



Software Architecture & Quality

Jean-Guillaume Louis
(j3l@ecam.be)



References

Clean Code: A Handbook of Agile Software Craftsmanship (Anglais) (Robert C. Martin)

Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin)

Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series (Fowler))

Refactoring: Improving the Design of Existing Code (Addison-Wesley Signature Series (Fowler))

[Interesting talks - Youtube playlist](#)

What is Software Architecture?



Why? What? How?

What we build is the architecture.

It's the system itself.

It's also the process that builds, test and deliver the software.



Why? What? How?

... But we first need to know **why** we build it.

... For **whom**?

... For **what**?

... Under what **constraints**?

... Optimized toward what **characteristics**?

Think “analysis”, “requirements gathering”,
“planning”, ‘scope definition”



Why? What? **How?**

Some call it “Design”.

It’s the nitty gritty of the system implementation. All the (ugly) details holding the system together.

Think “implementation choices”, “design pattern”, “tooling”



Why? What? How?


Where is the frontier?

Answer is: **you shouldn't care.**

There is not point at which, during a the building of a system, you should stop asking yourself these questions.

In practice, architecture is an ongoing activity.

A much better definition



“A shared understanding of the system”

- *Ralph E. Johnson*

The challenge being in how you communicate that knowledge.

Architecting is about communicating the intent of your system.

What is Quality?



Quality

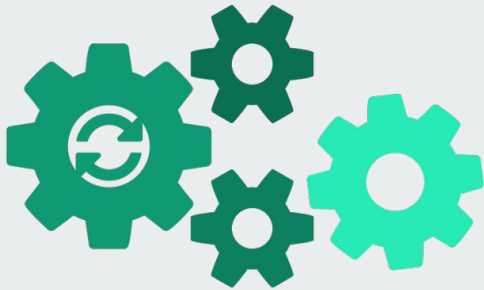
Quality of a system is typically on a multi-criteria scale.

To improve quality of a system, you need to improve the characteristics that matters.

What matters for one system won't matter the same for another.



Maintainability



Maintainability is a hidden quality metrics for the end user.

But it's arguably the only one that matters to the developer!

Always build with maintainability in mind.

Common used terminology when speaking about maintainability problem is “*technical debt*”

Technical debt

The accumulation of technical issues in a project:

- Bad code design
- Bugs
- Untested code
- Undocumented subjects
-

Leads to



... Time waste! (with interests)



Different type of technical debt

Accidental

Happens when you don't fully understand the problem at hand.

Favored by inexperience or lack of shared technical documentation.

Deliberate

Deadlines happen. There is not always the time to make things right.



“Bit rot” is a thing!

Code degrades over time.

Entire portion of codes become outdated, misunderstood or simply unused.

External factor might also factor in. Dependency are no longer satisfied.

Bug might become apparent in only certain circumstances



Security



Many aspects:

- Data protection
- Anonymisation (GDPR)
- Data integrity
- Authentication



Performance



Performance metrics comes in many flavours: processing time, memory usage, network latency...

Performance is measurable.

When you improve for performance, you need to start measuring.

... Remember: you cannot have it all!



Scalability



“The ability to scale the load of the system”

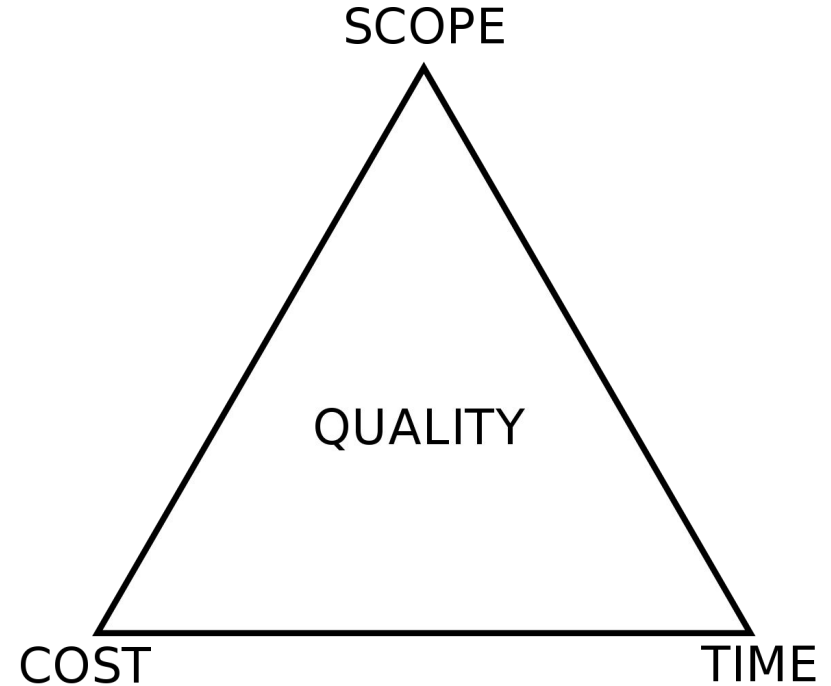
Whatever *load* means: network traffic, user count, concurrent requests, storage space, computing power.



The dilemma

Aka “project management triangle”

Or “Good, fast, cheap. Pick two.”

