



Software Maintenance

Mireille Blay-Fornarino

Site web : <https://mireilleblayfornarino.i3s.unice.fr/doku.php?id=teaching:reverse:2018:start>

“The code is the truth” – Sam Guckenheim

“The code is the truth, *but not the whole truth*”

– Grady Booch

Hypothèses de travail

- Vous connaissez
 - les principes SOLID, GRASP
 - les design patterns
 - les architectures logicielles de base
- La maintenance de code : ça vous dit quelque chose.
- Vous avez des capacités à l'abstraction
- Vous êtes des développeurs avancés
- **Vous avez envie d'apprendre par vous-même**

Pour comprendre/modifier
un système informatique
complexe, vous faites quoi ?

Pour vous ?



Enseigner la Maintenance Logicielle



GIFWAVE.COM

« réputée non-enseignable ».



Maintenance ? Rétro-Ingénierie ?

CodeScene @codescene · 14 déc.

Thanks for your kind words! The hotspot numbers indicate that roughly 17% of your development is in just 1.7% of the code, so any potential technical debt there is likely to come with a high interest rate. You can read more on the topic here: empear.com/blog/prioritiz...



Daniel Fesenmeyer @fesicraft

Just tried out @codescene on #EclipseDitto. I really like it so far! 👍

As I understand, we could reduce some coordination and merge effort by reducing the size of our hotspots....

Traduire le Tweet

3 4



Israel Herraiz @herraiz · 8 h

En réponse à @fabriengirardin

Technical debt as a tool. As Don Knuth said, premature optimization is the root of all evil. But to keep **technical debt** under control, you must be able to measure it and allocate and reserve time to repay it in the future. Good principles, also for any data science team.

Traduire le Tweet

1



Kuba Suder @kuba_suder · 13 nov.

you also shouldn't build every system preparing it from the first line of code to handle millions of users daily, because you'll make everything 100x more complicated than it needs to be and probably it'll all go to waste

Traduire le Tweet

1

1 1



Brian

@briennehning

Suivre

En réponse à @kuba_suder @marcoarment

Granted. But you should know where your technical debt is, and know what will be straightforward to fix later and what will be a nightmare

Traduire le Tweet

15:11 - 13 nov. 2018

Benjamin Benni a retweeté

Computer Science @CompSciFact · 23 h

Software reuse is more like an organ transplant than snapping together Lego blocks. ow.ly/FGDRX

Kent Beck a retweeté



Martin Fowler @martinfowler · 7 déc.

"the most important aspect of refactoring: practicing refactoring doesn't just improve our code, it helps us become better at reasoning about our code"

linkedin.com/pulse/testing-... @mfeathers

Traduire le Tweet



Testing Yourself - Growing Through Refactoring

These are interesting times for people who care about refactoring. The 2nd edition of Martin Fowler's book Refactoring has just been released.

linkedin.com

18

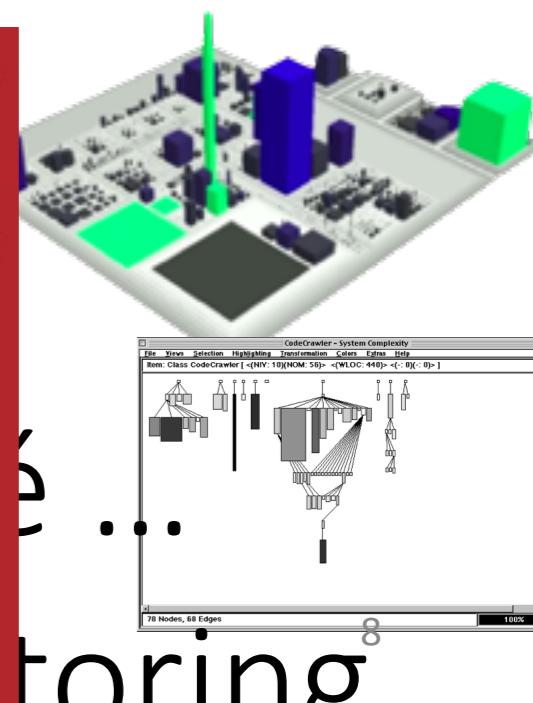
358

761

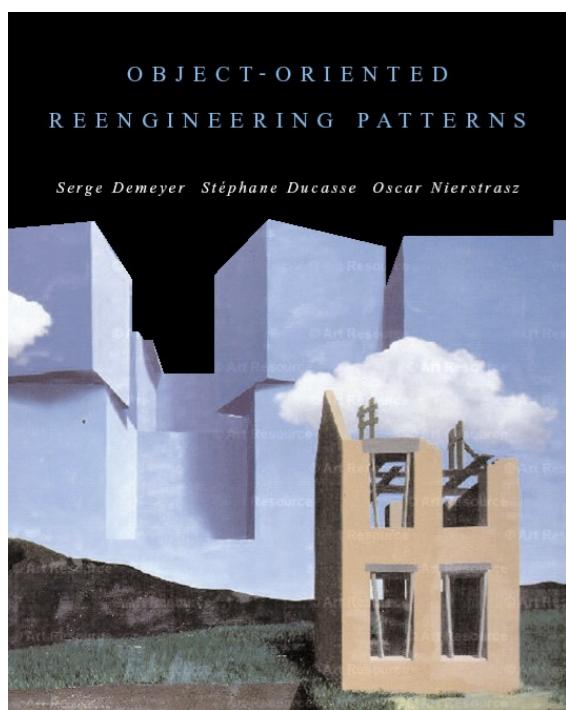
✉

Analyse de code ?

Code review analysis of software system using machine learning techniques. 2017



Refactoring... Maintenance?



Johan den Haan
@JohanDenHaan

Interesting research > Does Conway's Law apply to Linux? buff.ly/2rSwf7U

The Pragmatic Programmers

Investigator:
Date:
Case #:
Location:

Your Code as a Crime Scene

Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs

ISO/IEC 14764 IEEE Std 14764-2006 Second edition 2006-09-01

Subsystems

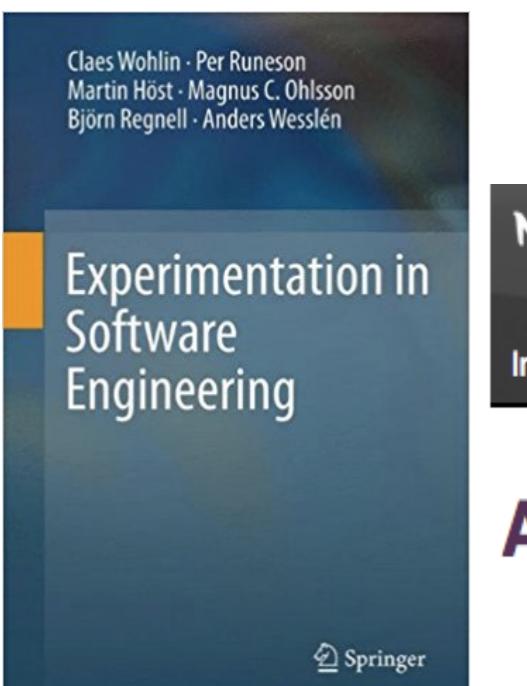
These arrows represent the "designed by" correspondence

(b) Percentage of specialists and generalists

Instruction of the atomic components in which designed in 14764

for (int j = 0; j < loc; j++) res[j] = buf[j];
return res;

public void checkRes(Res res) {
 for (int i = 0; i < res.length; i++) {
 if (res[i] != checkRes[i]) {
 Adam Tornhill
Foreword by Michael Feathers, author of Working Effectively with Legacy Code



MARTIN

The Pragmatic Programmers

Software Design X-Rays

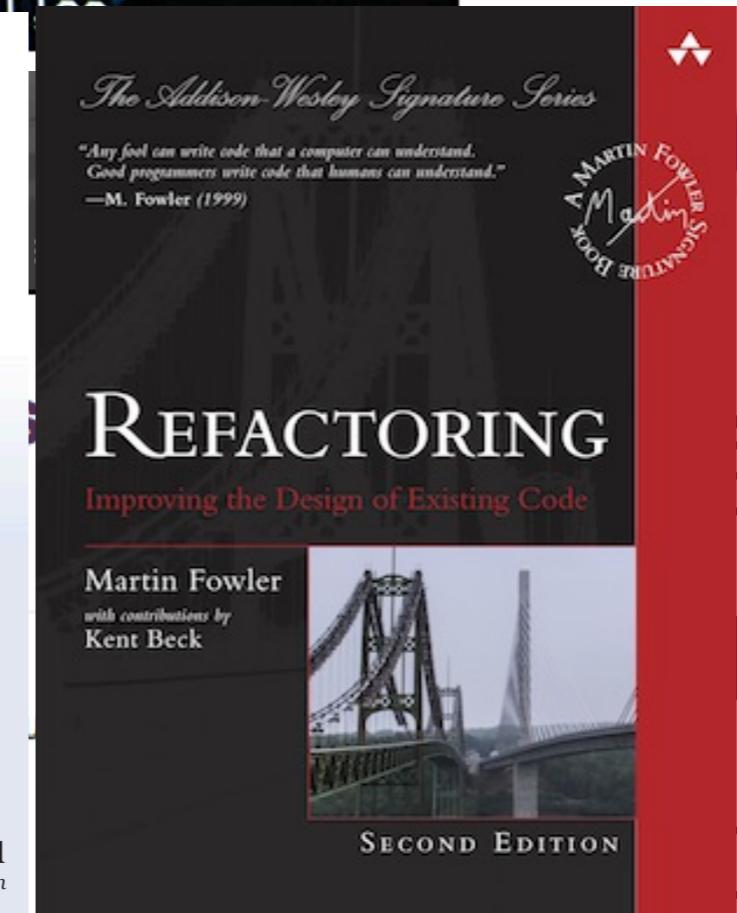
Fix Technical Debt with Behavioral Code Analysis

An Ap

Springer

Alex Terrieur @Tasakafei • 5
C'est marrant le métier d'arc meurtre et comprendre pour

1 1 1



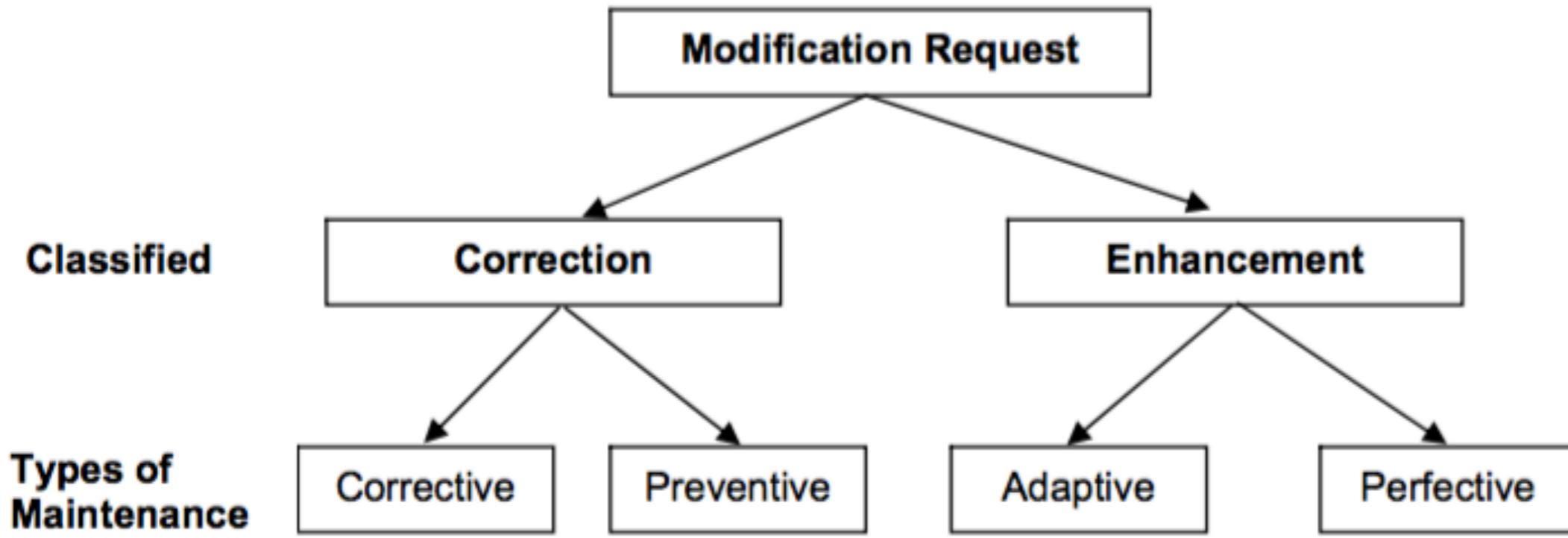
Adam Tornhill
edited by Adaobi Obi Tulton



La maintenance logicielle dans l'entreprise ?

Image : <http://bookkeepersme.com/software-maintenance-services-company-dubai/>

Definitions and terms



corrective maintenance : the reactive modification of a software product performed after delivery to correct discovered problems

emergency maintenance : an unscheduled modification performed to temporarily keep a system operational pending corrective maintenance

preventive maintenance : the modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults

adaptive maintenance : the modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment

maintenance enhancement : a modification to an existing software product to satisfy a new requirement

perfective maintenance : the modification of a software product after delivery to detect and correct latent faults in the software product before they are manifested as failures

Vocabulaire

Program Comprehension: Programmers spend considerable time reading and understanding programs in order to implement changes. **Code browsers** are key tools for program comprehension and are used to organize and present source code. **Clear and concise documentation** can also aid in program comprehension.

Reverse Engineering is the process of analyzing software to **identify** the software's components and their inter-relationships and to **create representations** of the software in another form or at higher levels of abstraction. Reverse engineering is passive; it does not change the software or result in new software. Reverse engineering efforts produce call graphs and control flow graphs from source code. One type of reverse engineering is **redocumentation**. Another type is **design recovery**. Finally, data reverse engineering, where logical schemas are recovered from physical databases, has grown in importance over the last few years. Tools are key for reverse engineering and related tasks such as redocumentation and design recovery.

Reengineering is defined as the examination and alteration of software to reconstitute it in a new form, and includes the subsequent implementation of the new form. It is often not undertaken to improve maintainability but to **replace aging legacy software**.

Refactoring is a reengineering technique that aims at reorganizing a program without changing its behavior. It seeks to improve a program structure and its maintainability. Refactoring techniques can be used during minor changes

A distinguer de

“*Forward Engineering* is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.”

Buts du Reverse

- Documenter, raisonner, **communiquer autour de l'application** dans un formalisme autre que du code
- **Comprendre la conception** ce qui permet de l'améliorer, la corriger, l'optimiser
- **Appréhender la complexité**: (taille, dépendances)
- **Retrouver des informations perdues** (e.g. quels changements? Pourquoi? qui?)
- ...

Comprehension activities

- A1. Investigating the functionality of (a part of) the system
- A2. Adding to or changing the system's functionality
- A3. Investigating the internal structure of an artefact
- A4. Investigating dependencies between artefacts
- A5. Investigating runtime interactions in the system
- A6. Investigating how much an artefact is used
- A7. Investigating patterns in the system's execution
- A8. Assessing the quality of the system's design
- A9. Understanding the domain of the system A set of typical software comprehension tasks should address all of these activities.

Reverse basé sur UML

Reverse Engineering: Aspect structurel

- L'opération de réverse génère un modèle UML à partir duquel il est possible d'extraire des vues (diagrammes)



Reverse Engineering: Aspect comportemental

- Un diagramme de séquences à partir d'une opération en Java
- Ne capture que la séquence d'appels d'opérations
- Indépendant d'un scénario en particulier

```
public void withdraw(double val) {  
    if (getBalance() >= val) {  
        setBalance(balance - val);  
        mHistory.incrementTransaction();  
    }  
    else  
        noAvailableFunds();  
}
```



Est-ce qu'un projet qui n'a
pas de « maintenance »
est meilleur?

Feedback loop

Because *the world irremediably changes*, .. software must be changed to keep it synchronized with its environment.

Software requirements also change, because *human processes are hard to define and state*, which also lead to modifications in the software.

Also, the very introduction of the system in the world will cause further demands for changes and new features. This last source of changes causes a feedback loop in the evolution of the program.

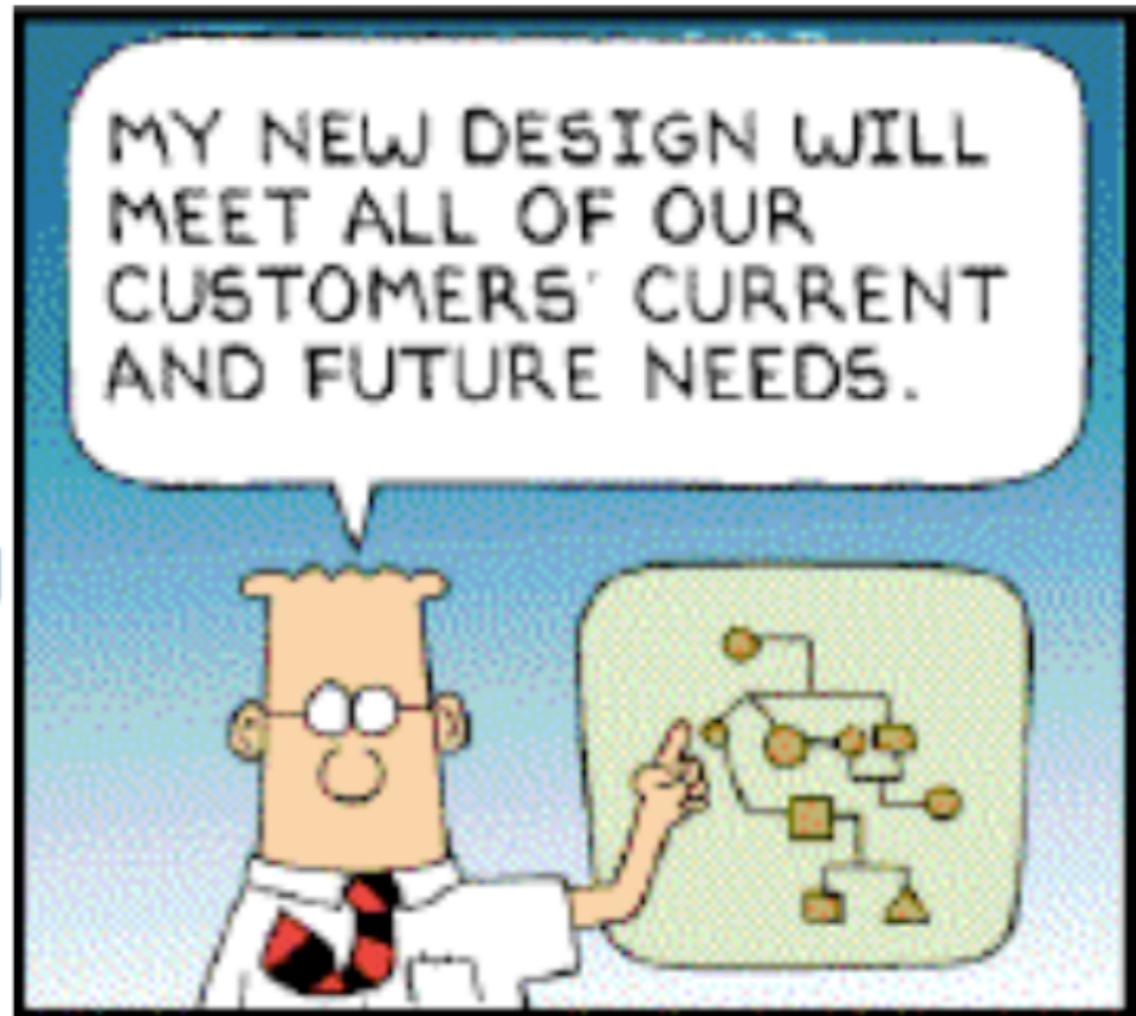


image : <http://tomcunniff.com/2012/03/old-vs-new-media-the-future-is-a-feedback-loop/>

Observations

Software change/evolution is

- inevitable
 - unpredictable
 - costly
 - difficult
 - time- and resource-consuming
 - poorly supported by tools, techniques, formalisms
 - underestimated by managers
 - poorly understood
-
- If performed well, a major success factor for business innovation !



Laws of software evolution [Lehman and Fernandez-Ramil 2006]

Three classes of programs:

- **S-type (specified)** programs are derivable from a static specification, and can be formally proven as correct or not.
- **P-type (problem solving)** programs attempt to solve problems that can be formulated formally, but which are not computationally affordable. Therefore the program must be based on heuristics or approximations to the theoretical problem.
- **E-type (evolutionary)** programs are reflections of human processes or of a part of the real world. These kind of programs try to solve an activity that somehow involves people or the real world.

Laws of software evolution [Lehman and Fernandez-Ramil 2006]

I Law of Continuing Change : An *E*-type system must be continually adapted, else it becomes progressively less satisfactory in use

II Law of Increasing Complexity : As an *E*-type is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.

III Law of Self Regulation : Global *E*-type system evolution is **feedback regulated**.

IV Law of Conservation of Organizational Stability : The work rate of an organization evolving an *E*-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.

V Law of Conservation of Familiarity : In general, the incremental growth (growth rate trend) of *E*-type systems is constrained by the need to maintain familiarity.

VI Law of Continuing Growth : The functional capability of *E*-type systems must be continually enhanced to maintain user satisfaction over system lifetime.

VII Law of Declining Quality : Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an *E*-type system will appear to be declining.

VIII Law of Feedback System: *E*-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

Laws of software evolution [Lehman and Fernandez-Ramil 2006]

Comment « vérifier » la véracité des lois ?

Quel est l'impact des changements des processus de développement ?

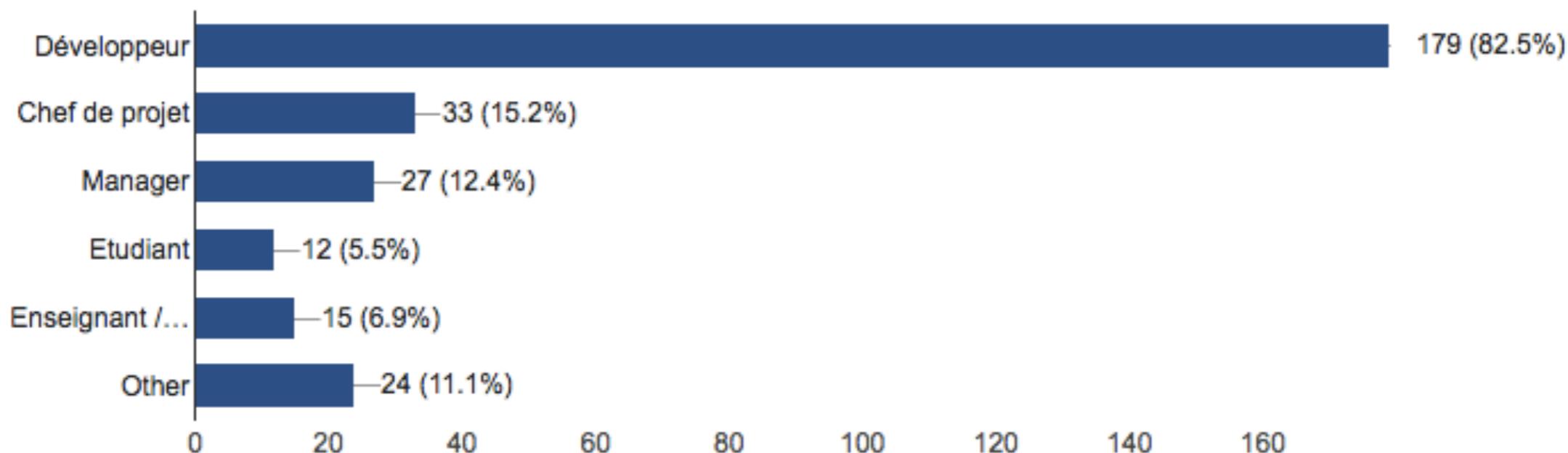
Ces règles s'appliquent-elles au logiciel libre qui n'intègre pas une notion de « hiérarchie »?

Un sondage...

