Software Engineering

Part 1: Introduction and architectural pattern

Guillaume Swaenepoel & Thomas Aubin

Plan

- Introduction
- Architectural patterns
- Conception & UML
- Git and continuous integration
- Tests
- Python

Introduction

Software Engineering definition and historic

Software engineering

What is it ?

Software engineering is a detailed study of engineering to the design, development and maintenance of software.

Goal is ensure that the software development is:

- Readable
- Correct
- Reliable
- Reusable
- Extendable
- Flexible
- Efficient

16%

In 1970 only 16% of applications developed with success

A little history ...

In 1970 the software crisis (1968) took place. The software not being reliable at that time and the deadlines to provide a software respecting the specifications became a feat and to maintain the software a heroic act it was mentioned during a conference Garminsch-Partenkirchen sponsored by NATO the creation of a science of software engineering. The aim of this science is to study the working methods and good practices of engineers. It involves identifying and using methods, practices and tools to optimize the chances of success of a project.

Why software engineering

- Delay of 1st space shuttle launch (1981)
- F16 airplane declared on the back as it passed the equator: the southern hemisphere frame of reference was not taken into account.
- Ariane 501 flight failure (1996)
- SNCF Socrate software (1993)
- TAURUS project on the London Stock Exchange (£ 100 million loss)

“Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.” (Edsger Dijkstra)

“Without requirements or design, programming is the art of adding bugs to an empty text file.” (Louis Srygley)

« Programmers talk about bugs to keep their mental health; Such a number of errors would be psychologically unsustainable » (Martin Hopkins, IBM).

Software project : life cycle

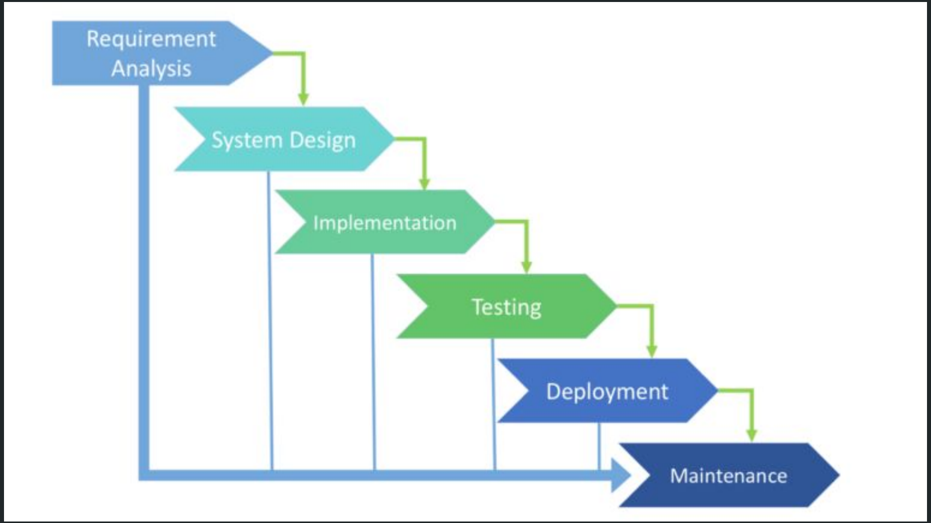
Organizing the development of a software : A cycle adapted to the needs.

Life cycle of a software project

- **Analyze of needs and capabilities (possibilities with your constraints)**
- **Software architecture**
  - What will it do ? When ? Versions ? ...
  - How is organized ? How the different parts will evolve ? ...
  - How the differents parts will communicates ?
  - What Database use ? What Physical and logical structure ?
  - Many more questions :)
- **Software design**
  - Precise architecture of every parts of the software
  - Source code organization.
- **Rules and production methods for source code**
- **Writing source code**
- **Testing with testing plan**
- **Software/version deployment**
- **Maintaining**
  - Every corrections (hotfixes and minor bugs)
  - Every software evolution (new functionality, Changing way of working for functionality, change visualization, change libraries...)