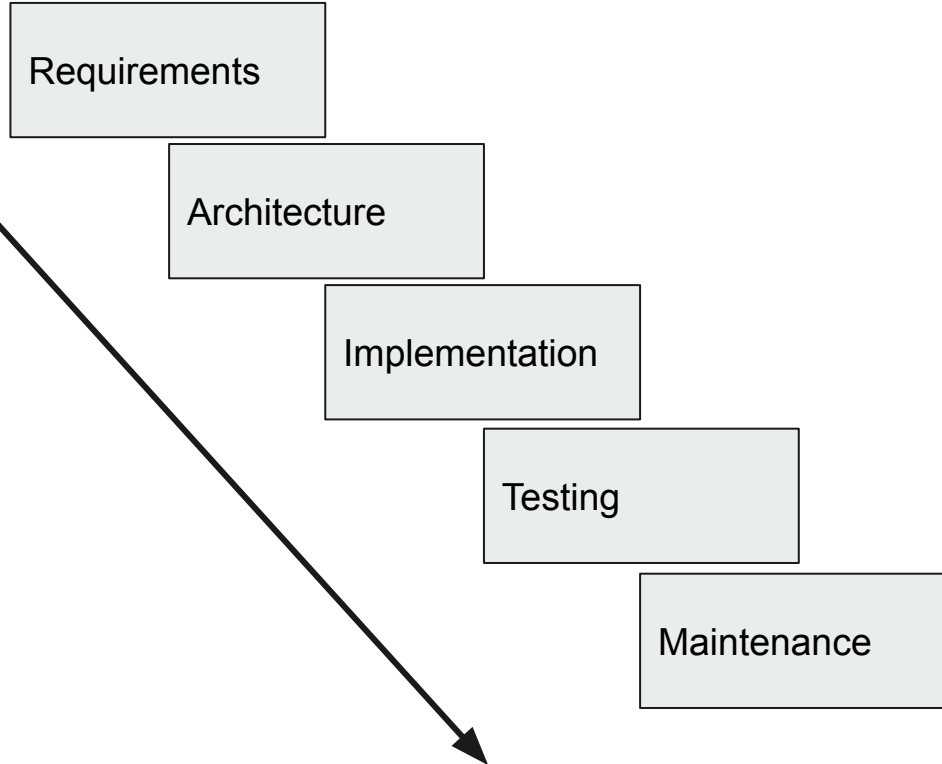


(...Except, this is not just a triangle)

