TIME TO MARKET

MAIF

TIME TO MARKET

TIME TO MARKET

[MVP]

Achieve your **Minimum Viable Product** to test your concept.

TIME TO MARKET

[MVP]

The **perimeter** of your MVP must be **limited** while allowing you to market your product.

Bet on **Early Adopters** and get a maximum of **feedback**.

Your MVP is deployed and usable in **production**.

TIME TO MARKET

[FAIL-FAST]

Fail fast to **learn** fast.

TIME TO MARKET

[FAIL-FAST]

Quickly experience the solution (a few weeks), collect **feedback** from your users and learn from your mistakes.

Do not be afraid to **change** everything.

Do not forget, **you will fail!**

TIME TO MARKET

[KISS]

Keep **I**t **S**imple and **S**tupid.

Why make it complicated when it can be simple?

TIME TO MARKET

[KISS]

Avoid over-engineering, if a "paper" model or a Google Form is enough to test your concept, do not go further.

Keep it simple! Both technically and functionally.

TIME TO MARKET

[PRODUCTIVITY]

Less specification, **more development**.

MAIF

TIME TO MARKET

[PRODUCTIVITY]

Limit your specifications to the bare minimum, **focus on the "what"** rather than the "how".

The product must be **self-documented** as much as possible.

Documentation must be versioned in the same way the code is.

TIME TO MARKET

[SAAS]

Always consider **SaaS** solutions.

MAIF

TIME TO MARKET

[SAAS]

SaaS solutions are **sustainable and cost-effective**.

In some cases **SaaS** can **speed up** the **MVP** implementation.

Think of the **target economic vision** regarding alternatives in terms of **total cost** (**TCO** : **T**otal **C**ost of **O**wnership) and not only in terms of license cost.

TIME TO MARKET

[CORE BUSINESS]

The **core business** should not hinder the construction of new services and applications.

TIME TO MARKET

[CORE BUSINESS]

The pace of evolution and delivery of the core business must be **compatible with the agility** of its consumer services.

The core business must **expose services**.

The core business must adopt an **Event-Driven** principle, it reports management actions in the form of events.

TIME TO MARKET

[CONTINUOUS DEPLOYMENT]

Deployment in production

is a non-event.

MAIF

TIME TO MARKET

[CONTINUOUS DEPLOYMENT]

Take advantage of **continuous deployment** to **adapt deployment** in **production** to business constraints and requirements, and not the other way around.

Deployments across all environments, up to **production**, must be **automated** and **frequent**.

MAIF

TIME TO MARKET

[PERPETUAL BETA]

A **perpetual beta** approach allows you to involve your users in the development process.

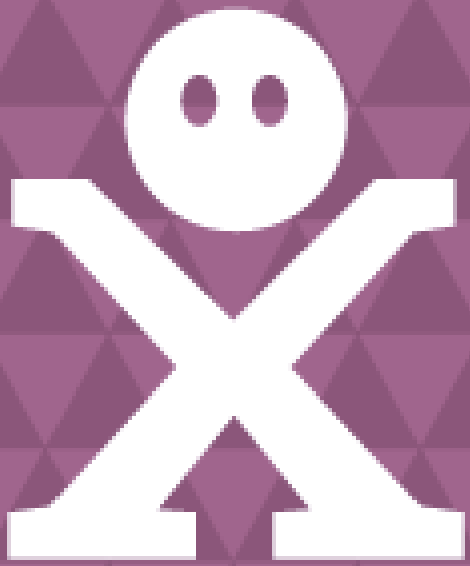
MAIF

TIME TO MARKET

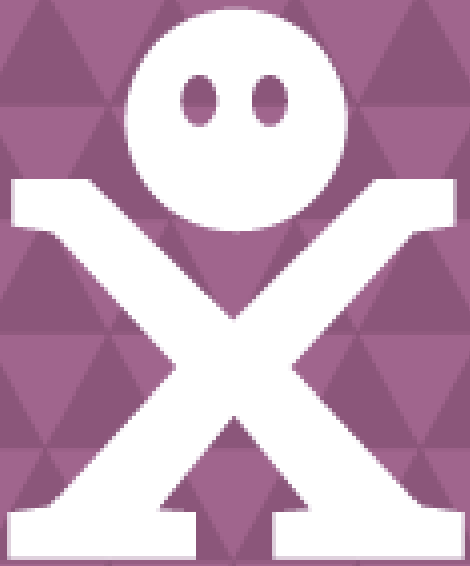
[PERPETUAL BETA]

Feel free to use the perpetual beta principle in which **users participate in development**.