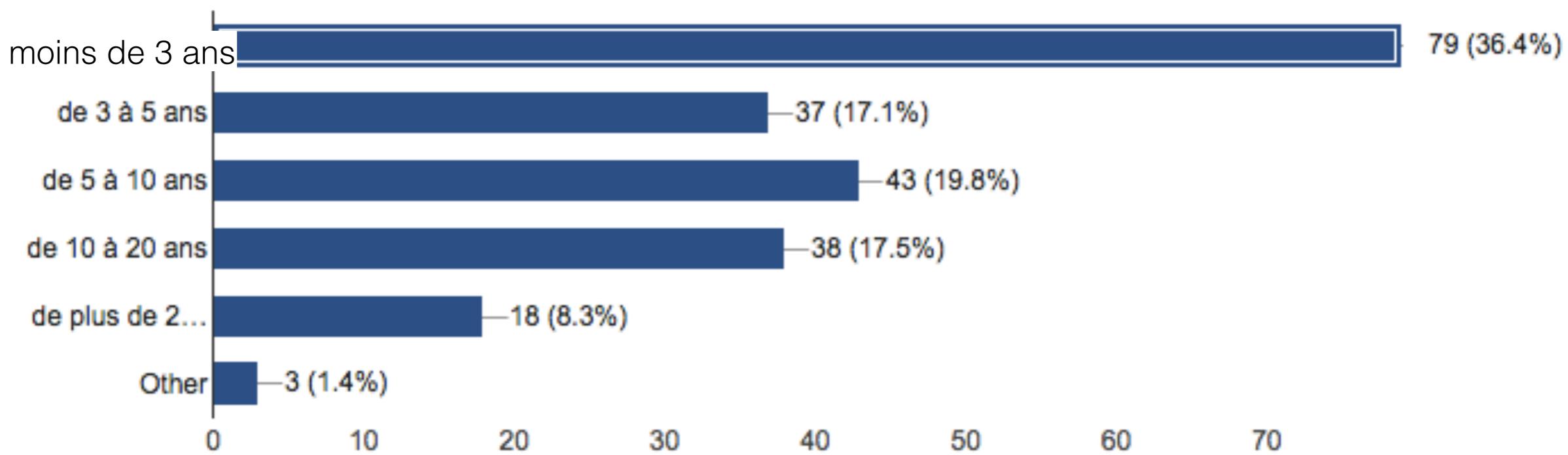
juillet à octobre 2016

Vous êtes (217 responses)

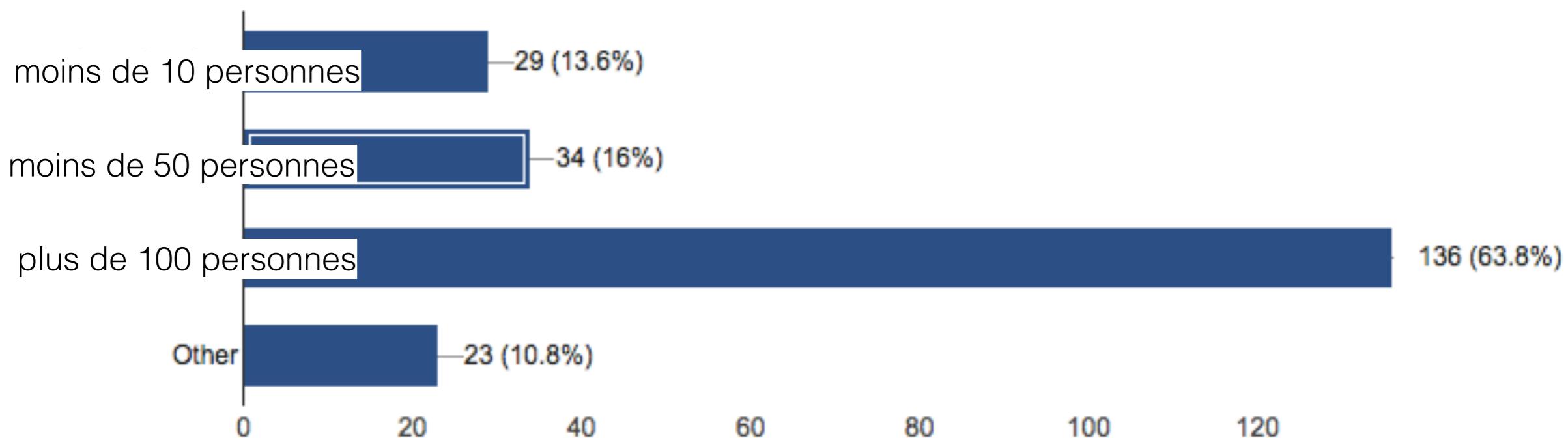
217 réponses



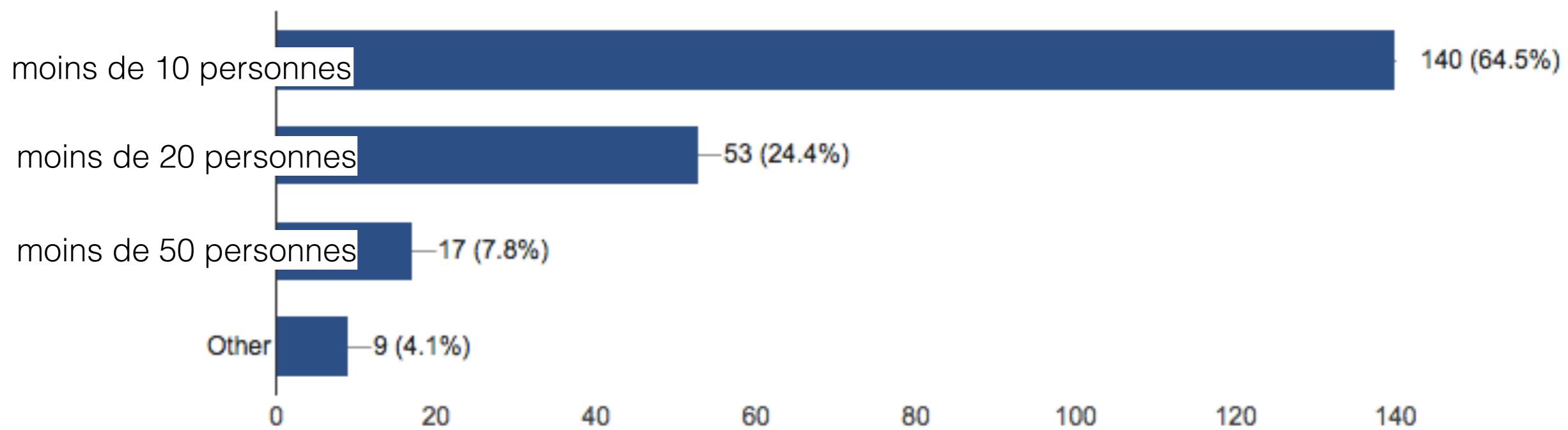
Vous avez une expérience professionnelle de (217 responses)



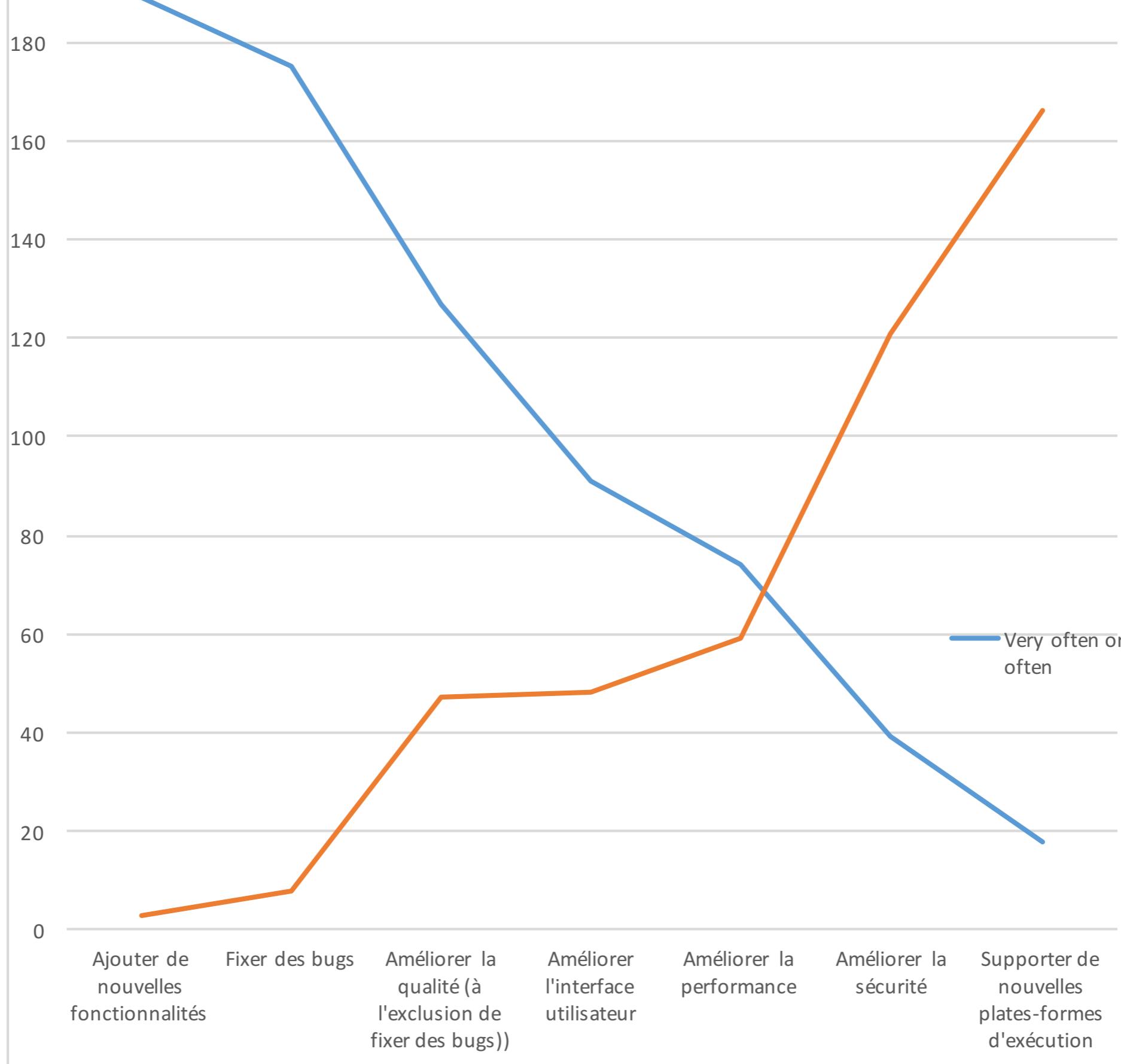
Vous travaillez pour une entreprise (213 responses)



Vous travaillez dans une équipe (217 responses)

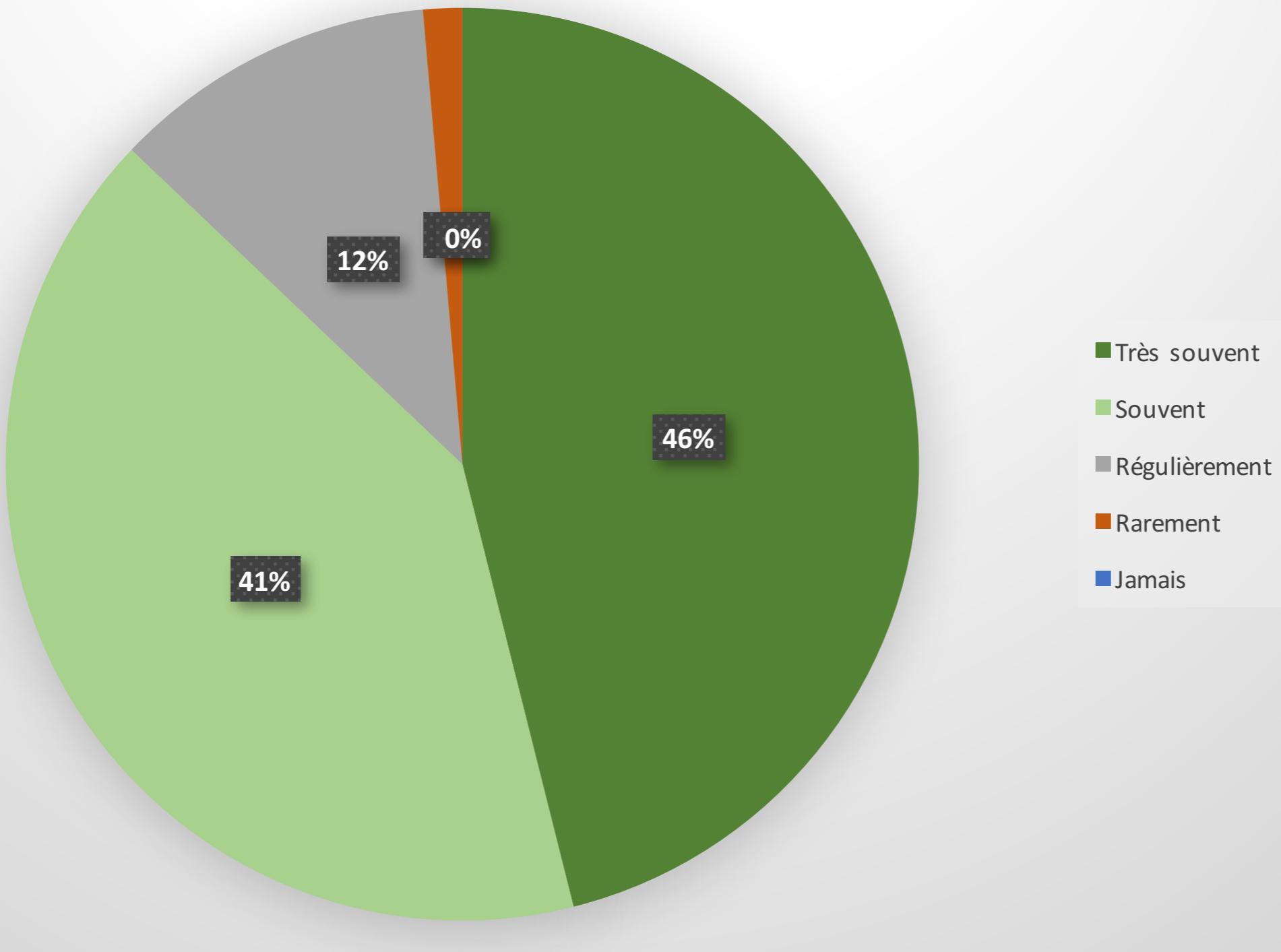


Quelle action de maintenance ?



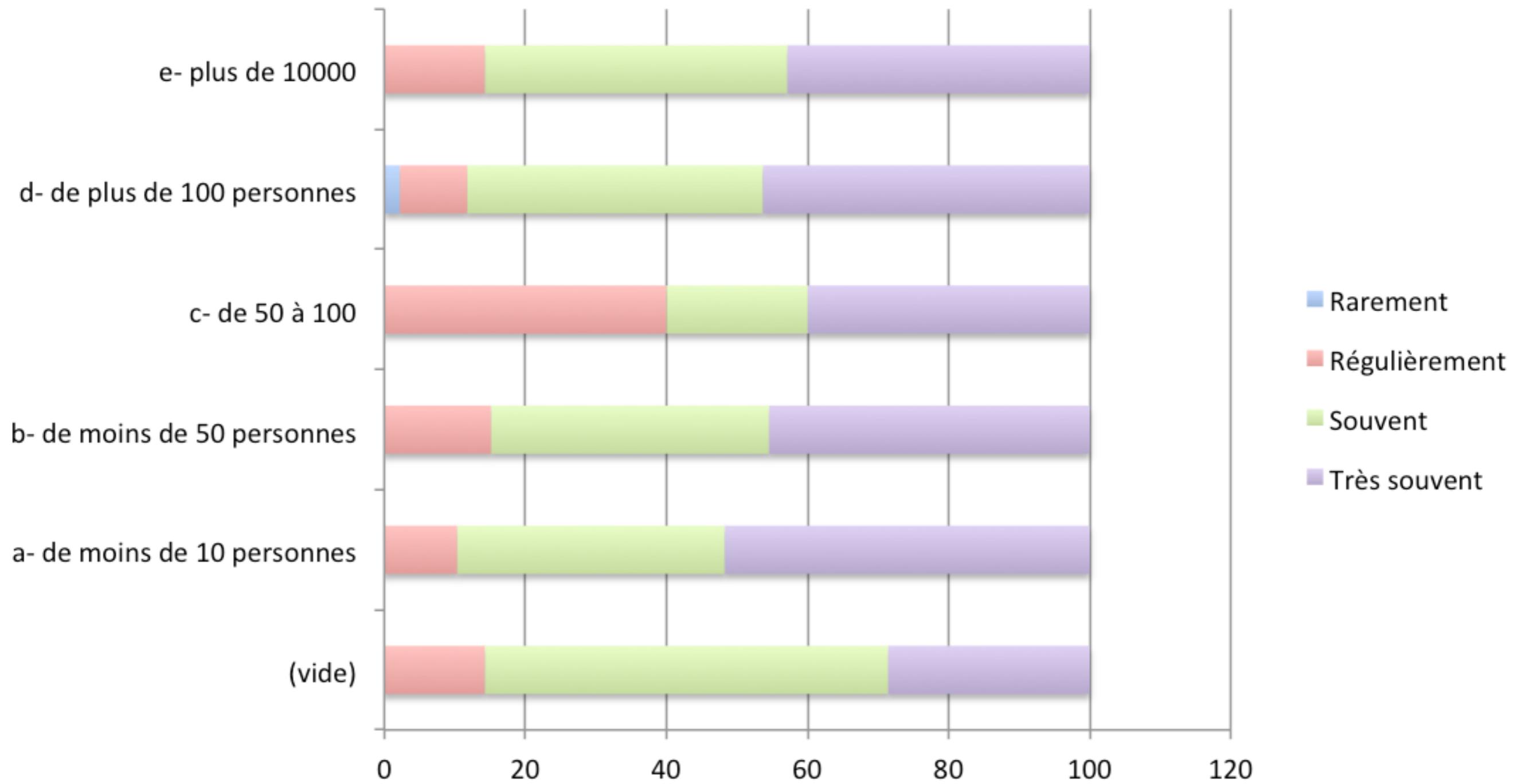
Vous modifiez votre code pour :

Ajouter de nouvelles fonctionnalités

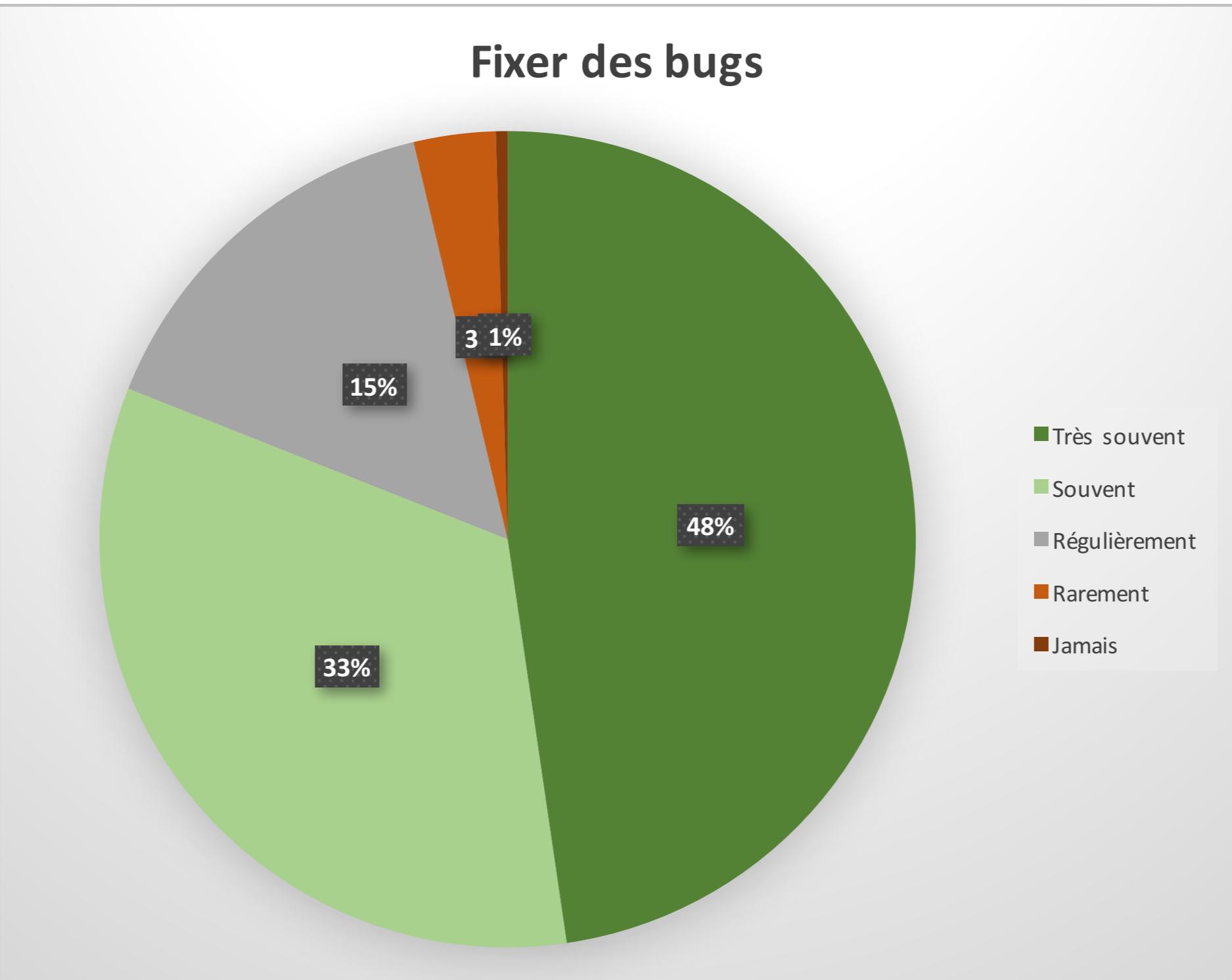


« we will never be able TO PREDICT THE FUTURE »

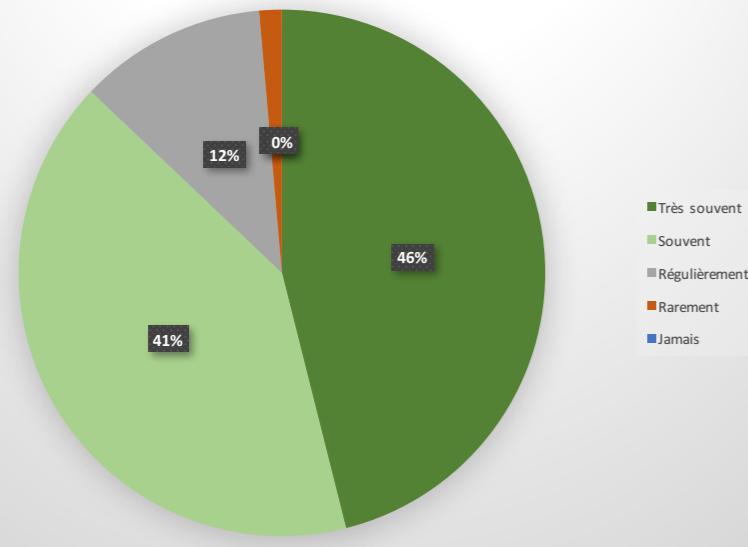
Adding functionality according to the company size