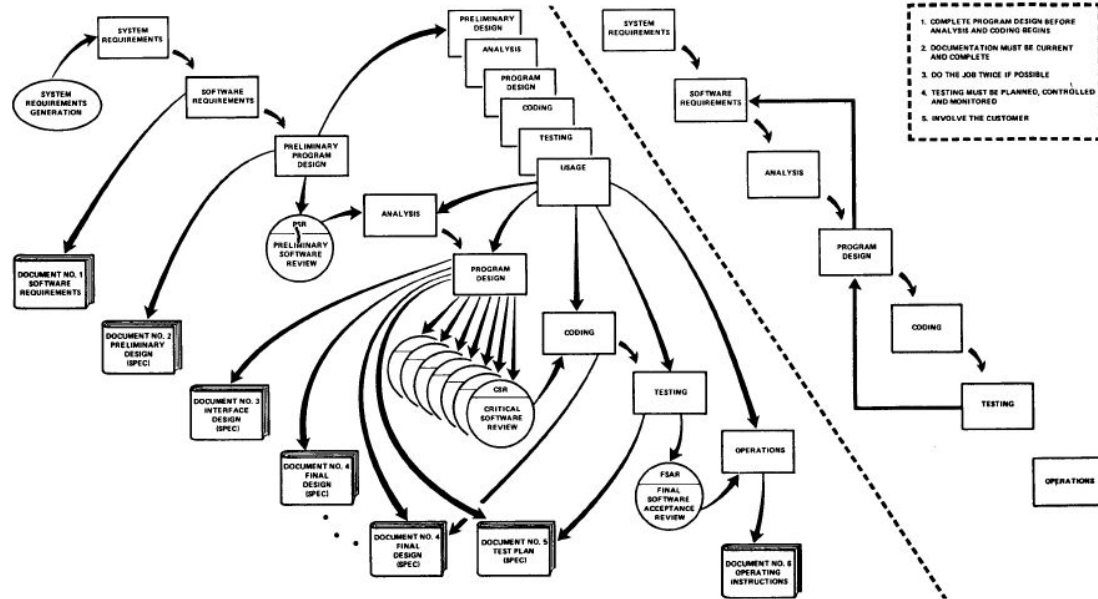
Methodologies



Waterfall

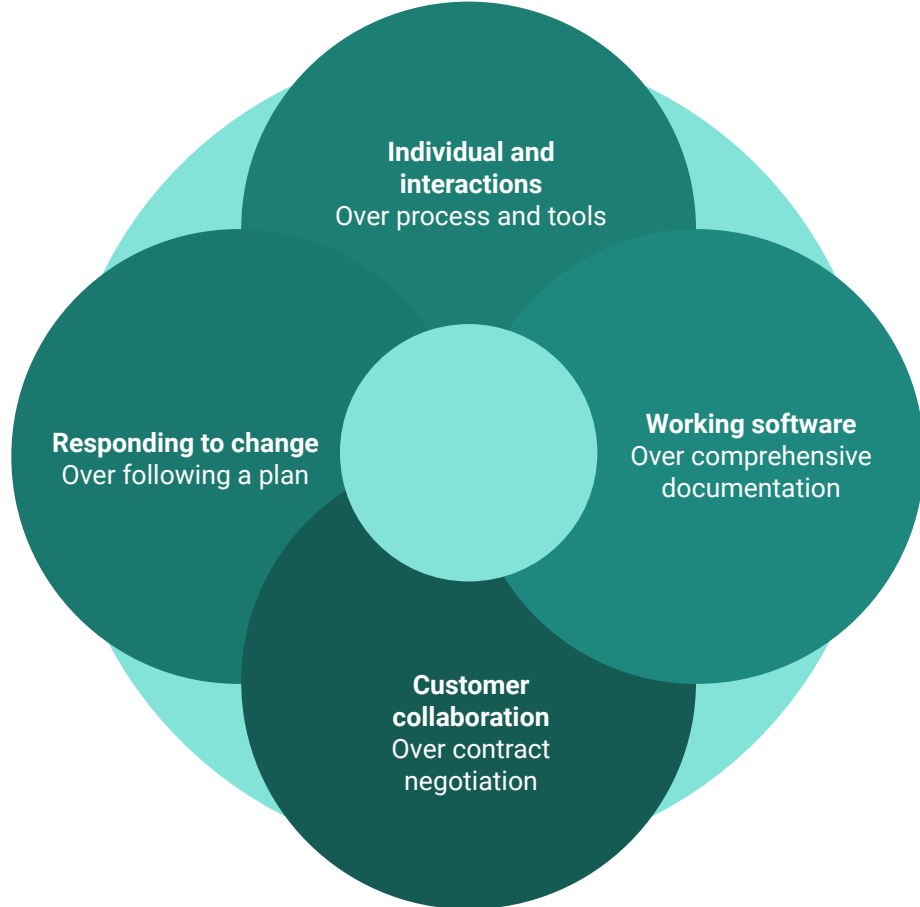


Improved Waterfall (?)





Agile



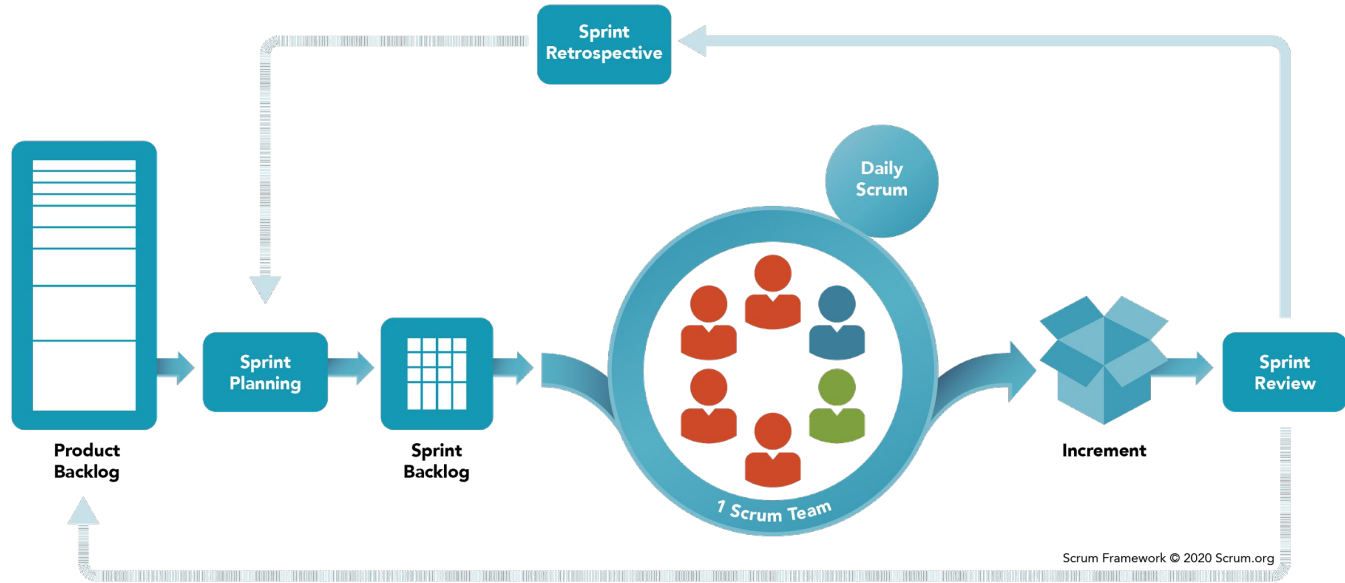
<https://agilemanifesto.org/>



Derived practices (a few examples)

