Measuring our efficiency

We are continuously trying to improve our practices, speed up our developments, secure our deliveries by:

- implementing automation and processes
- extending our culture of ‘well done’ things and quality
- aiming excellence in what we deliver
- and many other actions

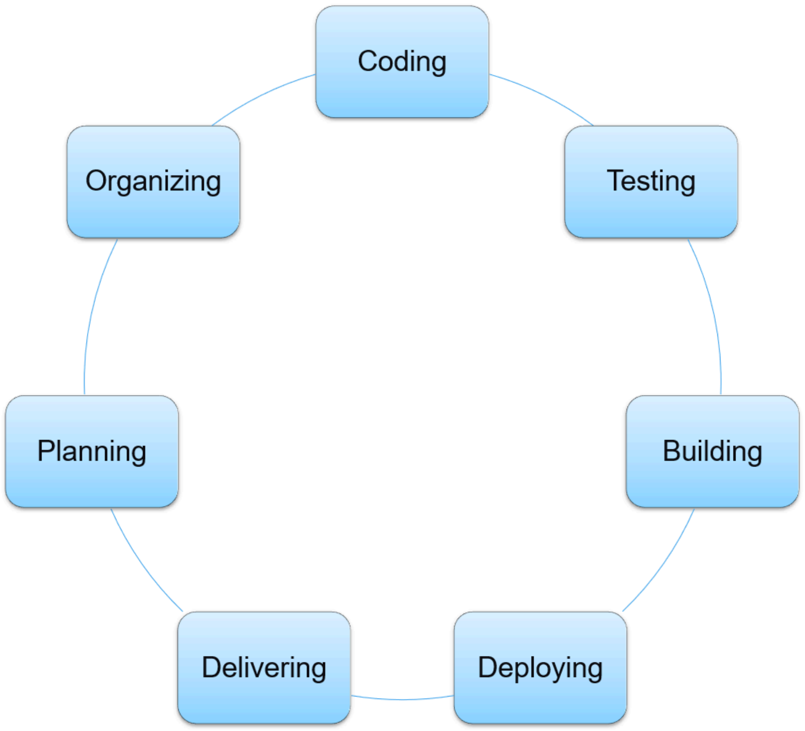
We are following, for a long time, the count of issues in production: this is a fundamental and important indicator to evaluate our customer’s experience using our product and to detect the ‘poor quality’ components in our solution.

This is allowing us to define proper investments to ‘heal’ our product where it is necessary but it’s not enough to know how efficient we are in our way of producing software all together. This is also useless to proactively invest in improvements area in an anticipation strategy.

Thus we need to measure other aspects of how we do things to improve more, define targets and follow our progression, not waiting to ‘only’ react to production effects.

Scope

This page relates to all activities that are part of producing software for our customers:



Everyone in tech department plays a crucial role in at least one of those activities and can actively contribute to our performance.

What is “our efficiency”?

