Software Maintenance

Mireille Blay-Fornarino

Site web : https://mireilleblayfornarino.i3s.unice.fr/doku.php?id=teaching:reverse:2017:start

image : https://www.linkedin.com/pulse/software-reengineering-successful-implementation-vasiliy-soloshchuk-1

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

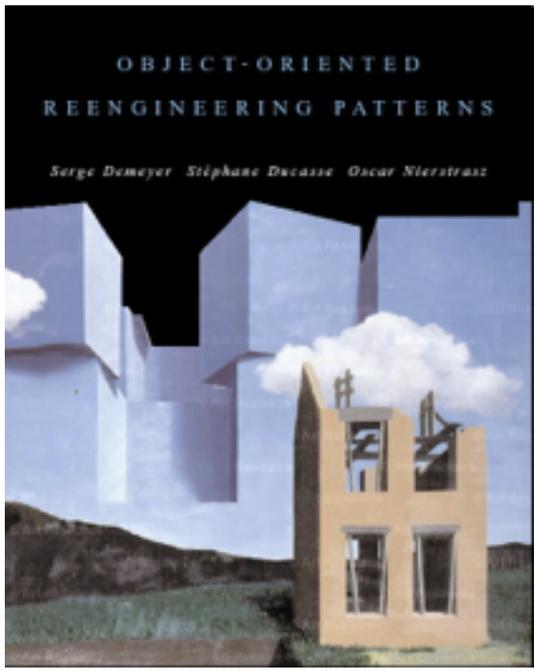
Pour comprendre/modifier un système informatique complexe, vous faîtes quoi ?

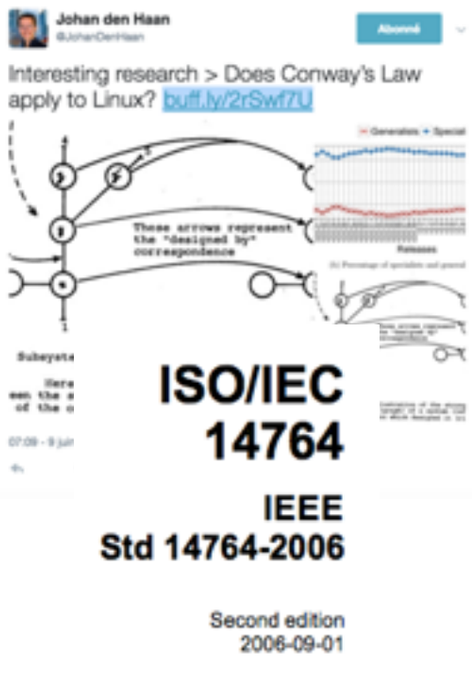
Maintenance ?
Rétro-Ingénierie ?

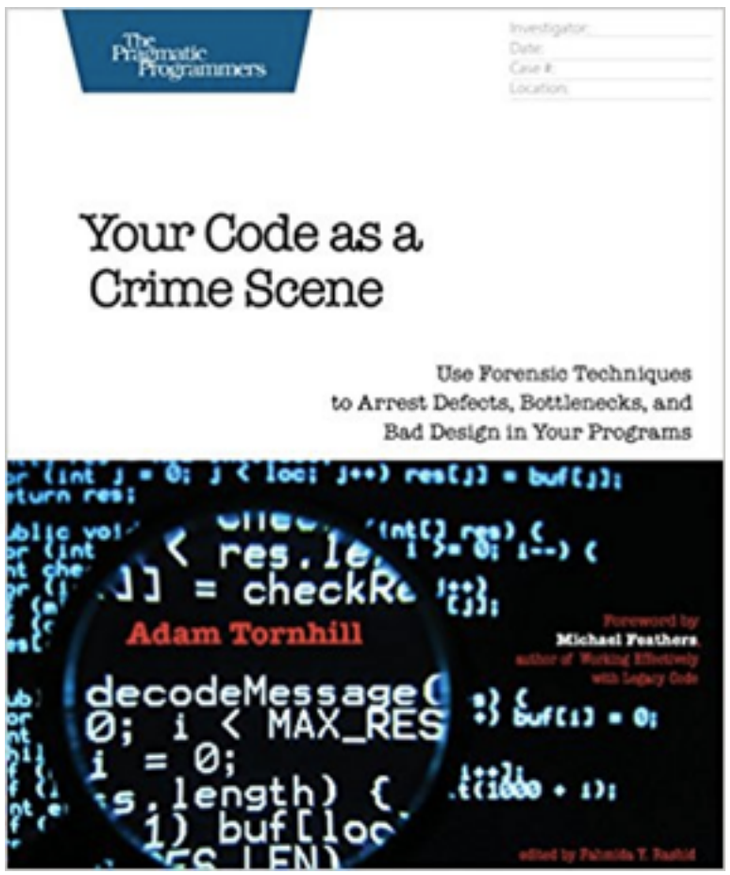
Pour vous ?

Enseigner la Maintenance Logicielle

« réputée non-enseignable ».

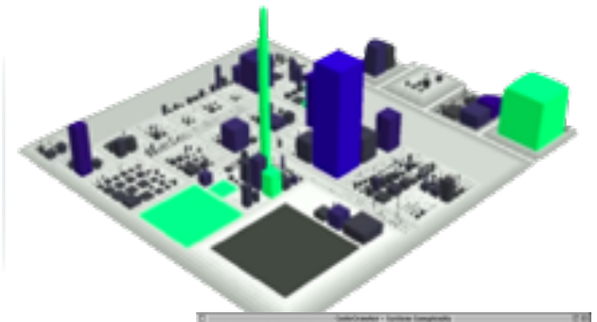


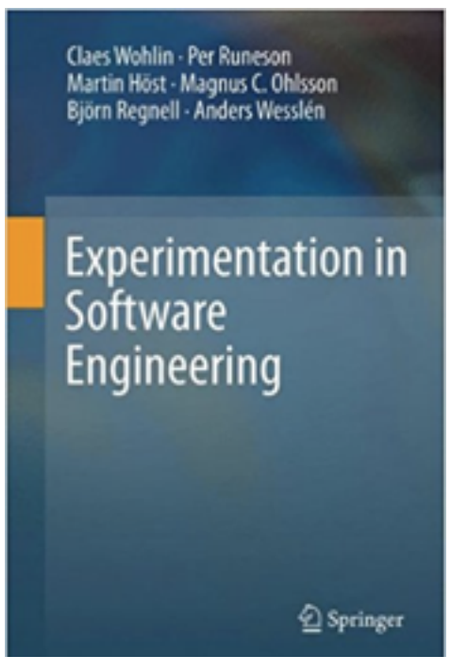


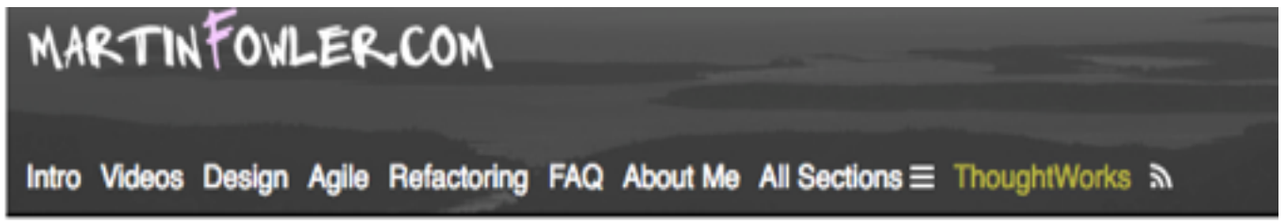


Analyse de code ?