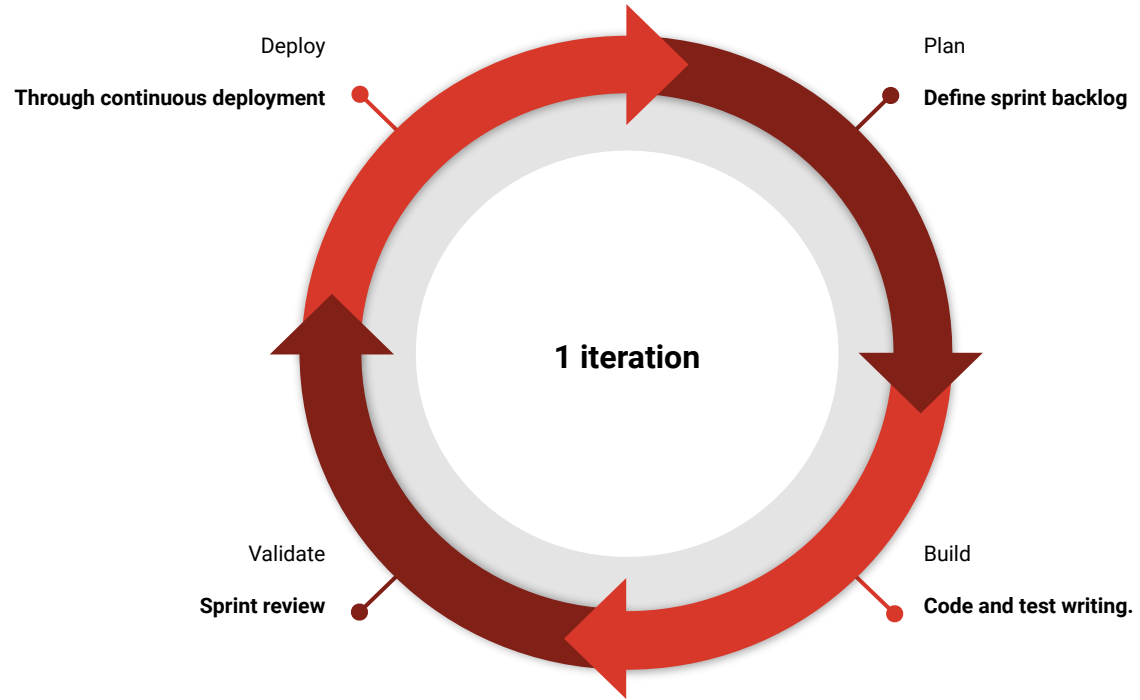
- Iterative development
- Test driven development
- Continuous integration
- Velocity metric and burndown charts
- Planning poker
- (Time-boxed) sprints
- Self organized teams

Scrum framework



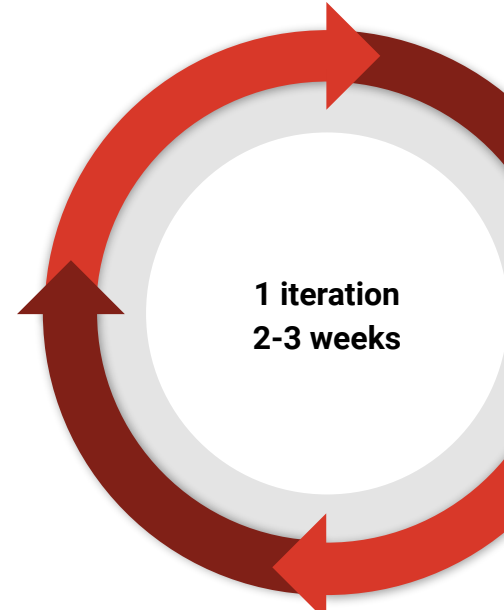
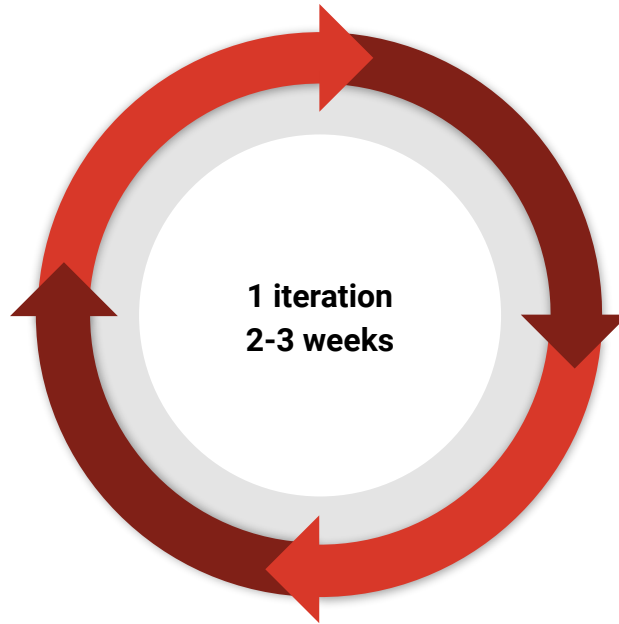
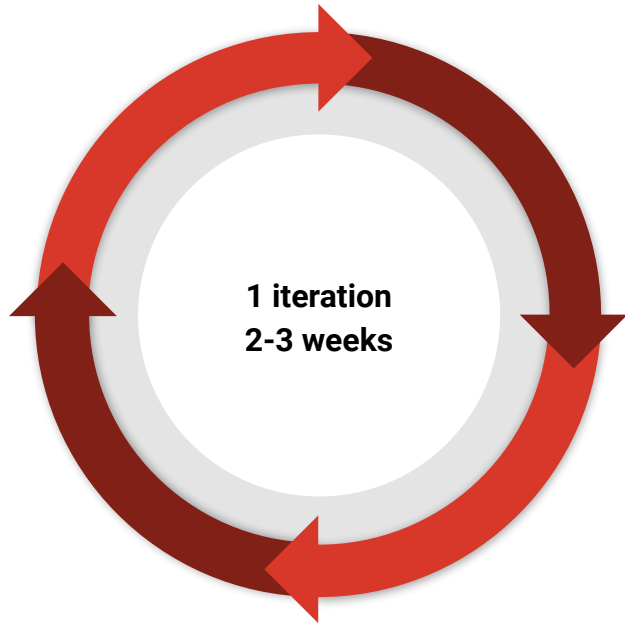


