

## Object - Oriented System Development Using UML, Java and Patterns

Professor. Rajib Mall

Department of Computer Science and Engineering

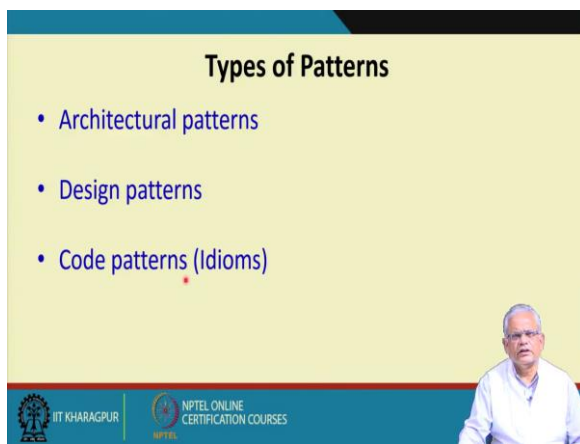
Indian Institute of Technology Kharagpur

Lecture 38

### GRASP Pattern: Introduction

Welcome to this ~~session~~! In the last ~~session~~ we had some introductory discussion about design patterns. Let us [try to](#) continue ~~our discussion~~ bit of introductory discussion on design patterns, broad idea about what are patterns, how do they help and so on and then we will start discussing about the GRASP pattern in ~~the section~~ [this session](#).

(Refer Slide Time: 00:45)



In last section we have remarked ~~that~~ there are ~~3~~ [three](#) broad types of patterns which are in use by software developers. The architectural patterns, the design patterns and the code patterns which are also called as idioms.

(Refer Slide Time: 01:01)

**Architectural Patterns**

- **Architectural designs concern the overall structure of software systems.**
  - Architectural designs cannot directly be programmed.
  - Form a basis for more detailed design.
- **Architectural patterns:**
  - Relevant to providing high-level solutions to large problems.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

*(Speaker image visible in bottom right corner)*

The architectural patterns deal with the overall design, high level solutions.

(Refer Slide Time: 01:10)

**Design Patterns**

- **A design pattern:**
  - Suggests classes in a design solution.
  - Also, defines the interactions required among the classes.
- Design pattern solutions are described in terms of:
  - Classes, their instances, their roles and collaborations, skeletal code.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

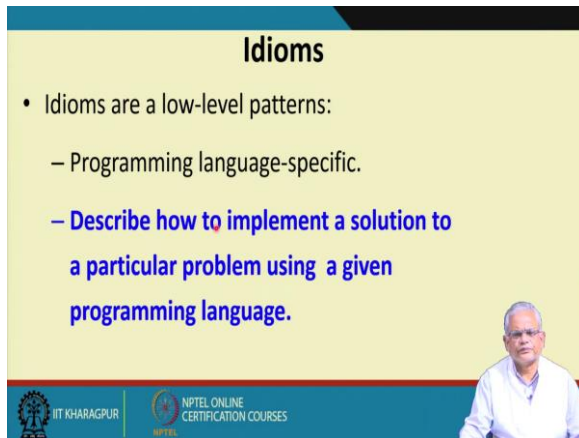
*(Speaker image visible in bottom right corner)*

Whereas the design patterns, they provide design solution to specific problems. They suggest a class diagram and also define interaction among the classes in the form of an interaction diagram and also we will have the object instances, we might give an object diagram, the different roles of the classes, the collaborations between the classes and also we will give java skeletal code for the GOF patterns.

For the GRASP patterns we will not really have the java code because the GRASP patterns are more of common sense, design common sense and if we have those common sense, design common sense, we can come up with good design but they do not really translate the

code or it may be not very meaningful to give code for the GRASP patterns, but for the GOF patterns, we will discuss about 10 to 15 GOF patterns. We will have the java code for sample uses of the specific GOF patterns.