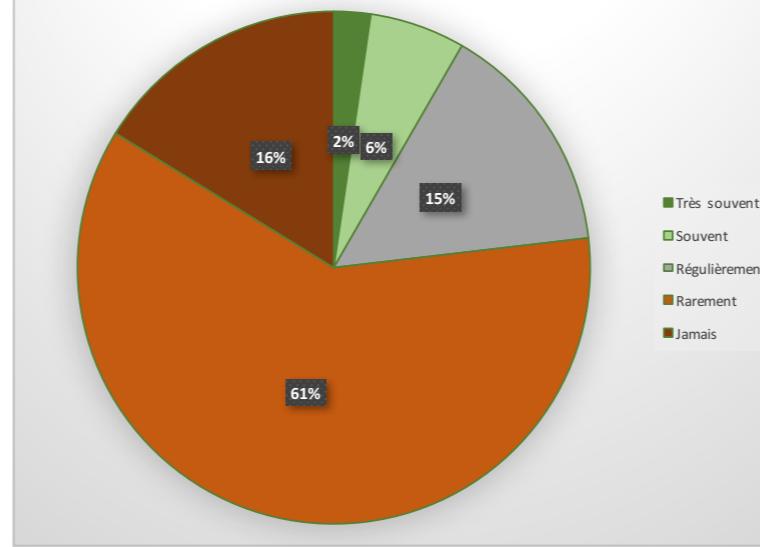
Fixer des bugs



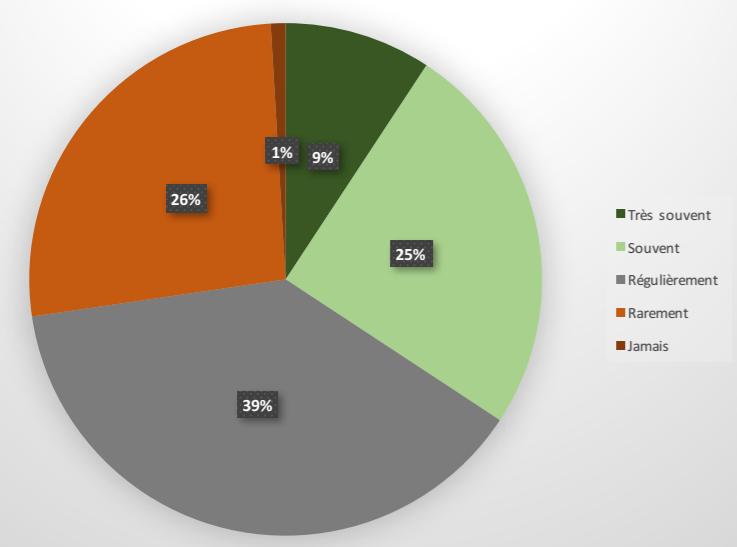
Ajouter de nouvelles fonctionnalités



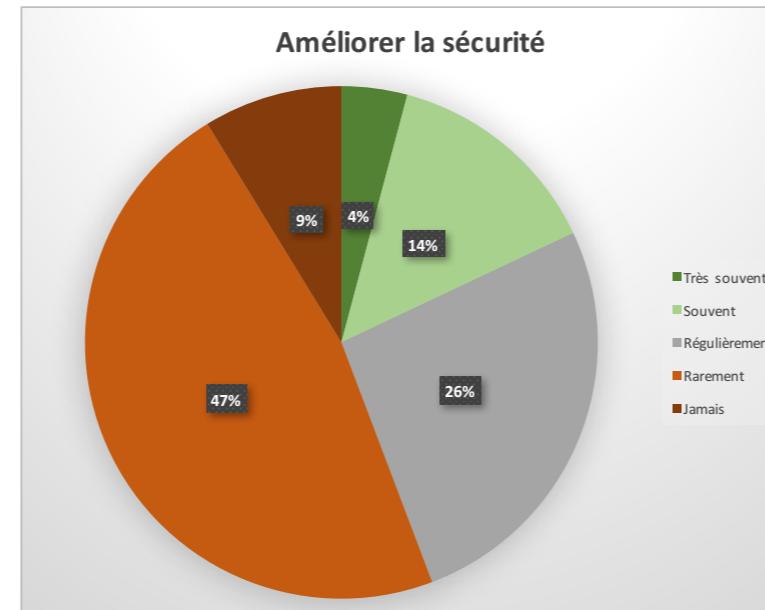
Supporter de nouvelles plateformes



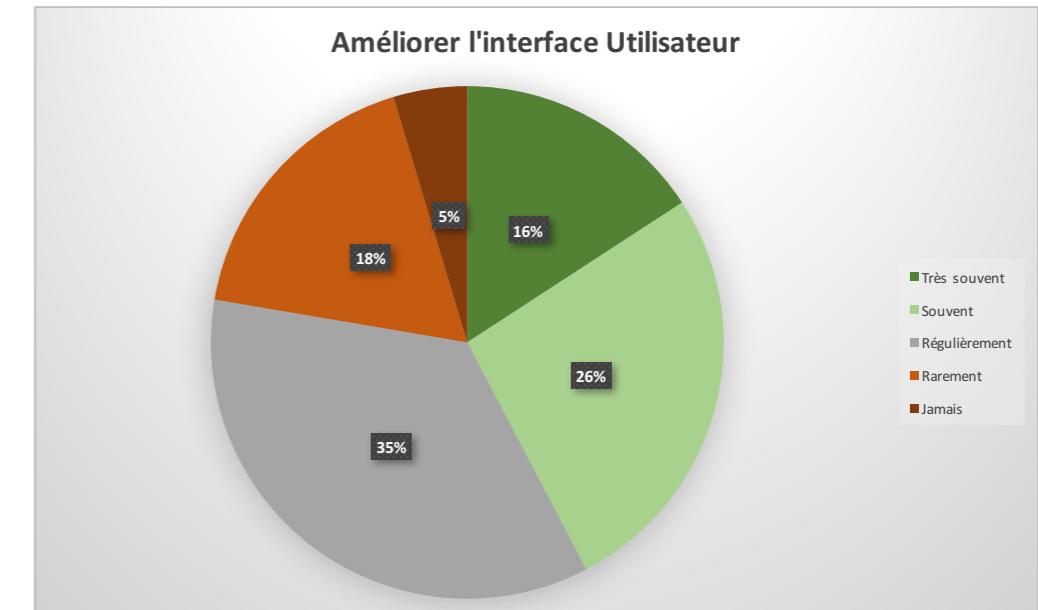
Améliorer les performances



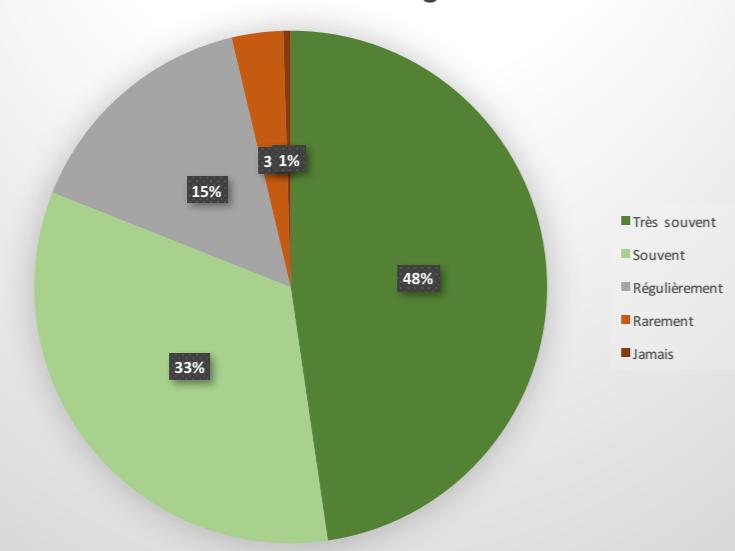
Améliorer la sécurité



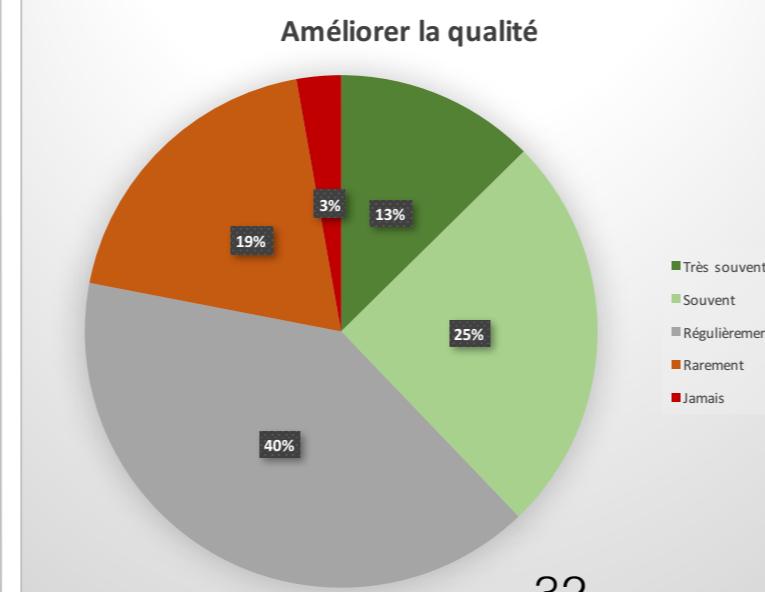
Améliorer l'interface Utilisateur



Fixer des bugs

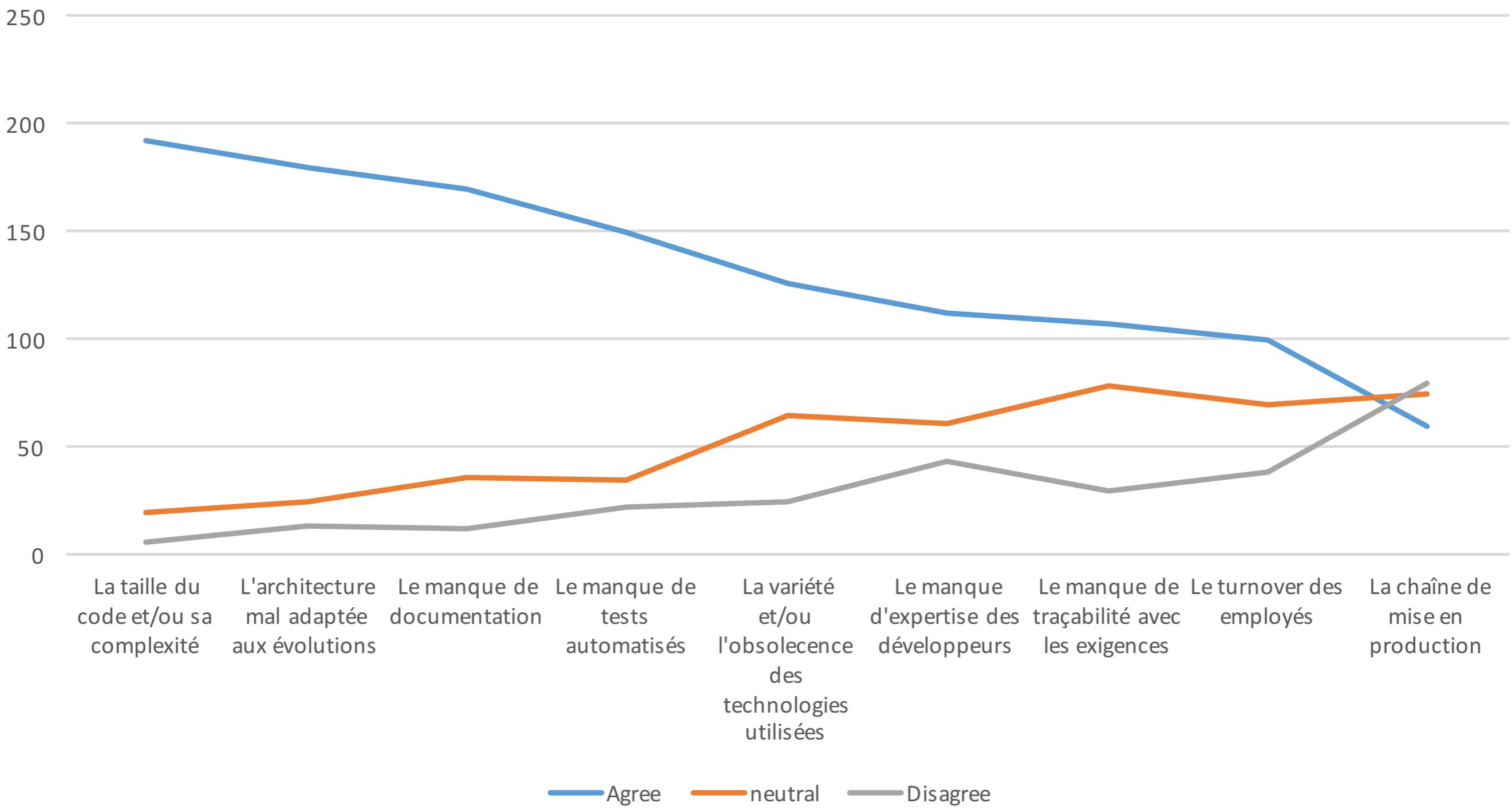


Améliorer la qualité

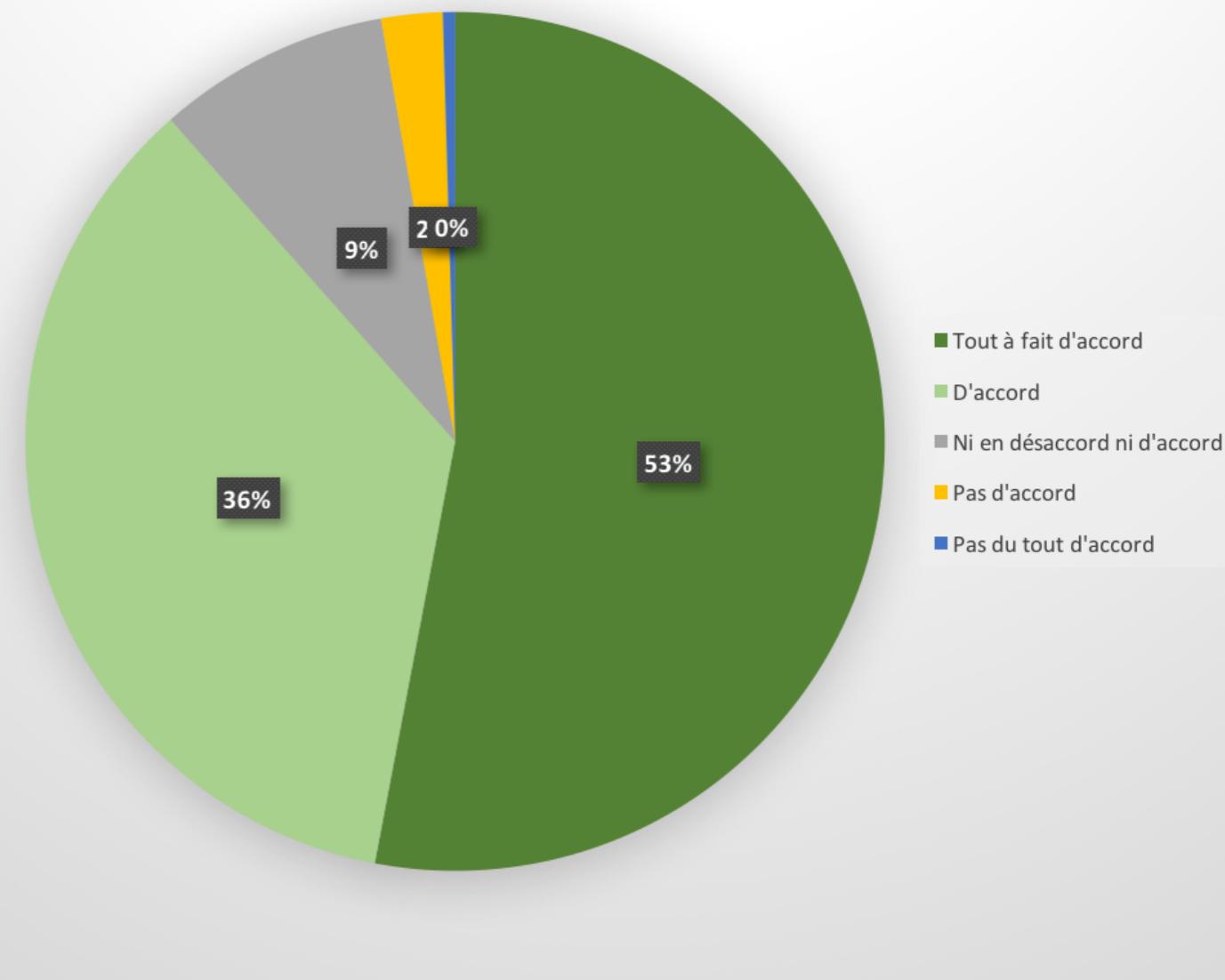


Pourquoi est-ce difficile ?

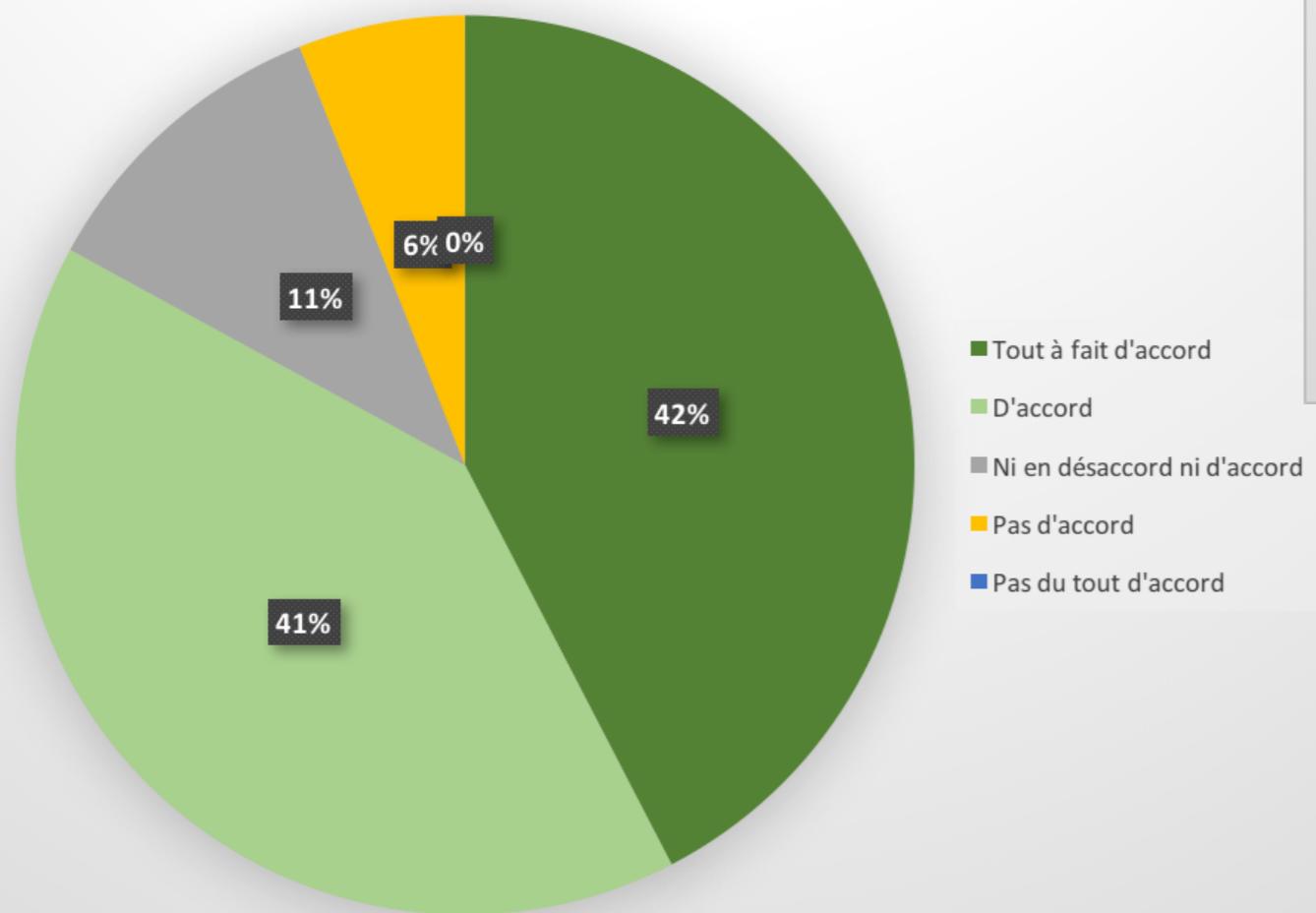
Why code maintenance is difficult?



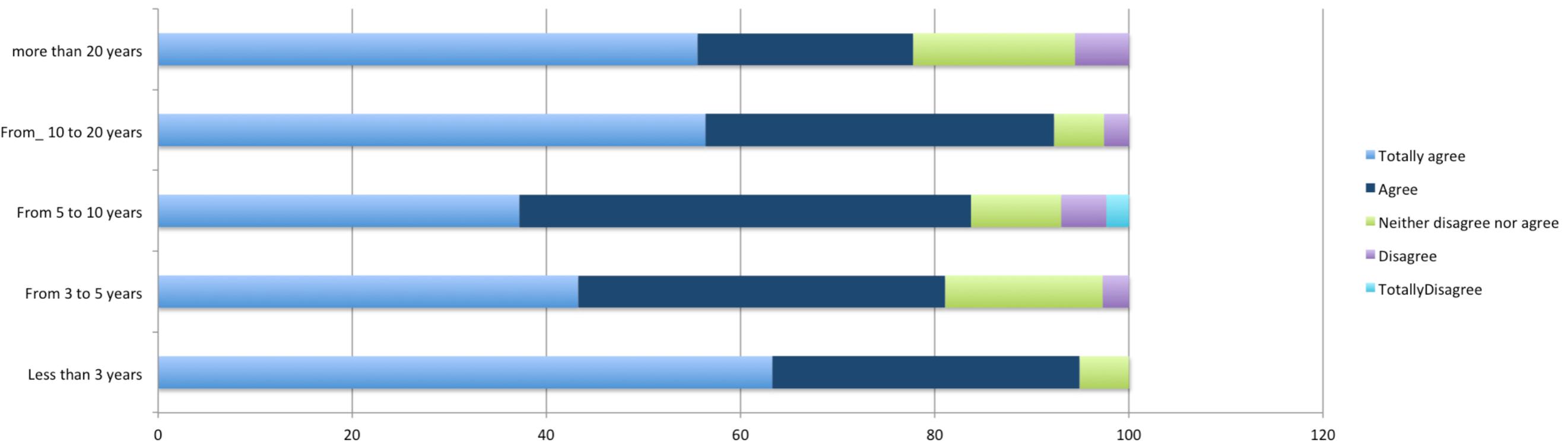
La taille du code et/ou sa complexité



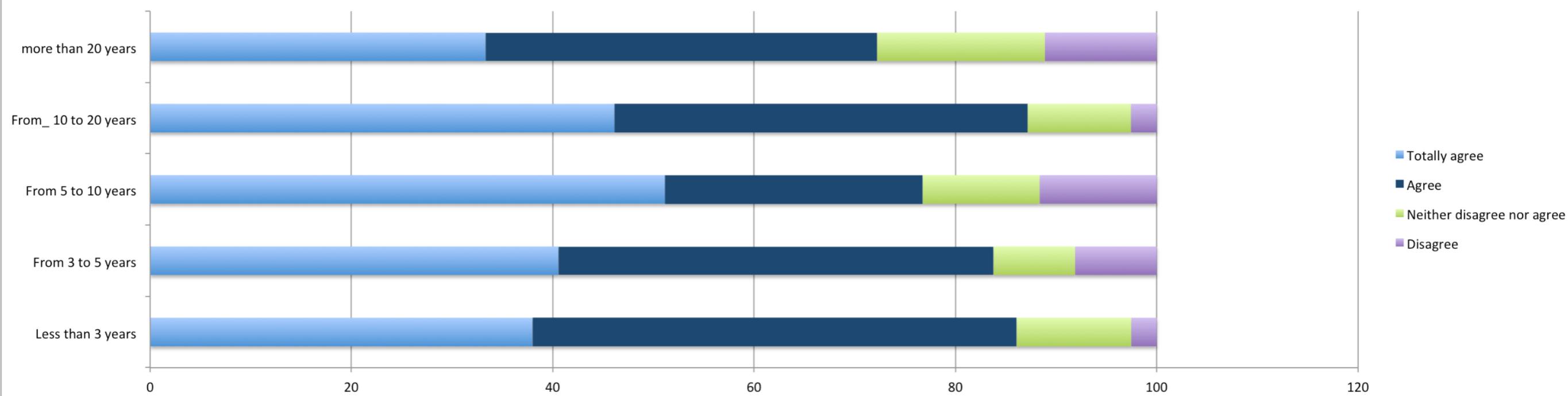
L'architecture mal adaptée...



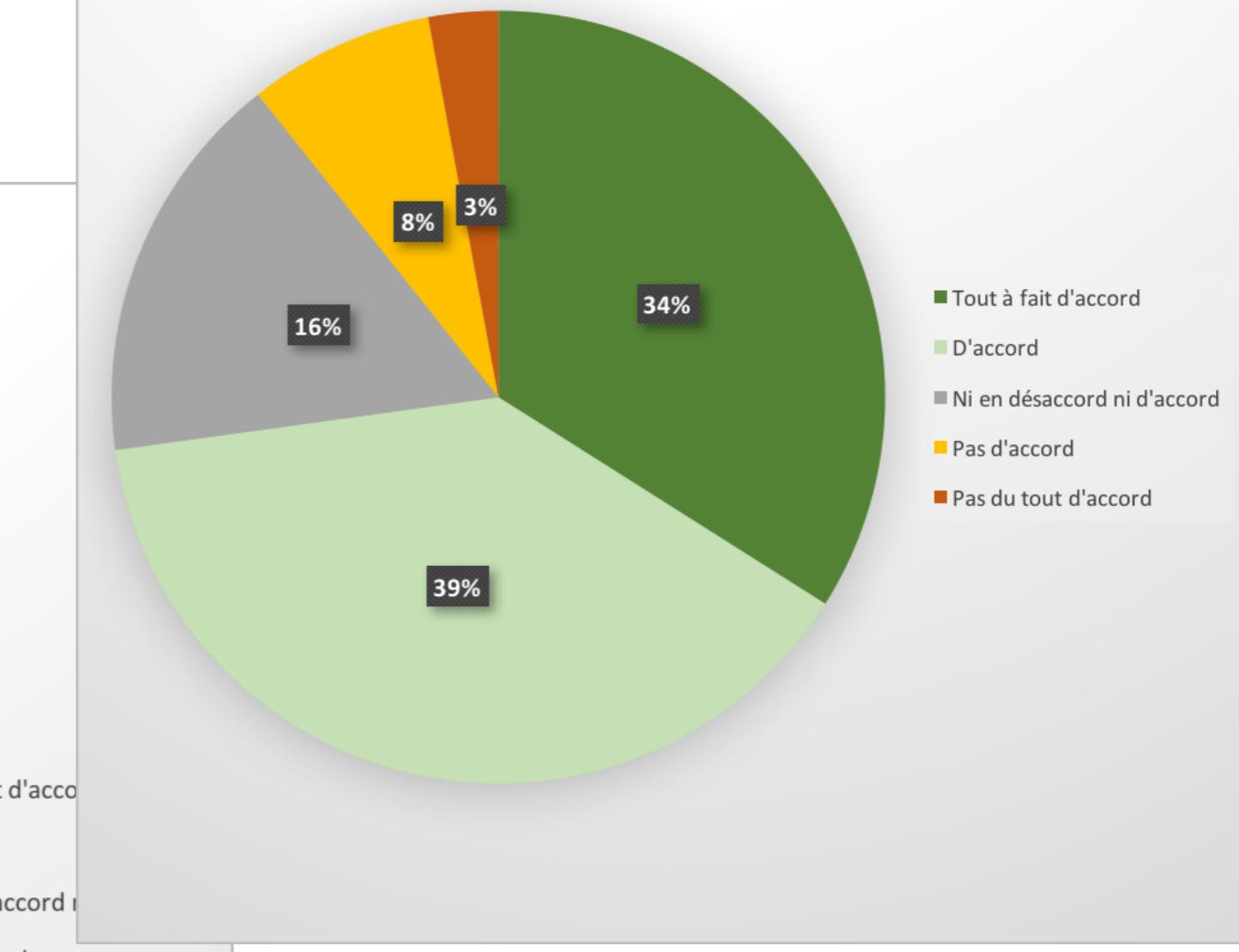
Size code evaluation according to developper experience