Software iteration

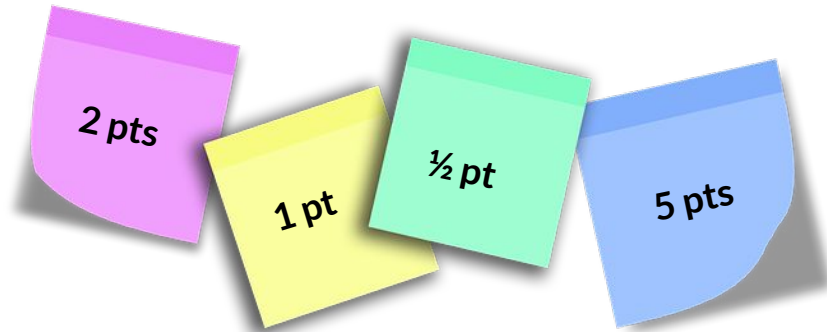




Project in motion, sprint by sprint

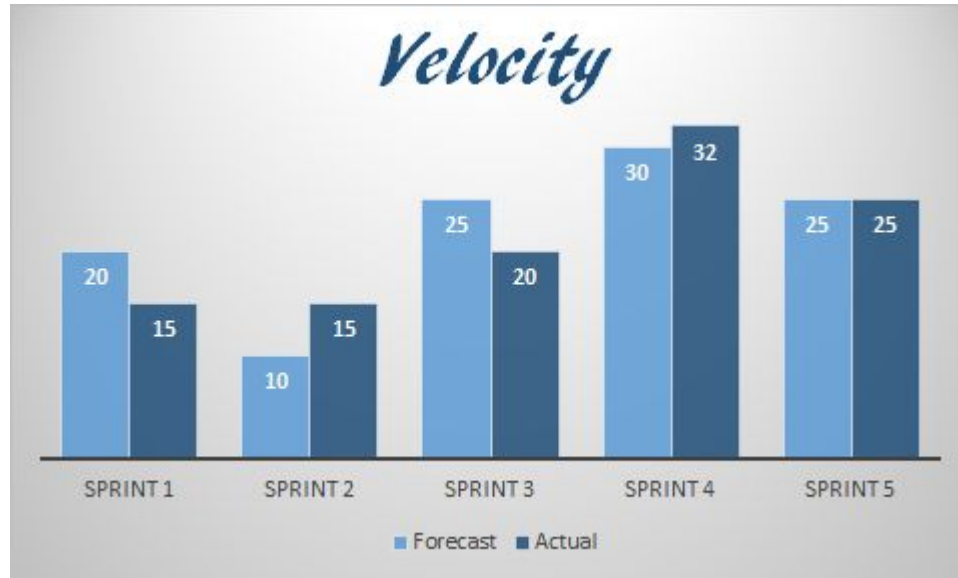


How much work in each iteration?



Assessing team
velocity after each
sprint

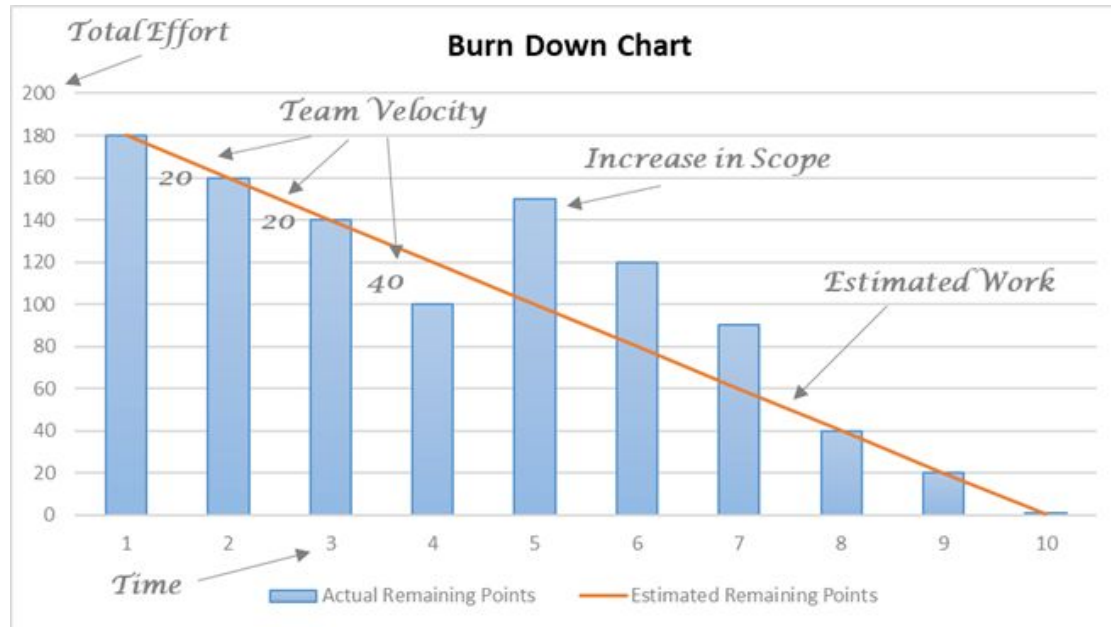
Velocity



Ask developers
“how much effort?”,
not “how much
time?”