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







Agilité …

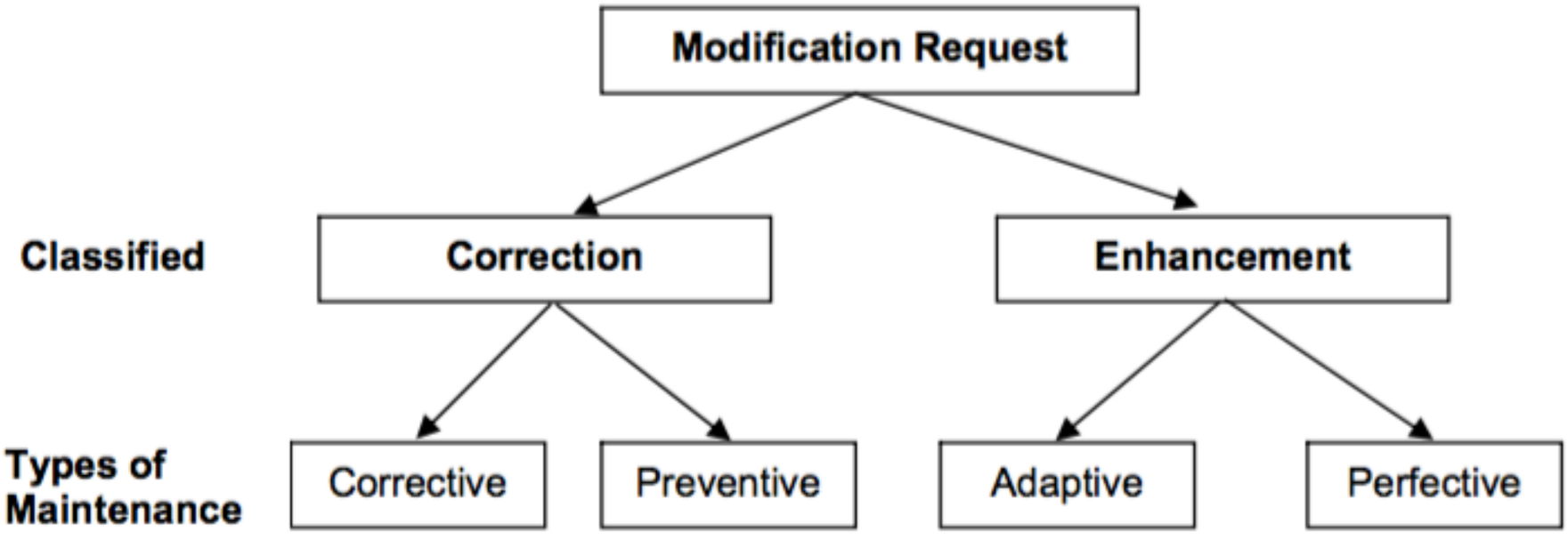
Refactoring…

Maintenance?

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

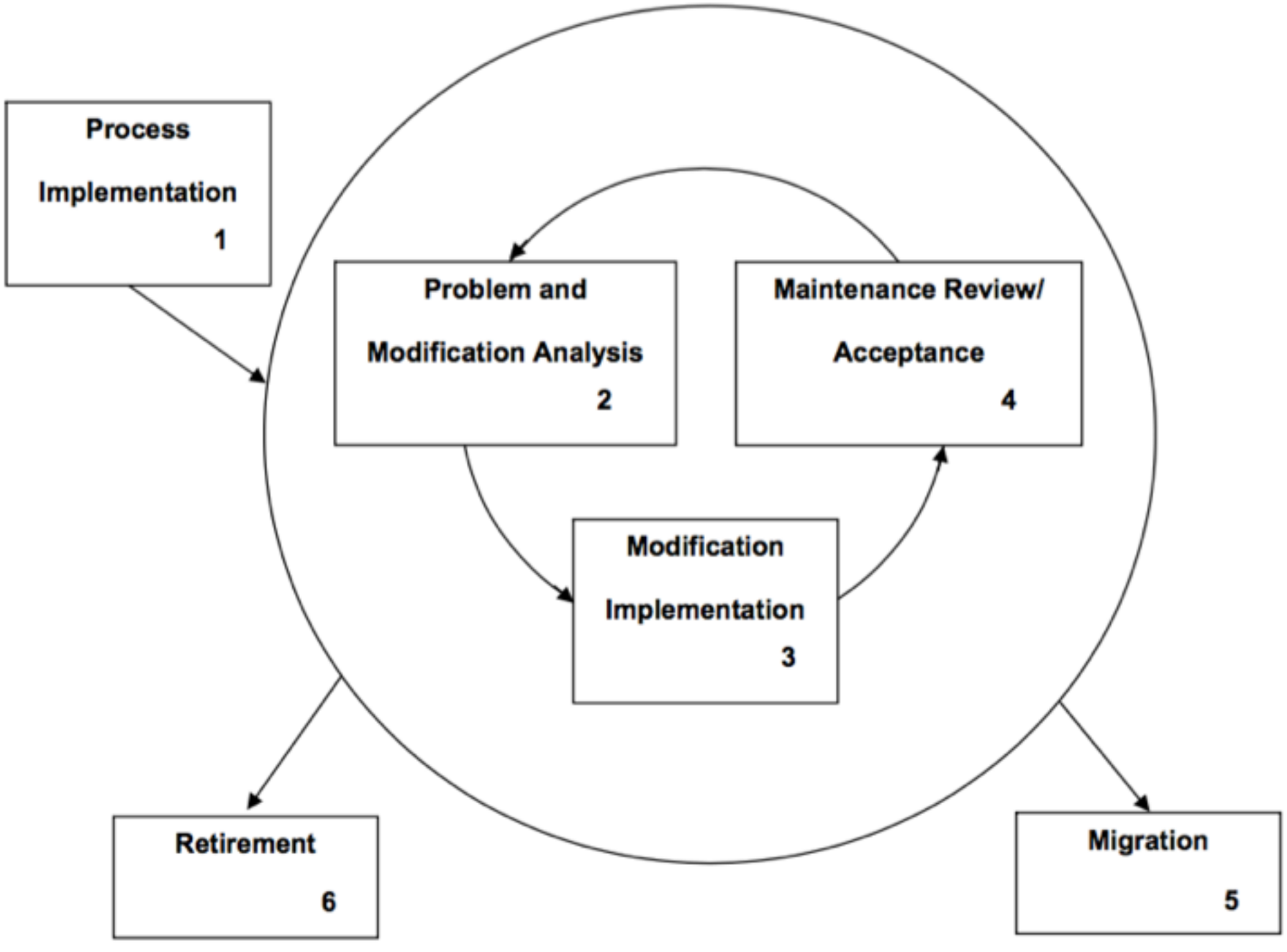
preventive maintenance : the modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults

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 : he modification of a software product afte delivery to detect and correct latent faults in the software product bef they are manifested as failures

Maintenance Process



Techniques pour la maintenance

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.

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

Reverse Engineering is the process of analyzing software to i***dentify*** 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 ow 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.

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 Re-engineering

- **Modulariser** : Diviser un système monolithique en modules qui peuvent être commercialisés séparément

- **Performance** : "Faites d'abord, puis faites-le bien, puis faites-le vite" - l'expérience montre que c'est la bonne séquence!