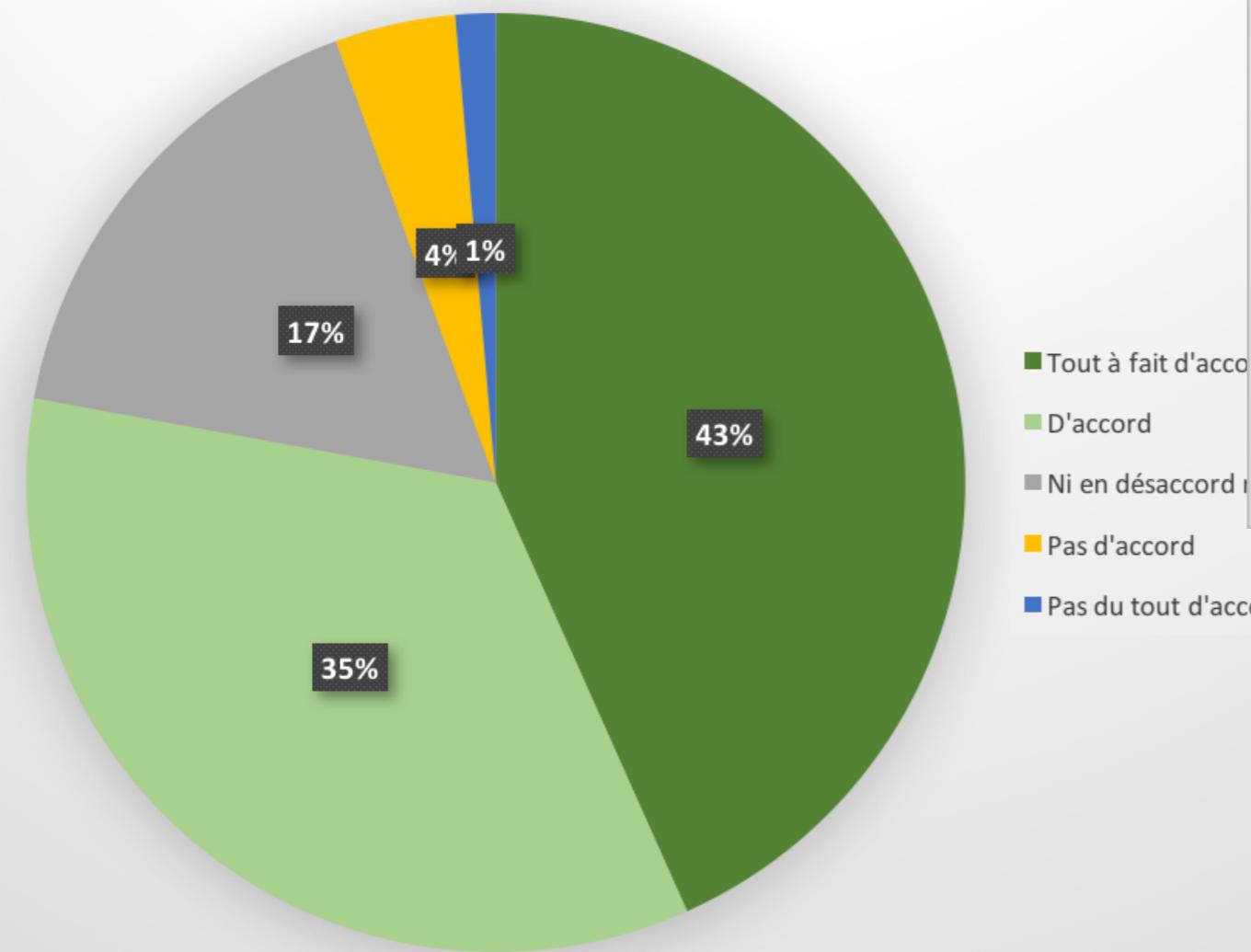
Architecture evaluation according to developper experience



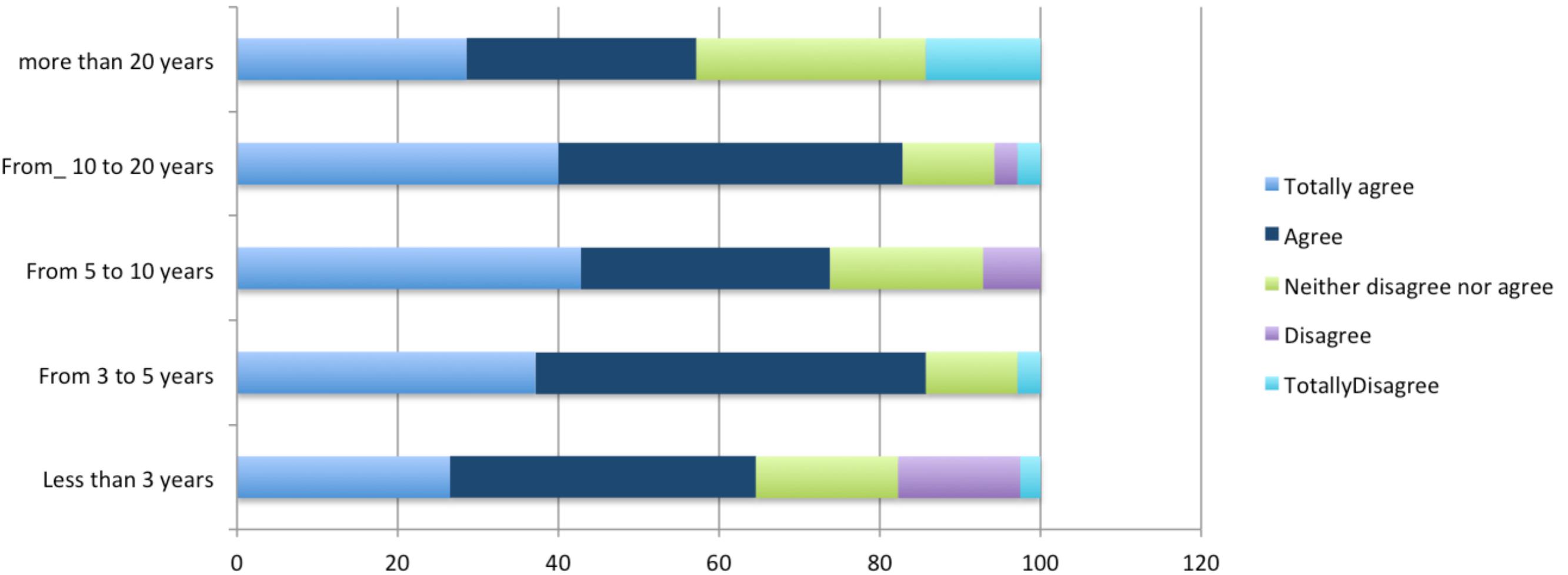
Le manque de tests automatisés



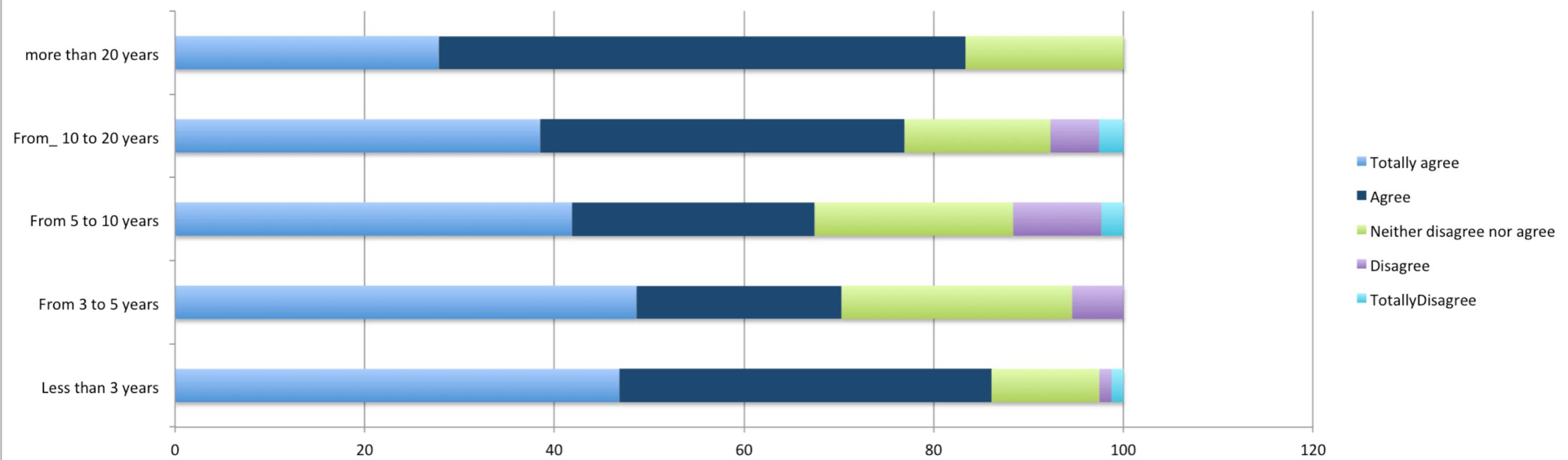
Le manque de documentation



Lack of tests versus developper experience

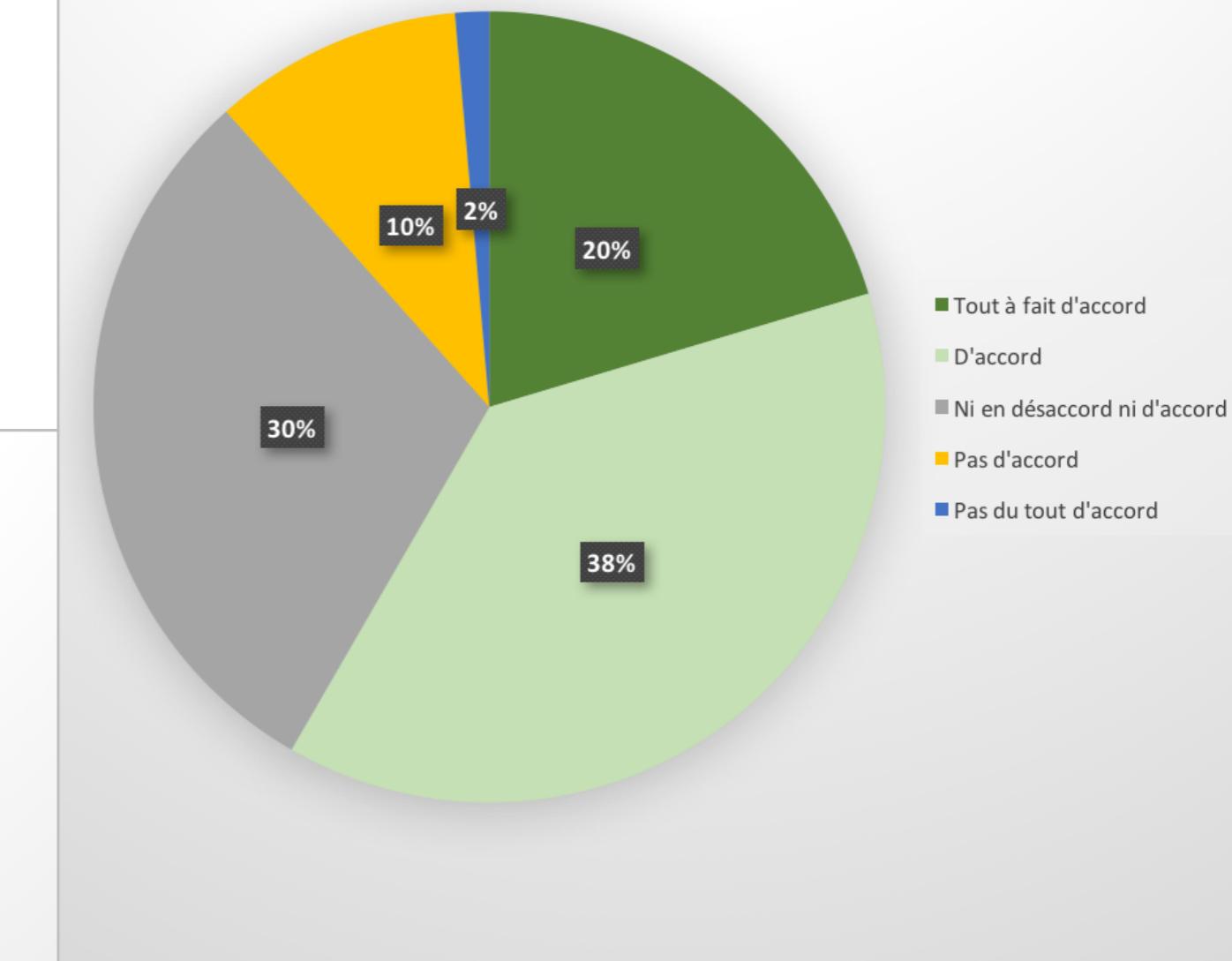
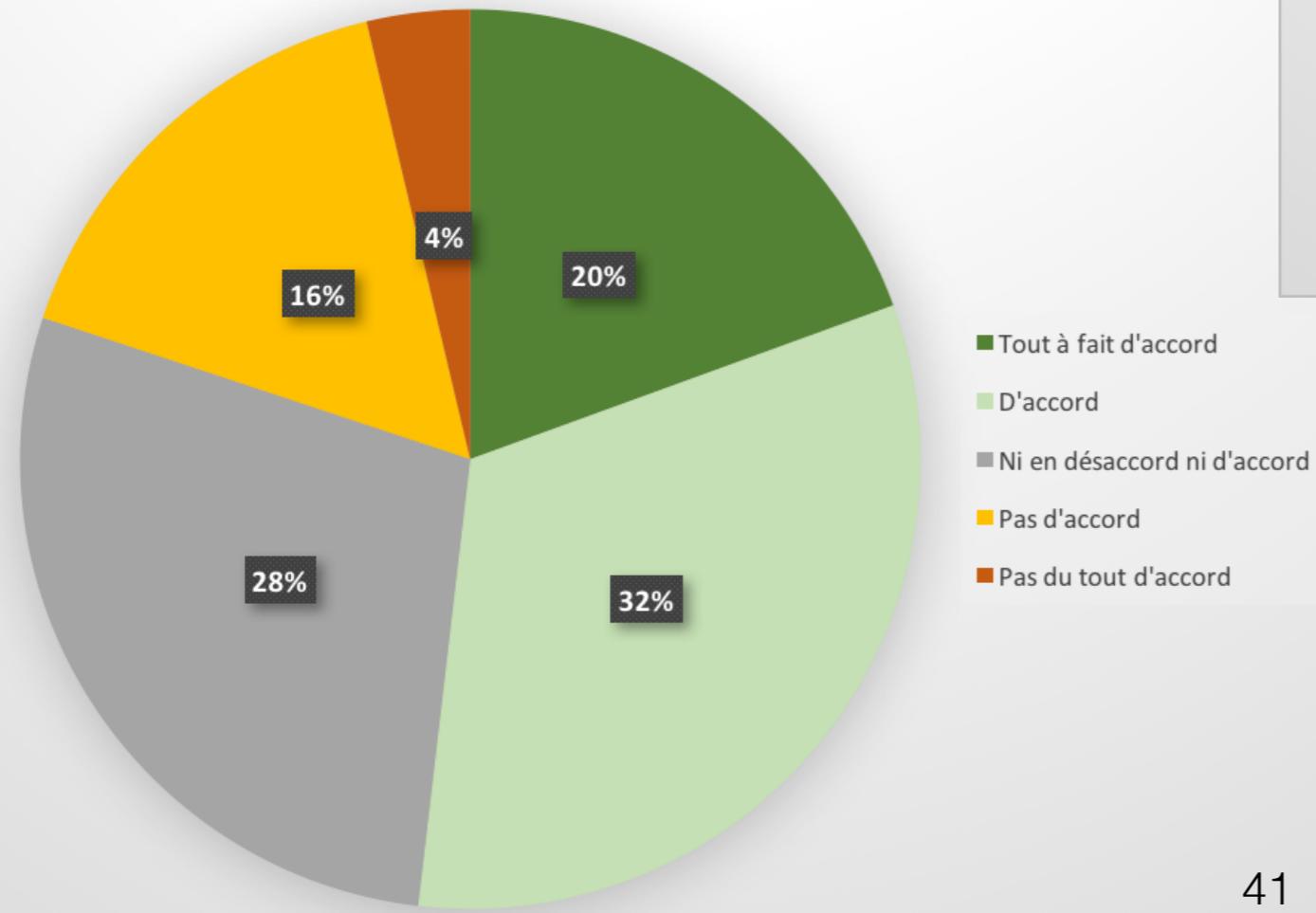


Documentation evaluation according to developper experience

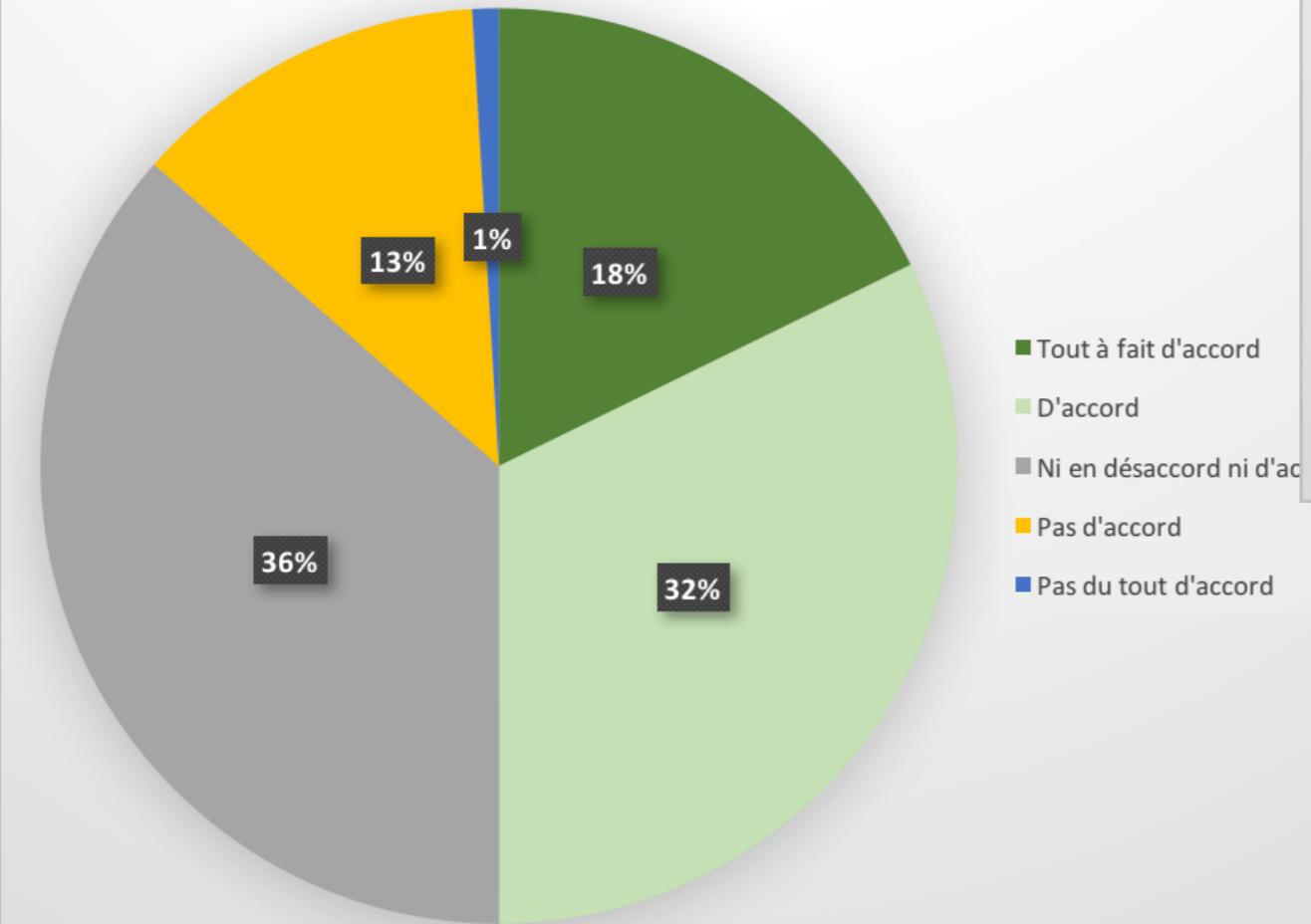


La variété et ou l'obsciescence ...

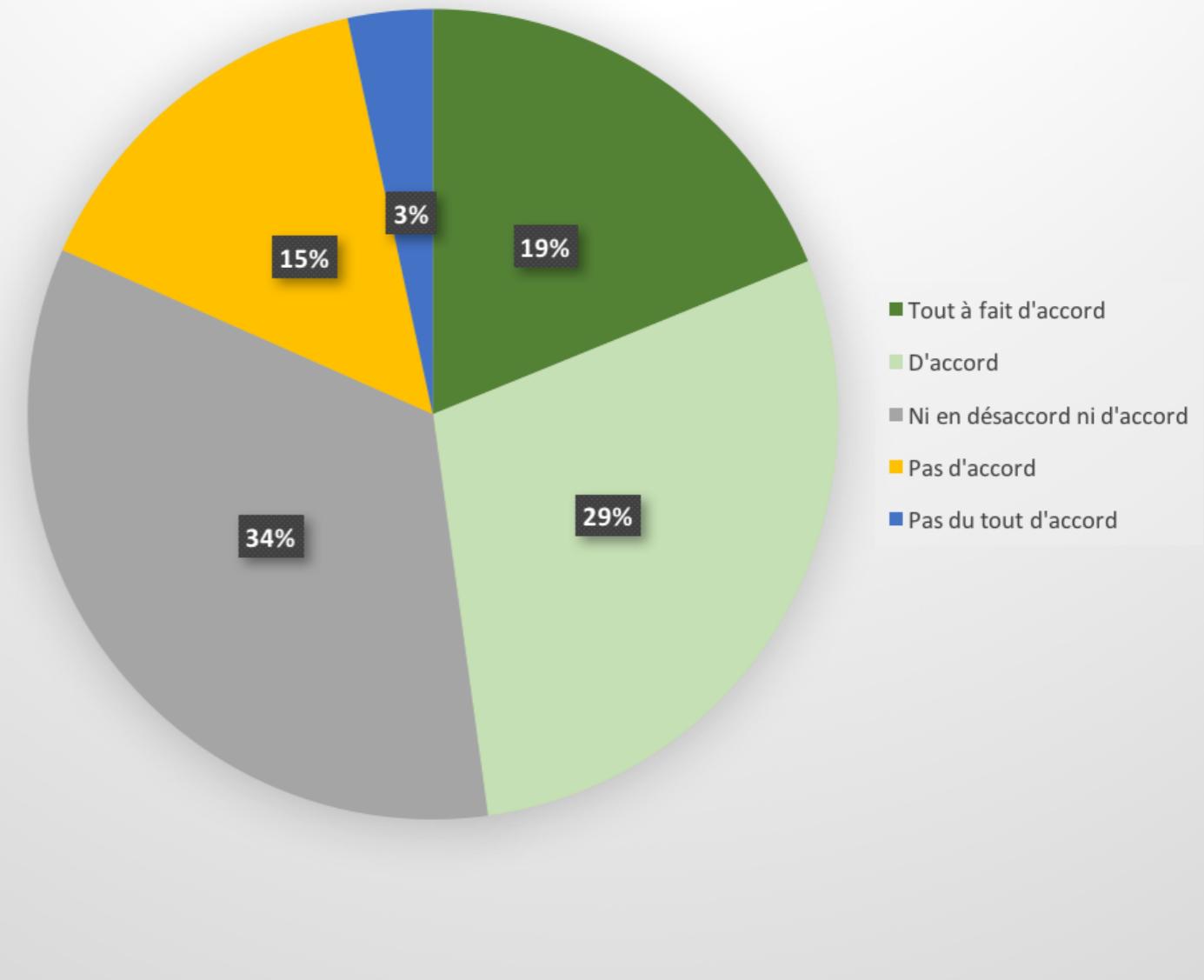
Le manque d'expertise des développeurs



Le manque de tracabilité des exigences



Le turnover des développeurs



Common Symptoms to reverse

Lack of Knowledge

- > *obsolete* or no documentation
- > *departure* of the original developers or users
- > *disappearance of inside knowledge* about the system
- > *limited understanding* of entire system
 - ⇒ *missing tests*

Process symptoms

- > *too long* to turn things over to production
- > need for *constant bug fixes*
- > *maintenance dependencies*
- > *difficulties separating* products
 - ⇒ *simple changes take too long*

Code symptoms

- *duplicated code*
 - *code smells*
- ⇒ *big build times*

Common Problems

Architectural Problems

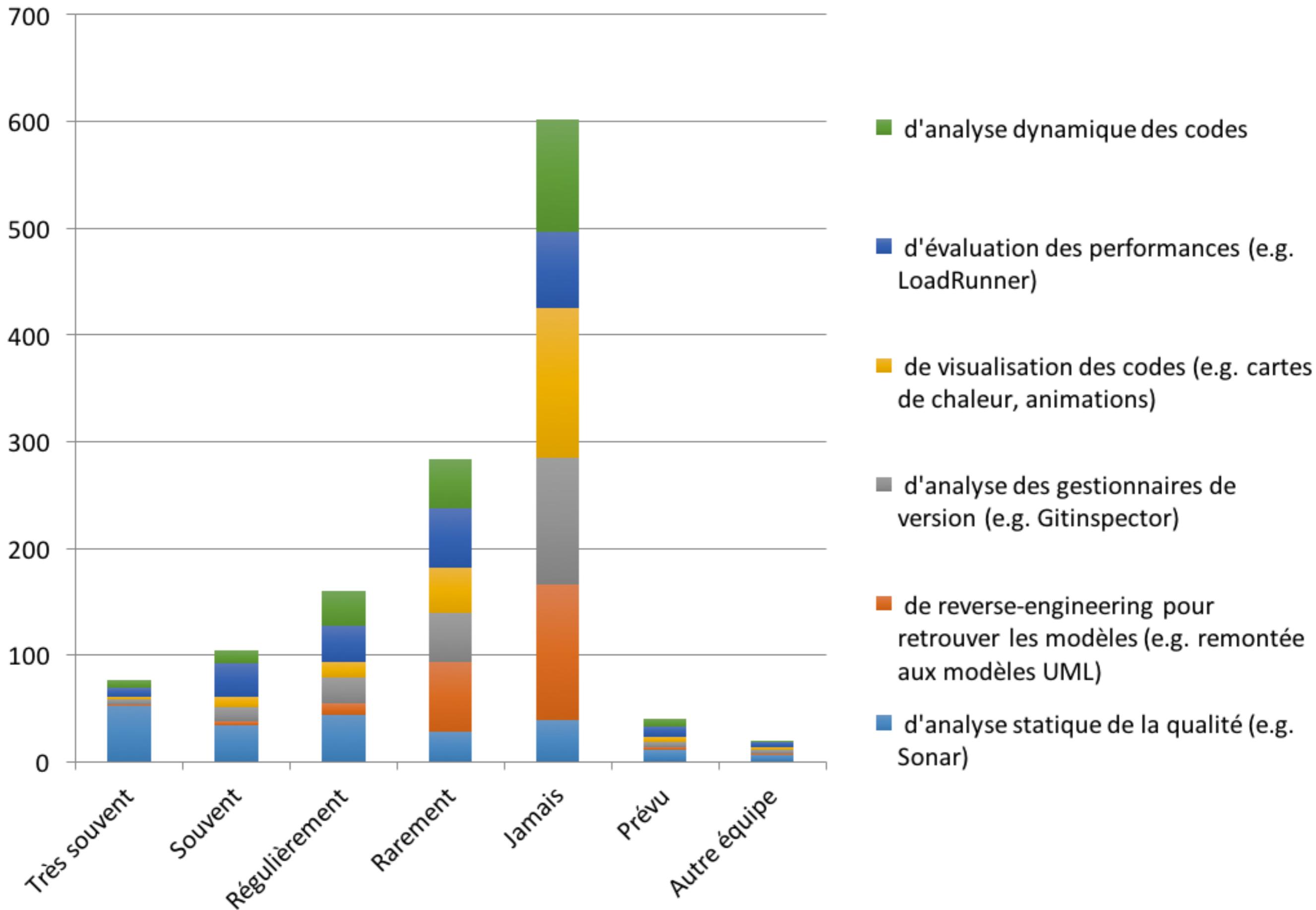
- > insufficient *documentation*
= non-existent or out-of-date
- > improper *layering*
= too few or too many layers
- > lack of *modularity*
= strong coupling
- > *duplicated code*
= copy, paste & edit code
- > duplicated *functionality*
= similar functionality
by separate teams