Burndown chart



Cow boy

“It’s faster to make the change in production”

“Rockstars programmers”

“Unit tests are a waste of time”

“It works on my machine”

“I don’t trust automation”



Drawing boundaries



Why it matters

Early decisions in a project can lead to costly consequences.

Choice of frameworks, database, web server, hardware and whatnot can lead to strong coupling.

You also don't want to absorb technical debt from outside.



What boundaries

Boundaries to the outside world

Network

Operating system

Third party services

Boundaries within your system

UI

Database

Framework, libraries

Hardware (abi)



Where to draw boundaries

Maximize interoperability.

Think of dependency flow.



Defer decisions

Avoid committing to early decisions.

Think your architecture so you can defer most of the decisions until you have enough information.

Because:

Later in the design, you will have a better understanding of your requirements.

Assessing your team's workflow



Version Control System



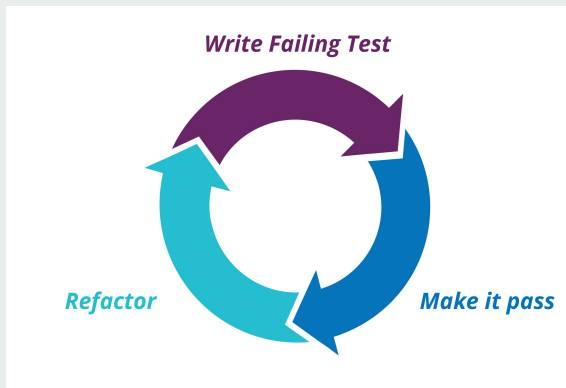
Provides:

- Code historization
- Backup
- Code sharing

Two types:

- Centralized (SVN, CVS, Perforce,)
- Decentralized (Git, Mercurial)

Automated Testing



- **Unit testing**

Test one “software unit” (a class, a method, a function) in **isolation**.

- **Integration testing**

Test multiple “software units” (possibly the whole software)

“You only test the parts of the application you want to work” - Uncle Bob

Continuous Integration/Delivery



Beware that as the products grows, the number of tests to validate the product grow. Some other part of the process might also complexify (deployment, maintenance, etc.)

Automation becomes necessary to keep that growth in check.

If you mostly rely on human operations in your process, the ability to deliver fast will deteriorate.

Code review



Benefits:

- Cross validation of the code
- Spread knowledge of the code across the team
- Ensure there is no retention of information
- Help communicating exigence

Architecture models

Layered architecture



Each layer is a **logical** separation.



Strictly closed layers

One layer has only access to the layer below.

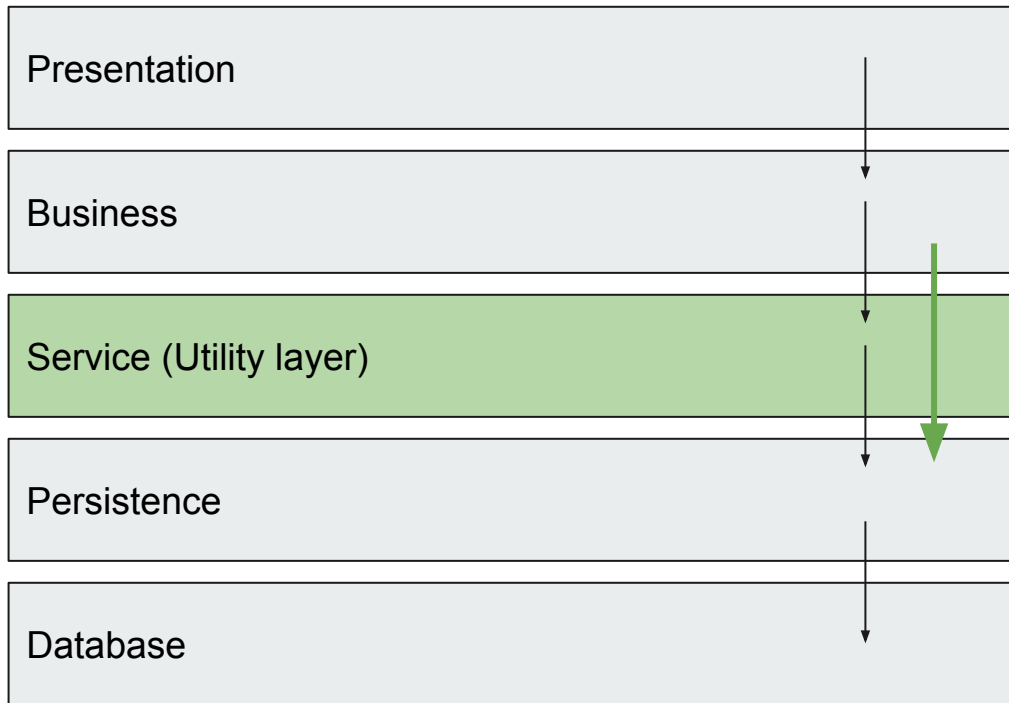
The layer is agnostic of the rest of the application.



