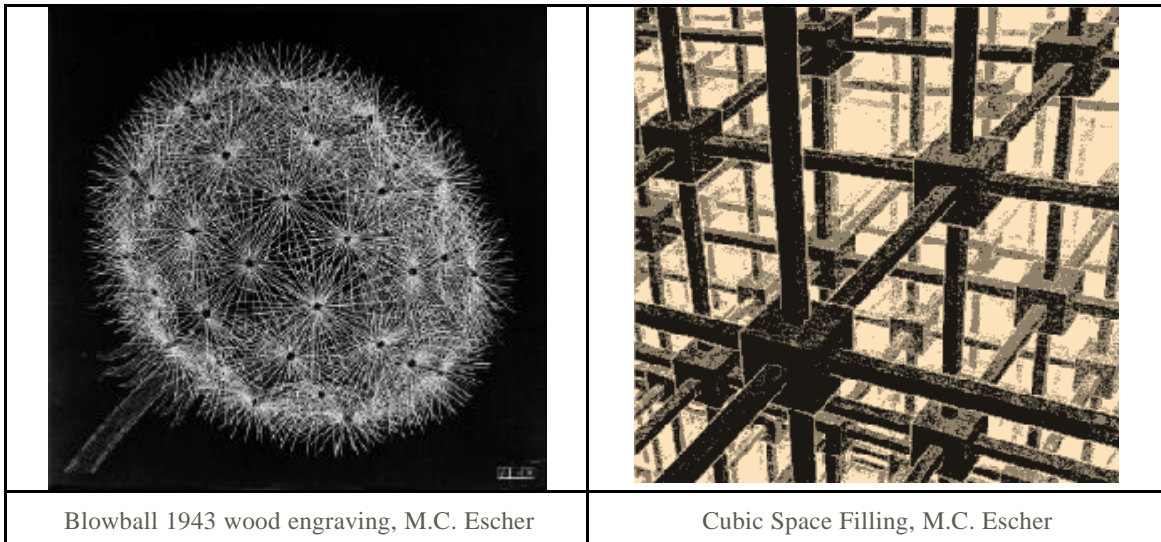


## **'How to develop a software architecture: pattern languages'**

- 1.- What a software architecture is and what it is for.
  - 2.- Architectures and Methodologies.
  - 3.- Types of Systems.
  - 4.- Pattern Languages.
  - 5.- New Designs.
- 

### **1.- What a software architecture is and what it is for.**



The concept of architecture is used in a wide sense and in very different fields, so that its meaning is somehow ... vague.

In the software field, the architecture identifies the more important elements of a system as well as their relationships between them. So, it show us a global vision of the system.

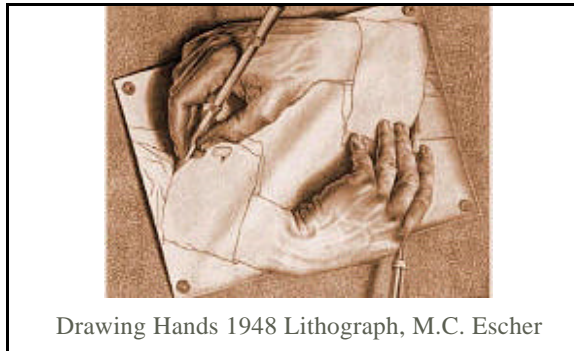
Why is this so important? Because we need an architecture to understand the system, organise its development, plan the software recycling and make it evolve.

Determining the elements that define an architecture is difficult and very important.

Usually, the development methodologies establish guides to identify and design an architecture, although, until now, the real help they can offer is very limited as these guides are very generic. In this article we are going to describe some techniques that can be very useful to build up software architectures.

The software architectures do not respond only to natural requirements, but they are related to other aspects such as performance, usability, recycling, economic and technological restrictions, and even to aesthetic questions.

## **2.- Architectures and Methodologies.**



Nowadays, there are many software developing methodologies, going from very “hard” and bureaucratic methods, methods that are adaptable to the project and to the developing conditions, to “light” methods that emerge as a response to the formal excesses of other methods.

Obviously, bearing in mind the guides given from so many and son different methodologies, it is very difficult to have a unified vision of the architectural design. However, we can remark some common elements to those that are more focused on this subject.

So, what are these common elements? The first one is the existence of a phase where a basic architecture is established or designed, and the second one is the very high dependence they define between the use cases and the architecture, defining a use case as a typical interaction (actions sequence) between the user and the system.