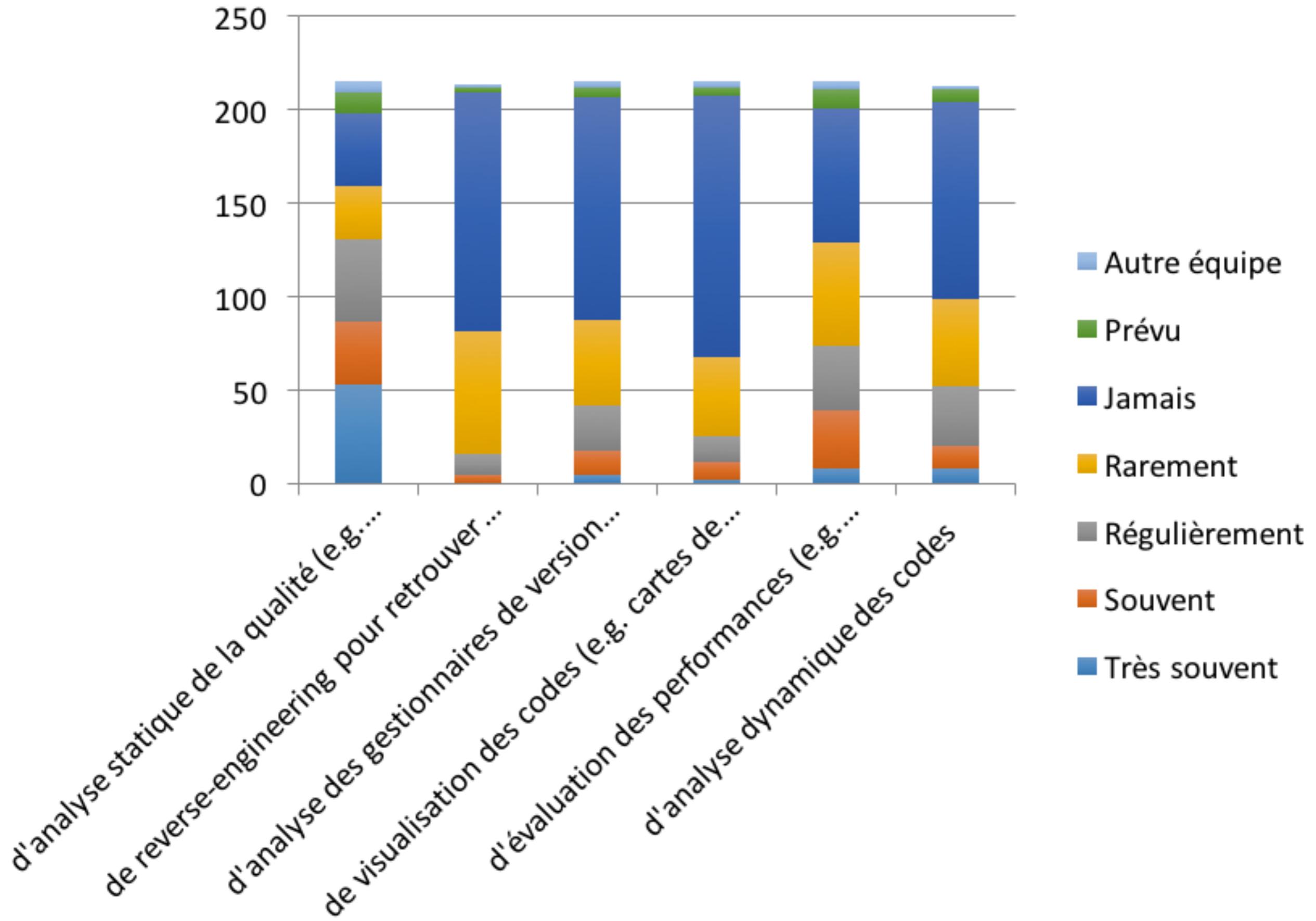
Refactoring opportunities

- > *misuse* of inheritance
= code reuse vs polymorphism
- > *missing* inheritance
= duplication, case-statements
- > *misplaced* operations
= operations outside classes
- > *violation* of encapsulation
= type-casting; C++ "friends"
- > *class abuse*
= classes as namespaces

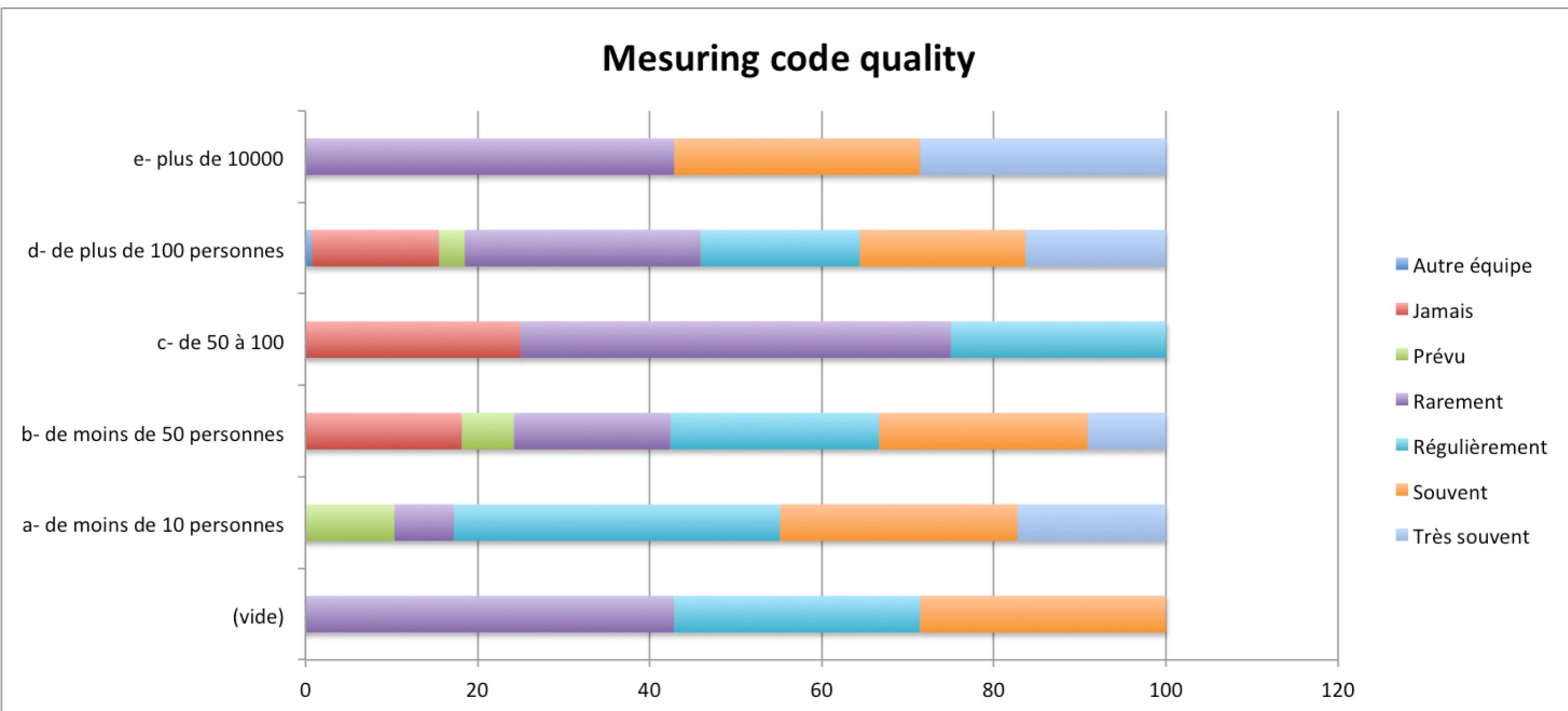
Les outils que vous utilisez,
servent à :







Y'a-t-il corrélation entre la taille de l'entreprise et les mesures de qualité de code ?



Et alors ...

- Quasi **absence d'utilisation d'outils** sauf la qualité par « Metrics »
- 2 principales difficultés : (i) la **taille** et la complexité des codes et (ii) **l'architecture** mal adaptée
- Importance de la maintenance dédiée à l'ajout de fonctionnalités et la correction de bugs.

The Challenges of Technical Debt at Scale

- Today's software systems consists of hundreds of thousands, often **million lines of code**. The scale of such systems make them virtually impossible to reason about; There are few people in the world who can keep a million lines of code in their head. Modern codebases are also built on several **different technologies**. As if this technical variety isn't complex enough, large-scale systems are also developed by **multiple programmers** organized into **different teams**. Each individual programmer is only exposed to a small part of the codebase, and frequently **no one has a holistic picture**.
- Our main challenges, if we want to understand and improve a codebase, is to balance these technical and organizational challenges. Unfortunately, as evident by sources like the CHAOS report - the majority of all projects fail to deliver on time or on budget - this is where organizations fail. I think there's a simple explanation for this repeated failure of our industry: The reason it's so hard to prioritize improvements is because most of the time we make our decisions based on what we see: the system as it looks today, its source code. But I will claim that the source code in isolation is incomplete information.



Reverse Engineering

- Conduire une réflexion sur la notion de « Métriques »
- Introduire d'autres formes d'analyse
- Travailler sur des projets réels de « grande » taille
- Mettre le focus sur le **rétro-ingénierie**

Tacking a step back



Le module

Planning

1. mar.18 déc. 2018

- 08:00 – 9:00 : Cours - Introduction (MBF)
- 9:00 – 10h00 : Autonomie
- 10:15 – 11:15 : TD - Choix et caractérisation du sujet d'étude (MBF,BB,PC)

lundi 7 janv. 2019 à 15h au plus tard **Livrable L.1**

2. mar.8 janv. 2019

- 8:00 – 9:00 : Cours - Comprendre un logiciel en regardant son histoire *Xavier Blanc(XB)*
- 9:00 – 11:00 : TD - Compléments sur le sujet en mode “coaching”(MBF,BB,PC,XB)
- 11:00 – 12:15 : Autonomie

3. mar.15 janv. 2019

- 08:00 – 09:30 : Cours : Adventure in a Docker world (BB)
- 09:45 – 12:15 : TD - Validations Métrics/KPI (MBF, BB)

4. mar.22 janv. 2019

- 08:00 – 11:00 : Oral (10mn exposé + 10mn questions) (MBF,BB) **Exposé E.1**

5. mar.29 janv. 2019

- 08:00 – 11:00 : TD - Travail sur la démarche & Métrics/KPI à base d'articles (BB) (Hypothétique)
- 11:15– 12:15 : Autonomie

6. mar.5 févr. 2019

- 09:45 – 12:15 : Autonomie

7. mar.12 févr. 2019

- 08:00 – 9:00 : Intervention d'un industriel
- 9:00 – 10:00 : TD - Compléments sur le sujet en mode “coaching”(MBF,BB,PC)

8. mar.19 févr. 2019 .

- Annonce de la sélection des articles utilisés pour l'examen
- 08:00 – 11:00 : Autonomie **Livrable L.3 & Livrable 4**

9. mar.26 févr. 2019

- 09:00 – 11:00 : Examen

On revient sur l'évaluation
du module à la fin du cours

Your Code As a Crime Scene

Use Forensic Techniques
to Arrest Defects, Bottlenecks, and
Bad Design in Your Programs



Adam Tornhill

edited by Fahmida Y. Rashid

Foreword by Michael Feathers,
author of *Working Effectively
with Legacy Code*

« The code alone doesn't tell the whole story of a software product. »

*Your code is a crime Scene,
Adam Tornhill*

Comment rapidement identifier les problèmes dans le code ?

« Profiler » le code

Pour ne pas lire des milliers de lignes, comprendre rapidement le code, identifier les parties à modifier, à protéger (par des tests) ou à contrôler régulièrement.



« Profiler » le code

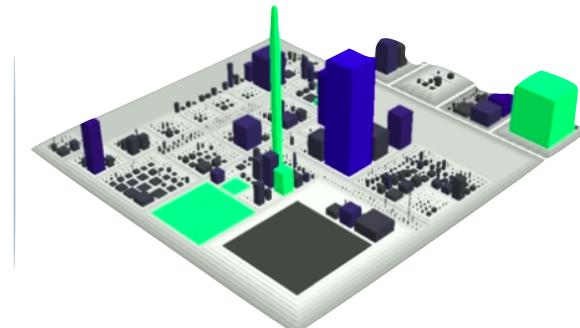
- Utiliser des metrics simples
- Visualiser des metrics pour avoir une vision globale
- Mettre le focus sur les anomalie en descendant dans les codes

I took a course in speed reading and read “War and Peace” in twenty minutes. It’s about Russia.

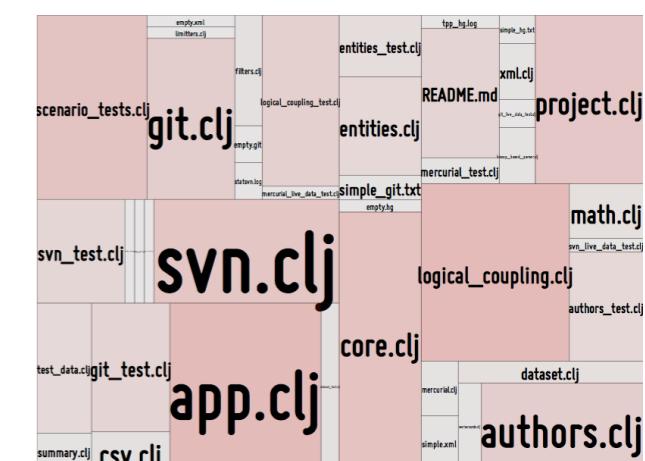
—

« Profiler » le code

- Analyse « géographique » : Visualiser le code (codeCity)

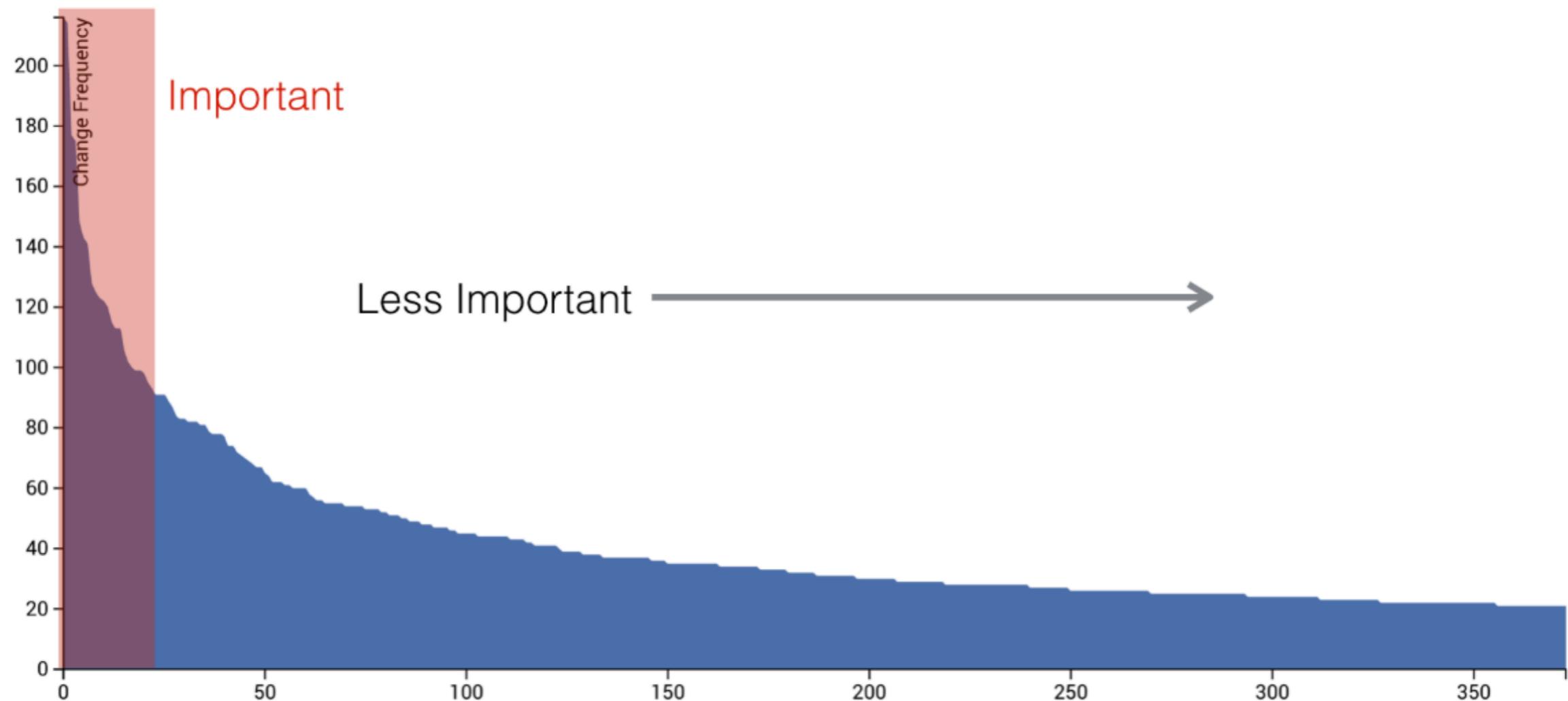


- Analyse des changements : qui, quoi, quand (MetricsTreeMap) :
 - Forte complexité et beaucoup de changements => un hotspot !
 - Beaucoup de changements sont signes de problèmes (Hassan 09)



Focus sur les codes « très modifiés »

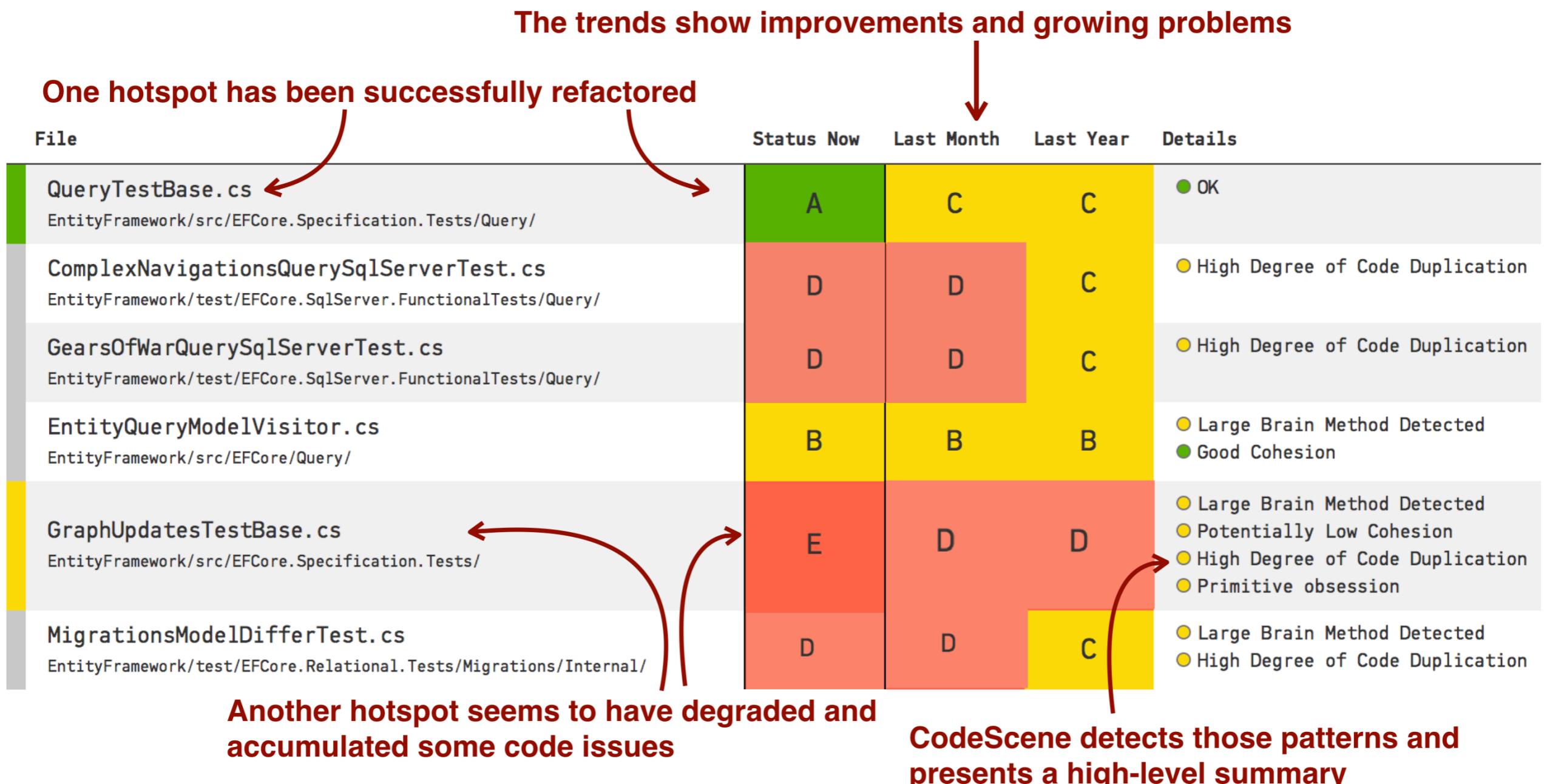
Nombre de changements



Fichiers ordonnés par la fréquence de changement

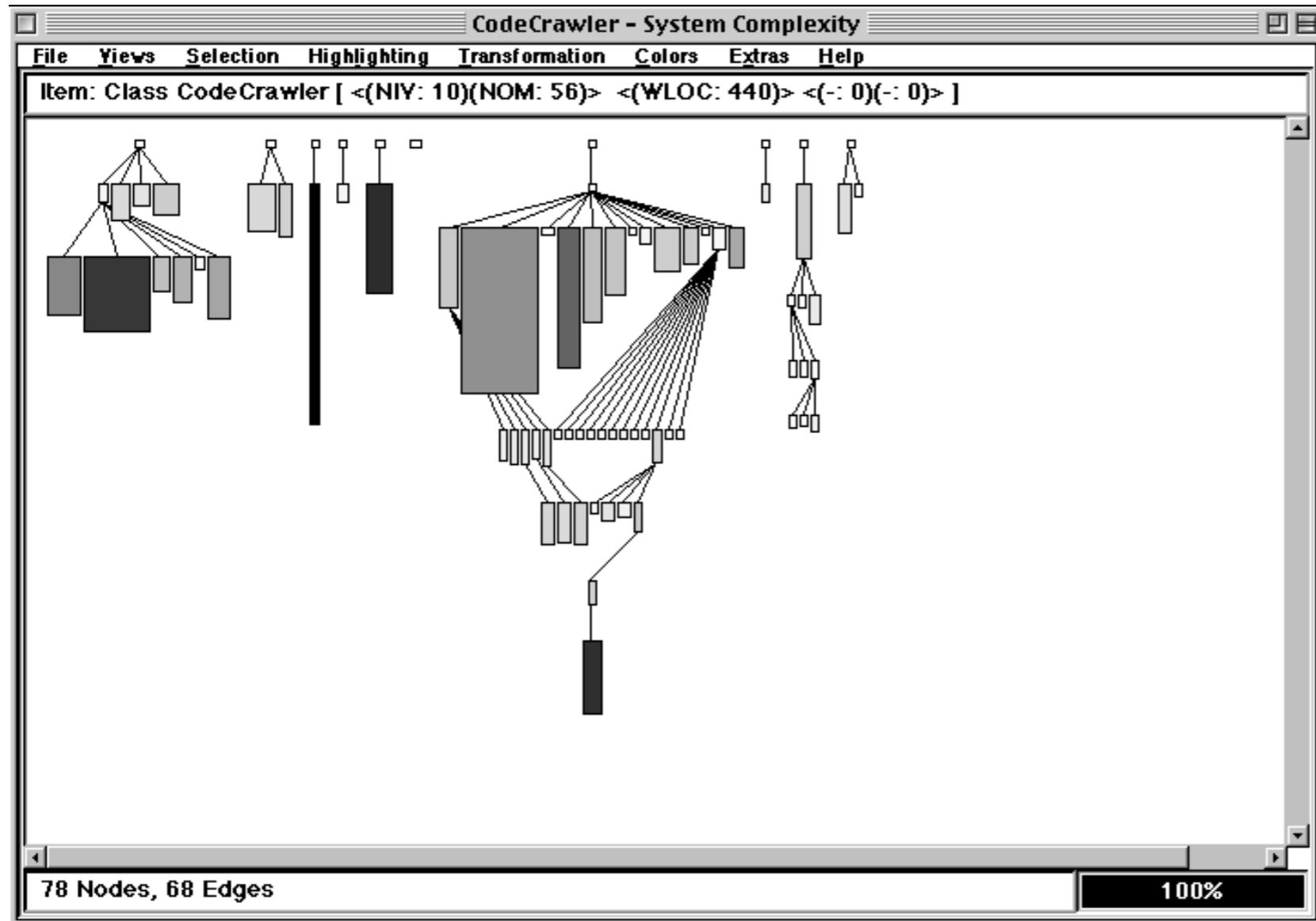
The distributions above show that most of our development activity is located in a relatively small part of the total codebase.