The term perpetual beta refers to an application developed just-in-time, **constantly evolving**, not to an incomplete product.



USER EXPERIENCE



USER EXPERIENCE

USER EXPERIENCE

[PERCEPTION]

The **experience perceived** by the user is fundamental.

Ergonomics is not negotiable.

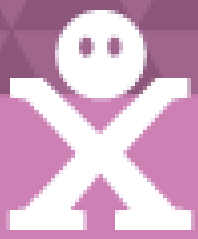
USER EXPERIENCE

[PERCEPTION]

Do not neglect the work of **UX designers**, it is fundamental in the development of an application.

Integrate the **feedback** of your users, it is essential.

USER EXPERIENCE



[PERFORMANCE]

Bet on **efficient interfaces** for both internal and external uses.

MAIF

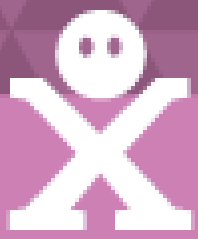
USER EXPERIENCE

[PERFORMANCE]

The interfaces are geared towards **efficiency**.

The **performance** of an interface **saves time**, increases users' **satisfaction** and therefore **reduce their frustration**.

USER EXPERIENCE



[MOBILE FIRST]

Adopt a **Mobile First** strategy.

MAIF

USER EXPERIENCE

[MOBILE FIRST]

Mobile devices are the **most important market share**.

Thinking mobile is thinking about the **essential**.

Responsive Design is the norm, it is a source of savings (**MVP**).

USER EXPERIENCE

[OMNI-CHANNEL]

Adapt to practices, **omni-channel** is the norm.

USER EXPERIENCE

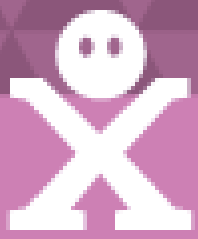
[OMNI-CHANNEL]

The omni-channel approach provides the user with a **unified experience** (example: Netflix).

The different **channels** are **synchronized** and **coherent** (unlike batch processes).

All the actors (clients, advisers) access the same information.

USER EXPERIENCE



[SELF-DATA]

Users are **owners** of their data and their journey.

USER EXPERIENCE

[SELF-DATA]

Allow **individuals**, at any time, to **control** their **personal** data.

Establish **trust** by giving users real-time traceability and control.

Subsystems must meet the same requirements.

USER EXPERIENCE

[CRM/SFA]

Customer relationship needs to be unified and contextualized with a flexible, unifying and event-driven **CRM/SFA.**

MAIF

USER EXPERIENCE

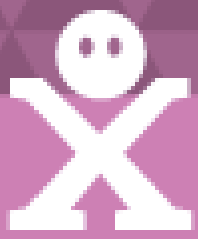
[CRM/SFA]

Opt for a **CRM** that manages both customer relationship and sales force automation (**SFA** : **S**ales **F**orce **A**utomation).

CRM must be **open** to new opportunities.

CRM produces **events** corresponding to management actions to fit into the **Event-Driven** logic of the platform.

USER EXPERIENCE



[BIG DATA]

The Big Data platform allows you to **centralize** and process the user's data to best serve him/her in his/her **journey**.

MAIF

USER EXPERIENCE

[BIG DATA]

Centralize **Maif Group**, **Partner** and **Vendor** data in a **journey** logic.

"Data Preparation" and processing help to **consolidate** the data.

Big Data teams collaborate with Feature Teams to ensure data **governance**.

USER EXPERIENCE

[WORKSTATION]

The workstation is adapted and adaptable to **modern practices and channels**.

USER EXPERIENCE

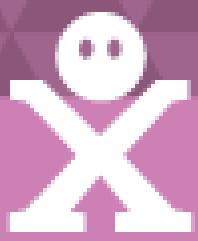
[WORKSTATION]

Adopt **identity federation** for a unified experience.

A **portal** offers an **overview**, it does not replace applications.

The workstation must be **mobile**, **multi-channel** and **standard** to allow opening within the **extended enterprise**.