(Refer Slide Time: 02:43)



The slide is titled "Idioms" and contains the following text:

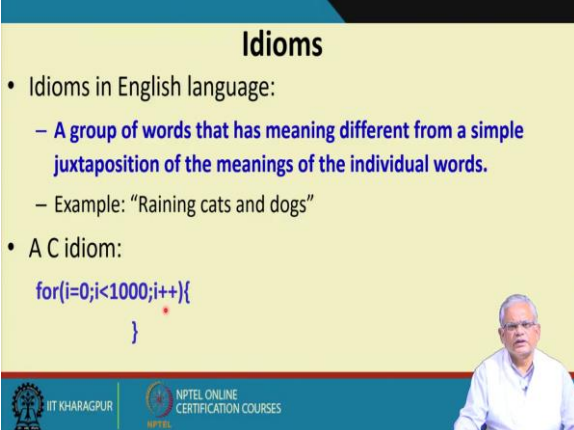
- Idioms are a low-level patterns:
  - Programming language-specific.
  - Describe how to implement a solution to a particular problem using a given programming language.

The slide also features the IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES logos at the bottom left, and a small portrait of a man in a white shirt at the bottom right.

Besides the architectural patterns and the design patterns, we have the idioms or the code patterns. These are low level patterns in the sense that these are programming language specific, these are not design patterns. The architectural patterns are high level patterns, the design patterns solve specific design problems, whereas the code patterns are code specific. They describe how to implement a solution for a given problem in a certain programming language.

But then, these are also very useful to the programmer, because if he knows the idioms to write code for a specific problem, he just falls back on the idiom and writes good code effortlessly. All programmers learn the idioms, the hard way, they keep on writing code until they are or they look in the books and examples and then they find that things are done in a certain way in certain programming languages and then they start using it without knowing that they are using an idiom.

(Refer Slide Time: 04:25)



The slide is titled "Idioms" and contains the following content:

- Idioms in English language:
  - A group of words that has meaning different from a simple juxtaposition of the meanings of the individual words.
  - Example: “Raining cats and dogs”
- A C idiom:

```
for(i=0;i<1000;i++){  
    }  
}
```

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small portrait of a man in the bottom right corner.

But what exactly the meaning of an idiom? If you look up the English dictionary or a grammar book, because grammar book will have a chapter on idiom. What does the term idiom mean? If you look up a dictionary, it will say that it is a group of words that has meaning different from a simple juxtaposition of the meanings of the individual words. An idiom is a group of words that has meaning different from simple juxtaposition of the meaning of the words.

For example, an idiom is it is “Raining cats and dogs” . It is the idiom is not simply putting the meaning of each word here, that outside you go and find that it is raining cats and dogs. But the implication is that it is raining very heavily. But then you have to use this specific words to be able to use this idiom, you have to exactly use the same thing as it is here, you cannot just change it and say that it is raining rats and lions.

People will laugh and that does not mean much. So, the idiom are a group of words that are used as it is and they imply some meaning which is different from the simple juxtaposition of the meanings of these different words. The same thing in the programming languages, a programmer learns idioms by reading books, it does not know that it is an idiom but he finds that programs are written in some way and later while writing he realises that it is a good way.

For example, to access all the elements of the array of 1000 elements in C, C++ or Java, you will write  $i \equiv 0; i \leq 1000; i++$ . That is for array traversal or looking at each element of an array of 1000 elements, you do not think that should I use a do-while loop, should I use a while-do loop, you use the for loop by

default for array traversal and not only that you said `i = 0`, you do not even think that should I write `i = 1; to i <= 1000; i++`.

You straight away write `i = 0; i < 1000; i++, i++`, this is a standard idiom which all C, C++, java programs have mastered without knowing that this is a idiom. There are large number of idioms for any programming language which if you learn will improve your programming skills, but then for this course this is not the focus, we will not discuss about the idioms, we will discuss about the design patterns.