From an architectural point of view, not every use case have the same importance, being more remarkable those that help us to reduce the more important risks and, above all, those that represent the basic functionality of the system we want to build.

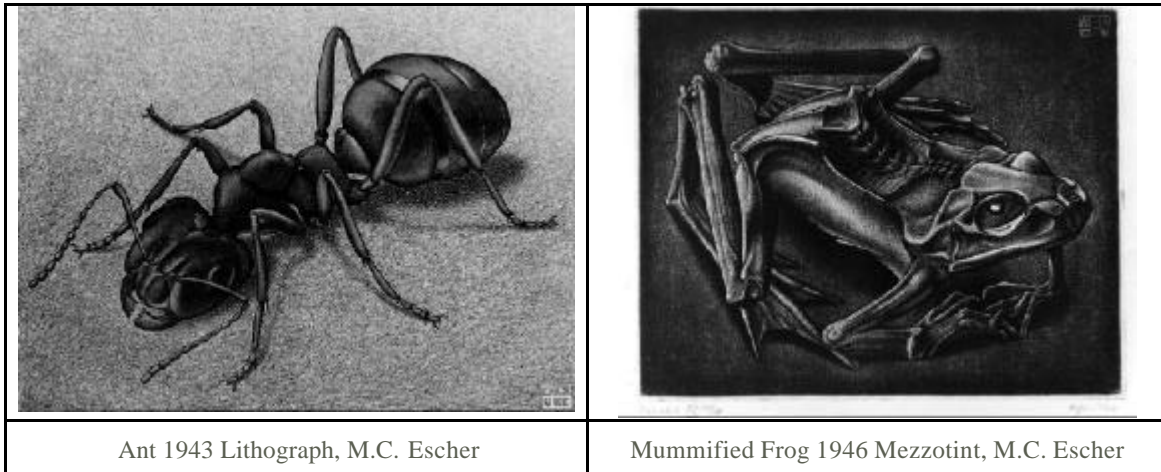
This basic architecture will be specified by diagrams that show subsystems, interfaces between them, components diagrams, classes, diverse descriptions, and by the basic group of use cases.

This group of specifications let us validate the architecture with the clients and the developers, and make sure that it is right to implement the wished basic functionality.

From this point, we would develop the system in an iterative way until having it functionally complete.

Up to here everything is perfect, isn't it? Now we really know how useful a software architecture is, but... we still don't know how to design one.

### 3.- Types of Systems.



An alternative vision would be to identify the type of system we want to build. Everybody knows that there are not two equal applications, but we know that there are clear parallelisms between applications that are built to solve similar problems.

Looking at applications of the same type has many advantages as that help us to understand the client needs and the what has been done to cover them.

Whereas using a traditional development methodology:

- ?? Makes us focus only on a part of the problem (knowing deeply the problem, its requirements and its microstructure), getting round subjects like performance, security, communication protocols, hardware, software and economic restrictions, and so on.
- ?? Gives us a narrow view, as the client does not usually present all his/her requirements in a suitable way, because he/she does not know them.

Let's imagine that we are developing a web site. Experience shows us that from a certain number of pages, it is useful to develop a system based on Xslt templates, because of the decrease of maintenance costs it all implies.

This is something that any functional requirement will show us, it will not arise either in a "natural" way with the design models.

We get to this type of solutions (design pattern) by the experience of web system developing and by the knowledge of the technologies existing on the market.

Thanks this experience, from the beginning of an application developing, we can search for components which are capable to implement these technologies or certain functionalities, and so we can integrate the searching of components and their use into the software developing process.

Identifying the type of system we have to build lets us examine the architecture of systems already built, understand the requirements they are facing and contrast them with our clients. If we have in mind that there are similar needs in any type of system (e.g. web sites), some components used on their development are usually the same (e.g. Xml parsers, Xslt transformation engines, data access components, search engines, shopping trolleys, content managers, and so on).

The methodologies that manage in a direct way the architectural and structural subjects will be able to produce not only products of higher quality but with a lower cost and in less time. This is due to fact that the architectural risks are lower and are more controlled, and also because being able to integrate a component-oriented view increases the possibilities of reusing already developed software, with all its advantages.

Building an architecture is both an activity to develop new ideas and an opportunity to use the gathered experience. The developer has usually the responsibility of creating a product of quality, and then, of knowing the type of system he/she has to build. Luckily, pattern languages can help us to do that.