USER EXPERIENCE



[COLLEAGUES]

Do not forget that your **colleagues** are using modern applications with modern UX at home.

MAIF

USER EXPERIENCE

[COLLEAGUES]

Treat **all your users as "customers"**: Internet users, managers, ops, developers, etc.

Do not underestimate the **UX effort** needed for internal-use management applications.

USER EXPERIENCE

[MEASURING EVERYTHING]

Everything that can be measured must be.

Without measure, everything is only opinion.

USER EXPERIENCE

[MEASURING EVERYTHING]

Think of the metrics during the **development** of the application. **Logs** must have a **business as well as technical** dimension.

Do not neglect **performance metrics**, they are fundamental.

The Feature Team ensures **operation**: it is responsible for making the **application operable**.

USER EXPERIENCE

[A/B TESTING]

A/B Testing saves time by letting **feedback** decide.

USER EXPERIENCE

[A/B TESTING]

Rather than arbitrarily decide between two solutions, do not hesitate to set up **A/B testing**.

This pattern consists of presenting **two different versions** of the same application and choosing one of them based on **objective measures** of user activity.

USER EXPERIENCE

[DEGRADATION]

Consider degradation rather than service interruption in the event of a failure.

USER EXPERIENCE

[DEGRADATION]

In case of **failure** of one of the subsystems, **a degraded** version of the service must **be considered** in the first place rather than an interruption.

With the help of **Circuit Breakers**, **isolate a breakdown** to **avoid** its **impact** and **propagation** to the entire **system**.

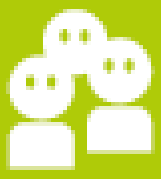
HUMAN

HUMAN

HUMAN

[FEATURE TEAM]

The team is organized around a **product** or a provided **service**.



HUMAN

[FEATURE TEAM]

Teams are **Feature Teams**, organized around a coherent functional scope, and comprise all the **skills** necessary for this scope.

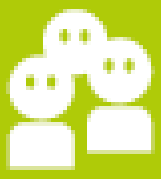
For example: Business Expert + Web Developer + Java Developer + Architect + DBA + Ops.

The **responsibility** is **collective**, the Feature Team has the necessary power for this responsibility.

HUMAN

[2-PIZZA TEAM]

Limit the **size of Feature Teams** (from 5 to 12 people).



HUMAN

[2-PIZZA TEAM]

Limit the size of a Feature Team: **between 5 and 12 people**.

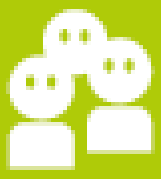
Below 5, the team is too sensitive to external events and lacks creativity. Above 12, it loses productivity.

The term **"2-Pizza Team"** indicates that the size of the Feature Team should not exceed the number of people that can be fed with two pizzas.

HUMAN

[SOFTWARE CRAFSTMAN]

Bet on versatile people who **know how to do** and who **like to do**.



HUMAN

[SOFTWARE CRAFSTMAN]

The most important are **development culture**, **scalability** and **adaptability**.

Recruit **software craftsman and full-stack developers**, they bring a real added value through their know-how and their overall vision.

Nonetheless, mobile developers - for example - are usually **specialized developers**.

HUMAN

[RECRUITMENT]

Be **attractive** to recruit the **best**.

HUMAN

[RECRUITMENT]

Offer ways of working adapted to employees: **mobility**, **home working**, **CYOD** (**C**hoose **Y**our **O**wn **D**evice).

Allow time for experimentation and ensure it happens **during working hours**.

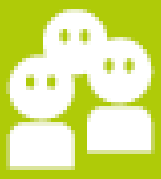
HUMAN

[WATCH]

The organization must be a

watch apparatus

Watch is part of the job.



HUMAN

[WATCH]

The organization must be a **watch** apparatus by setting up plans like **continuous learning** or **corporate universities**.

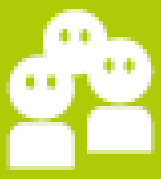
Feel free to combine them with more **informal** ways such as: **Coding Dojos**, **Brown Bag Lunches**, **External** Conferences.

HUMAN

[CO-CONSTRUCTION]

Break barriers between jobs, bet on objectives **convergence**.