Waterfall cycle

The development of the software is split in multiples phases.



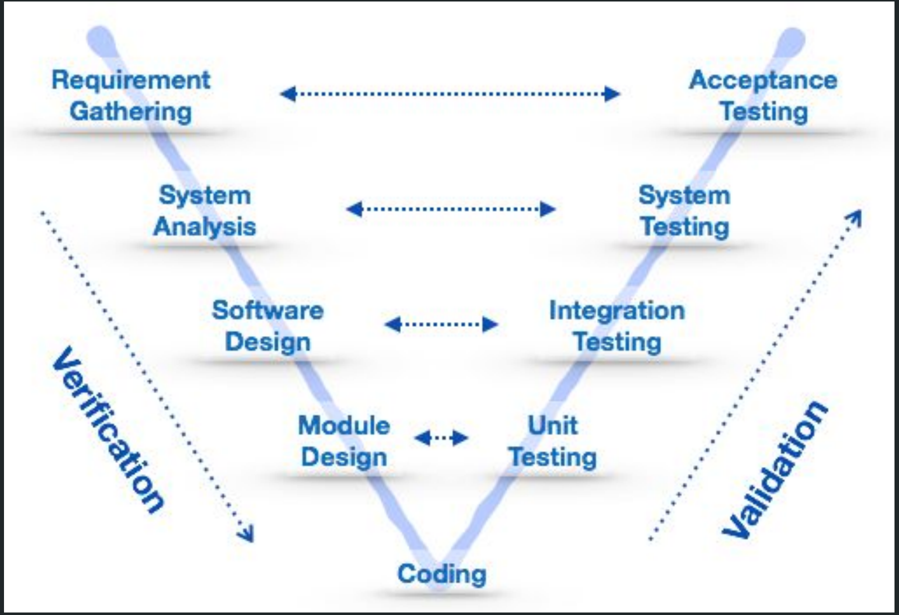
Software development phases

1. requirements: Redaction of software requirements.
2. design: Search of the best architecture
3. Implementation: Write code of the software
4. testing: Search bugs, check that the requirements are fulfilled, ...
5. deployment: deliver the software to customers, run in production
6. maintenance: fix bugs, new features ...

The main problem of the model is the lack of flexibility. Projects may take longer to deliver, compared to using an iterative methodology such as Agile.

V Cycle

Similar to waterfall cycle but include tests for each steps of the software development. Each phase is related to with a compilation a tests.