#### **4.- Pattern Languages.**



Plane Filling II 1957 Lithograph, M.C. Escher

"Pattern Languages" can be defined as it follows: *"Specification of a series of elements (patterns) and their relationships (patterns) in a way that let us describe good solutions to different problems indifferent contexts"*.

The objective of the *design* patterns is to capture good practices that let us improve the quality of a system design, determining elements able to support useful roles in that context, *encapsulating complexity* and *making it more flexible*.

Usually, it is said that the function defines the form, that means that the structure or the architecture of any system is very related to what that system has to do.

This is the reason why systems with very similar objectives have also a common architecture, well defined processes and a group of very similar elements (design patterns). Similar functionality and service, similar structure.

When we develop a system which can be classified as a certain type, it is very useful to consult pattern languages that deal with the field we are in. A pattern language acts as a conceptual reference of the field of the problem, as they are given as a solution to a group of use cases and interactions with specific actors. Besides, they constitute a conceptual frame for the design of our systems architecture, since, as the function defines the form, they summarize architectural and structural solutions that are well proven and very useful in the type of problem they shape.

In some way, patterns let us identify and complete the basic use cases proposed by the client, they let us understand the system architecture, build their problems, and search already develop components that complies with the requirements of the type of system we have to build (that means that they let us obtain in a easy way the basic architecture we search during the architectural design phase).

Unluckily, pattern languages are not either the panacea, they present many gaps. Above all, we must remember that all this documentation of design movement starts in the middle nineties and, though a hard work has been done, there is no standardisation yet of how to deal with these languages development, nor a classification that relations them to each other.

From my point of view, a pattern language that shapes a type of system should be described including the following information:

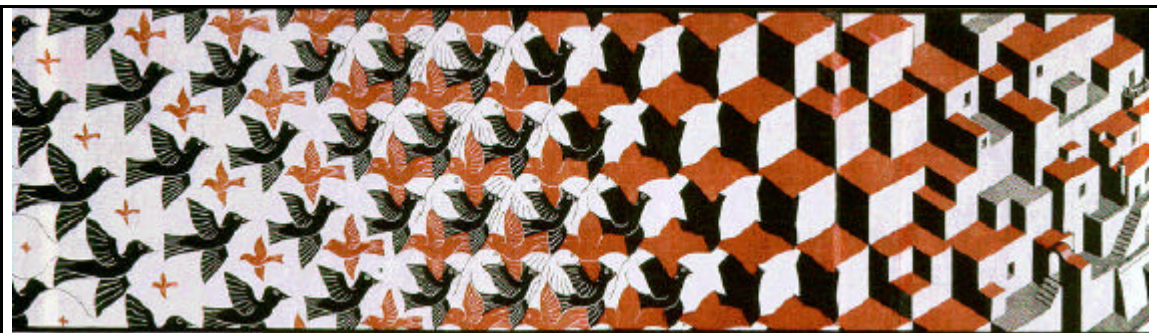
?? *Basic characteristics that define and differentiate it.* For example: a we system is characterized by

- ✍ microlight clients,
- ✍ using of protocols **without a state,**

- ↯ centralization of the execution software in servers in such a way that the applications is shared by all users,
  - ↯ **distribution of the information in levels of magnitude to its recording,**
  - ↯ the vast majority of the transactions in this type of systems are very simple,
  - ↯ the system is the responsible of building the interface of the user in a more diligent way than in other systems,
  - ↯ and so on.
- ?? *Definition of the main actors* that participates on the system as well as the basic use cases, obviously described in a generic way.
- ?? *Specification of the main functional components of the system* as well as the relationship between them.
- ?? *Logic architecture* and information flows that structure the different subsystems, the exchange of their information, and so on. For instance: reflexive architectures, model-view-controller, and so on.
- ?? *Components architecture*. That consist of mapping the functional components in the logic architecture of the application.
- ?? *Physic architecture*. Specification of the deployment of the components.

These languages should also have a view oriented to software building and to build this software as elements able to integrate into the applications development process.

## **5.- New designs.**



Metamorphosis II 1940 woodcut in black, green and brown, printed from 20 blocks on 3 combined sheets, M.C. Escher

I have talk a lot about recycling of the already gathered experience, however, it is to emphasize that in such a changing field as it is software, everything needs to evolve, and nothing is definite. The need to innovate and think about will continue being a very important element for the applications development.

Nonetheless, as Isaac Newton said "If I have seen further [than others], it is by standing on the shoulders of giants". Let's profit this piece of advice and let's build and innovate using the experience others have left to us.