Managed Chaos

Naresh Jain's Random Thoughts on Software Development and Adventure Sports

« HelloWorld: Haskell

Different approaches to OO Design »

Why are Design Patterns Important?

Short: They help us solve recurring design problems. Note that : design patterns don't solve the problem themselves, they help us solve the problem.

Detailed answers:

» **Communication, Learning and Enhanced Insight**: Over the last decade design patterns have become part of every developer's vocabulary. This really helps in communication. One can easy tell another developer on the team, "I've used Command pattern here" and the other developer understands not just the design, but can also easily figure out the rationale behind it. Design Patterns really help in learning, esp. when you are new on a project. Also this helps in providing developers with better insight about parts of the application or 3rd party frameworks they use.

» **Decomposing System into Objects** : The hard part about OO Design is finding the appropriate objects and decomposing a system. One has to think about *encapsulation, granularity, dependencies, flexibility, performance, evolution, reusability and so on.* They all influence decomposition, often in conflicting ways. Design Patterns really helps identify less obvious abstractions. These objects are seldom found during analysis or even the early design, they're discovered later in the course of making a design more flexible and reusable.

» **Determining Object Granularity**: One thing I struggle a lot with is finding the right level of abstraction and granularity. Design patterns helps in coming up with objects with different levels of granularity that makes sense.

» **Specifying Object Interface** : Identifying the right interface and the relationship between various interface is not a one-shot activity. Usually it takes several iterations to identify the right composition of interfaces. Forget interfaces, most of the times, coming up with a method signature can also be quite tricky. Design Patterns really helps in this area.

» **Specifying Object Implementation** : How should we implement an Object? Given an interface there could be multiple concrete classes of that type, each one can have very different implementations. Design Patterns provide guidance like Program to an interface (type) not an implementation (concrete class) which can result in really good OO code.

» **Ensuring right reuse mechanism** : When to use Inheritance, when to use Composition, when to use Parameterized Types? Is delegation the right design decision in this context? There are various questions that comes to a programmer's mind when they are trying to design highly reusable and maintainable code. Knowledge of design patterns can really come handy when making such decisions.

» **Relating run-time and compile time structures** : An object oriented program's run-time structure often bares little resembles to this code structure. Sometimes looking at the code does not give us the insights into run-time structure. Knowledge of design patterns can make some of the hidden structure obvious.

» **Designing for change** : We all know that lack of continuous refactoring and design that doesn't take change into account risks major redesign in the future. Over the years we've also learnt that big upfront designs can't standup against the constant change/additon of software requirements. We've leant grouping elements with similar change life cycle together yields in far more flexible and extendable design. If we think some behavior or element of behavior is most likely to change, we try to abstract that behavior in one place. While we understand these concepts are important design patterns really make it possible to design such systems. Each design pattern lets some aspect of the system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change.

Related posts:

1. Patterns and Anti-Patterns: Acceptance Testing with FitNesse Patterns: Organizing Tests – Allowing customers to add new tests without breaking the build. This is possible by using the...
2. While the debate on upfront v/s JIT design goes on… Most software "engineers" (engineers in double quotes) don't set out to create "bad designs". Yet most software eventually degrades to...
3. Retrospectives: Anti-patterns It is important to understand what a retrospective is and why we need one before we jump into antipatterns. Based...
4. Fred George's Object Bootcamp Day 13 Patterns: Gives us a very powerful vocabulary to explain or talk about the program. In agile we need to be...
5. RECAP: Patterns Model View Controller MVC is nothing but the Mediator pattern. The controller is the mediator in the MVC world. Usually,...
6. Storytests and Domain Driven Design with FitLibrary by Rick Mugridge I'm organizing the next Agile Philly User Group [http://groups.yahoo.com/group/agilephilly] meeting. Following are the details: Date : 15th Feb Venue :...
7. Simple Design And Testing Conference in Philly Greatly influenced by the CIT conference, with the help of folks from the Agile Philly user group, I'm planning to...
8. Mentoring College Grads If you want to groom/mentor a college grad (fresher) to be a software developer in any language what are the...
9. Distributed Agile presentation at the Round Table CTO meet Recently I met Mr. Joel Adams, CEO of Devon Consulting, at the Agile Philly user group. He organizes a meeting...
10. eXtreme Programming (XP) 2nd edition As XP is catching pace, the 12 practices of XP have grown to 25…26…(whatever). Though the basic principles and core...

This entry was posted on Friday, September 5th, 2008 at 7:23 PM and is filed under Agile, Design. You can follow any responses to this entry through the RSS 2.0 feed.

You can leave a response, or trackback from your own site.

2010 Agile Coach Camp agileindia agile india agileindia2012 Agile India 2012 AgileIndia2013 Agile India 2013 Agile India 2013 Workshops agileindia2014 Agile India 2014 agile mumbai 2010 ASCI Attendees Profile Code Coverage code smells Conference Continuous Deployment Continuous Integration cyclomatic complexity Design duplicate code Evolutionary Design Extreme Programming Industrial Logic Lean Lean Startup LinkedIn Open Source pair programming post modern agile primitive obsession smell Refactoring Refactoring Teaser Scrum SDTConf simple design TDD technical debt Test Driven Development Testing unit testing user stories video

Recent Comments

» Fahad M Rafiq

WordPress is a highly customizable CMS that have been adopted by millions around the web....

Change WordPress Table Prefix