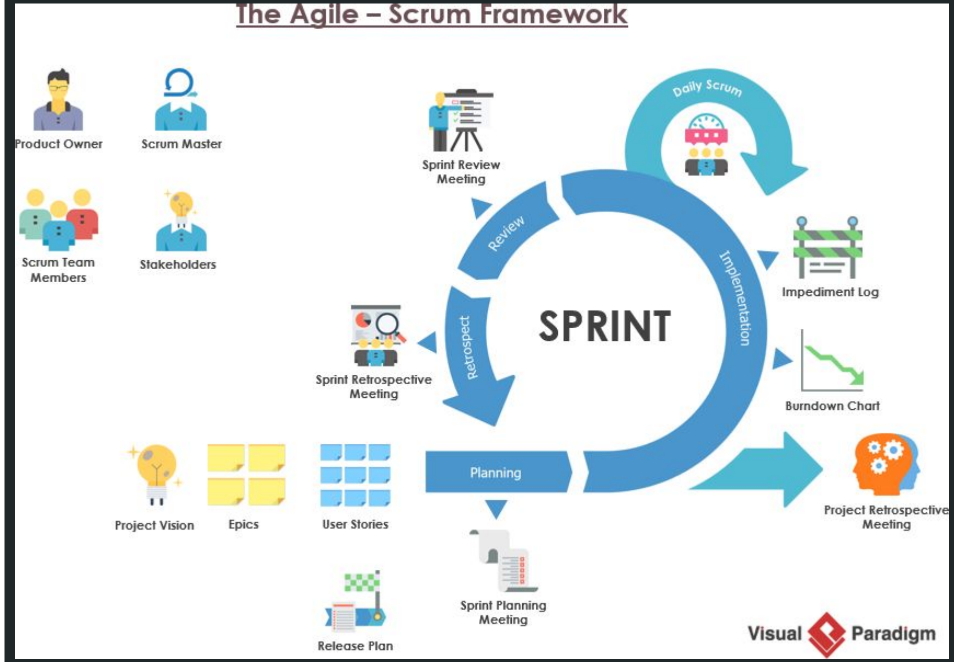


V Cycle

PRO	COINS
Stable definition of the goals	Not adapted for small development/evolution
efficients for big development	can lead to lack of communication
Doesn’t need many reunions	Not adaptable
Tasks and architecture define at the beginning	Validation phase can be hard due to many tests to be done

Agile Cycle

Customers needs is part of the software roadmap. The software is contently challenged and the deployment is made in many phases.



Agile cycle

The Agile methodology is based on a simple principle : Plan the totality of your project and every detail is counterproductive

At the beginning of the project

- Clarifying organisation of work
- Main project goals define by project owner or with clients