- **Portage** vers une autre plateforme : l'architecture doit distinguer les modules dépendant de la plate-forme

- **Refactoriser** : améliorer la maintenabilité, la portabilité, etc.

- **Exploitation** de nouvelles technologies

Chikofsky and Cross [in Arnold, 1993]

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?)

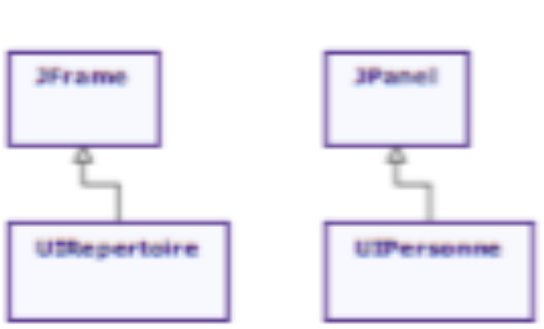
- …

Chikofsky and Cross [in Arnold, 1993]

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)

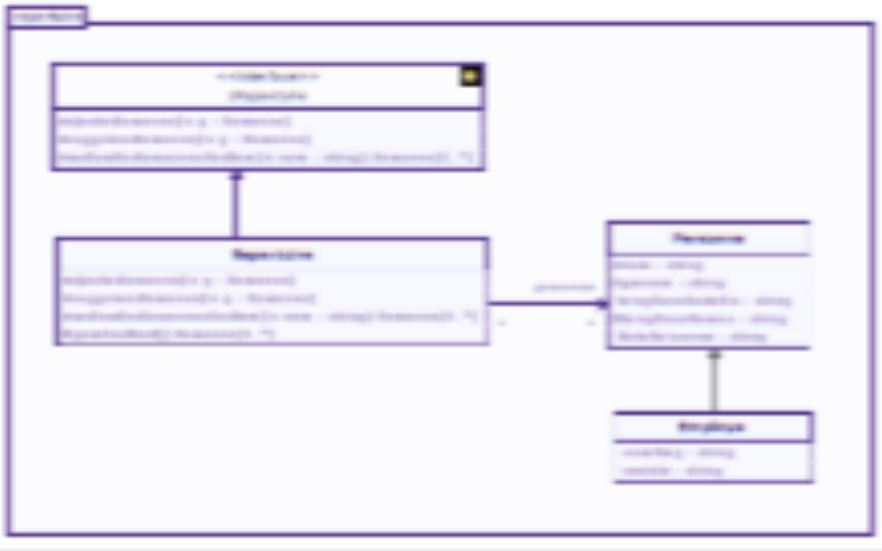


Vue sur les généralisations

Vue sur les packages

Vue sur les associations

Zoom sur le contenu d'un package

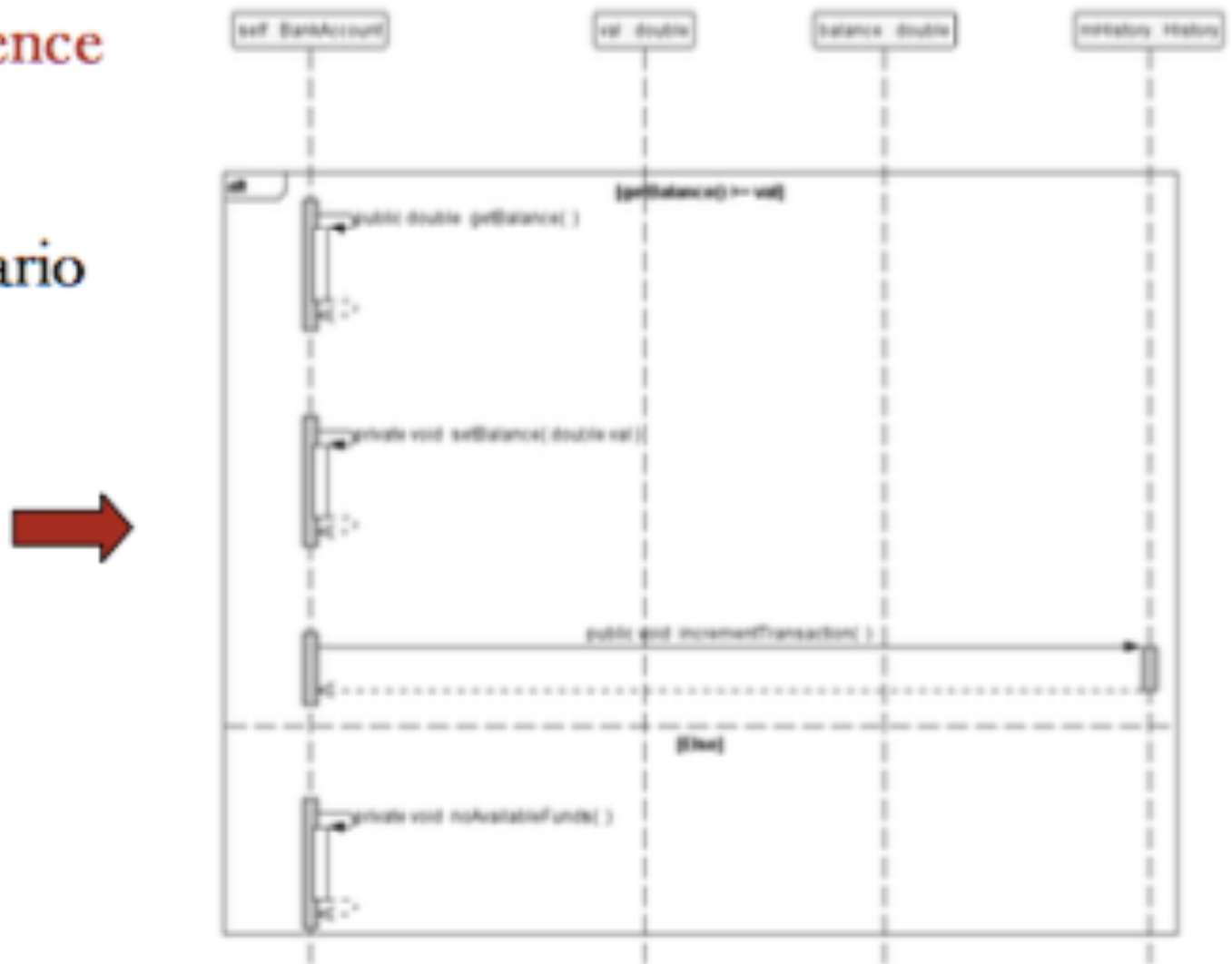


Reverse Engineering: Aspect comportemental