MAIF



HUMAN

[CO-CONSTRUCTION]

To break barriers between jobs, it is not enough to group people around a common product in a common place.

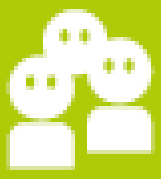
The **Agile Methodologies** can help to remove these barriers and ensure **objectives convergence**.

These practices are an integral part of the keys to success, the organization is responsible for it.

HUMAN

[DEVOPS]

DevOps practices help to break walls between Build and Run.



HUMAN

[DEVOPS]

Adopt **DevOps** to help **Dev** and **Ops** converge towards a common goal: **serve the organization**.

Both jobs remain different ! DevOps does not mean that the same person performs the tasks of Dev and Ops. **Developers** and **Ops** are required to **collaborate** in order to **benefit** from each other's **skills** and to improve **empathy**.

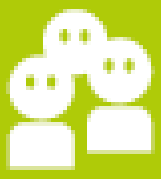
HUMAN

[PAIN]

Painful tasks are performed **by the Feature Team**.

Automation comes out of it.

MAIF



HUMAN

[PAIN]

In a traditional organization, a **lack of understanding** between teams is usually related to distance and **lack of communication**.

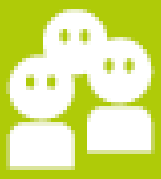
The **members of a Feature Team** are **co-responsible** and **united** facing up all tasks.

Pain is a key factor in **Continuous Improvement**.

HUMAN

[Service Center]

Service centers are difficult to reconcile with the **collective commitment**.



HUMAN

[Service Center]

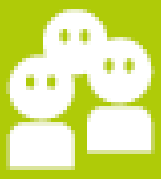
Feature Teams are built around principles that rely heavily on **collaboration** and **collective engagement**.

Service centers tend towards rationalization and consolidation of IT by area of expertise, which is contrary to this notion of collective commitment.

HUMAN

[VALIDATION]

The organization has a **validation role**, without being dogmatic.



HUMAN

[VALIDATION]

Ensure that the organization retains its **validation role** on tools and uses. In particular regarding the **tools that affect assets** (example: source code management).

Provide Feature Teams with **means** to support their choices.

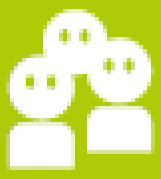
Do not be **dogmatic** and ensure to **encourage experimentation**.

HUMAN

[TRANSVERSALITY]

Feature Teams are expected to **communicate** and share their **experience** and **skills**.

MAIF



HUMAN

[TRANSVERSALITY]

Do not create barriers between **Feature Teams**.

Set up an **organization** and the required **agility** for Feature Teams to communicate with each other and share their skills and experiences.

The organization of transversality at **Spotify** (Tribes, Chapters and Guilds) is an **eloquent example**.

INTEROPERABILITY

INTEROPERABILITY

INTEROPERABILITY

[API FOR ALL]

APIs for all usages:
internal, customers and partners, public.

MAIF

INTEROPERABILITY

[API FOR ALL]

Expand your organization to new usages and new customers with **Public APIs**.

As part of **commercial** partnerships, whether with **customers** or **providers**, APIs are the standard exchange format.

APIs are also meant to be used **internally** inside the organization.

INTEROPERABILITY

[SELF SERVICE]

Using an API must be **simple** and **fast**.

INTEROPERABILITY

[SELF SERVICE]

Using APIs should be as simple as possible. Think about the **developer experience**.

The best solution to validate adequacy with the need is to **test the API quickly**: a few minutes must be enough!

The platform must offer a **graphical interface** to test the API in a simple manner.

INTEROPERABILITY

[API MANAGEMENT]

APIs usage must be **controlled** and **understood**.

MAIF

INTEROPERABILITY

[API MANAGEMENT]

Implement an API Management solution to manage **quotas**, **throttling**, **authentication**, and **logging**.

Collect metrics to manage **monitoring**, **filtering**, and **reporting**.

INTEROPERABILITY

[REQUIREMENTS]

Set **requirements** for **external systems and services** integrated into the platform.

MAIF

INTEROPERABILITY

[REQUIREMENTS]

Require **external systems** to meet the same **requirements** as **internal systems**.

External systems must publish **events** and allow **technical** monitoring.

In the case where the external systems data must be integrated, **total** synchronization have to be **possible**.