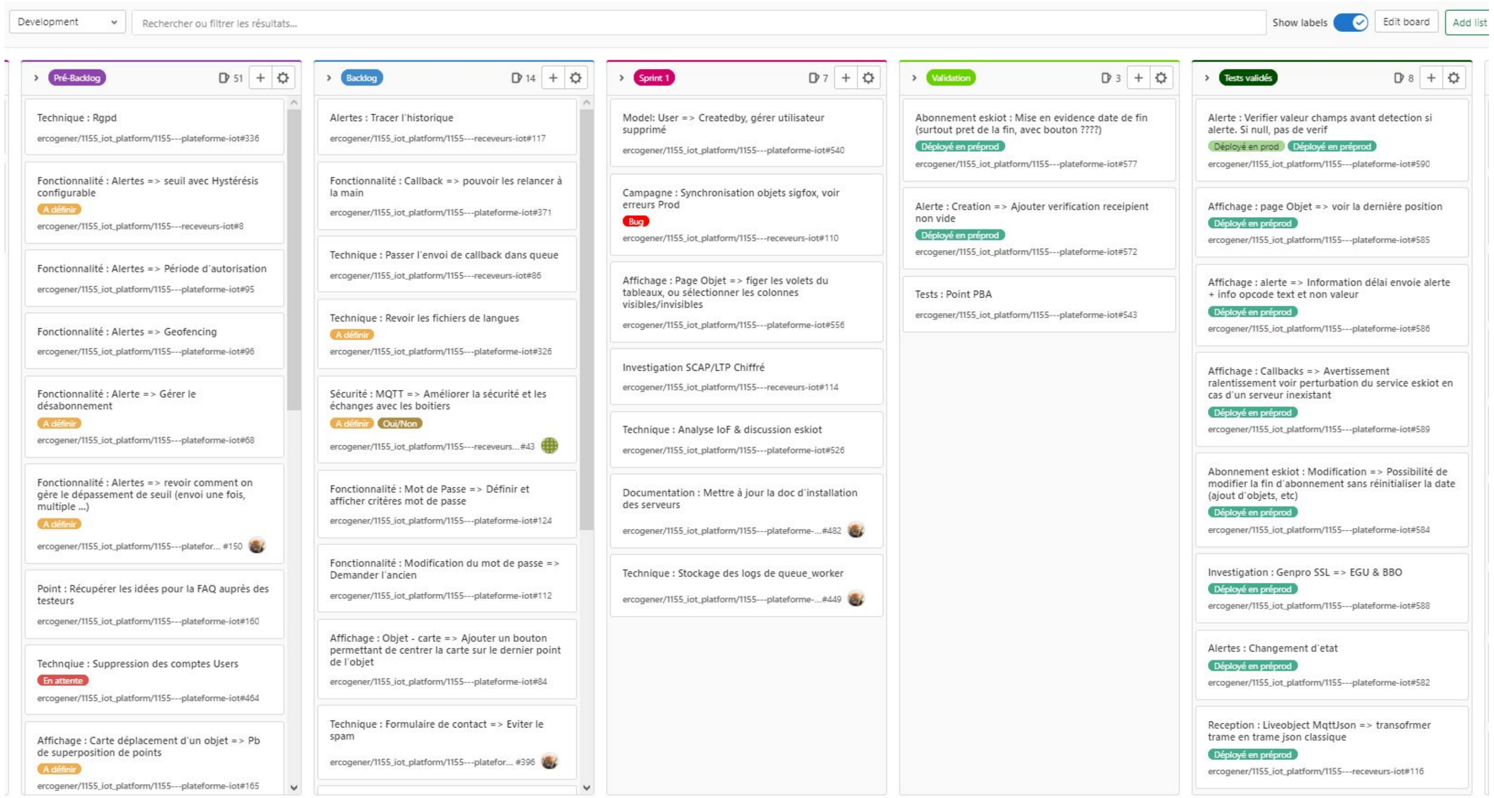
The sprint cycle

- Creation of tasks of the sprint (depends on project owner or client demands)
- Define priority
- Define tasks attribution
- Resolving tasks
- Testing
- Validating
- Presenting (to the team of to other services/user)

The scrum master

- Member of the team with scrum tasks and normal tasks
- Not a chief
- Responsable of the good execution of the Agile cycle
- Organize meetings
- Organize presentations
- Organize tasks
- Harmony and relate people
- “game master” of Agile

Exemple : Gitlab tasks



Pré-Backlog 51

- Technique : Rgpd
  ercogener/1155_iot_platform/1155---plateforme-iot#336
- Fonctionnalité : Alertes => seuil avec Hystérésis configurable
  A définir
  ercogener/1155_iot_platform/1155---receveurs-iot#8
- Fonctionnalité : Alertes => Période d'autorisation
  ercogener/1155_iot_platform/1155---plateforme-iot#95
- Fonctionnalité : Alertes => Geofencing
  ercogener/1155_iot_platform/1155---plateforme-iot#96
- Fonctionnalité : Alerte => Gérer le désabonnement
  A définir
  ercogener/1155_iot_platform/1155---plateforme-iot#68
- Fonctionnalité : Alertes => revoir comment on gère le dépassement de seuil (envoi une fois, multiple ...)
  A définir
  ercogener/1155_iot_platform/1155---platefor... #150
- Point : Récupérer les idées pour la FAQ auprès des testeurs
  ercogener/1155_iot_platform/1155---plateforme-iot#160
- Technqiue : Suppression des comptes Users
  En attente
  ercogener/1155_iot_platform/1155---plateforme-iot#464
- Affichage : Carte déplacement d'un objet => Pb de superposition de points
  A définir
  ercogener/1155_iot_platform/1155---plateforme-iot#165

Backlog 14

- Alertes : Tracer l'historique
  ercogener/1155_iot_platform/1155---receveurs-iot#117
- Fonctionnalité : Callback => pouvoir les relancer à la main
  ercogener/1155_iot_platform/1155---plateforme-iot#371
- Technique : Passer l'envoi de callback dans queue
  ercogener/1155_iot_platform/1155---receveurs-iot#86
- Technique : Revoir les fichiers de langues
  A définir
  ercogener/1155_iot_platform/1155---plateforme-iot#326
- Sécurité : MQTT => Améliorer la sécurité et les échanges avec les boitiers
  A définir Oui/Non
  ercogener/1155_iot_platform/1155---receveurs...#43
- Fonctionnalité : Mot de Passe => Définir et afficher critères mot de passe
  ercogener/1155_iot_platform/1155---plateforme-iot#124
- Fonctionnalité : Modification du mot de passe => Demander l'ancien
  ercogener/1155_iot_platform/1155---plateforme-iot#112
- Affichage : Objet - carte => Ajouter un bouton permettant de centrer la carte sur le dernier point de l'objet
  ercogener/1155_iot_platform/1155---plateforme-iot#84
- Technique : Formulaire de contact => Eviter le spam
  ercogener/1155_iot_platform/1155---platefor... #396