(Refer Slide Time: 808:05)

The slide is titled "Patterns versus Idioms" and contains the following text:

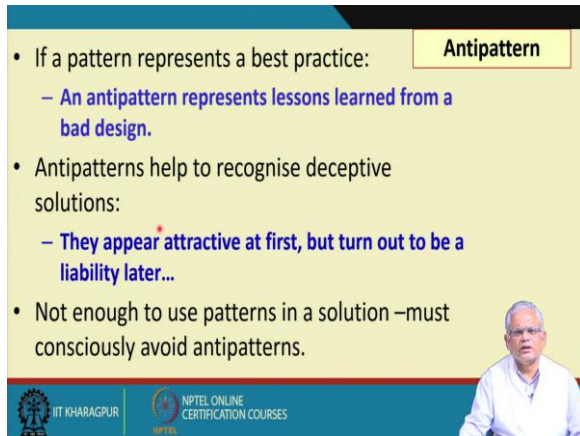
- A pattern:
  - Describes a recurring problem
  - Describes a core solution
  - Used for generating many distinct designs
- An Idiom though describes solution to a recurring problem, is rather restricted:
  - Provides only a specific solution, with fewer variations.
  - Applies only in a narrow context
    - e.g., C++ or Java language

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small portrait of a man in the bottom right corner.

The design pattern versus the idioms, there are very large number of differences but then there are some common things. One is that both idioms and design patterns, they try to solve some commonly occurring problem. They also offer a solution, a core solution which we may adapt a little bit for a specific problem in hand, but there are large number of differences as well. The idioms, they provide a specific solution with very few variations.

For example, the array access that we are saying, is that always you start writing for `i = 0`, `i < 1000`, `i++` with very few variations, maybe you will write `i = 10`, `i < 10000`, `i++`, etc. only small variation. And also this apply in a very narrow context, if we are using C++, and you are trying to do something, you will use some idiom and for java language coding you will use some idioms. Whereas, the design patterns have broad applicability irrespective of the programming language.

(Refer Slide Time: 0-9:39)



**Antipattern**

- If a pattern represents a best practice:
  - An antipattern represents lessons learned from a bad design.
- Antipatterns help to recognise deceptive solutions:
  - They appear attractive at first, but turn out to be a liability later...
- Not enough to use patterns in a solution –must consciously avoid antipatterns.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

*(Speaker's video feed is visible in the bottom right corner of the slide)*

Now before we discuss about the design patterns, let us look at the antipatterns. The antipatterns are also important in the sense that they describe design solutions which appear to be good initially but later turn out to be a bad decision. This comes from the experience of designers who use this solution thinking that these are good solutions but later found that it is really a bad decision. If we know the antipatterns, we will consciously try to avoid those solutions.

We will be able to recognize the deceptive solutions, deceptively good-looking solutions which appear attractive but latter turn out to be bad decision. It does not really help to come up with the exact solution but it tells us that the solution that we are thinking of whether ~~it is~~ it will turn out to be a bad solution. So, the antipatterns are also important but since our number of hours limited, we will not really discuss about antipatterns.

(Refer Slide Time: 11:03)

**Patterns versus Algorithms**

- **Are patterns and algorithms identical concepts?**
  - After all, both target to provide reusable solutions to problems!
- Algorithms primarily focus on solving problems with reduced space and/or time requirements:
  - **Patterns focus on understandability and maintainability of design and easier development.**

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

*(A small video inset of a man speaking is visible on the right side of the slide.)*

But you must be clear about one thing, that the intension of both patterns and algorithms is to help reuse of good solutions. The algorithms, some of the algorithms have surprising solutions, which is very difficult for somebody to think of and come up. But then if you know the algorithm, you can make use of it, reuse somebody's solution. For example, the quick sort algorithm or the Dijkstra's shortest path algorithm and so on.