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

```
public void withdraw(double val) {

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



Tools for maintenance

«A potential means of containing software maintenance costs is to use CASE tools. These tools aid software maintenance activities. .. This interrelated collection of CASE tools should be brought together in the form of a Software Engineering Environment (**SEE**) to support the methods, policies, guidelines, and standards that support software maintenance activities. A Software Test Environment (**STE**) should also be provided for the maintainer so that the modified software product can be tested in a non-operational environment…*To date the adoption of CASE tools has met with limited success.* »

De quels outils avons-nous besoin ?

ISO/IEC 14764:2006(E) IEEE Std 14764-2006

Outils pour la maintenance

program slicers : which select only parts of a program affected by a change;
static analyzers : which allow general viewing and summaries of a program content;
dynamic analyzers : which allow the maintainer to trace the execution path of a program;
data flow analyzers : which allow the maintainer to track all possible data flows of a program;
cross-referencers : which generate indices of program components;
dependency analyzers : which help maintainers analyze and understand the interrelation- ships between components of a program.

…. Code Orienté …

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

Feedback loop

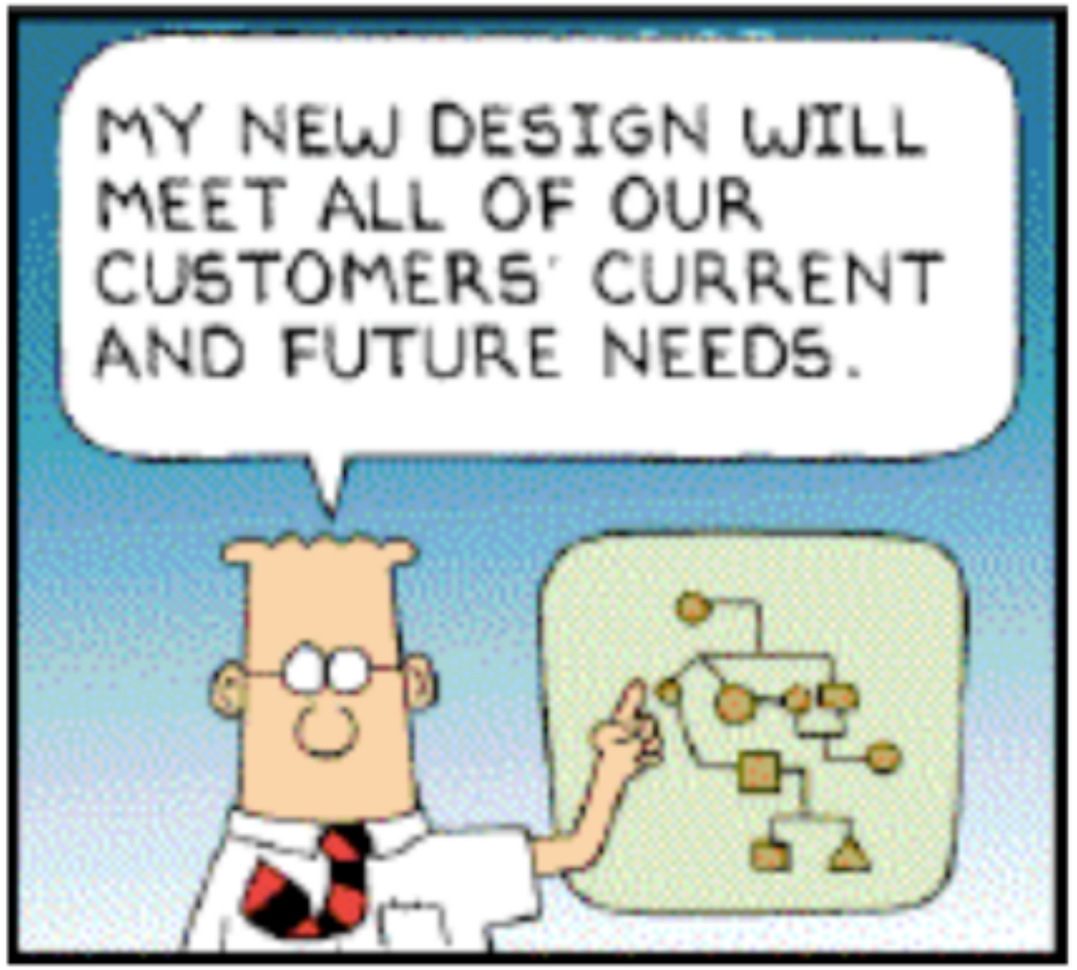
Because t*he world irremediably changes*, .. software must be changed to keep it synchronized with its environment.
Software requirements also change, because *human processes are hard to define and state*, which also lead to modifications in the software. ….
Also, the very introduction of the system 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 !

http://www.slideshare.net/tommens/future-research-challenges-in-software-evolution?from_action=save

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.

The evolution of the laws of software evolution. A discussion based on a systematic literature review. 2013