Sprint 1 7

- Model: User => Createdby, gérer utilisateur supprimé
  ercogener/1155_iot_platform/1155---plateforme-iot#540
- Campagne : Synchronisation objets sigfox, voir erreurs Prod
  Bug
  ercogener/1155_iot_platform/1155---receveurs-iot#110
- Affichage : Page Objet => figer les volets du tableaux, ou sélectionner les colonnes visibles/invisibles
  ercogener/1155_iot_platform/1155---plateforme-iot#556
- Investigation SCAP/LTP Chiffré
  ercogener/1155_iot_platform/1155---receveurs-iot#114
- Technique : Analyse IoF & discussion eskiot
  ercogener/1155_iot_platform/1155---plateforme-iot#526
- Documentation : Mettre à jour la doc d'installation des serveurs
  ercogener/1155_iot_platform/1155---plateforme-...#482
- Technique : Stockage des logs de queue_worker
  ercogener/1155_iot_platform/1155---plateforme-...#449

Validation 3

- Abonnement eskiot : Mise en evidence date de fin (surtout pret de la fin, avec bouton ????)
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#577
- Alerte : Creation => Ajouter verification receipient non vide
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#572
- Tests : Point PBA
  ercogener/1155_iot_platform/1155---plateforme-iot#543

Tests validés 8

- Alerte : Verifier valeur champs avant detection si alerte. Si null, pas de verif
  Déployé en prod Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#590
- Affichage : page Objet => voir la dernière position
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#585
- Affichage : alerte => Information délai envoie alerte + info opcode text et non valeur
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#586
- Affichage : Callbacks => Avertissement ralentissement voir perturbation du service eskiot en cas d'un serveur inexistant
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#589
- Abonnement eskiot : Modification => Possibilité de modifier la fin d'abonnement sans réinitialiser la date (ajout d'objets, etc)
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#584
- Investigation : Genpro SSL => EGU & BBO
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#588
- Alertes : Changement d'etat
  Déployé en préprod
  ercogener/1155_iot_platform/1155---plateforme-iot#582
- Reception : Liveobject MqttJson => transofrmer trame en trame json classique
  Déployé en préprod
  ercogener/1155_iot_platform/1155---receveurs-iot#116