These are worked out by some very eminent persons in the area and for a normal programmer, it becomes very difficult to come up with those algorithm unless he knows them. A and therefore, the algorithms help to reuse knowledge of the experts. Same is with design patterns, they help reuse the good solutions worked out by experts. But then, what are the differences between the patterns and algorithms we must be aware? One is that the focus of the algorithms is to come up with solutions which take less time, they are time efficient and also space efficient.

But finally the objectives are different. The algorithms mainly try to achieve good efficiency, runtime efficiency and space efficiency. Whereas, the patterns focus on maintainability, understandability, testability and so on.

(Refer Slide Time: 13:56)

**Pros of Design Patterns**

- Help capture and disseminate expert knowledge.
  - Promotes reuse and helps avoid mistakes.
- Provide a common vocabulary:
  - Help improve communication among the developers.
  - “The hardest part of programming is coming up with good variable and function names.”

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

If we know the design patterns, we will be able to avoid mistakes, come up with good design, reuse other solution what others have thought of. ~~Once it also ones~~ we know the patterns, we can communicate well with the other professional designers in a company environment, we can understand what others are speaking, if we do not know the pattern and somebody says that he is using composite pattern, it will mean very little unless we know the composite pattern.

And also, these gives the names which form part of one's vocabulary and remember that somebody had told that the hardest part in any programming is to come up with good variable and function names. ~~A~~ and here, we will see that the pattern names really help. These are good and thoughtfully given names and help us to communicate well, document well the designs.

(Refer Slide Time: 15:16)





It reduces the number of design iterations, helps to improve the design quality and the designer's productivity.

(Refer Slide Time: 15:25)




And the good solutions they have come from use of abstraction, encapsulation, different principles which we discussed, the solid principles that is the SRP, Single Responsibility, Open Closed Principle, [Least of Liskov](#) Substitution Principle, Interface Segregation Principle and the Dependency Inversion Principle. The separation of concerns, coupling and cohesion issues, use of [d](#)Divide and conquer and so on. These are all good principles and if you use design patterns, you are implicitly, making use of these good principles because the design patterns have been founded on these important principles.

(Refer Slide Time: 16:14)

### Cons of Design Patterns

- Design patterns do not directly lead to code reuse.
- To help select the right design pattern at the right point during a design exercise.

**– At present no systematic methodology exists.**



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES


But are there any problems of the design patterns? ~~W~~We must be aware before we start using the design patterns. One thing is that just knowing the design pattern does not help in code reuse. You have to make use of the design patterns but unfortunately, there is no systematic methodology that how one would go about spotting these places where you will use the design pattern, it will largely intuition and experience.

As you solve more problems using the design patterns, you see that you understand where exactly to use it. Possibly this is a shortcoming of the design patterns that we have no systematic way as to where to use a design pattern.

(Refer Slide Time: 17:11)

### Why learn Design Patterns?

- **Your own designs will improve**
  - borrowing well-tested ideas
  - pattern descriptions contain some analysis of tradeoffs
- **You will be able to describe complex design ideas to others**
  - assuming that they also know the same patterns
- **You can use patterns to “refactor” existing code**
  - Improve the structure of existing code without adding new functionality
- **You can understand why some aspects of Java language are that way.**



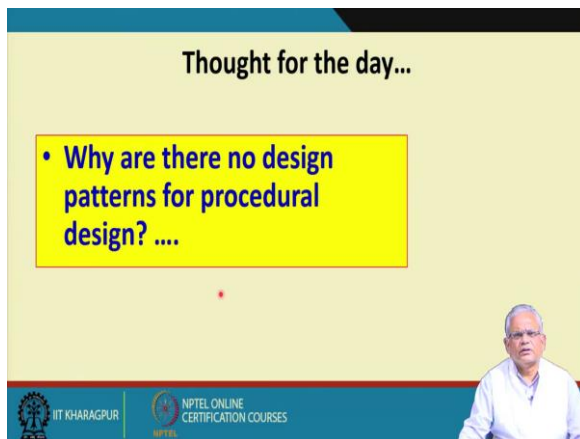
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

At the end of our discussion on design patterns, the objective is that once you know the design pattern, your own design will improve. You will be able to use well

tested ideas and you will see that some analysis and trade off that have been done, you can make use of that, you can describe your complex design ideas to others assuming that they also know the patterns.