Use Code Biomarkers To Identify Expensive Hotspots

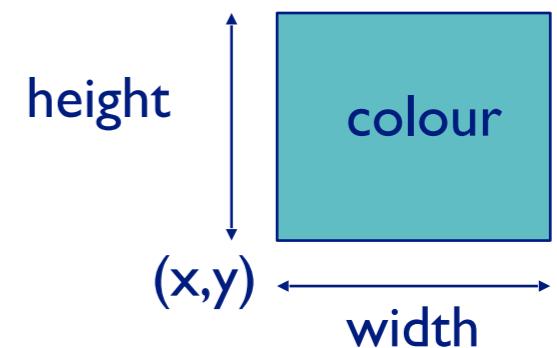


<https://empear.com/blog/prioritize-technical-debt/>

Visualizing Metrics

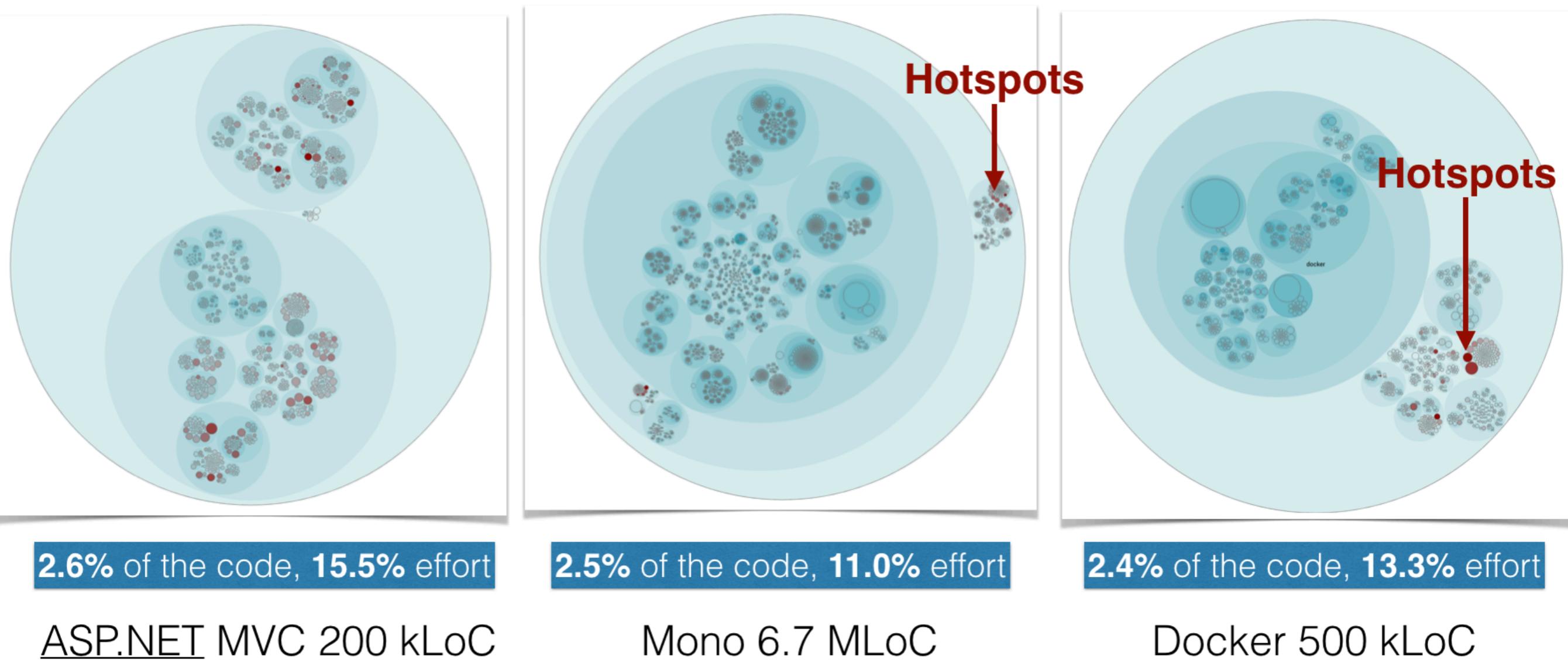


Use *simple* metrics and layout algorithms



Visualizes up to 5 metrics per node

Use Code Biomarkers To Identify Expensive Hotspots



Et Après

- Si les points critiques représentent 4 à 6% du code (Hibernate 400.000 LOC=> 24000 LOC)
 - Comment éliminer rapidement les faux positifs ?

Names makes the code fit your head !

- Un nom c'est plus qu'une description.
- « *Our brain's working memory is short term and serves as the mental workbench of the mind : limited cognitively!* »
- On regroupe les fonctions de calcul dans les noms => Un code moins cher à maintenir !
- Exemple de noms suspects :
 - Conjonction e.g. ConnectionAndSessionPool (cohesion? same concept?)
 - Manager, Util, Impl : Vraiment pas de logiques ?

Hotspot & Nom suspect

- Nom général et un composant très gros.
- AbstractEntityPersister.java => ~4000 LOC
- Configuration.java => ~2600loc

Code Complexity, d'un coup d'oeil?

```
final static int libre = 0, empile = 1, exclus = 2, pointdentree = 3, dansC  
  
int dfsCfc(HashMap<Sommet, Integer> etat,  
           HashMap<Sommet, Integer> etat_cfc,  
           HashMap<Arc<Sommet>, String> etat_a,  
           LinkedList<Sommet> pile,  
           HashMap<Sommet, Integer> rang, int rg,  
           HashMap<Sommet, Integer> rat,  
           Sommet racine, Sommet s){  
    etat.put(s, encours);  
    System.err.println("J'explore "+s);  
    rang.put(s, rg++);  
    rat.put(s, rang.get(s));  
    pile.addFirst(s);  
    etat_cfc.put(s, empile);  
    for(Arc<Sommet> a : voisins(s)){  
        Sommet t = a.destination();  
        if(etat_cfc.get(t) == inexploré){  
            System.err.println("(" + s + ", " + t + ") liaison");  
            etat_a.put(a, "arcliaison");  
            rg = dfsCfc(etat, etat_cfc, etat_a, pile, rang, rg, rat, racine, t);  
            if(rat.get(t) < rat.get(s)){  
                rat.put(s, rat.get(t));  
            }  
        }  
        else{  
            if(etat_cfc.get(t) == empile){  
                if(rang.get(t) > rang.get(s))  
                    // (s, t) est avant  
                    etat_a.put(a, "arcavant");  
                else{  
                    // (s, t) est arrière ou intra-arbre  
                    // t est dans C(s) car t descendant de s et s -> t  
                    if(etat.get(t) == encours){  
                        System.err.println("(" + s + ", " + t + ") arrière");  
                        etat_a.put(a, "arcarrière");  
                    }  
                    else{  
                        System.err.println("(" + s + ", " + t + ") intra-arbre");  
                        etat_a.put(a, "arcintra");  
                    }  
                }  
                System.err.println(t + " est dans C(" + s + ")");  
                if(rang.get(t) < rat.get(s))  
                    rat.put(s, rang.get(t));  
            }  
            else{  
                // t est exclus  
                System.err.println(t + " exclus");  
                if(rang.get(t) < rang.get(racine))  
                    // (s, t) est inter-arbre  
                    etat_a.put(a, "arcinter");  
                else  
                    etat_a.put(a, "arcintra");  
            }  
        }  
    }  
}
```

```
@Test  
public void testTailleInitiale() { assertEquals(0, graphe.taille()); }  
  
@Test  
public void testAjouterSommetInitial() {  
    graphe.ajouterSommet(s1);  
    assertEquals(1, graphe.taille());  
    assertEquals(s1, graphe.getSommet("S1"));  
}  
  
@Test  
public void testAjouterArcInitial() {  
    graphe.ajouterSommet(s1);  
    graphe.ajouterSommet(s2);  
    graphe.ajouterSommet(s3);  
    graphe.ajouterArc(s1, s2, distance_s1_s2);  
    assertEquals(3, graphe.taille());  
    assertTrue(graphe.existeArc(s1, s2));  
    assertFalse(graphe.existeArc(s1, s3));  
    ArrayList<Arc<Sommet>> arcs = graphe.arcs(s1, s2);  
    assertEquals(1, arcs.size());  
}  
  
@Test  
public void testVoisinsSommet() {  
    initGlobalGraph();  
    Collection<Arc<Sommet>> voisins = graphe.voisins(s1);  
    assertEquals(3, voisins.size());  
    //System.out.println("voisins de s1 : " + voisins);  
    voisins = graphe.voisins(s2);  
    assertEquals("voisins de s2 : " + voisins, 2, voisins.size());  
    //System.out.println("voisins de s2 : " + voisins);  
}  
  
@Test  
public void testArcs() {  
    initGlobalGraph();  
    ArrayList<Arc<Sommet>> arcs = graphe.arcs(s1, s2);  
    //System.out.println("de s1 a s2 : " + arcs);  
    assertEquals(2, arcs.size());  
    arcs = graphe.arcs(s1, s3);  
    //System.out.println("de s1 a s3 : " + arcs);  
    assertEquals(1, arcs.size());  
    arcs = graphe.arcs(s1, s4);  
    //System.out.println("de s1 a s4 : " + arcs);  
    assertTrue(arcs==null);  
    arcs = graphe.arcs(s4, s1);  
    //System.out.println("de s4 a s1 : " + arcs);  
    assertEquals(1, arcs.size());  
}  
  
@Test  
public void testParcoursCheminsDe() {  
    initGlobalGraph();  
    ArrayList<Chemin> chemins = parcours.chemins(s1);  
    assertEquals(3, chemins.size());  
    System.out.println("Chemins à partir de s1: " + chemins);  
    Chemin chemin = chemins.get(0);  
}
```

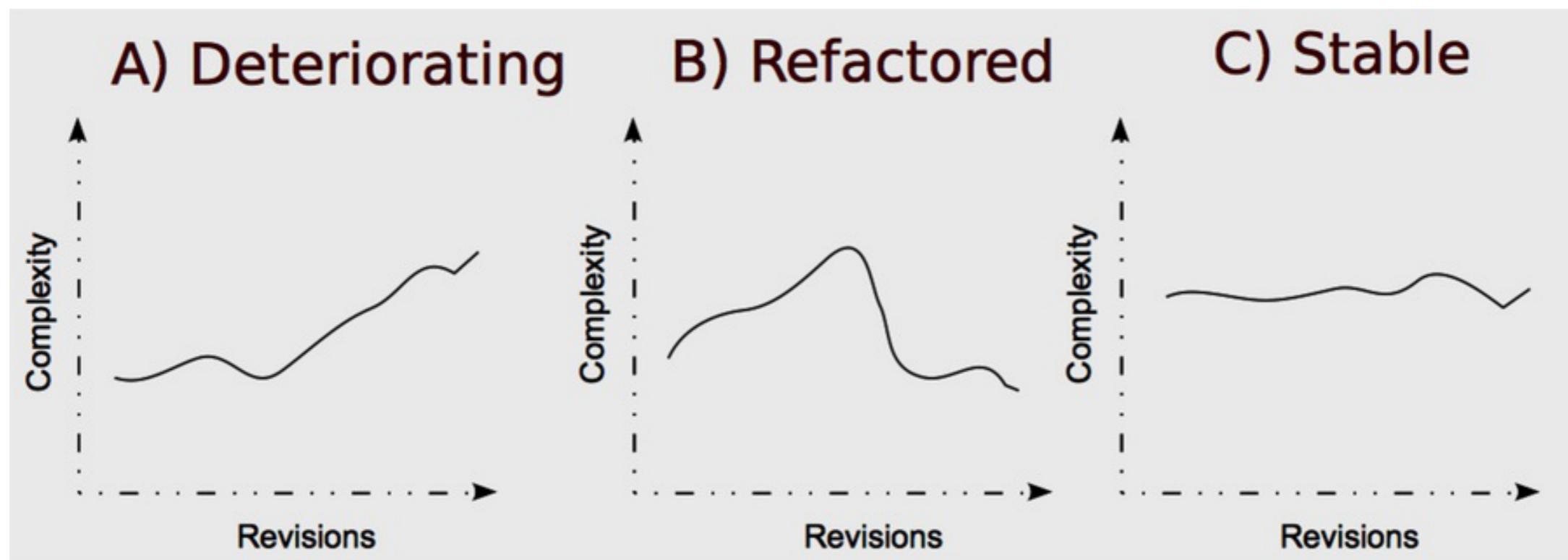
Code Complexity, d'un coup d'oeil?

Configuration.java un max de 14 indentations : il semble y avoir de la logique métier dans le fichier de configuration...

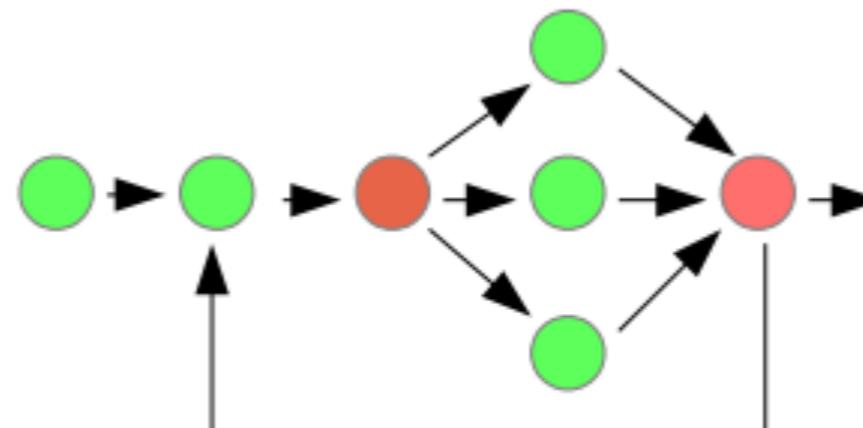
Evolution de la complexité du code

En se basant sur le nombre d'indentation (vue comme le nombre total d'indentations)

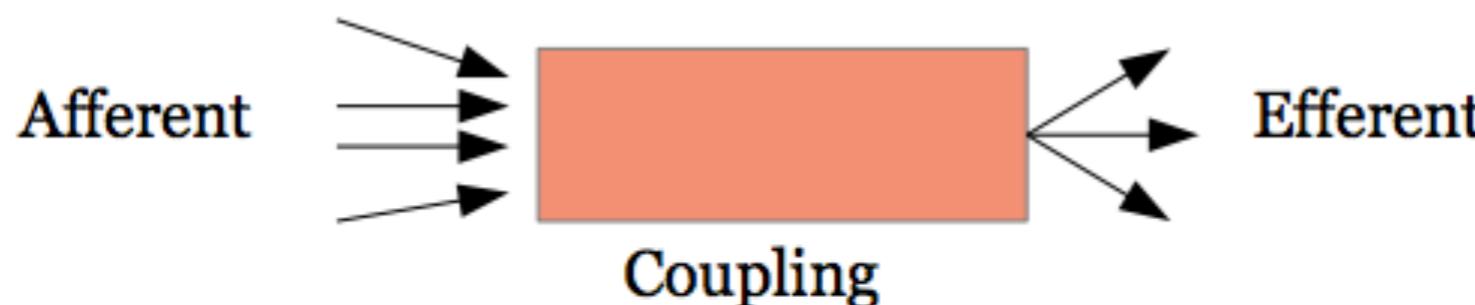
- ajout de code ou modification ? on calcule la moyenne et l'écart type : s'ils diminuent c'est plutôt bon signe.



Complexity Measures to the Rescue?



Cyclomatic Complexity



“Syntactic complexity metrics cannot capture the whole picture of software complexity” (Herraiz & Hassan)

“The use of metrics to manage software projects has not even reached a state of infancy” (Glass)

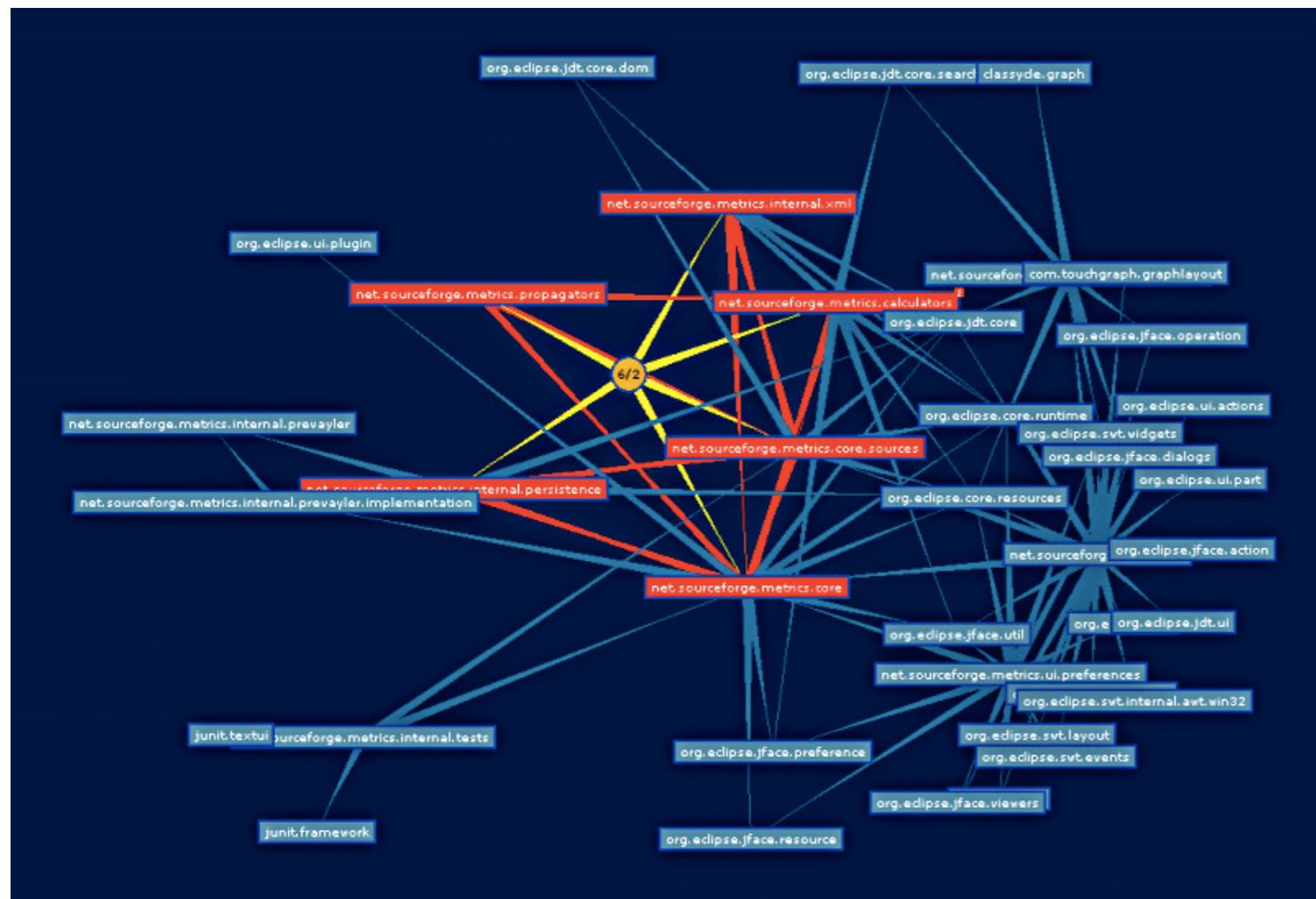
Metrics ?

The average cyclomatic complexity is decreasing in Linux. This is due to the high rate of aggregation of small functions, that dilute the average value of the complexity.

Having only a single data point per release, Lehman decided to use releases as a pseudo-unit of time, formalized as Release Sequence Number (RSN). He decided to use the number of modules as the base unit for size as well.

Evolution du code & Dépendances

- Modifier A => modifier B mais aussi C par exemple un logger....



ok... mais pour les dépendances non « structurelles » ?

Evolution du code & Dépendances

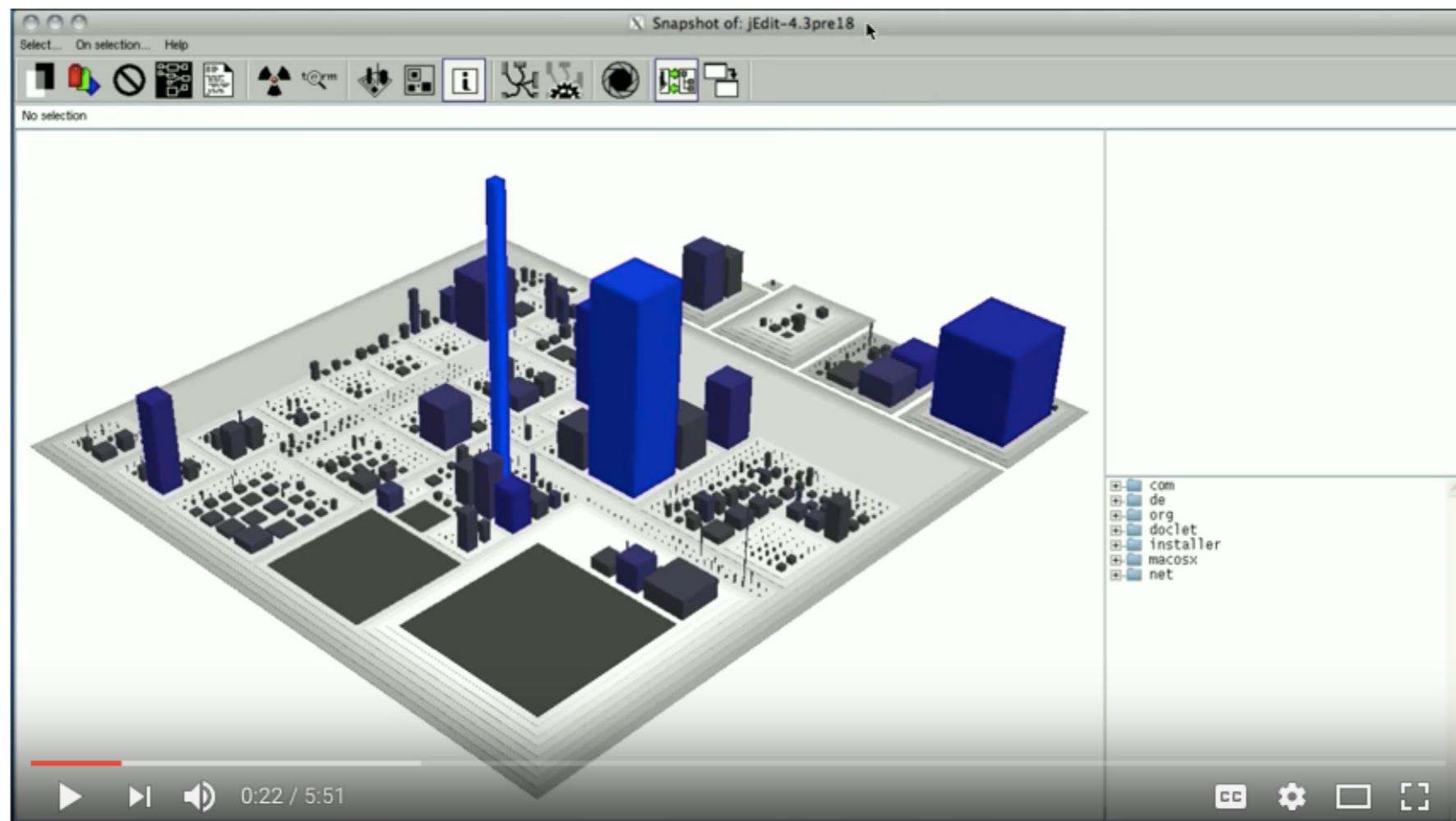
- Code dupliqué...
- Mauvaise encapsulation ou répartition des responsabilités
- Producteur & Consommateur

Connaître les couplages implicites permet de : mieux travailler sur les impacts de modifications; identifier des faiblesses architecturales.

Comment identifier ces dépendances ?

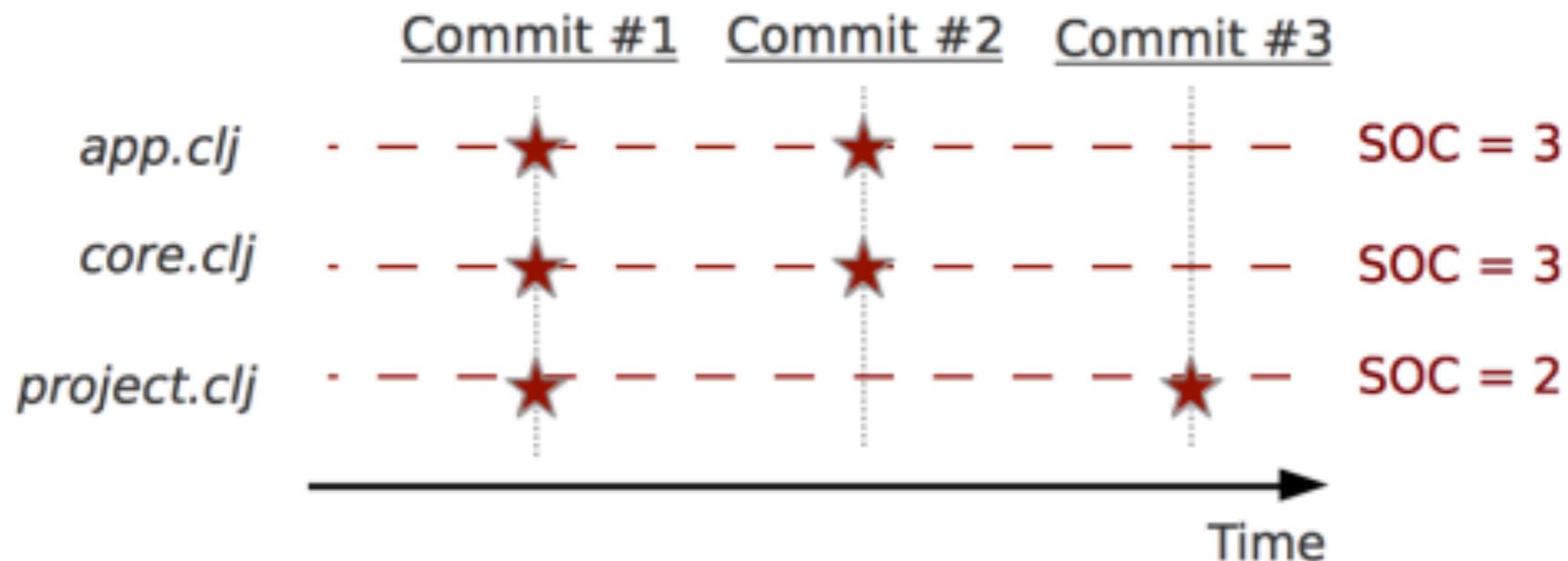
Evolution du code & dépendances : apprendre des commits : couplage temporel !