Agile Cycle

PRO

Flexible

Dynamic

Lot of communication

Good organization

COINS

Lost of the main goal due to too many subtasks

tasks can change at every cycle and can be opposite -> lost of motivation

Need good habits and organization

too much reunions

Development models

Structure of code, for easy maintenance and handling lot of functionalities

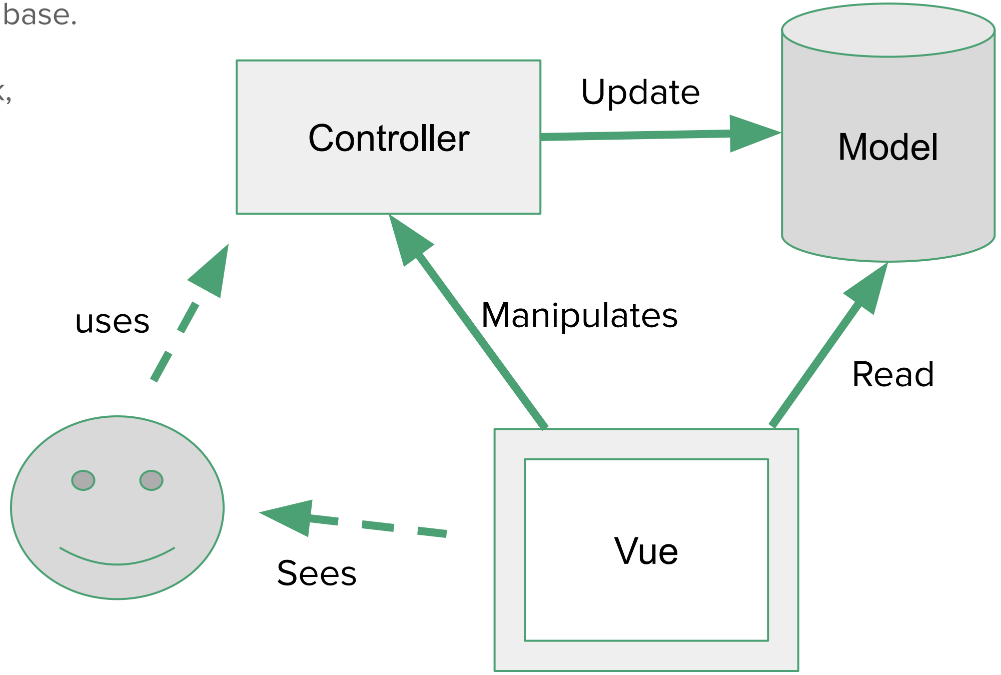
Issues

- hardware performance limitation
- high availability
- Minimize business risks

MVC (Model-Vue-Controller)

Model :
Objects and functions communicate with database.
Controller :
Objects and functions handle data (get, check, processing, ...)
Vue
Objects and functions used for user display.

It allows to clearly separate the different aspects of each problem.



Model

allows you to worry only about the way in which we will represent the different terms (nouns and verbs) that appear in the needs analysis, without worrying about the way in which we will represent the data or the actions to 'user.

Allow a representation of all the software data. Your model is most of the time a connector to your database.

Example:

For the storage of a date we don’t care about the way to display it or the timezone.

The model will only check if the date is valid (31 days for january, 28 for february except leap years. From the model the date will be a structure of three variables day, month, year and three fonctions to check leap year, an other to get the number of days in a month and the last one using precedent one to check the date validity.

17 may 2018 -> valid
32 february 2020 -> invalid

View

allows you to worry only about the way to display/return data to users.

The view can be also intended for other software. For example a REST API returning response in JSON format.

Taking the previous example, the view will be in charge to transform the date structure in a comprehensive string. It can for example transmit in the right country format 12th February 2012 (EN) or 12 February 2012 (FR).

Controller

The model and the view are two independant part of your software.

The controller will aim to connect these two extremes in a coherent way:

- If a view changes data, the controller will ensure that the "new value" is consistent before notifying the model of the modification.
- when a data is added or updated at the model level, the controller will take care of signaling the change(s) to the view.

PRO & COINS

+ **Segregation of tasks**: separate business logic, user interface and system dynamics.

+ **Specialization**: a clear definition of the areas of intervention of developers. UI developers can focus exclusively on the interface, without being hampered by the rest of the application. The same goes for developers of business logic or system dynamics.

+ **Development** can be split easily, different person can work on independant parts. (example: one guy write HTML sources for the view, an other guy the model and controller in PHP).

+ **Reusability and time saving:** The model or the view can be completely change or use somewhere else easily.