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.

The evolution of the laws of software evolution. A discussion based on a systematic literature review. 2013

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 ?

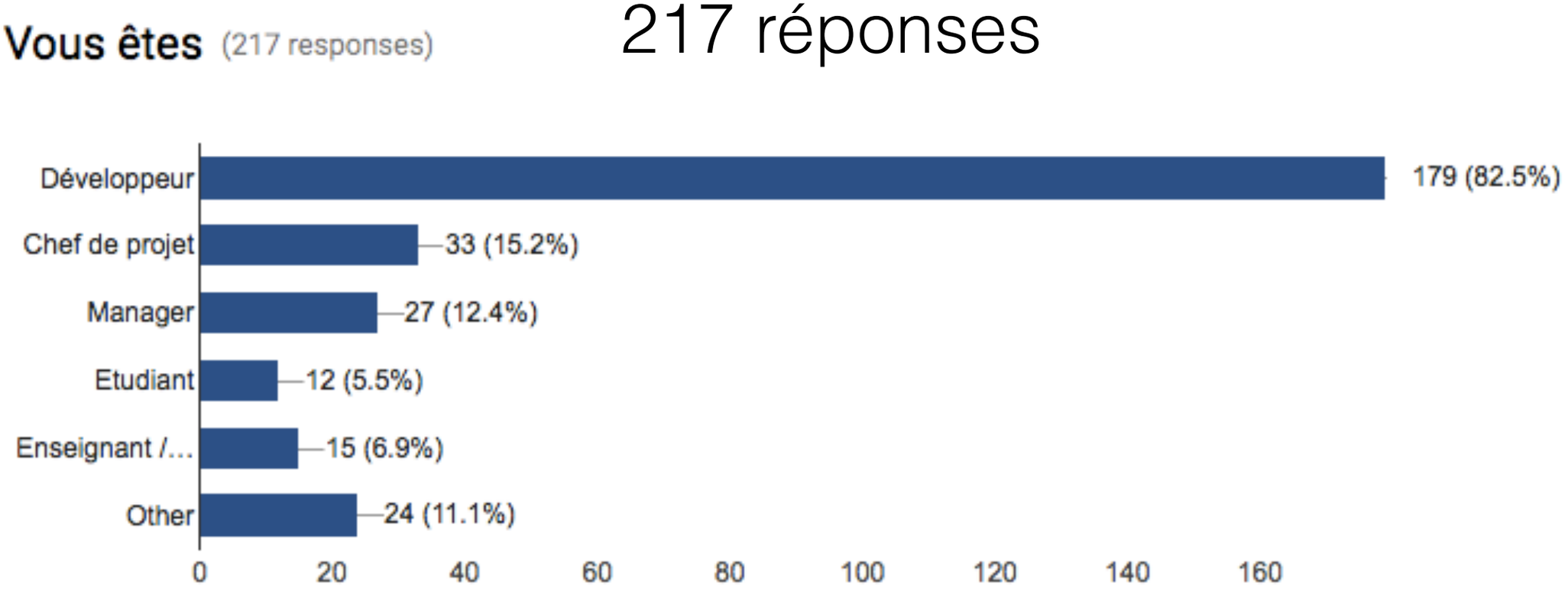
Ces règles s'appliquent-elles au logiciel libre qui n'intègre pas une notion de « hiérarchie »?

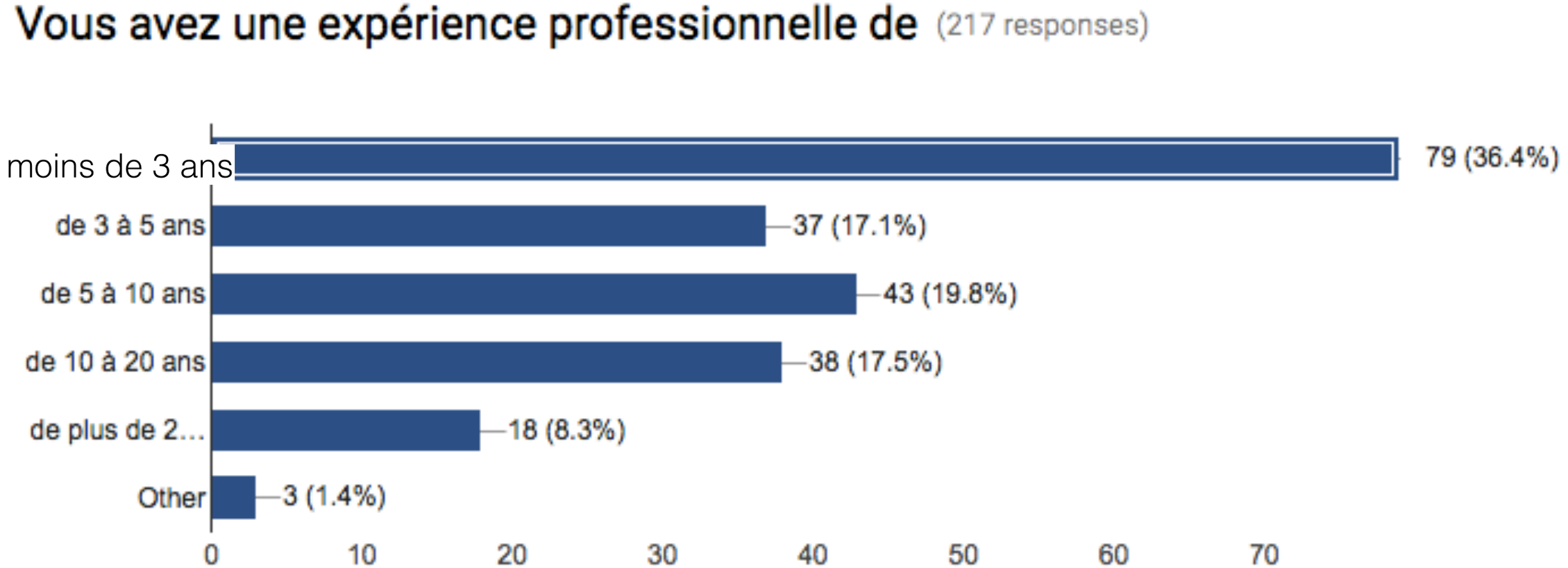
The evolution of the laws of software evolution. A discussion based on a systematic literature review. 2013

Un sondage…

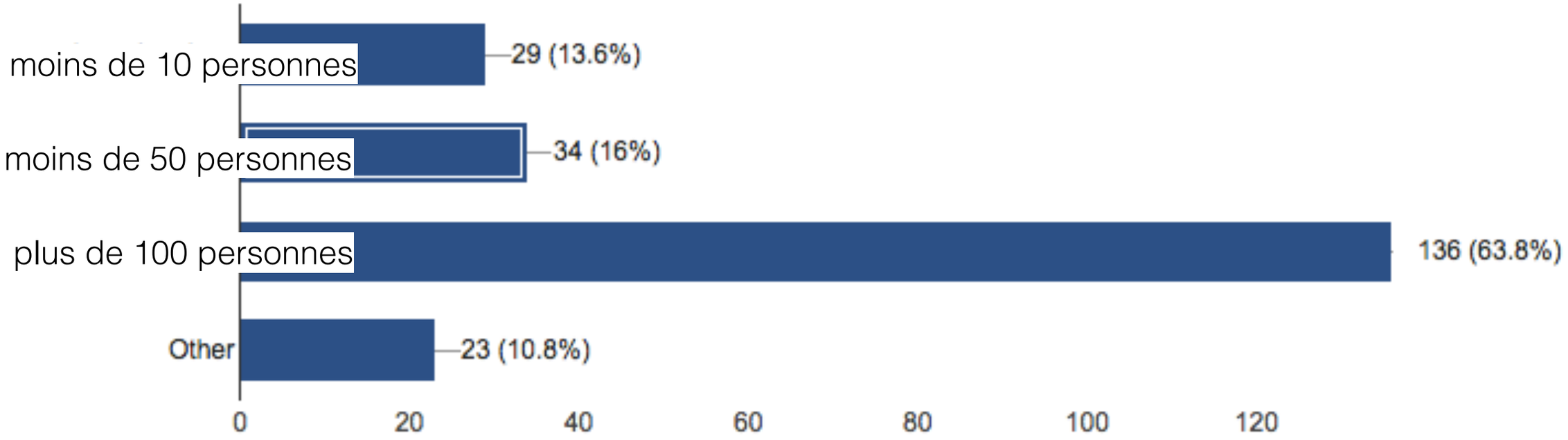
juillet à octobre 2016

217 réponses

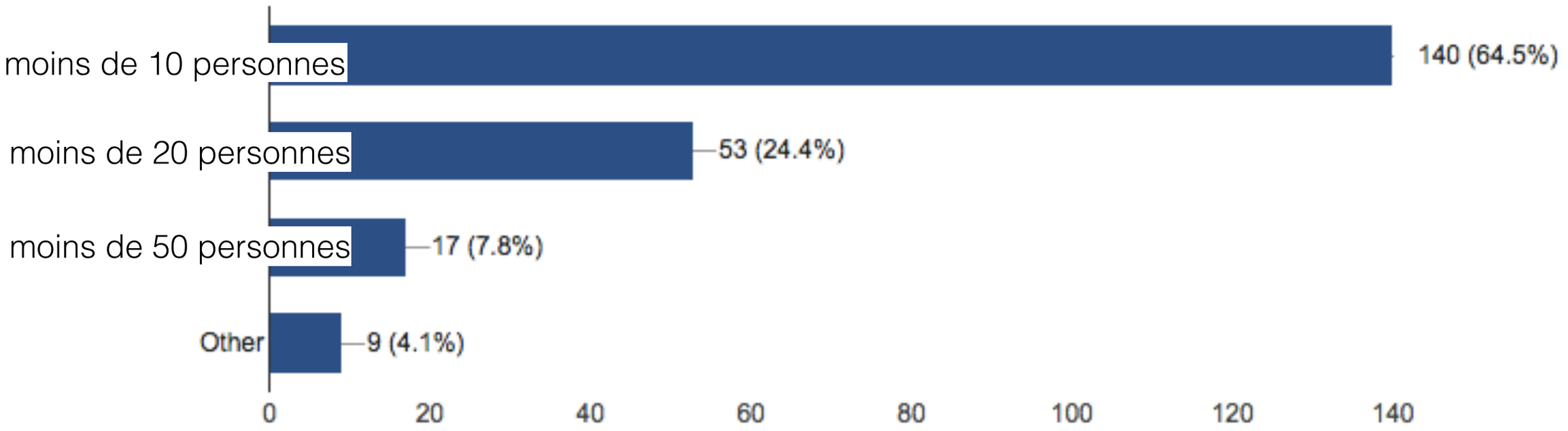




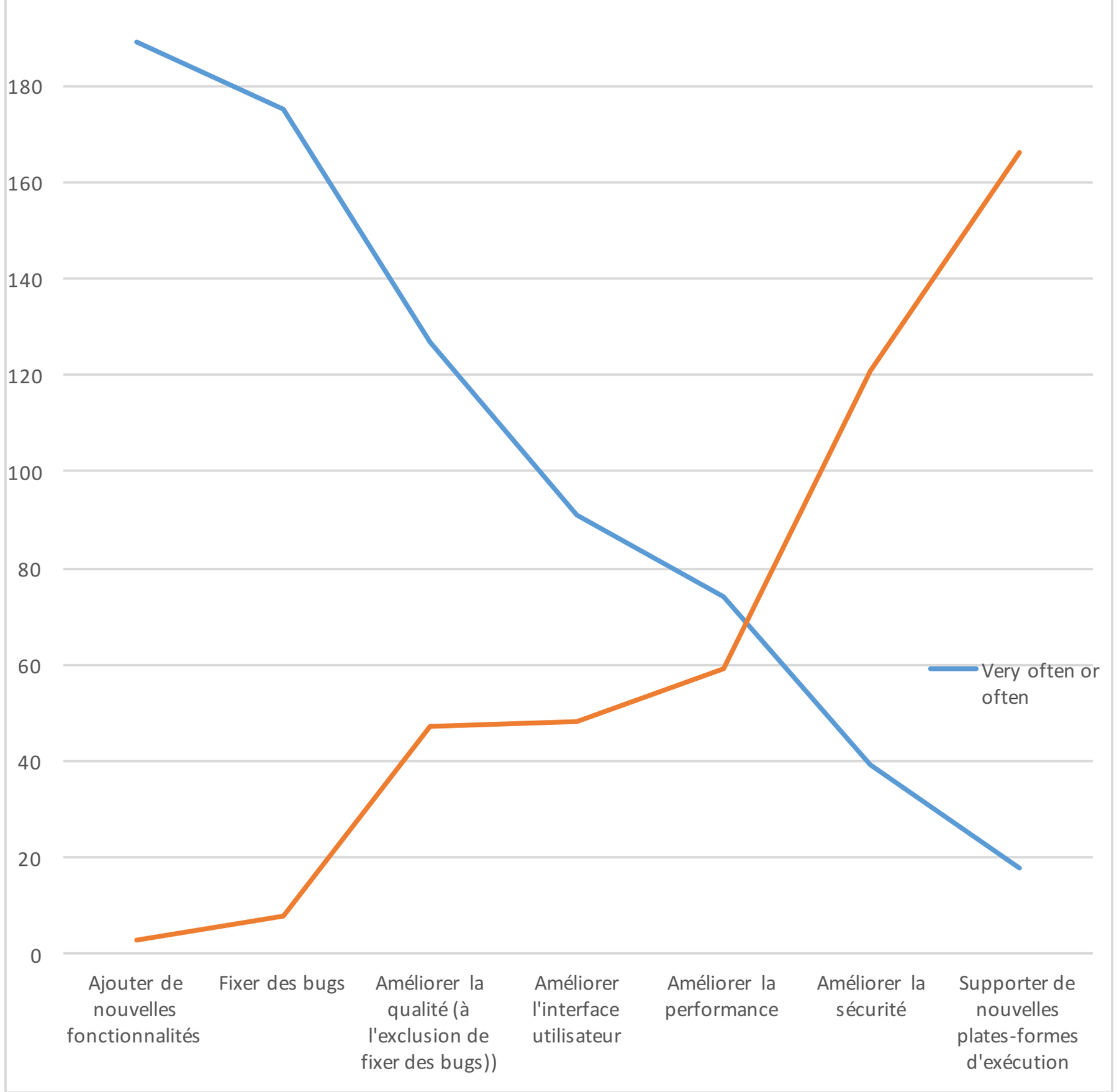
Vous travaillez pour une entreprise (213 responses)



