argue that you are trying to change the wrong thing at the wrong level if you have to apply a design pattern to make it more amenable to the actual change. – MSN Jan 21, 2009 at 18:36

12 In my experience, preemptively introducing abstractions / design patterns to make change easier in the future often achieves exactly the opposite by bloating the code. The easiest-to-change system is still nearly always the smallest one. – Michael Borgwardt Jan 21, 2009 at 20:14

I've never run into bloat with a module being abstracted by a Facade. If having that file adds a few KBs, it's worth it for the decreased coupling. – geowa4 Jan 21, 2009 at 21:07

If you've abstracted everything properly, then you have something in there that in OO-languages is called a "pattern". If, however, the language provided you the proper means of abstraction, you could #include <factory.h> instead of writing it. – edgar.holleis Jan 22, 2009 at 2:26

There are already a number of fully configurable ways of specifying how a process should be done, and they're called "programming languages." Glibness aside, getting carried away with making things easy to swap out can make your code hard to understand and debug (and maybe less easy to change than you thought). – Casey Jan 29, 2015 at 18:40

12

I believe this part from Paul Graham's quote is important:

> [...] I'm using abstractions that aren't powerful enough-- often that I'm generating by hand the expansions of some macro that I need to write [...]

It shows that if you can build these abstractions in your language (and you can do pretty much anything with Lisp, which is what he probably has in mind saying the above), you don't need to model them with patterns.

But not everybody uses Lisp, so you will only be able to build abstractions where you can, and use patterns where you can't. As an example, in languages without higher-order functions, these are often modeled using the strategy pattern.

It seems to be a matter of how you look at it: the pattern can be seen as a symptom of a problem (e.g. the lack of higher-order functions), or as a solution of the problem (e.g. making something like higher-order functions possible via the strategy pattern).

Share Improve this answer answered Jan 21, 2009 at 18:49

right on point. the GoF patterns that are so commonly cited as mantra for everybody are really only best for Java programmers. other tools demand other solutions. most will be 'pattern like', but with widely different pros/cons, so using the same names wouldn't be appropriate. – Javier Jan 21, 2009 at 19:25

1 I've used the strategy pattern in a language that has higher-order functions (specifically, Common Lisp). Having "strategies" that inherited from one another, and contained easily accessible state, greatly simplified the problem I was working on. I could had done it all with closures, but it would have been cumbersome. The thing is, I didn't even know I had used the pattern until I described it to a friend and he mentioned the phrase. I looked it up and saw it matched exactly. I don't see how the fact that the technique I used said anything at all about the language I used it in. – Pillsy Aug 20, 2009 at 15:19

1 Pillsy, perhaps we are thinking of rather different things when saying 'strategy pattern'. I was thinking of simple things like a sorting strategy for collections or a matching strategy for queries. No state involved, no inheritance. These are usually done with higher-order functions if possible and with the strategy pattern if not. What you describe sounds like a more complex case, and I can imagine how it can make sense to use objects instead of functions to define strategies in some cases, even if higher-order functions are available. – Fabian Steeg Aug 20, 2009 at 16:03

My thought was that the strategy pattern, for example, is still one that makes sense to use in some circumstances even when you have higher-order functions. In simpler cases, you're just making up for that lack, but there are more complex cases where the strategies generalize in a way that first-order functions don't. If you will, there's a tipping point where objects stop being the poor man's closure and closures start being the poor man's object. – Pillsy Aug 21, 2009 at 14:19

8

[sigh] How many rants against good practices do we have to debunk before people start using their own judgement?

Short answer: if you're doing real work in an object-oriented language, there's a very high probability that you have implemented some of the GoF design patterns to get the job done. Whether you realize you've done this or not is a matter of education, perspective, and introspection. Denying that you've ever done this or that patterns don't exist or aren't "necessary" or are "overused" is ridiculous - unless you never write anything more complex than 'hello world' ;-)

Why, just the other day i had to implement a visitor facade for a singleton just to get the adapter strategy to work :-P

Share Improve this answer Follow

edited Feb 12, 2009 at 8:21

brian d foy
132k ● 31 ● 211 ● 604

answered Jan 21, 2009 at 19:57

Steven A. Lowe
61.1k ● 19 ● 135 ● 204

The term Design Patterns is overloaded and confusing.