- Regarder comment les modules évoluent ensemble dans le temps.



Evolution du code & Dépendances

- Compter combien de fois un module évolue en même temps qu'un autre, ce qui donne son « couplage »
- Puis on regarde avec qui il bouge



.../app.clj,.../core.clj -> 60%

.../app.clj,.../scenario_tests.clj, -> 57%

LIMITES ?

Tests et dépendances

« A good test system should encapsulate details and avoid depending on the internals of the code being tested. »

- Pouvez-vous clairement identifier où sont les tests ?
- La fréquence de modification des tests est-elle « proportionnelle » à celle du code ?
- Savez-vous distinguer Tests unitaires et tests système?

Tests et dépendances

« A good test system should encapsulate details and avoid depending on the internals of the code being tested. »

On attend des changements ~40% des tests unitaires...
mais si on est en test-driven développement, c'est différent...

Pour les tests systèmes qui sont à la jointure du système,
un tel score est étonnant...

On s'attend à ce que l'investissement sur les tests reste inférieur à celui sur l'application... sans perte sur la couverture de tests, évidemment !

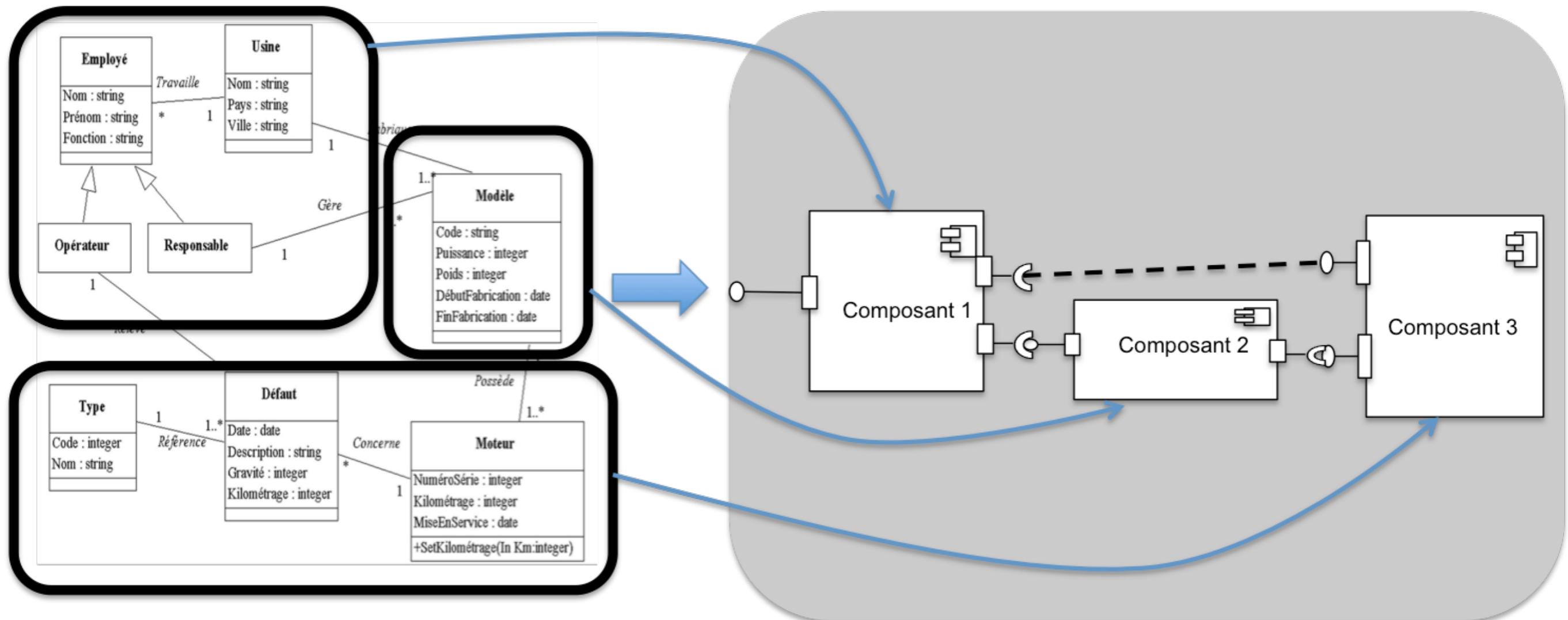
Comprendre l'architecture



- Si on trouve un couplage inattendu (e.g. les vues bougent en même temps que les modèles), regarder les codes impactés en détail.
- Repérer les modules qui bougent le plus (on veut éviter du couplage avec eux!)

Une question récurrente : Une classe a-t-elle trop de responsabilités?

Comprendre l'architecture



Importance de l'équipe

Pluralistic ignorance

Vérifier par des questions et des faits



Commits



Point de départ pour
une discussion



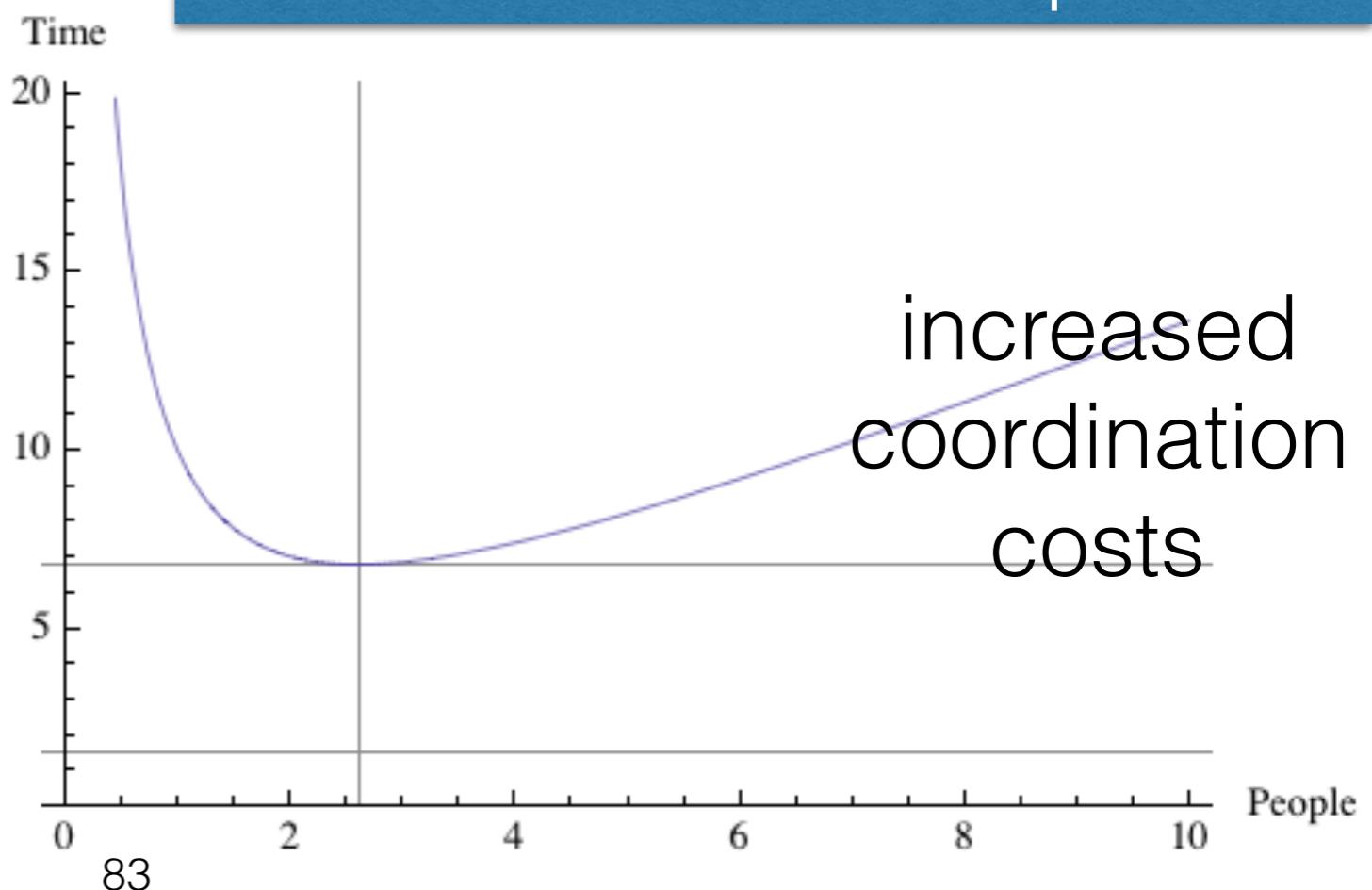
Brooks's Law: Adding manpower to a late software project makes it later.

(Fred Brooks)

izquotes.com

- Baisse de la participation aux discussions
- Baisse de la production
- Dilution des responsabilités

Il faut éviter d'avoir plusieurs développeurs sur les mêmes morceaux de code => identifier ces hotspots



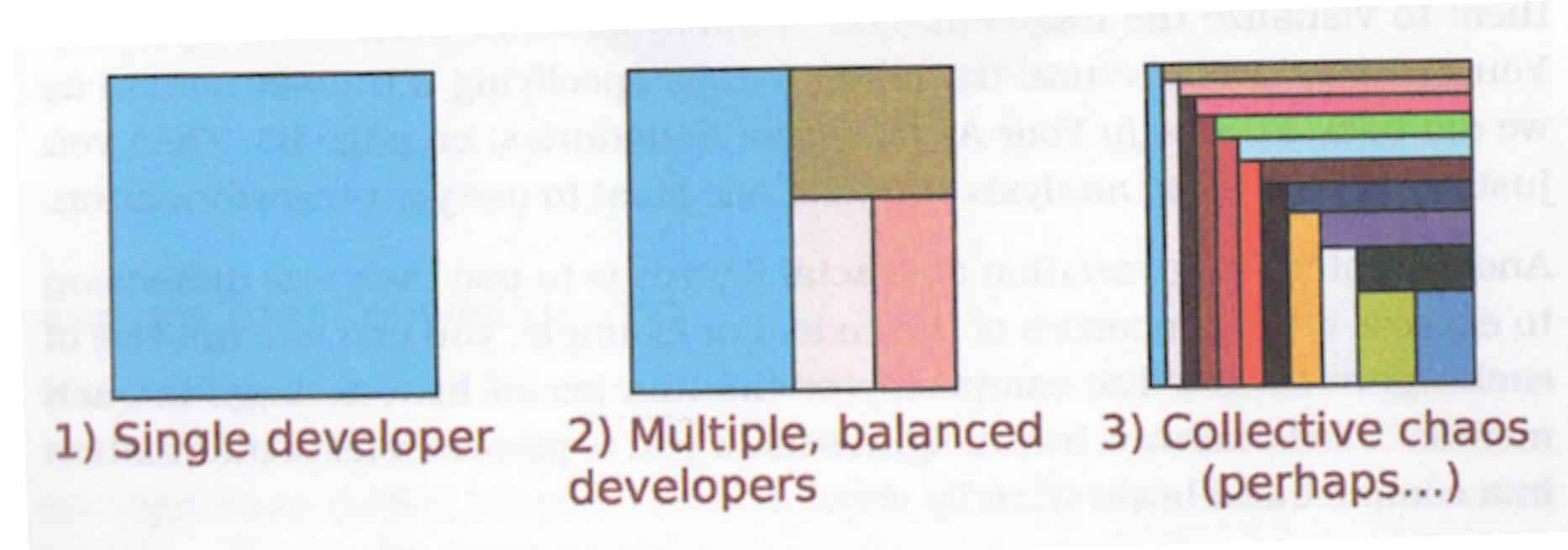
organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

— M. Conway (68)

- Regarder les commits sur une journée, repérer des changements corrélés,
- identifier les principaux contributeurs des modules corrélés, c'est eux qui doivent communiquer.

Un métric : le principal contributeur = celui qui a comité le plus de lignes.

Sont-ils distants?
dans la même
équipe ?

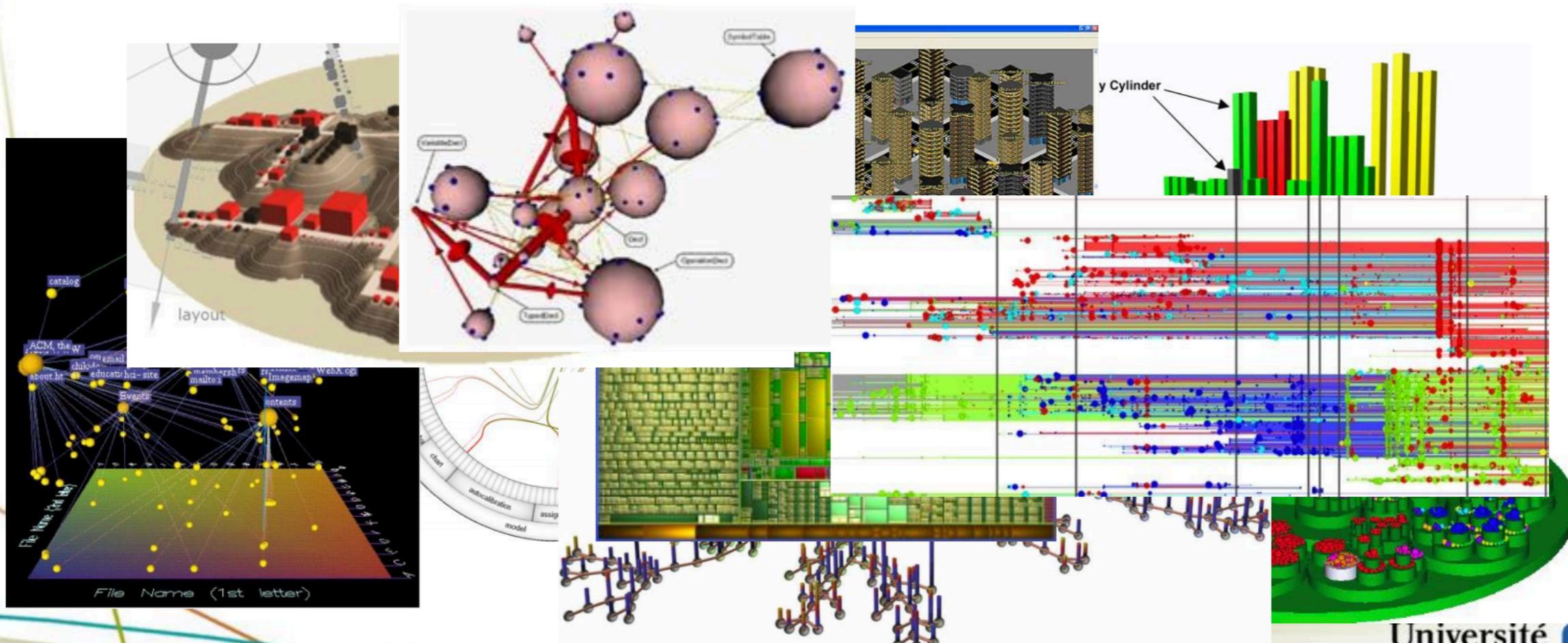


An Appropriate Use of Metrics

*Management love their metrics. The thinking goes something like this, "We need a number to measure how we're doing. **Numbers focus people and help us measure success.**" Whilst well intentioned, **management by numbers** unintuitively leads to problematic behavior and **ultimately detracts from broader project and organizational goals.** Metrics inherently aren't a bad thing; just often, inappropriately used. This essay demonstrates many of the issues caused by management's traditional use of metrics and offers an alternative to address these dysfunctions.*

Visualization Tools for Maintenance

- Dozens of tools proposed each year in VLHCC, SOFTVIS, VISSOFT, SE journals, conferences and workshops, etc.



Université de Montréal

Houari Sahraoui - Université de Montréal - juin 2018

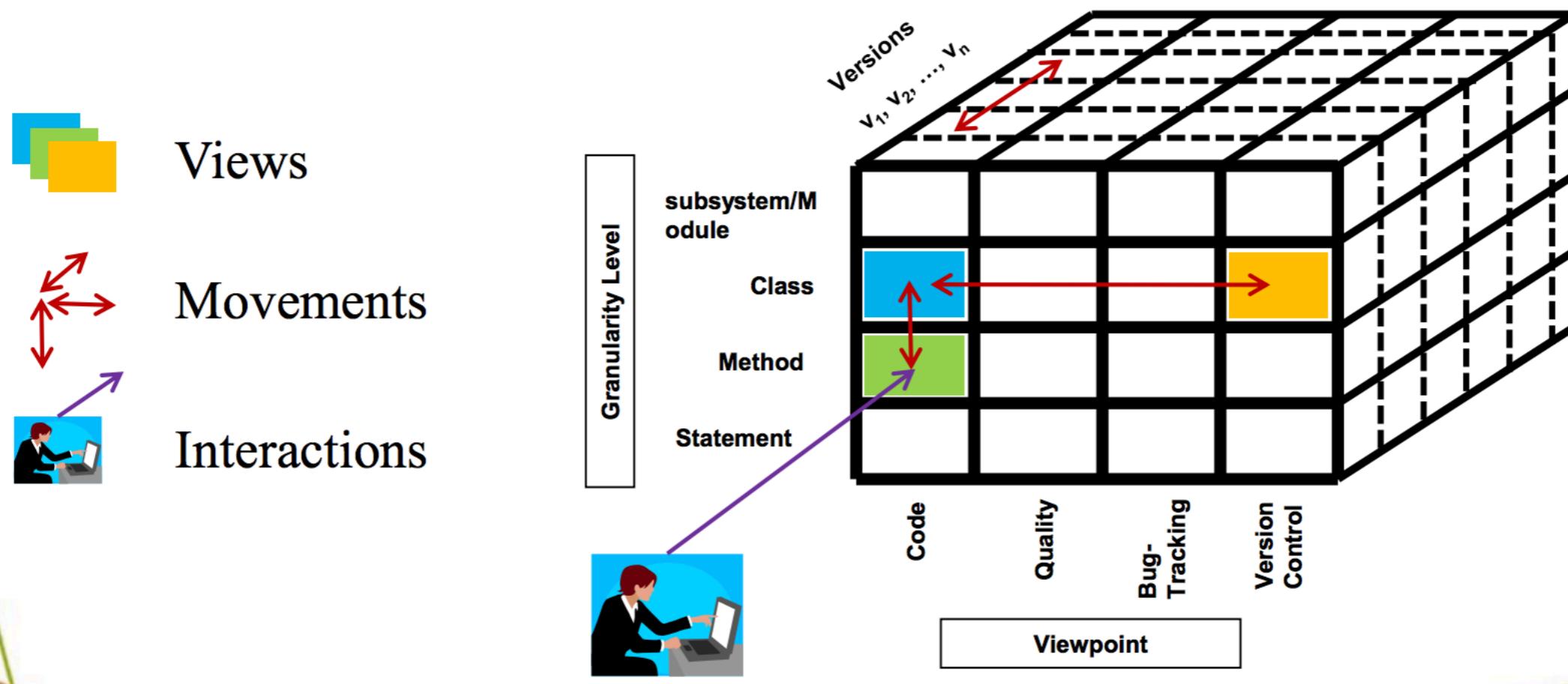
http://gdr-gpl.cnrs.fr/sites/default/files/imagesGPL/JourneesNationales/JourneesNationales2018/visualization_GPL_2018Share.pdf

Modeling Maintenance as Interactive Visualization

- Example : detecting anomalies
 - Example of anomaly definition
 - A Blob is a *controller* class, *abnormally large*, with *almost* no parents and no children. It *mainly uses* data classes, i.e. *very small* classes with *almost* no parents and no children.
 - Model
 - Entities: classes, methods
 - Properties: coupling, cohesion, complexity
 - Relationships: invocations, inheritance
 - Structure: architecture
 - Viewpoints: code, metrics
 - Time: multiple version
 - Operations: evaluate conditions, etc.

Modeling Maintenance as Interactive Visualization

- Visualization world



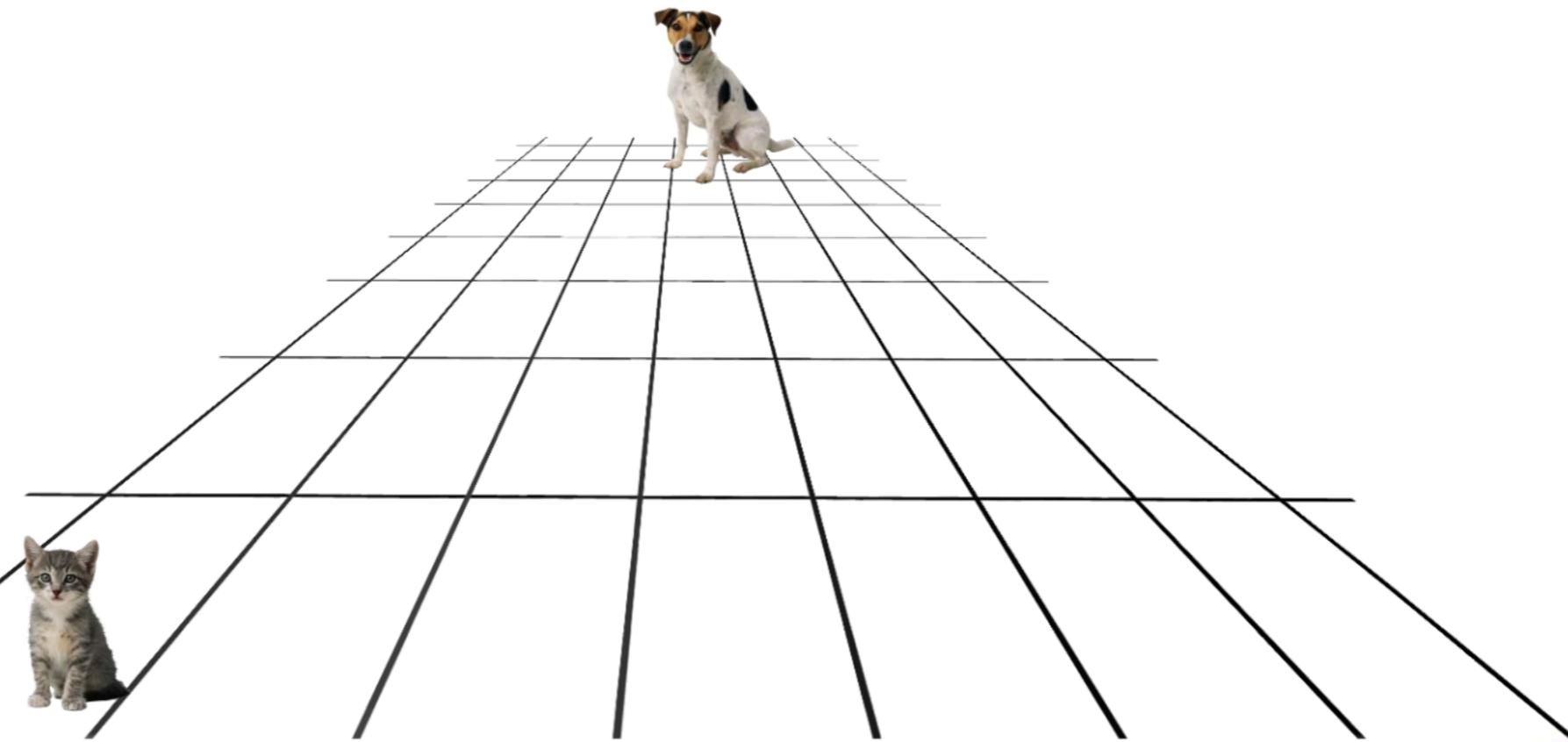
Defining Views

- Software is intangible
- Representations of
 - Entities by shapes
 - Properties by graphical attributes of shapes
 - Relationships by connecting shapes
 - Structures by spatially organizing shapes