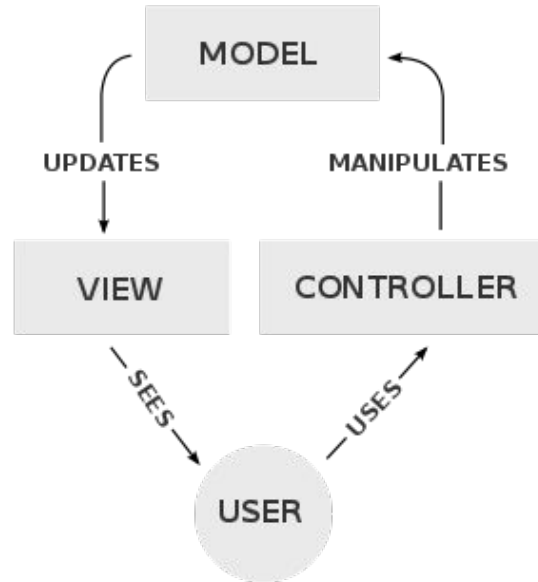


Open Layers

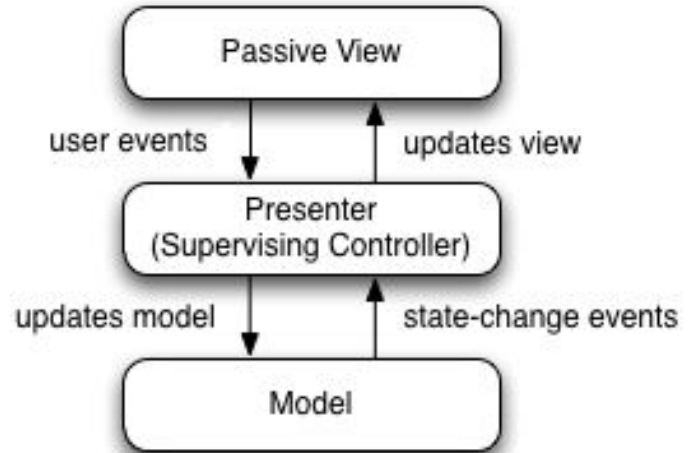
An open layer may be traversed.



Model View Controller

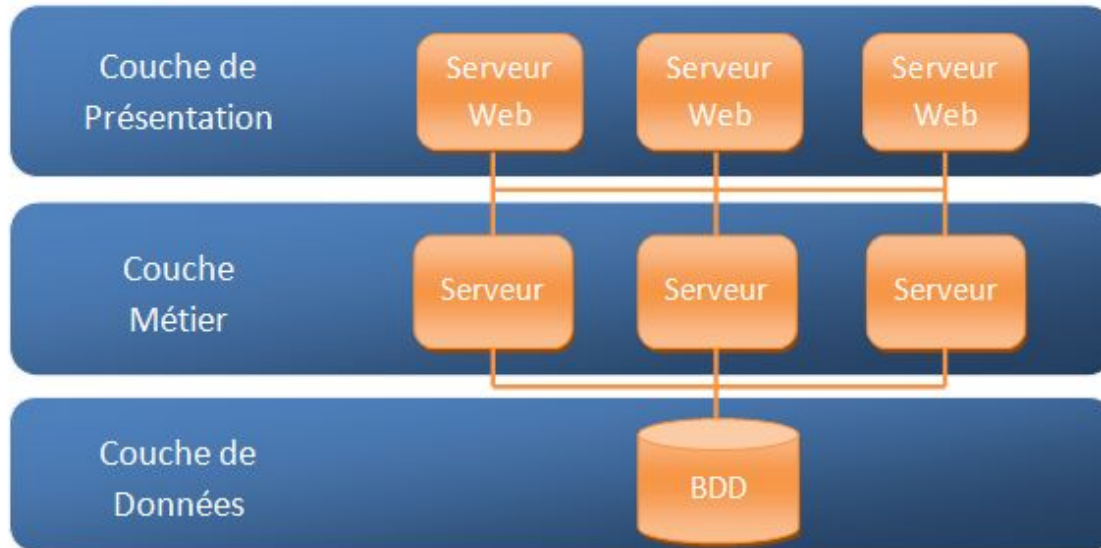


Model View Presenter



Osi Model		TCP/IP Model	Description
7	Application	Application	HTTP, SMTP, FTP
6	Presentation		Jpeg, Mpeg, ASCII, UTF-8
5	Session		AppleTalk, NetBIOS, RPC
4	Transport		TCP/UDP
3	Network		Ip addresses and routing protocols
2	Data Link		Transmission & Control of data frames Mac addresses and switch
1	Physical		Raw bit streams over copper and/or optical fibers