INTEROPERABILITY

[MULTI-TENANT]

The architecture must be designed as **multi-tenant**.

MAIF

INTEROPERABILITY

[MULTI-TENANT]

Even if white labelling is not considered at the beginning, set up a multi-tenant architecture. Your initial **application** is the first **tenant**.

Think of the **functional multi-instantiation** of the system right from the start.

INTEROPERABILITY

[CONFIGURATION]

Systems must be **natively configurable**.

MAIF

INTEROPERABILITY

[CONFIGURATION]

Languages, currencies, business rules, security profiles must be simple to configure.

Beware of **hyper-genericity**, it is often useless and **costly**.

The **configuration** must be **scalable** and fast depending on needs.

INTEROPERABILITY

[FEATURE FLIPPING]

Create flexible and generic systems using **feature flipping**.

MAIF

INTEROPERABILITY

[FEATURE FLIPPING]

Feature flipping is about designing an app as a set of **features** that can be **enabled** or **disabled** live, in **production**.

In a **multi-tenant** application, feature flipping allows you to **customize** by tenant.

Feature flipping **simplifies A/B testing**.

RULES OF THE GAME

MAIF

RULES OF THE GAME

MAIF

RULES OF THE GAME

[TECHNICAL CHOICES]

The **technical choices** are **made** and **assumed** by the **Feature Team**.

RULES OF THE GAME

[TECHNICAL CHOICES]

The Feature Team must act **responsibly** to identify the choices that impact itself exclusively and the choices that impact the organization.

The **choices** that **exceed the scope** of the Feature Team (eg, license, infrequent programming language) must be **validated** by the organization or by peer convergence process .

RULES OF THE GAME

[GOOD USE]

The **right tool** for the **good use** is a source of savings.

RULES OF THE GAME

[GOOD USE]

A **bad tool** imposed on everyone is a **risk**. The **misuse** of a good tool can have very **damaging consequences**. For example, poorly used Agile methods are dangerous.

Tools must be **questioned**.

Excel is often a **rational** choice but it is **not a tool to do everything** (CRM, ERP, Datamart, ...)

RULES OF THE GAME

[BUILD VS. BUY]

Prefer **Build** for the core business.

Consider **Buy** for the rest, case by case.

RULES OF THE GAME

[BUILD VS. BUY]

The more a tool brings a **differentiating feature** for the organization, the more it should be **built**. The core business must allow **specificity** and **adapt quickly and often**. Some **enterprise software** are **sometimes adapted** to this need.

For **the rest**: SaaS, Open Source, Build or Vendor are to be studied **case by case**.

MAIF

RULES OF THE GAME

[OPEN SOURCE]

Opt for Open Source.

Alternative choices must be explained.

RULES OF THE GAME

[OPEN SOURCE]

Proprietary solutions are a **risk** for the organization which must be able to take over maintenance if needed.

Few are the proprietary tools that do not have **open source alternatives**.

The organization **benefits** from the **Open Source Community** and can **pay it back with its contributions**.

RULES OF THE GAME

[MICRO-SERVICES]

Develop **stand-alone** and **loosely coupled** services.

MAIF

RULES OF THE GAME

[MICRO-SERVICES]

Loose coupling must be the norm.

Each micro-service has a **clearly defined interface**.

This **interface** determines the **coupling** between **micro-services**.

Domain Driven Design helps, especially through **Bounded Contexts**, to anticipate this problem.

RULES OF THE GAME

[DATA]

Each service has its **own** data storage system.

RULES OF THE GAME

[DATA]

A **Data Store** is intended to be **coupled** only with **a single micro-service**.

Data access from one micro-service to another is done **exclusively via its interface**.

This design implies **consistency over time** across the platform. It must be **apprehended at all levels**, including UX.

RULES OF THE GAME

[PERIMETER]

Each micro-service must have a reasonable functional perimeter, which **"fits in the head"**.

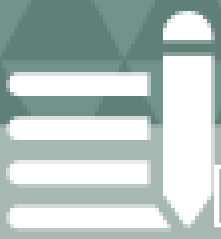
RULES OF THE GAME

[PERIMETER]

A micro-service offers a **reasonable number of features**.

Do not hesitate to split a micro-service when it begins to grow.

A service of reasonable size makes it possible to **consider** serenely the **rewriting**, if the need arises.