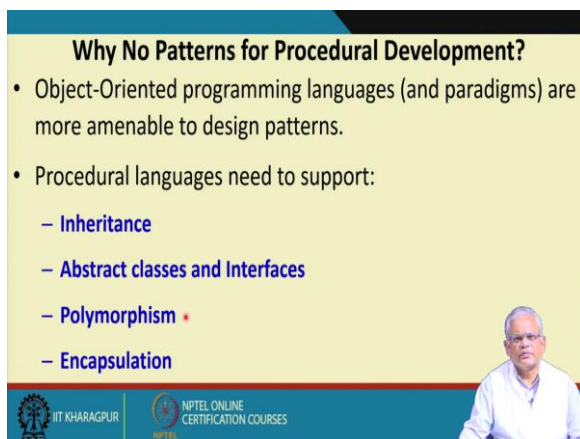
Given a design, you will be able to spot where to improve the design by using the patterns and you can understand why some aspects of the java language and other languages are the way that they are. We will also point out few places where these have been directly implemented in the java programming language. You have been programming in java without knowing the exact pattern that they have used and as you understand the pattern, it will give a new meaning as to why ~~you were~~ you were so far doing somethings in certain ways.

(Refer Slide Time: 18:41)



But one thought you can think about that for object-oriented design, we say that there are design patterns which you can understand and then use it to improve your design and so on. But for procedural designs, hardly there are any design patterns and nobody even has proposed a pattern, why? [If you reflect on this enough you will come up to the solution.](#)

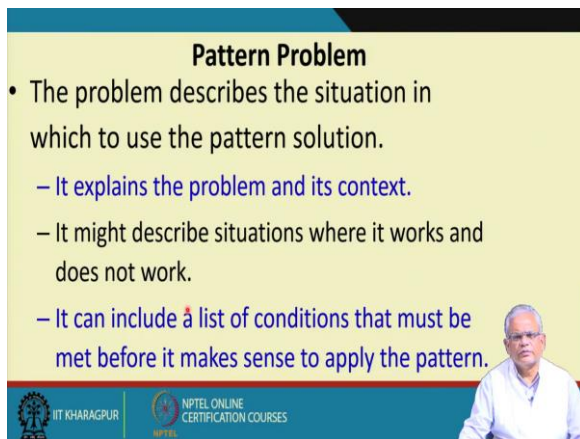
(Refer Slide Time: 19:25)



~~If you reflect on this and of you will come up to the solution~~ That the object oriented programming languages have certain features using which you can come up with good designs and these features are absent in procedural languages and therefore, our procedural languages ~~it~~ becomes very difficult, have some guide lines of a good designs other than some general guidelines. No specific patterns are there.

The pattern solutions that we will discuss make use of inheritance, abstract classes, interfaces, polymorphism, encapsulation and so on. A and such features are not present in the procedural languages. This is one possible reason why there are no ~~patterns~~ design patterns. design patterns were procedural languages.

(Refer Slide Time: 20:19)