5

There is a narrow way to think of it: basically as a collective name for the OO concepts listed in the GoF book, e.g., Singleton, Facade, Strategy. You probably use this definition if you put Design Patterns as a competence on your CV.

One project can easily contain dozens of singleton objects. This pattern pretty obviously benefits from being coded as an instantiable abstraction, e.g., as a macro or abstract class. Likewise for many of the other patterns in the GoF book.

It seems to me that Paul Graham's thesis is: design patterns, being standard ways to repeat yourself, should be codified as instantiable abstractions, and by corollary you should use a language that has a very flexible way to define instantiable abstraction.

But the concept is more general — I think he misrepresents design patterns. Christopher Alexander invented the concept in order to apply it to the design of countries, cities, towns, and houses. Game designers have languages of patterns like Fog-of-War, Asymmetrical-Abilities, and so on.

In this general sense, it becomes much less obvious that design patterns should all be coded as instantiable abstractions.

If you haven't read A Pattern Language, check out the scope of it here. Imagine something like that for the world of software. The GoF patterns would be way down at the bottom of the ToC: they are implementation details.

What other patterns could you find above them?

I guess one chapter would have game-related patterns. It could incorporate both game design patterns and implementation techniques, — and I submit that this is not a clear-cut distinction —, like the Game Loop, the World Entity, the Spatial Hash, the Height-Map Landscape, and so on. These are all design patterns.

Now, in the scope of one game project, you will probably only have one World Entity implementation. Anyone working in any language would have the sense to make this World Entity an instantiable abstraction like a base class.

But what the pattern describes is how this abstraction is formed: it contains some notion of position, it is renderable, and so on. It's not obvious whether this Design Pattern would benefit from being coded as a macro or abstract class in itself — the pattern already describes an abstraction! Would you make a metaclass for world entity implementations? A macro for defining world entity classes, with all kinds of ungodly parameters?

Looking further up the imaginary Table of Contents, we see that these patterns are subordinate to a more general pattern, that of the Game. Is this worth coding as a macro or abstract class? Maybe. But not obviously. We're talking about a game framework, where you define a computer game by passing parameters, filling in the blanks, etc. That could be productive and fun, but it's not the only way to do things.

The Erlang platform has instantiable abstractions for patterns as general as the Server, which is cool. And I guess an Erlang project has as many Servers as a Java project has Singletons.

But for your specific purposes, if you're working in Lisp, or Haskell, or whatever, and writing a server, sometimes it's enough to just follow the Server pattern, implementing it as good old functions and objects, without trying to make an instantiable abstraction of the whole thing.

All design patterns are not low-level textual patterns in your source code.

Share Improve this answer Follow

answered Jan 21, 2009 at 20:33

Mikael Brockman
51 ● 1

5

I think Paul Graham is missing the mark big time on this one. Using patterns isn't about picking up a book that lists them and applying some of them to your code. I would agree that would be a poor choice but that's not what patterns are about.

Design patterns are simply a way of recognizing "Hey, I recognize that this is similar to another problem I've solved before" and learning to create a good abstraction of the solution so it is applicable in appropriate situations.

In fact, odds are, if you've programmed anything of any reasonable size, you've used commonly known design patterns and just didn't know there was a name for it. For example, I was using the "factory" pattern for a long time before I had any clue it was a "factory." Knowing what it's called officially just prevents me from reinventing the wheel again.

Sure some well known patterns are a bit crappy; singleton is definitely suspect. But that one was a clear cut case of a overly clever solution looking for a problem to solve and not doing it very well. But that's why you can find a lot of literature both for and against it—because people recognize it may not be a good solution.