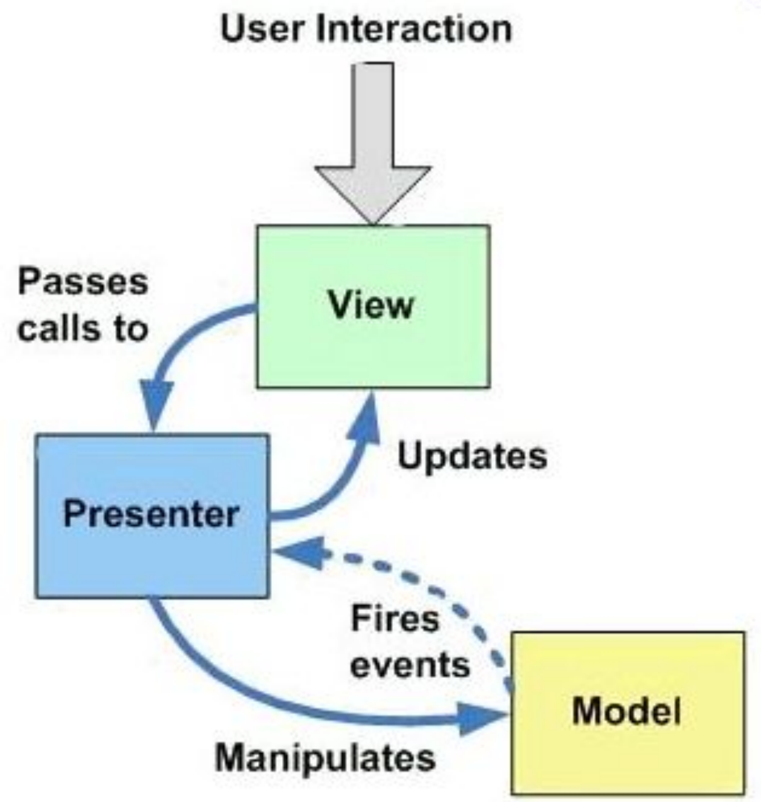
- **Complexity:** The team must be well synchronized to exchange coherent data.

- **A huge number of files** to handle, in fact the separation of the different layers requires the creation of more files (3 times for the different layers).

But for large-scale projects, this detail seems less relevant, if the design has been done well.

MPC (Model-Presentation-Controller)

This is a derivation of the MVC model. The interaction between the view and the model is make by the presentation. This part organized the data for the view.



Architectural Patterns

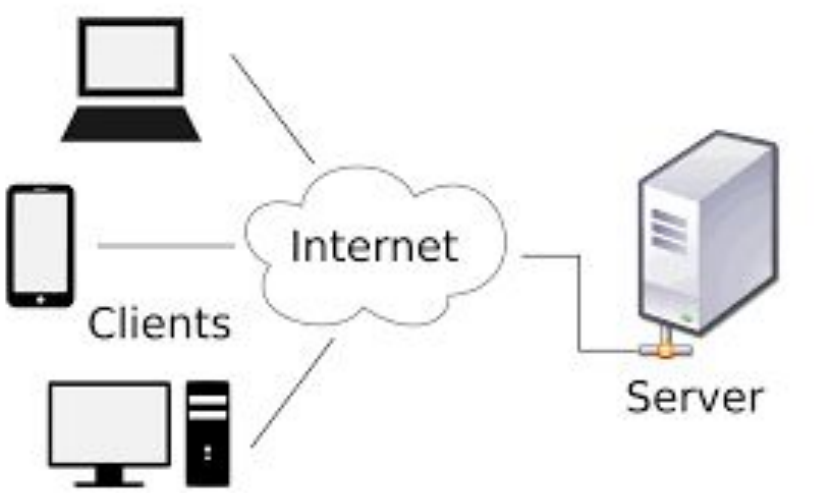
General, reusable solution to a commonly occurring problem within a given context

Many frameworks already include architectural patterns

Client/server

also named 2-tier Architecture

based on network exchanges between one or more clients and a server.

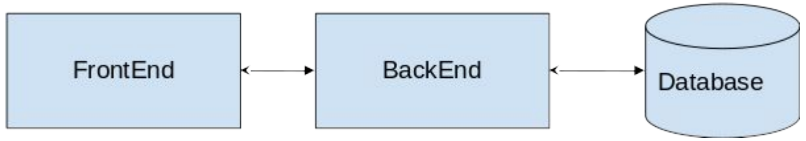


Layered Architecture

n-tiered patterns where the components are organized in horizontal layers.

Traditional method for designing most software and is meant to be self-independent. This means that all the components are interconnected but do not depend on each other.