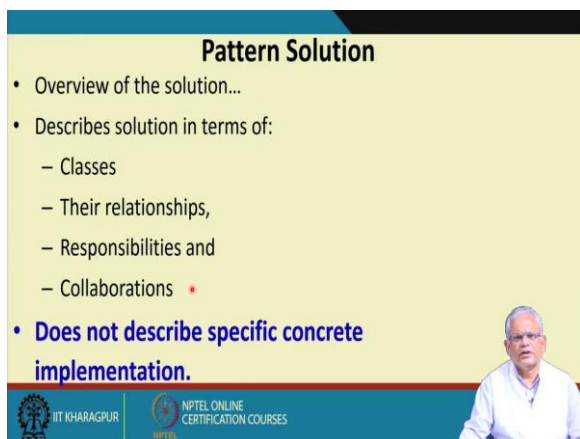
**Pattern Problem**

- The problem describes the situation in which to use the pattern solution.
  - It explains the problem and its context.
  - It might describe situations where it works and does not work.
  - It can include a list of conditions that must be met before it makes sense to apply the pattern.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now the pattern problem all the patterns we will discuss, we will give a name to the pattern we will discuss about the problem that it tries to solve the context in which it occurs, we will also mention the situations where it will work and it will not work and the conditions that must be met before we can use the pattern.

(Refer Slide Time: 20:52)



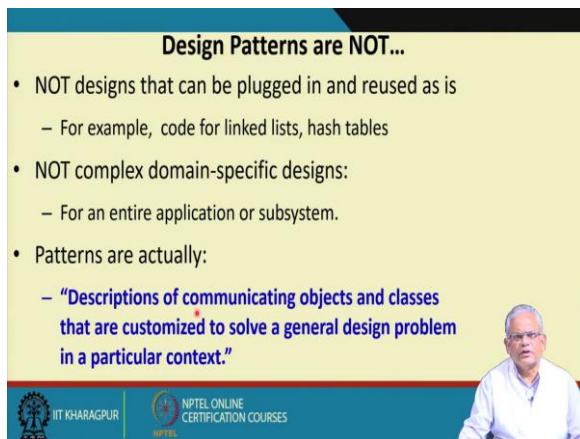
**Pattern Solution**

- Overview of the solution...
- Describes solution in terms of:
  - Classes
  - Their relationships,
  - Responsibilities and
  - Collaborations
- **Does not describe specific concrete implementation.**

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And the pattern solution will give an overview of the solution then will give the class, class relations, responsibility, collaborations and one thing we must remember that it does not give a concrete solution which we just pick and plug, it gives a broad solution which we have to tailor it and adopt it to our specific problem.

(Refer Slide Time: 21:23)



**Design Patterns are NOT...**

- NOT designs that can be plugged in and reused as is
  - For example, code for linked lists, hash tables
- NOT complex domain-specific designs:
  - For an entire application or subsystem.
- Patterns are actually:
  - **“Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.”**

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

And we must be clear that it is not plugged and re-used. Unlike algorithms or [libraries](#), we do not re-use them as it is and this are only parts it is not for an entire application or sub-system ~~a~~. And the patterns are actually communicating objects and classes that are customized to solve some broad design problem in a particular context.

(Refer Slide Time: 21:57)



**GRASP Patterns**

The slide features a large yellow box with the text 'GRASP Patterns' in the center. It includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

Now let us start discussing about the GRASP patterns. We said that these are the simplest but then widely used in almost every design we do from ~~now~~ now; we will be able to use the GRASP pattern. It is a very intuitive and extremely simple. ~~We~~ We will discuss about 8, 9 GRASP patterns but hardly we will need half an hour that is one session or slightly more than that. ~~But~~ GOF patterns, we will find that there are more sophisticated but that their applicability is less than GRASP pattern.

(Refer Slide Time: 22:54)



The slide is titled "GRASP Patterns" and contains the following content:

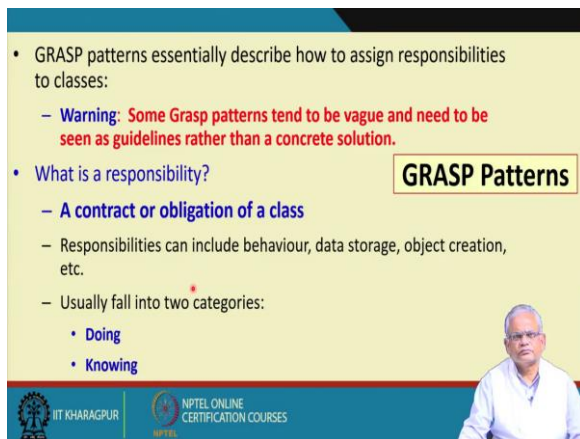
- GRASP: **Generalized Responsibility Assignment Software Patterns:**
  - Larman, "Applying UML and Patterns"
- GRASP patterns can more accurately be described as best practices:
  - If used judiciously, will lead to maintainable, reusable, understandable, and easy to develop software