But patterns as a whole are a very good thing, if anything they *encourage* abstraction.

Share Improve this answer Follow

edited Feb 12, 2009 at 8:20
brian d foy
132k ● 31 ● 211 ● 604

answered Jan 21, 2009 at 21:16
Evan Teran
90.3k ● 32 ● 187 ● 243

1 If the language was abstract enough you would #include <factory.h> instead of writing it. – edgar.holleis Jan 22, 2009 at 2:14

2 That sounds like a library issue and not a language issue. And doesn't really effect the usefulness of the pattern. Basically you are saying that the language should supply more patterns for you. Sure, why not. They are still patterns though. – Evan Teran Jan 25, 2009 at 5:10

1 There isn't a library for every possible situation. Sometimes we have to code. Having a template to work from is beneficial. It helps us cover the bases. It helps those that follow us to figure the code out faster. – Tony Ennis Oct 7, 2010 at 3:30

@Tony: of course I understand that, but edgar said "if the language..." which is pointing the finger at the wrong thing. If you want more patterns to be available by default, then that is something that should be added to the standard library, probably not the language itself. – Evan Teran Oct 7, 2010 at 17:30

4

I believe Paul Graham's claim is that design patterns should be expressed in the language. If there is a X pattern, it means that people are forced to rewrite sequences of code to do it, and there should be a language-specific way of expressing it. This could be built into the language (although that's awkward, and can create new design patterns), or it could be automatable within the language (such as Common Lisp macros; C++ templates, particularly when used in weird ways, can do much the same sometimes).

For example, let's consider a much simpler pattern: the incrementing loop. Having to do something on each element in a sequence is quite common, and if we didn't have something corresponding to a for statement we would doubtless adopt some common coding convention to express it, and somebody (or some foursome) would package that together with other similarly elementary constructs and write a book about it.

Similarly, if using a Factory pattern, either a language could include a Factory feature, or it could be possible to automate it within the language. Specifically, Paul wants to be able to write a Lisp macro to implement it, rather than write Factories himself over and over again.

The danger with using a design pattern is that it's a familiar code framework we keep typing, and therefore we have chunks of code that we tend not to read and write with full attention. The inconvenience is that they're code we have to type again and again, and the feeling that the rewriting really should be automated.

Whether patterns are an essential part of using languages, like the GoF says, or can be abstracted away, like PG says, is fundamentally an empirical question, probably best solved by looking for languages where the patterns themselves are unnecessary and observing to see if they really need new patterns.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:43

David Thornley
57k ● 9 ● 95 ● 158

4

Design Patterns are defined as the following (from wikipedia but I have read the same in some book too)

> In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

If you look at applying patterns right from the offset without getting into the problem domain extensively can lead to problems. Use design patterns as a guide or more so as a tip/hint to solve a problem. Forcefully applied design pattern will surely not give a solution thats abstract enough. It is often seen that in Enterprise software a variant or a combination of a few design patterns are used..kind of hybrids. If you say you have never used design patterns then you would be happy to know that something like a foreach loop is actually an interator pattern and you can hunt for more obvious implementations near you!

Share Improve this answer

Follow

answered Jan 21, 2009 at 18:43

Perpetualcoder

13.6k ● 9 ● 66 ● 101

4

In a nutshell patterns are just formalized ways of solving common tasks in various languages. As such there were "design patterns" even before someone coined the term. For every language there are "best practices" and design patterns are really just that - a set of best practices for problems that occur on a regular basis.

One of the main benefits of labeling the patterns is that the shared terminology allows us to talk about abstract concepts.

Obviously design patterns can be misapplied or overused in any given context, but I really fail to see how design patterns as a concept can be a problem.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:49

Brian Rasmussen
116k ● 34 ● 224 ● 323

3

When I programmed in LISP I didn't use GoF design patterns at all, it just was not applicable for the programs that I was writing. Now I'm programming in C# and run across situations all the time where a design pattern actually simplifies the program being written.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:17

DavGarcia
18.8k ● 14 ● 61 ● 97

2

There is a time and place for everything. I've seen code with too many patterns, which would agree with your point, but also with not enough patterns, making the code difficult to maintain and error prone when making changes.