RULES OF THE GAME

[REACTIVE]

The **Reactive Manifesto** opens the way towards the design of reactive architectures.

RULES OF THE GAME

[REACTIVE]

Responsive programming focuses on the flow of data and the propagation of change. It is based on the "**Observer**" pattern, contrary to the more traditional "**Iterator**" approach.

The Reactive Manifesto sets some fundamental axis: **availability** and speed, **resilience** to failure, **flexibility**, **elasticity** and **message oriented**.

RULES OF THE GAME

[ASYNC-FIRST]

Asynchronous processes foster **decoupling** and **scalability** in favor of **performance**.

MAIF

RULES OF THE GAME

[ASYNC-FIRST]

Exchanges between applications must be **asynchronous** first.

Asynchronous exchanges **naturally** allow **loose coupling**, **isolation** and flow **control** (**back-pressure**).

Synchronous communication should only be considered **when required**.

RULES OF THE GAME

[EVENTS]

The information system must be **event-driven**.

RULES OF THE GAME

[EVENTS]

The **event-driven** functional **processes** are **naturally** implemented in an **asynchronous** manner.

Being **event-driven** allows for approaches such as **C**ommand **Q**uery **R**esponsibility **S**egregation (**CQRS**) and **Event Sourcing**.

RULES OF THE GAME

[MESSAGE BROKER]

Prefer a **simple**, robust and powerful **message-broker** to a "smart pipe".

MAIF

RULES OF THE GAME

[MESSAGE BROKER]

ESB showed their **limits**: **scalable maintenance** is **critical**, both from a **technical** and **organizational** standpoint.

Message brokers like **Kafka** offer a **simple**, **durable** and **resilient** solution.

Smart endpoints and **dumb pipes** is an architecture that works at scale: it's **Internet**.

RULES OF THE GAME

[SYNCHRONIZATION]

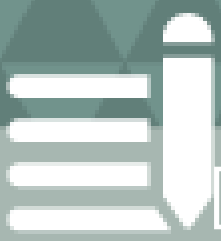
The **full synchronization** of the system should be thought of during its **design**.

RULES OF THE GAME

[SYNCHRONIZATION]

If the **synchronization** between two systems is ensured by an **event flow**, the total **resynchronization** of these systems must be **planned at design time**.

An automatic **synchronization audit** (example: by samples) allows to **measure** and **detect** any possible synchronization **errors**.



RULES OF THE GAME

[CENTRALIZATION]

Services **configuration** is **centralized**, their **discovery** is managed through a **directory**.

MAIF

RULES OF THE GAME

[CENTRALIZATION]

Micro-services configuration is **centralized** for all **environments**.

A central **directory** ensures **dynamic discovery** of **micro-services**.

The IS' global **scalability** depends on this **directory**.

RULES OF THE GAME

[SANDBOX]

Feature Teams provide a **sandbox environment**.

RULES OF THE GAME

[SANDBOX]

Feature Teams maintain a **sandbox environment** (current and upcoming versions) to allow other teams to develop **at scale**.

In **non-nominal cases**, **some features** may be **disabled** in the **development** environment.

RULES OF THE GAME

[DESIGN FOR FAILURE]

Your system will crash!

Design it so that it is fault-tolerant.

MAIF

RULES OF THE GAME

[DESIGN FOR FAILURE]

Your **system will fail**, it's inevitable. It must be designed for this (**Design For Failure**).

Plan **redundancy** at every level: **hardware** (network, disk, etc.), **application** (multiple instances of applications), geographical **zones**, **providers** (example: AWS + OVH).

RULES OF THE GAME

[TOOLKITS]

Provide **toolkits**, do not impose strict frameworks.

RULES OF THE GAME

[TOOLKITS]

Beware of transversal in-house technical components! They are restrictive, expensive and difficult to maintain.

Accelerators, **toolkits**, **technical stacks** can be **pooled**, **at the choice** of the Feature Teams, avoiding a dogmatic approach.

RULES OF THE GAME

[CLOUD]

Public, private or hybrid, the **cloud** (**IaaS** or **PaaS**) is the standard for production.

RULES OF THE GAME

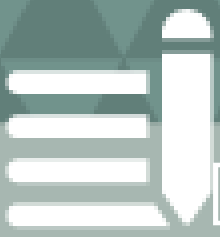
[CLOUD]

PaaS services are **preferred**, **simple**, and scale quickly.