The slide also features a small image of the book cover for "Applying UML and Patterns" by Eric Larman, and a small portrait of a man in the bottom right corner. Logos for IIT Kharagpur and NPTEL Online Certification Courses are visible at the bottom.

The GRASP stands for Generalized Responsibility Assignment Software Pattern. In our discussion so far, we have discussed about responsibility, responsibility of a class and we had remarked that responsibility of a class is basically the method it needs to support. The uses here ~~is~~ are the same that the patterns discuss a deal with which class should have what methods. Largely based on Larman, Larman had proposed this in his book Applying UML and Pattern but then as we have been remarking that these are actually best practices.

It is not really a solution, sophisticated solution which we make use in our design but these are best practices if we use judiciously, it will lead to maintainable, reusable, understandable and easy to use software. So, this the book applying UML and patterns based on which we will discuss the GRASP patterns.

(Refer Slide Time: 24:31)



The slide is titled "GRASP Patterns" and contains the following text:

- GRASP patterns essentially describe how to assign responsibilities to classes:
  - **Warning: Some Grasp patterns tend to be vague and need to be seen as guidelines rather than a concrete solution.**
- What is a responsibility?
  - **A contract or obligation of a class**
  - Responsibilities can include behaviour, data storage, object creation, etc.
  - Usually fall into two categories:
    - **Doing**
    - **Knowing**

The slide also features a video feed of a speaker in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

The GRASP patterns deal with how to assign responsibility to classes. But then some patterns tend to be bit vague and are to be seen as guidelines rather than concrete solutions unlike the GOF patterns where we have specific problems and very specific solutions. Here, will see that these are rather some of this are broad guidelines rather than being real design patterns.

Responsibility we had said that this are methods of a class, but then this can be also information present in a class. Not only doing but also knowing that is the data present in the class ~~that~~ is also a responsibility. So, these patterns they assign responsibility that is which class should have something, some information and also what are the methods that will be supported by the class.

(Refer Slide Time: 25:51)



**GRASP Patterns**

- **Creator**
  - Who creates an object?
- **Information Expert**
  - Which class should be responsible?
- **Low Coupling**
  - Support low dependency and increased reuse
- **Controller**
  - Who handles a system event?
- **High Cohesion**
  - How to keep complexity manageable?
- **Polymorphism**
  - How to handle behavior that varies by type?
- **Pure Fabrication**
  - How to handle a situation, when you do not want to violate High Cohesion and Low Coupling?
- **Indirection**
  - How to avoid direct coupling?
- **Law of Demeter (Don't talk to strangers)**
  - How to avoid knowing about unassociated objects?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There are 9 of these. Creator ~~this~~ deals with who creates an object. Expert, which class is responsible for doing something ~~like~~. ~~Low coupling~~, how to have low coupling and increase reuse. Controller, who handles an event may be a user request or a system event who, which class is responsible to handle it. High Cohesion, how to keep the complexity manageable. Polymorphism, how to handle ~~behavior~~~~behaviour~~ that varies by the type of the class. Pure Fabrication, how to handle a situation when you do not want to violate high cohesion and low coupling?

Your design has quotation and low coupling and you want to avoid. You do not want to violate ~~the~~~~is~~ principal high cohesion and low coupling. how do you go around and handle the situation and that is given by the pure fabrication pattern. Indirection, how do you handle direct coupling between classes.

~~A~~ and then finally the low of Demeter which basically do not talk to strangers that is how can a class avoid communicating with unrelated classes or ~~unassociated~~~~associated~~ objects.