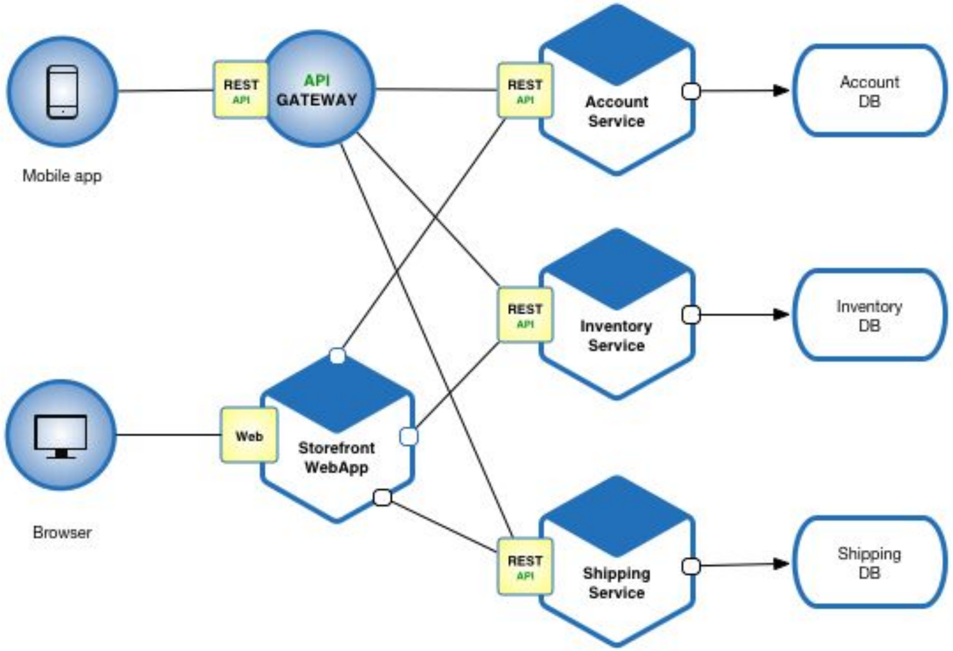
Example : architecture 3-tiered



Micro services

massively used in cloud infrastructure. Allow deployment of very large and complex applications. (Example: Netflix, Amazon, Google ...)

The software is divided in several services communicating together. Services can be easily dispatch between servers.



Microservices : Benefits

- Enables the continuous delivery and deployment of large, complex applications.
- Each microservice is relatively small:
  - Easier for a developer to understand
  - The IDE is faster making developers more productive
  - The application starts faster, which makes developers more productive, and speeds up deployments
- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack. Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

Microservices : Drawbacks

- Developers must deal with the additional complexity of creating a distributed system:
  - Developers must implement the inter-service communication mechanism and deal with partial failure
  - Implementing requests that span multiple services is more difficult
  - Testing the interactions between services is more difficult
  - Implementing requests that span multiple services requires careful coordination between the teams
  - Developer tools/IDEs are oriented on building monolithic applications and don’t provide explicit support for developing distributed applications.
- Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different services.
- Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g. EC2 instance), as is the case at Netflix, the overhead is even higher.

When to use the microservice architecture?

One challenge with using this approach is deciding when it makes sense to use it. When developing the first version of an application, you often do not have the problems that this approach solves. Moreover, using an elaborate, distributed architecture will slow down development. This can be a major problem for startups whose biggest challenge is often how to rapidly evolve the business model and accompanying application. Using Y-axis splits might make it much more difficult to iterate rapidly. Later on, however, when the challenge is how to scale and you need to use functional decomposition, the tangled dependencies might make it difficult to decompose your monolithic application into a set of services.

Web app & API REST

Not an architecture pattern but a functionality that has to be considered

Authentification secured -> OAuth2 (standard used everwhere)

Access to functionalities via URL

Exemple : https://myurl.com/api/addDevice -> POST Request with variable content

Conclusion

The emergency of Docker and Kubernetes and their exploitation by cloud infrastructure operator such as AWS, Azure, OVH allow company to easily buy a cluster and deploy their applications in it.

It permits to expand the possibilities and the functionalities of web services. It requires many adaptation and changing in the way of developing the project.

If the utilisation of microservices, you can have a better gestion of the ressources and a better load balancing (by replicating and spreading your services on demand).

But this method is hard and long to create and may be not adapted to small web services.

Questions ?