IaaS services allow for greater **flexibility** but require more operational work.

A private cloud is not a traditional virtualization environment, it relies on **commodity hardware**.

RULES OF THE GAME



[INFRASTRUCTURE]

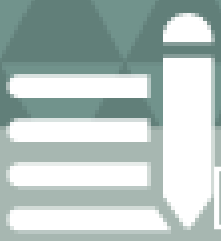
Feature Teams do not manage the infrastructure, it is **provided and maintained by the organization**.

MAIF

RULES OF THE GAME

[INFRASTRUCTURE]

Infrastructure issues are not the responsability of the **Feature Teams**. Infrastructure must be **provided** and **maintained** by a **transversal** service.



RULES OF THE GAME

[CONTAINERS]

Containers provide the flexibility needed for heterogeneous tooling.

MAIF

RULES OF THE GAME

[CONTAINERS]

Containers provide the **flexibility** needed by Feature Teams to enable **heterogeneous tooling** in a **homogeneous context**.

MAIF

RULES OF THE GAME

[ENVIRONMENTS]

The use of **containers** makes it possible to overcome the problems of **technical environments**.

RULES OF THE GAME

[ENVIRONMENTS]

The containers (example: **Docker**) make it possible **to abstract** the environments differences.

The **deployment** process must be **agnostic** to the environment.

Some components such as databases should not be deployed in containers. Their deployment is still automated.

RULES OF THE GAME

[METRICS]

Metrics must be **centralized** and **accessible** to all.

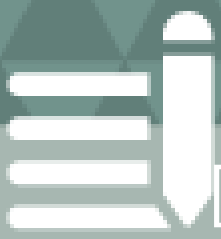
RULES OF THE GAME

[METRICS]

Metrics are **accessible** to everyone with different levels of granularity: detailed view for the relevant Feature Team, aggregations for other members of the organization.

Access to metrics **does not imply access to data**, access must be controlled to maintain confidentiality.

All environments are concerned.



RULES OF THE GAME

[QUALITY]

Software quality is a **key factor**.

RULES OF THE GAME

[QUALITY]

Code reviews are **systematic**. They are conducted by members of the Feature Team or other members of the organization, as part of **Continuous Improvement**.

That **is not you being audited but your code**: "You are not your code!".

Qualimetry can be partly automated, but nothing beats the **"fresh eye"**.

RULES OF THE GAME

[AUTOMATED TESTING]

Automated testing is a non-negotiable prerequisite for continuous deployment.

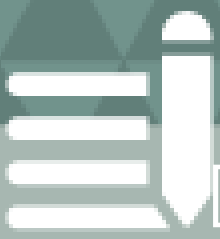
RULES OF THE GAME

[AUTOMATED TESTING]

Automated **testing** ensures **quality** of the product **over time**.

It is a **prerequisite** to continuous deployment, it allows for **frequent changes and deployments**.

Production rollout becomes a **non-event**!



RULES OF THE GAME

[TEST LEVELS]

Tests at all levels : unit, integration, functional, resilience, performance.

RULES OF THE GAME

[TEST LEVELS]

Integration and **functional** tests are the most important ones, they **ensure** the correct **behavior** of the **product**.

Unit tests are suitable for **development**.

Performance tests measure performance **over time**.

Resilience tests help to anticipate **failures**.

MAIF

RULES OF THE GAME

[COVERAGE]

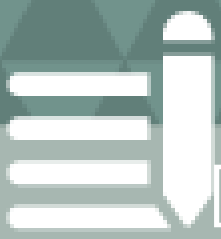
Coverage is the primary unbiased indicator of test quality.

RULES OF THE GAME

[COVERAGE]

The tests **code coverage** is a **good** metric of code quality.

It is a **necessary condition** but **not sufficient**, the coverage of a **bad** test strategy can be high without guarantee that the code is of good quality.



RULES OF THE GAME

[SECURITY]

Security is a **process**, it should not be treated in response to problems.

MAIF

RULES OF THE GAME

[SECURITY]

Security experts can be **integrated** directly into Feature Teams **if needed**.

Security experts are available in the organization for **audit** purposes, **awareness** and **sharing**.