Generalizing something like this is going to be logically flawed, but if you rephrase your question to something along the lines of "can there be too many patterns in code that is symptomatic of a bigger problem, that is, insufficient levels of abstraction" then the answer would be yes. Is it always the case? No.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:14

Elie
13.8k ● 25 ● 78 ● 128

2

There are a lot of maxims we use (and which are common around SO) that, if you trace them back to the source, originally were asserted with reservations; as in " ... when you are developing using ..." This is one of those cases. Patterns fit well when you are doing OOP. YMMV when you are in another paradigm.

"Refactoring" is another similar case.

When I'm writing SQL, the "patterns" part of my brain switches off.

Note in your quotation: "I'm generating by hand the expansions of some macro that I need to write.". So he recognizes patterns and abstracts them into a macro - he doesn't recode it another way. Just good-ol' DRY. It's not the patterns, it's failing to do the right thing with the ones we find. I'd say his comment could go into any wiki entry expanding on the virtues of DRY, and how to own them. GoF would concur completely - recognize the desgin patterns, then use what you know about them to implement (or refactor) them appropriately.

Share Improve this answer Follow

edited Jan 21, 2009 at 18:36

answered Jan 21, 2009 at 18:24

dkretz
37.6k ● 13 ● 83 ● 140

2

During the design phase of my programs, I often **break down the problem first**, and once I have it broken down then I can easily start to recognize patterns inherent within the design. At that point, I can begin to apply well-known design patterns so that when it comes time to communicate or implement the design, I have a common language I can use for communication, and hopefully I can re-use some generic objects that were previously implemented during the implementation of a similar pattern.

Often times, when design patterns are applied incorrectly, it's because the process happens in reverse. Programmer Joe (my apologies to those of you named Joe) reads a book on Design Patterns, and says "Ok, I understand Design Pattern X, now how can I apply it to my application?" **This is wrong.**

Design patterns can be a mighty weapon, but like anything else, they should be used appropriately, and the programmer should always be prepared to incorporate some original thought into his design, as well.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:38

jeremyalan
4,786 ● 2 ● 31 ● 39

2

I think that sometimes when you learn a pattern, you don't just learn the pattern. But you aquire a new view at the problem domain which you might not have had before. And maybe its exactly what you wanted.

Without context for the "problem" we can't say if design patterns are the solution or not.

Share Improve this answer Follow

answered Jan 21, 2009 at 20:53

JSmyth
12.2k ● 3 ● 25 ● 18

There is never a *need* to use design patterns—you can always write code that ignores everything that has been

2

learned about coding up to now—it may even work. But design patterns can make things easier, giving you a shared language for discussing design.

Diesign patterns don't represent too little abstraction—they are an attempt to increase the level of abstraction. You can say "this bit is a Visitor" rather than "here is some code that recursively traverses a tree of objects, performing on operation on them."

Share Improve this answer Follow

edited Feb 12, 2009 at 8:19
brian d foy
132k ● 31 ● 211 ● 604

answered Jan 21, 2009 at 22:21
DJClayworth
26.8k ● 9 ● 58 ● 80

2

Patterns are the problem if they aren't a solution. By that I mean: if a pattern is introduced for the pattern's sake and not to solve a real, existing design problem in the application, it's likely to cause issues rather than solve or prevent them.
I keep telling people here at work that the Gang of Four book (Design Patterns: Elements of Reusable Object-Oriented Software) is a reference, not a handbook for good design.