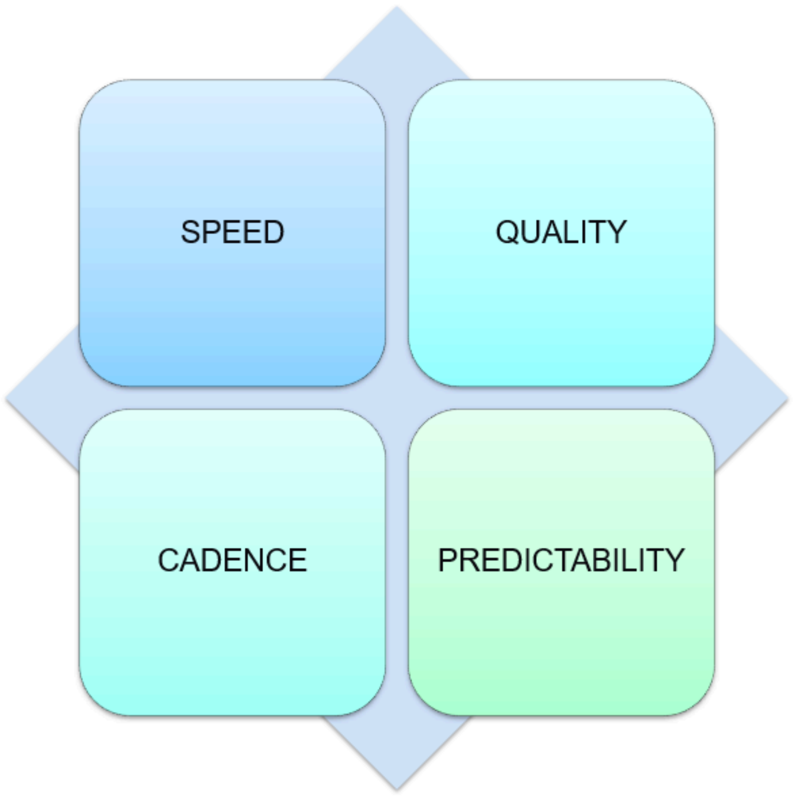
Our efficiency in producing software may be considered as the ROI (Return On Investment) of engaged efforts by everyone in Tech Department.

There are many options to improve this efficiency:

- reduce effort by automating manual tasks (CI, test, deployment etc…)
- accelerate by improving tools/processes
- facilitate daily work by improving workflows
- remove obstacles when detected (both technical / organizational)
- reduce wastes in a lean continuous improvement approach (partially done work/re-work, relearning, delays, context switching….)
- improve developer experience
- removing/reducing “inefficiency”: feel free to contribute to this!

What can we measure to know if we “do well”?

In order to know where we are, to define goals and evaluate our evolution over time, we need to measure things! Indicators should be setup to give us proper observability of our way of producing software.



“Doing well” is really subjective and can be considered factually as:

- How **GOOD** is the software we deliver to our customers?
- How **FAST** do we deliver software to our customers?
- How **REGULAR** are our software deliveries to our customers?
- How **PREDICTABLE** are our software deliveries to our customers?

Indicators

We have a lot of data related to our way of producing software, mainly available from the tools we are using in our daily work. Here are indicators we can follow to confirm our efficiency in different ways.

How GOOD are our software?

Debt ratio

- Definition
  - The ratio between the cost to develop the software and the cost to fix it (as defined in sonarqube)
- Impact
  - Maintainability
  - Team's productivity
  - Product's evolutivity
  - Developer experience
- Measure
  - Sonarqube
- Evolution
  - **BAD WHEN INCREASED**

Nb production bugs

- Definition
  - Count of bugs identified on production environment
- Impact
  - Functional suitability (completeness, correctness, appropriateness)
  - Customer's satisfaction
  - Time spent on maintenance
- Measure
  - JIRA
- Evolution
  - **BAD WHEN INCREASED**

How FAST do we deliver ?

Lead Time (stories)

- Definition
  - Time from item (story) creation to last transition to DONE status
  - Lead time
- Impact
  - Time to market
  - New features availability
- Measure
  - Jira / Gitlab
- Evolution

    - BAD WHEN INCREASED

EPIC stretch

- Definition
    - Time between EPIC’s first and last child (story) implementation (or to EPIC’s transition to DONE if reached before all stories are implemented)
- Impact
    - Time to market
    - Focus / context switching
- Measure
    - Jira
- Evolution
    - BAD WHEN INCREASED

How REGULAR are our deliveries ?

Merge Requests count

- Definition
    - Count of merge requests merged
- Impact
    - Long living branches
    - Not delivered code
    - Branches maintenance cost (merge/rebase)
- Measure
    - Gitlab
- Evolution
    - GOOD WHEN INCREASED

Merge Requests size

- Definition
    - Size of merge requests (amount of code added/modified/deleted)
- Impact
    - Delivery risk
    - Integration into main branch costs
- Measure
    - Gitlab
- Evolution
    - BAD WHEN INCREASED

How PREDICTABLE are our deliveries ?