Defining Views

- Representing properties (principles)
 - Interaction between visual properties

Size vs depth

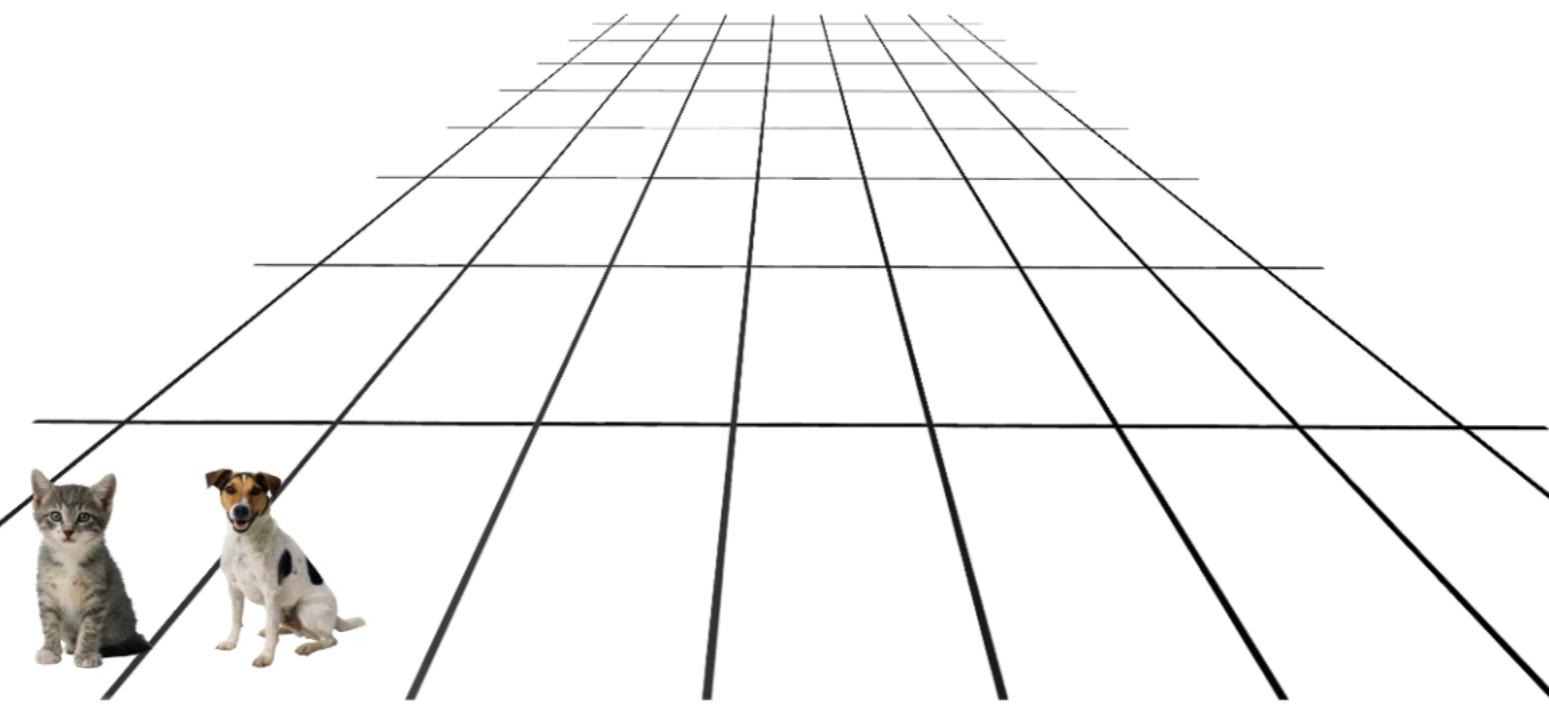


90

Defining Views

- Representing properties (principles)
 - Interaction between visual properties

Size vs depth



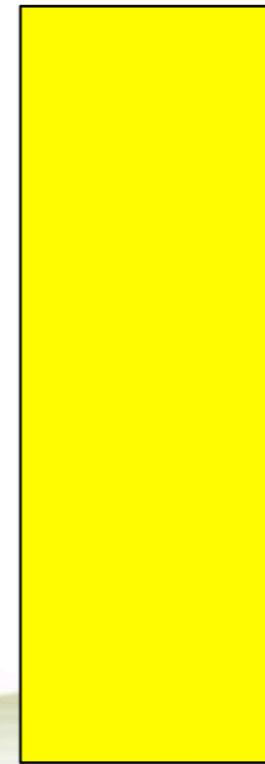
91

Defining Views

- Representing properties (principles)
 - Interaction between visual properties



Size vs orientation



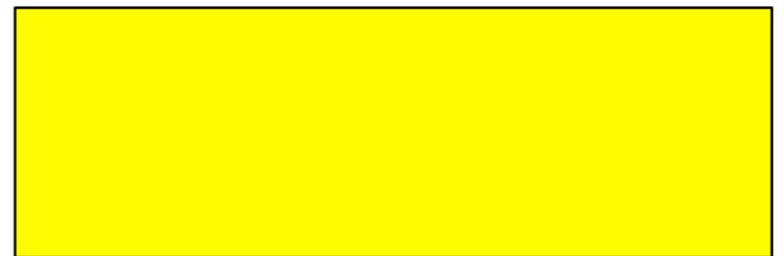
Université
de Montréal

Defining Views

- Representing properties (principles)
 - Interaction between visual properties

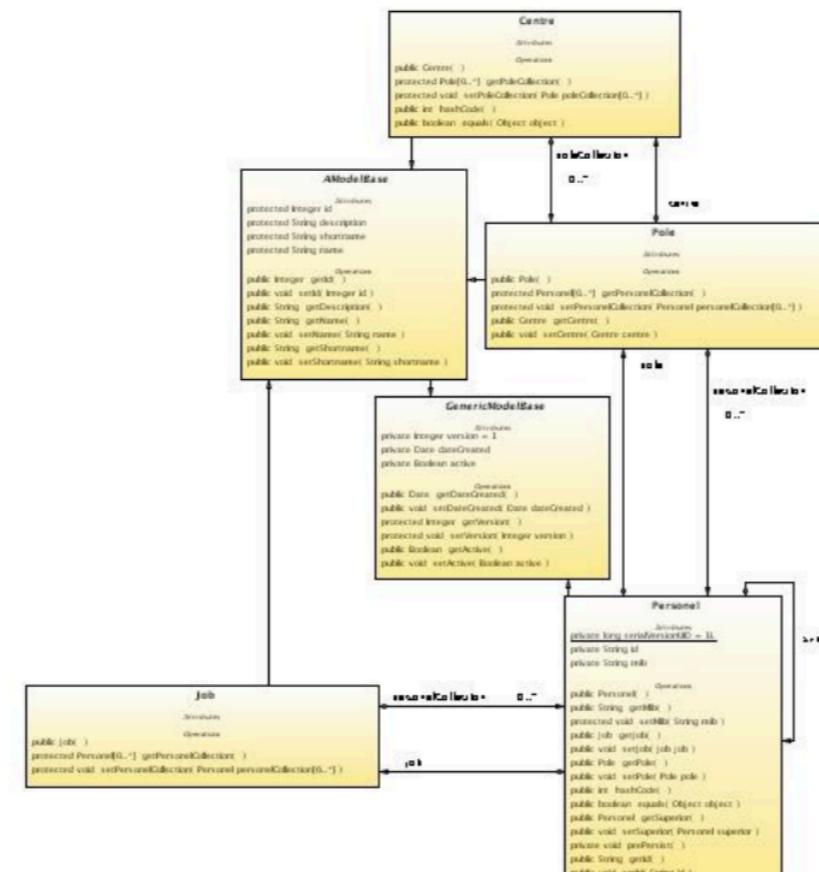


Size vs orientation



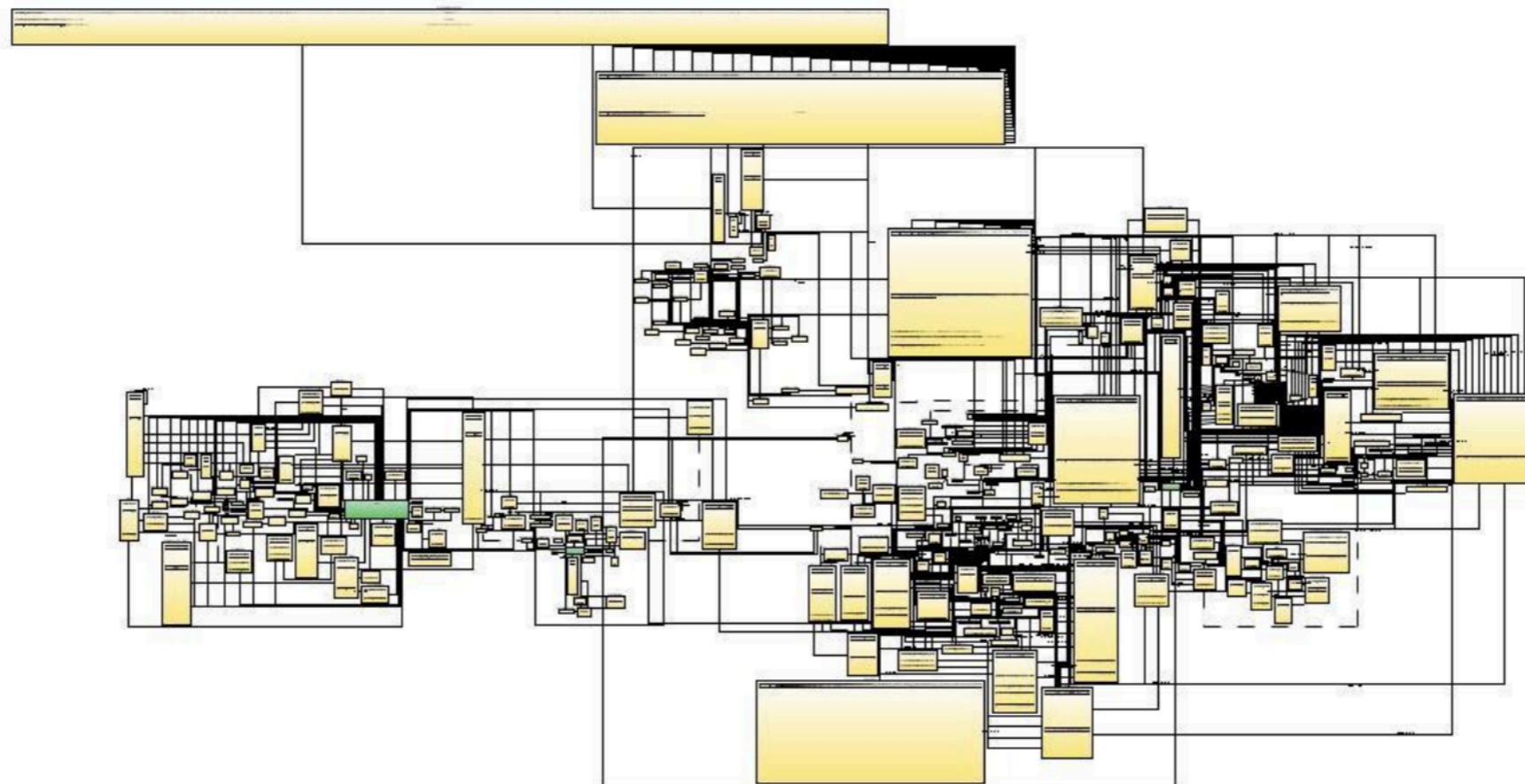
Defining Views

- Representing relationships (principles)
 - Explicit representation vs size



Defining Views

- Representing relationships (principles)
 - Explicit representation vs size



Evaluation du module

- Les rendus associés à un livre

- **Rendu L.1.:** “Voici mon sujet” Date limite : 7/01 à 15h.
- (0.5) **Exposé E.1. :** Présentation du projet, 23/1
- (2) **Rendu L.2 :** Contenu du livre
- (1) **Rendu L.3 :** Codes/Résultats Brutes

- (3) Examen

A book (Github book)

Un chapitre par sous-groupe

Quelques chapitres de l'an dernier et de l'année d'avant

Conclusion

A book subchapter : Une question par chapitre

- I. Research context /Project
- II. Observations/General question
- III. Information Gathering
- IV. Hypothesis & Experiences
- V. Result Analysis and Conclusion
- VI. Tools (facultatif)
- VI. References

Sujets proposés

- **Sujet GIT - Make merging great again**
- **Sujet ROCKFlows - Does its architecture rock?**
- **Sujet XWiki : Will you still improve the quality of an open-source project?**
- **Sujet Feature toggle: Something new, just toggle it...**

EXAMPLE

What are the impacts of Test-Driven Development on code quality, code maintainability and test coverage ?

En 2016-17

- Does TDD reduce the number of issues to fix during the development ?
- Does TDD reduce the overall complexity of a project compared to the common Test Last (TL) method ?
- Does TDD projects always have a high test coverage ?

What are the impacts of Test-Driven Development on code quality, code maintainability and test coverage ?

Projects Studied

TDD

FitNess : project from Uncle Bob, ...more than 5000 commits.

JUnit4 .. using TDD throughout around 2,000 commits.

JFreeChart around 3,000 commits.

OpenCover ...code coverage tool for .NET around 1,200 commits).

TL

Google Gson ...around 1,300 commits

JaCoCo ...around 1,400 commits

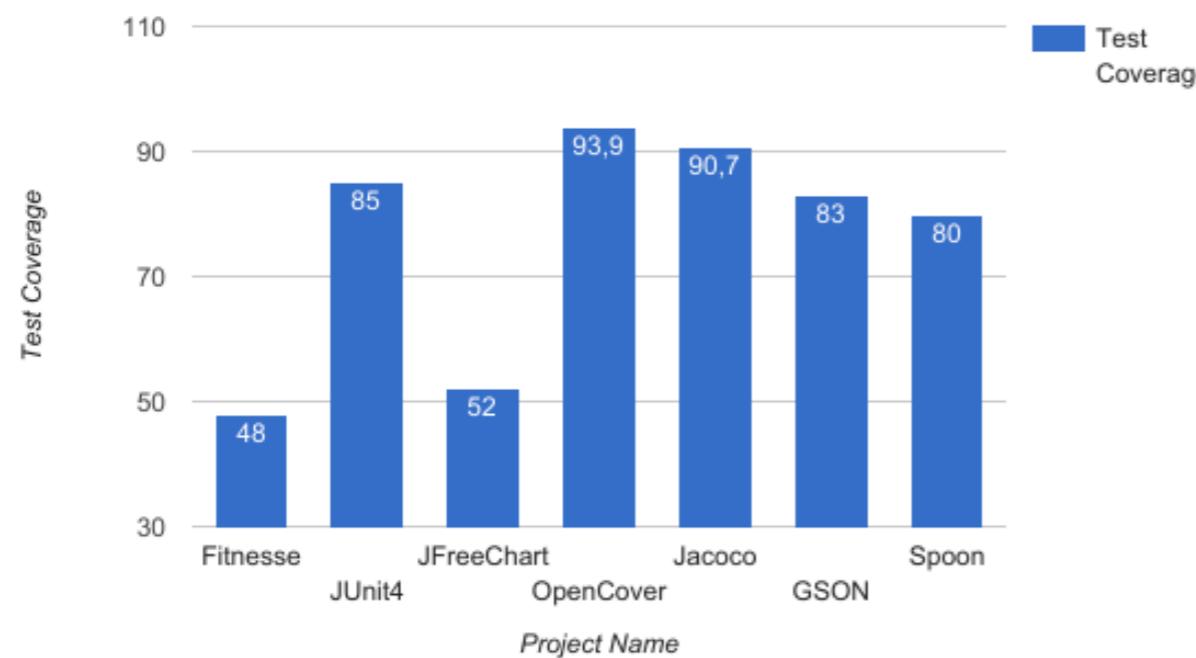
Spoon ..around 1,800 commits

JaCoCo :
couverture de tests

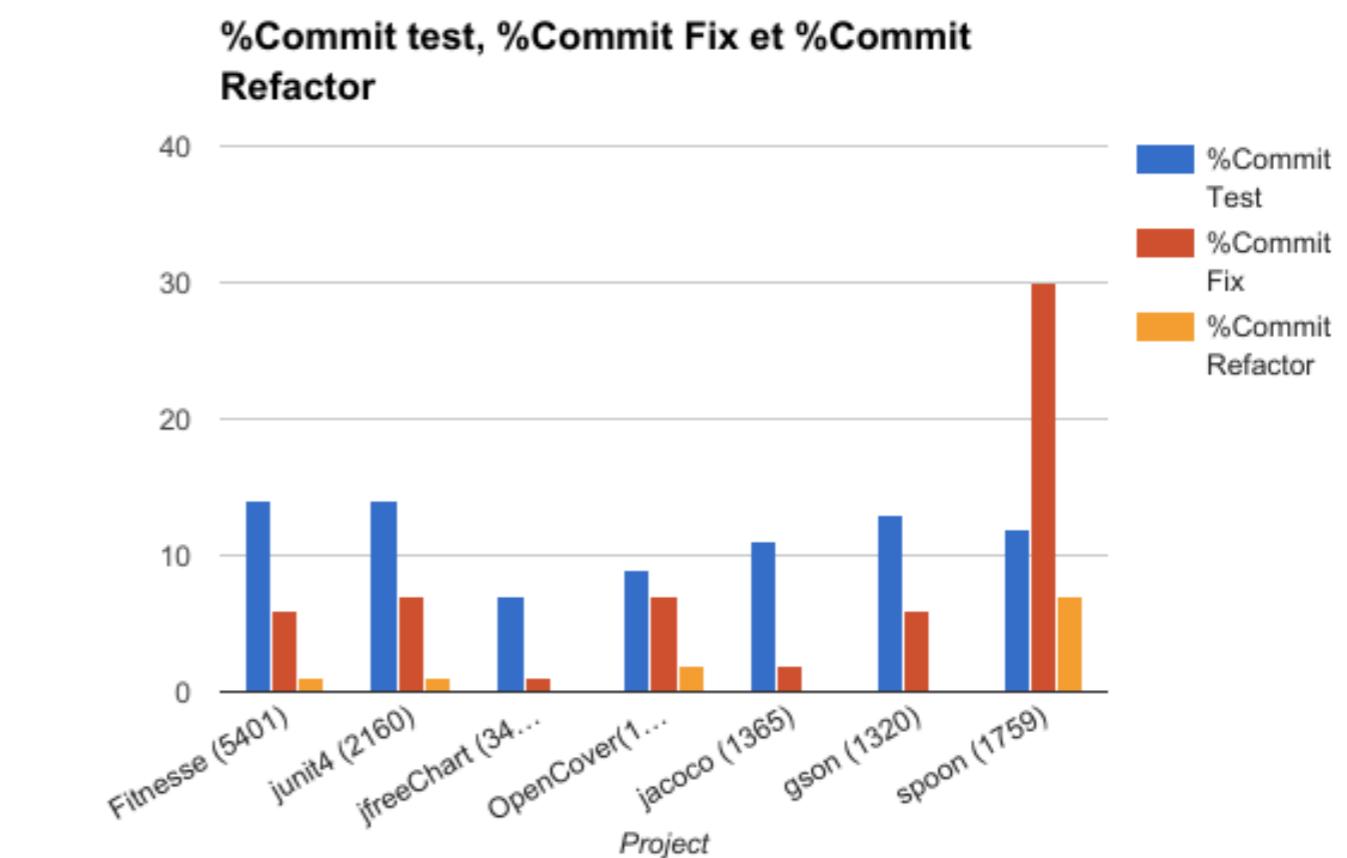
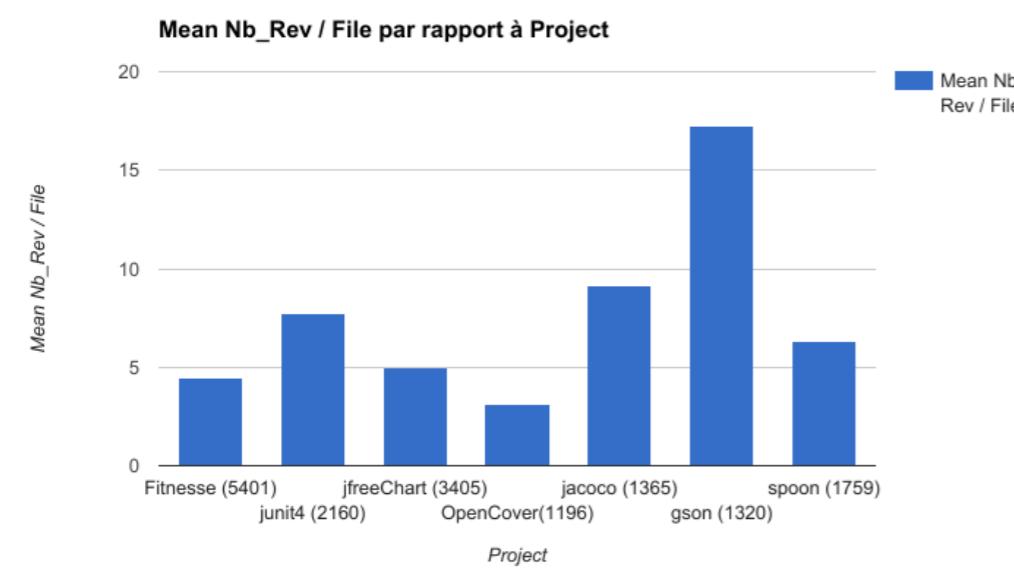
Sonar : issues, métriques et visualisation

Code maat :
analyse de dépôts Git

Metrics	Test-Driven Development				Test-Last Development		
	Fitnesse	JUnit4	JFreeChart	OpenCover	Spoon	GSON	JaCoCo
Code Coverage	48%	85%	45%	93.9%	90.7%	83%	80%
Sonar issues	1927	833	5039	286	2341	592	200
Complexity	8612	2061	19323	1568	7635	1945	1962
Code Age	48.3%	21.5%	18.1%	83.6%	9%	50%	35%
Average number of reviews/files	4.46	7.72	4.97	3.15	6.35	17.26	9.12 >
% "Fix" Commit	6%	7%	1%	7%	30%	6%	2%
% "Refactor" Commit	1%	1%	0%	2%	7%	0%	0%
% "Test" commit	14%	14%	7%	9%	12%	13%	11%



=> le calcul sur Spoon pour les commits "Fix" vient sûrement du fait qu'on utilise le terme "fix" dans les commits pour clore les tickets que l'on ouvre sur Github.
@simon.urli



15/06/2017

Ciel 2017

- Alexandre Cazala <alexandre.cazala@gmail.com>
- Nicolas Lecourtois <lecourtoisn@gmail.com>
- Lisa Joanno <lisa.joanno@gmail.com>
- Pierre Massanès <pierre.massanes@gmail.com>

103

Visualisation : SoftVis3D

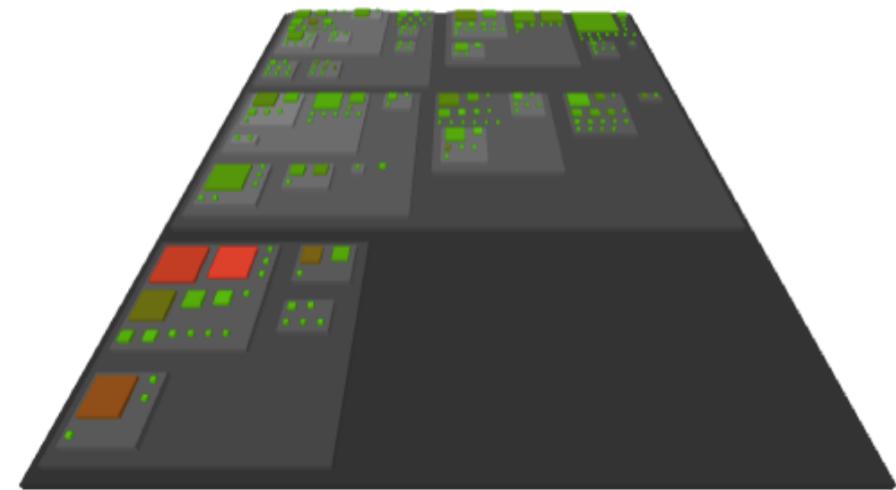


Figure 4 : SoftVis3D results for JUnit4

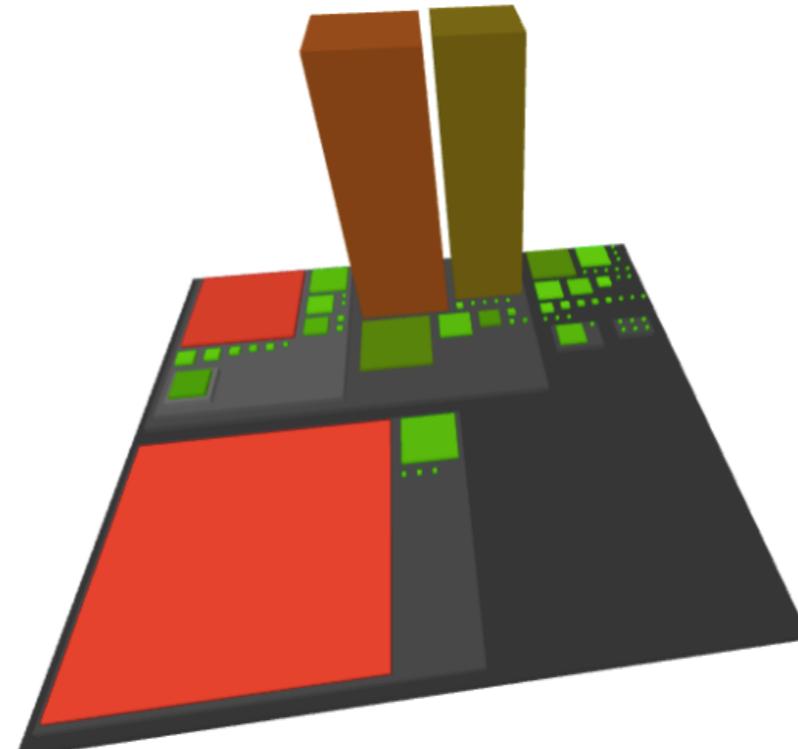


Figure 8 : Google GSON

For our project, we use it to judge the global **cleanliness** of a project. We used the complexity as footprint, the number of duplicated lines as height, and number of Sonar issues as the color.

- Alexandre Cazala <alexandre.cazala@gmail.com>
- Nicolas Lecourtois <lecourtoisn@gmail.com>
- Lisa Joanno <lisa.joanno@gmail.com>
- Pierre Massanès <pierre.massanes@gmail.com>

What are the impacts of Test-Driven Development on code quality, code maintainability and test coverage ?

The results that we analysed shows that the TDD projects have an overall complexity and code quality better than TL projects, according to our expectations. However, the results gathered about code coverage are not matching our expectations. Half of the TDD projects have a code coverage lower than 60% and all the TL projects have a coverage higher than 80%.

EXAMPLE

Comment est organisé le développement d'un projet Open Source de Machine Learning ?

En 2017-18

- Robin Alonzo <alonzo.robin@etu.unice.fr>
- Antoine Aubé <aube.antoine@etu.unice.fr>
- Thierno Balde <balde.thierno@etu.unice.fr>
- Mathieu Mérino <merino.mathieu@etu.unice.fr>

1. Ces projets sont-ils menés par des chercheurs ?

2. Qui écrit les algorithmes d'apprentissage automatiques dans ces projets ?

3. La qualité logicielle est-elle une préoccupation de ces projets ?

Successivement, nous avons observé l'importance de la contribution du monde de la recherche dans ces projets, de l'existence d'un fort *code ownership* des algorithmes et d'un effort particulier pour maintenir une bonne qualité.