N-Tiers

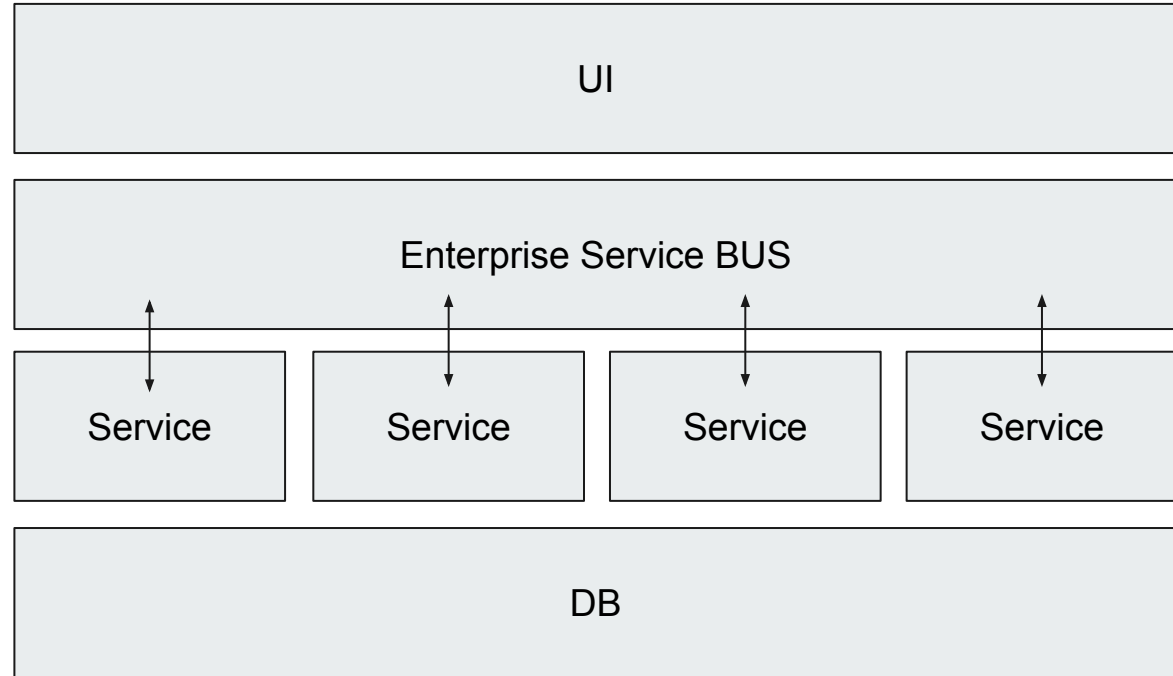


Each tier is **physically*** independent, favoring scalability.

*each component might reside in different processes or machines.

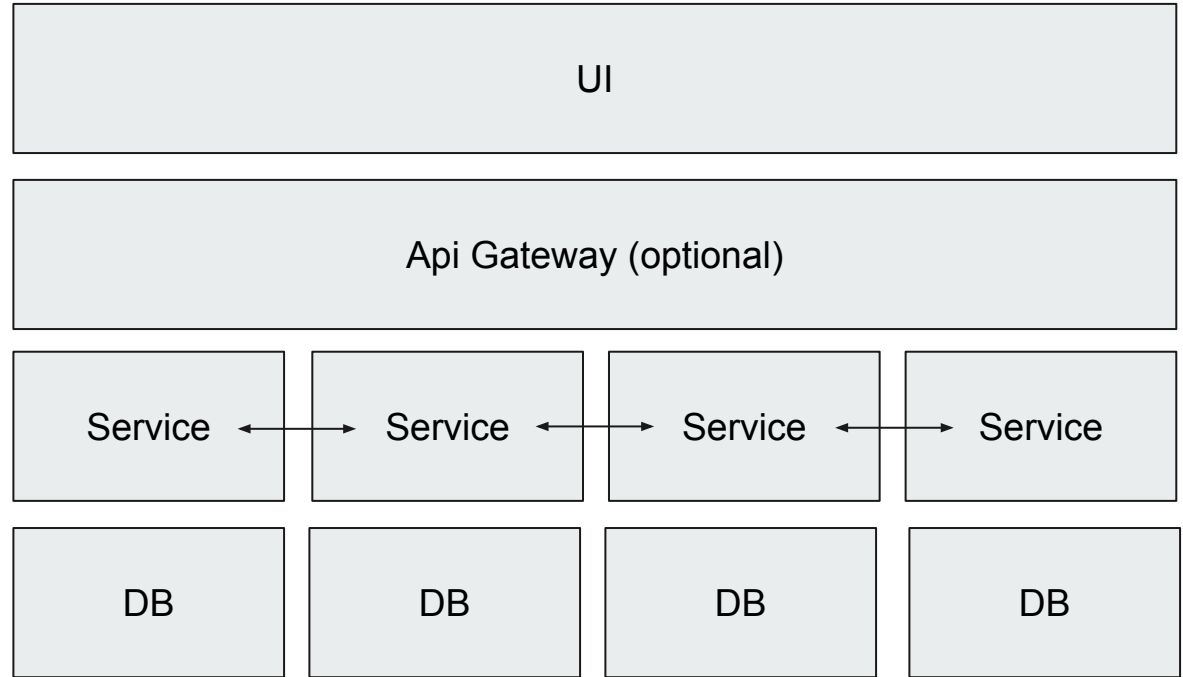


SOA





Microservices



Sample questions

One of the 4 core values in the Agile Manifesto is
"working software over comprehensive documentation".

How do you ensure your software is staying in a working
state?

When is scalability important to consider?

Can you give an example of an existing software where scalability is a critical criterion?

In the Scrum methodology, what's the reason for asking developers for "points" instead of "hours" when they are evaluating a task?

What do points represent?

Your team has trouble meeting deadlines when it comes to delivering software. Your delivery process is getting longer every iteration.

What could be the possible root causes? How would you try to solve the issue(s) at hand?