% PI Objectives reached

- Definition
    - Ratio of defined objectives reached on time (as planned by the teams)

- Impact
    - Planning
    - Roadmap delays
    - Commitment
- Measure
    - Manual
- Evolution
    - GOOD WHEN INCREASED

Leaked items

- Impact
    - Partially done work
- Measure
    - ‘Sprint’ / ‘fix version’ changes (JIRA)
- Evolution
    - BAD WHEN INCREASED

Measuring indicators

Interpreting and measuring indicators can easily lead to wrong conclusions. It’s important to consider different aspects of ‘measuring things'.

Goodhart’s law and biases

As stated by Goodhart in “Goodhart’s law”, *every measure which becomes a target becomes a bad measure.* That’s exactly why all indicators should be considered together to feel our efficiency in producing software.

It’s good to increase some, decrease others but it’s always important to consider all indicators together to state that things are evolving in the right direction.

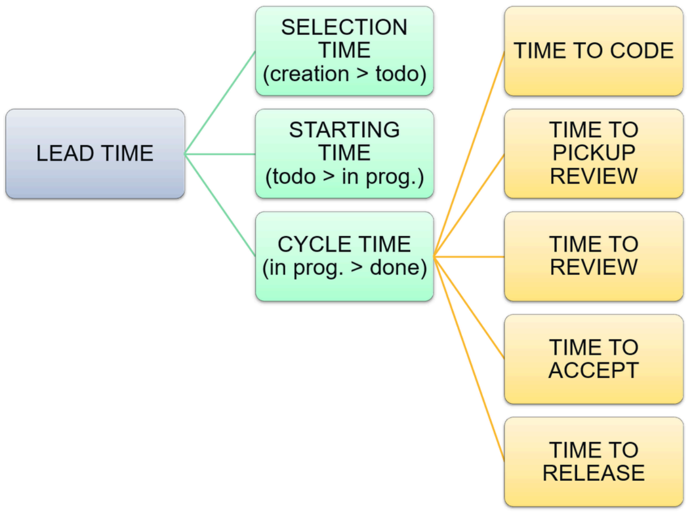
Don’t define indicator increase/decrease as an objective : define potential improvements, then confirm them by all indicators evolution.

One single indicator can easily be misinterpreted, as it sometimes does not contain all information we need to take the right decision.

- Is it good to reach 100% planned objectives? It depends if your planning is too much or not enough challenging…
- Is it good to merge the code faster? It depends if it raises the count of bugs in production…
- Is it a performance to have reduced lead time? It depends if you increased debt ratio in the code…
- …

Correlations : focus on lead time

Indicators could be more or less correlated, “Lead time” is a good example of a ‘global’ indicator, composed by many others.


LEAD TIME
SELECTION TIME (creation > todo)
STARTING TIME (todo > in prog.)
CYCLE TIME (in prog. > done)
TIME TO CODE
TIME TO PICKUP REVIEW
TIME TO REVIEW
TIME TO ACCEPT
TIME TO RELEASE

Indicators composing “Lead time”

It’s then useless to report all of them separately as they are strongly correlated. But all indicators composing ‘lead time’ can be interesting to identify bottlenecks and pain points, then define actions to improve.

For example, we can work on improving team’s reactivity when code is ready for review, this will improve ‘time to pickup review' which should have a positive impact on lead time. Pain points might be different, depending on teams, it’s important to have a detailed view on specific workflows by considering those more detailed measures.

Indicators in PowerBI

Detailed data

Granular data, gathered from different tools, is available in PowerBI. Feel free to explore your state of art:

- Sonarqube: Power BI
- Gitlab / JIRA: Power BI
- QA indicators (JIRA PAOSUP issues) : Power BI

Global dashboard

We are working on a global (and more visual) overview of our efficiency. This is coming soon…