Share Improve this answer Follow

answered Feb 12, 2009 at 8:24
Joris Timmermans
11k ● 2 ● 52 ● 78

Or, more likely, they are introduced as an 'almost fitting the' pattern. Octagonal peg in a square hole. – Jay Sep 3, 2010 at 0:04

1

Yes, pg is right there. If you see similar code throughout your application, it means that you are missing an abstraction of that code. Less code is better.

Here is another article on the subject: http://blog.plover.com/prog/design-patterns.html

Share Improve this answer Follow

answered Jan 21, 2009 at 18:13

jrockway
42.6k ● 9 ● 64 ● 87

1

I would rather work with too many patterns than none at all. That said, the proper balance is of course the goal.

Share Improve this answer Follow

answered Jan 21, 2009 at 18:16

Jim Petkus
4,508 ● 1 ● 27 ● 19

1

That sentence makes no sense: *design patterns are a sign of not enough abstraction* As design patters **are** an abstraction! Following the answers here I agree that programming languages should have a way to express design patterns. However that is obviously not possible for every situation.

Share Improve this answer Follow

answered Jul 8, 2010 at 15:21

Angel O'Sphere
2,660 ● 20 ● 18

0

I recall an interview with one of the GoF where someone asked about the common claim that design patterns are tools to replace features which are missing from a language. The author argued that A) any language has design patterns (not the same ones) since design patterns are tools built to work around language idiosyncrasies and B) it is not feasible to get rid of their need by adding more features to a language.

Share Improve this answer Follow

edited Jan 21, 2009 at 19:32

answered Jan 21, 2009 at 18:17

Brian
25.8k ● 18 ● 86 ● 178

Wow...a feasibility study that came up with a "No this isn't feasible"? I thought those didn't exist ;)
– Restore The Data Dumps Again Jan 21, 2009 at 18:47

"design patterns are tools built to work around language idiosyncrasies" that means that the GoF patterns are limited to Java and similar languages (C#). for other languages, there might be recognizable patterns; but won't be exactly those. i'd love to stop hearing those new names for old ideas! – Javier Jan 21, 2009 at 19:21

0

Design Patterns are solutions for common problems.

But first, you need to know what or where is the problem. And that's the point where people fail to use the patterns correctly.

Share Improve this answer

Follow

edited Feb 12, 2009 at 8:21

brian d foy
132k ● 31 ● 211 ● 604

answered Jan 21, 2009 at 19:42

DonOctavioDelFlores
363 ● 1 ● 7

0

I've also not yet found a use for GoF "Design Patterns" in my code. The coderati appears to have latched on to the GoF book, and it is now expected in the majority of companies that you know them and apply them. Just today I had an interview and was asked which patterns I knew and had used, and how I would apply them to an enterprise app for a major bank.

The generic idea of learning from our previous projects obviously makes sense. What doesn't make sense is the hype and cult around the specific GoF patterns, people mistaking them for good-coding practice, and being expected to love and embrace them in order to be called a competent developer.

So to answer your question, I'd say the GoF idea of design patterns is misunderstood and overused by most. We need to mature into a higher-level use of design patterns as a general learning tool that applies to the way we learn and refine OO--not just as 20 cookie-cutter ideas to memorize and impose on programs. ***There is no silver bullet.***

Share Improve this answer Follow

answered May 3, 2009 at 1:00

alchemical
13.9k ● 24 ● 84 ● 111

0

Programming languages, like spoken ones, provide you with the vocabulary to say whatever you want. Patterns only describe a way to say things that allows people to know what you're talking about at a higher level.

If you'll indulge me in a metaphor: Writing music is a common 'problem' and music is often (obviously not always) loosely composed as some kind of variation on the following:

verse chorus verse chorus verse chorus chorus

Which, indeed is a 'pattern'. You still need to write the song though and not every song written will work well using this pattern.

The point I'm trying to get at is that patterns aren't a plug-play-done solution for programming, or music. They're a guideline to get you started and a springboard from which you can make something that suits your need.

Share Improve this answer Follow

answered May 15, 2009 at 22:22

Steven Evers
17.2k ● 22 ● 81 ● 132