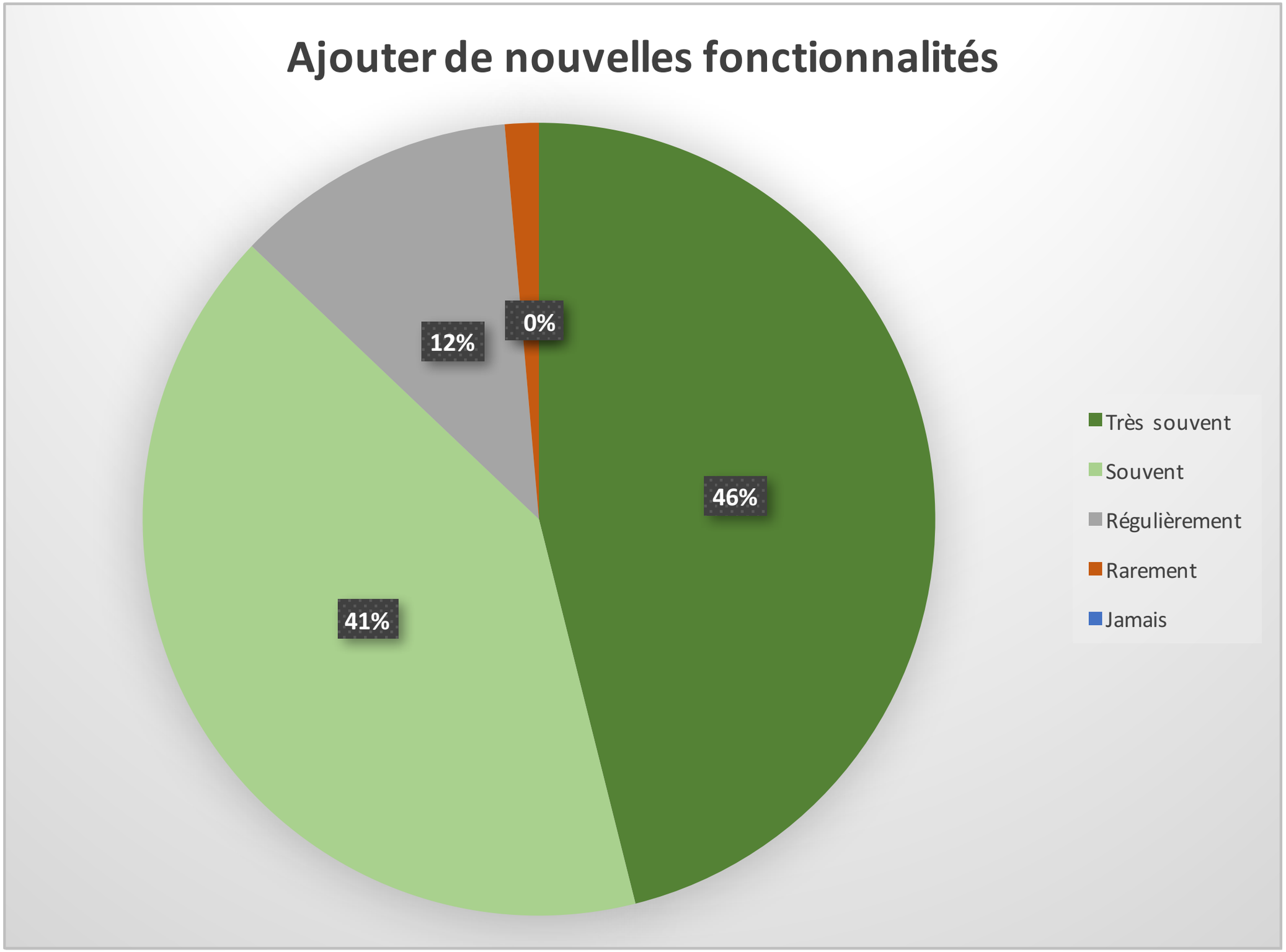
Vous travaillez dans une équipe (217 responses)



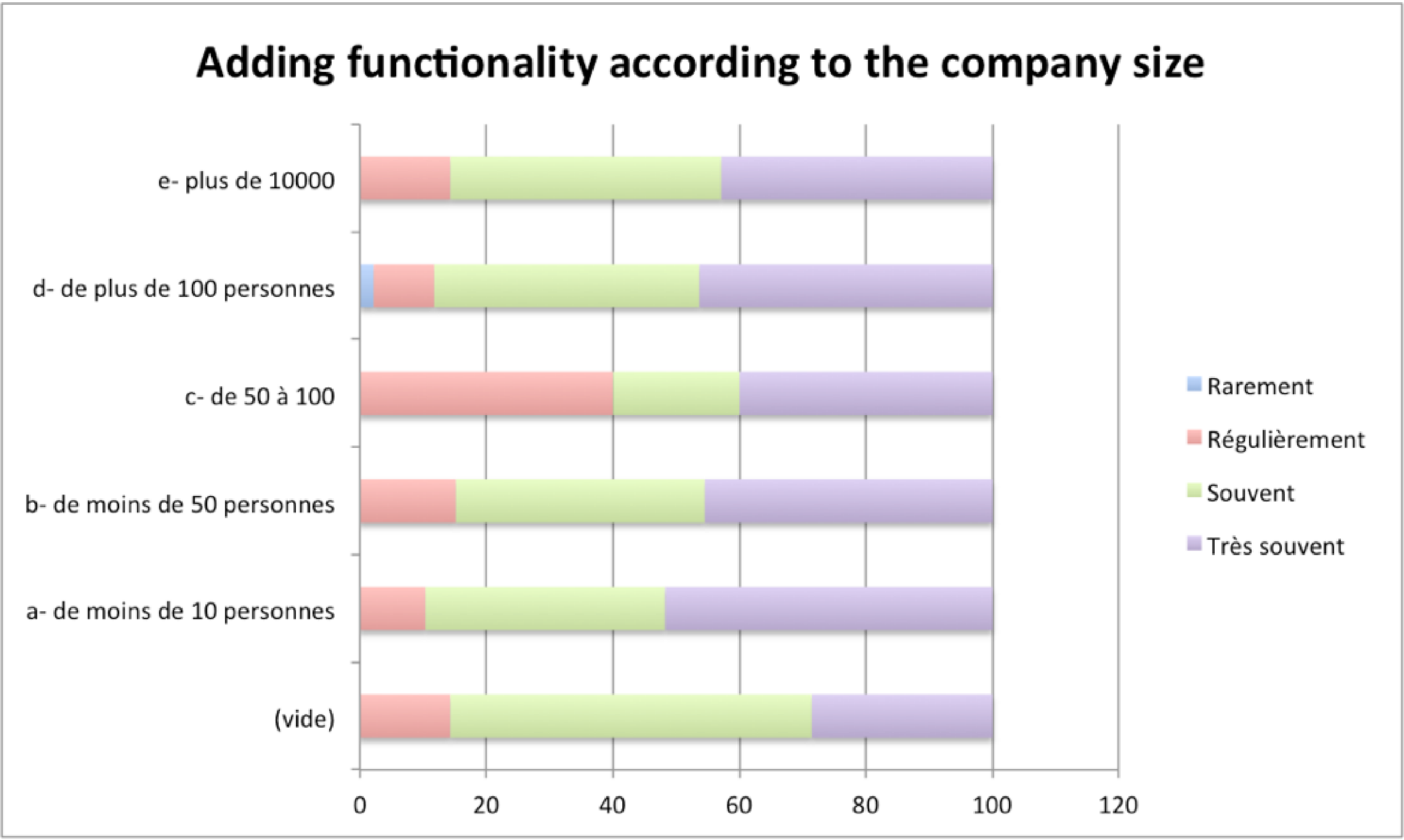
Quelle action de maintenance ?

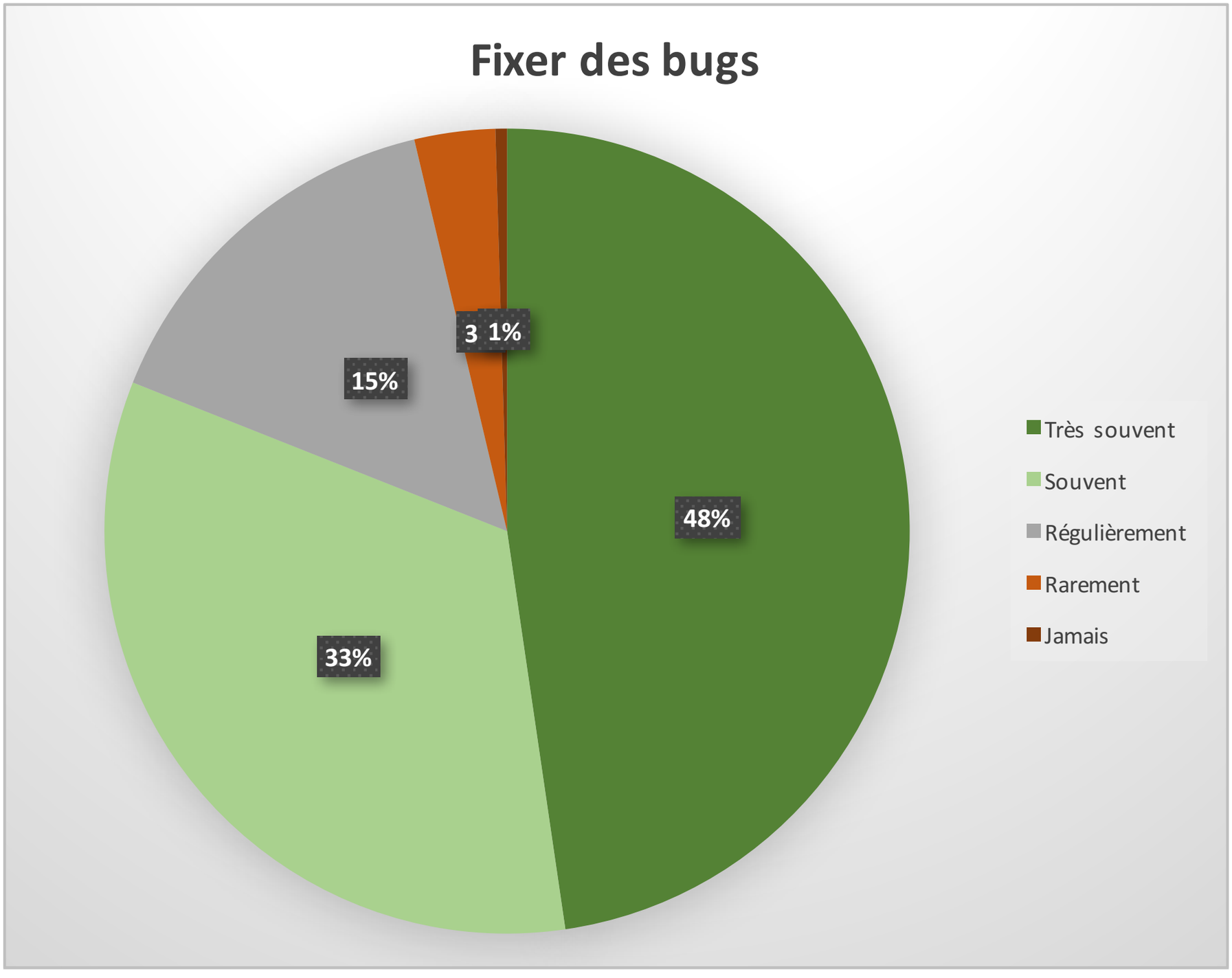


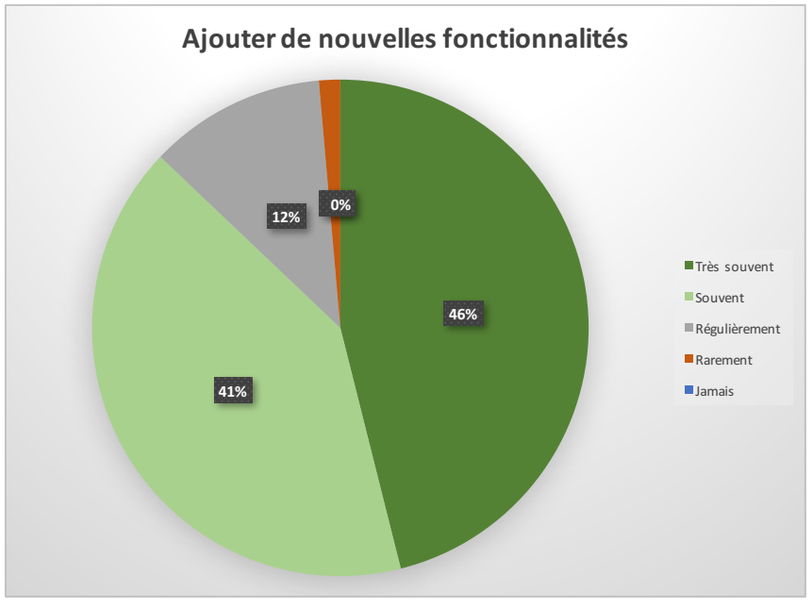
Vous modifiez votre code pour :

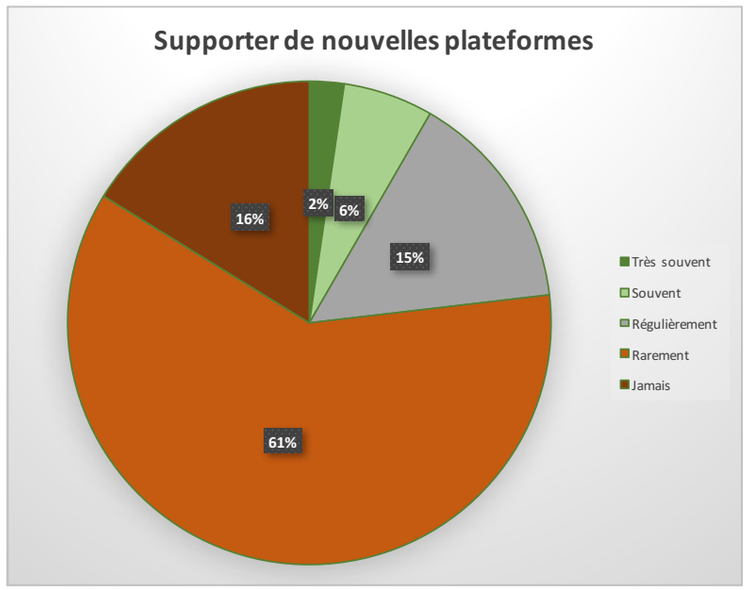


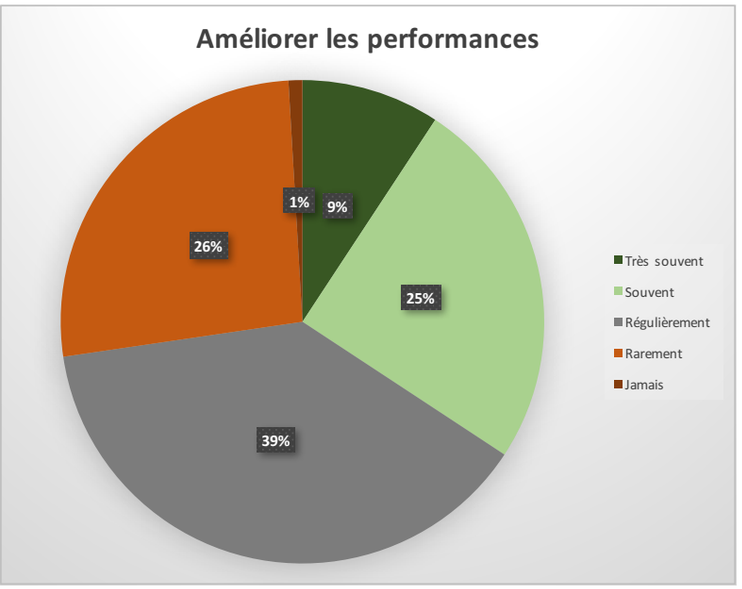
« we will never be able TO PREDICT THE FUTURE »

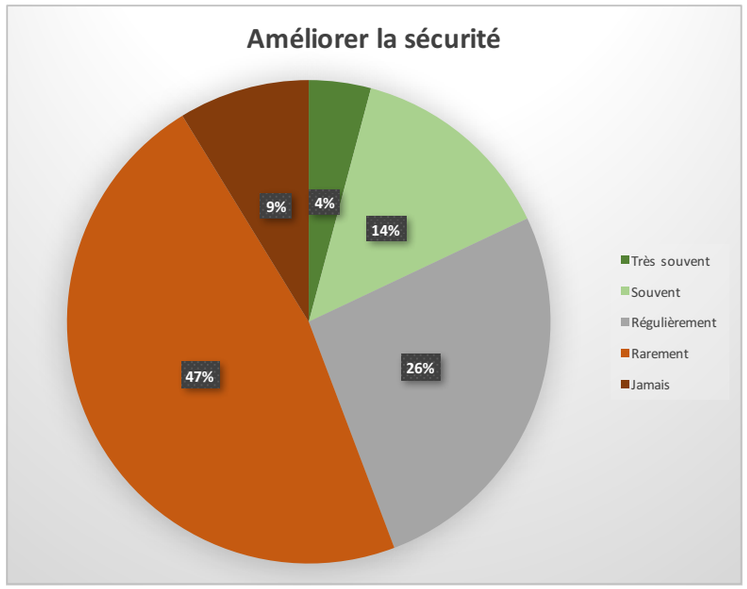

Adding functionality according to the company size
e- plus de 10000
d- de plus de 100 personnes
c- de 50 à 100
b- de moins de 50 personnes
a- de moins de 10 personnes
(vide)
0
20
40
60
80
100
120
Rarement
Régulièrement
Souvent
Très souvent

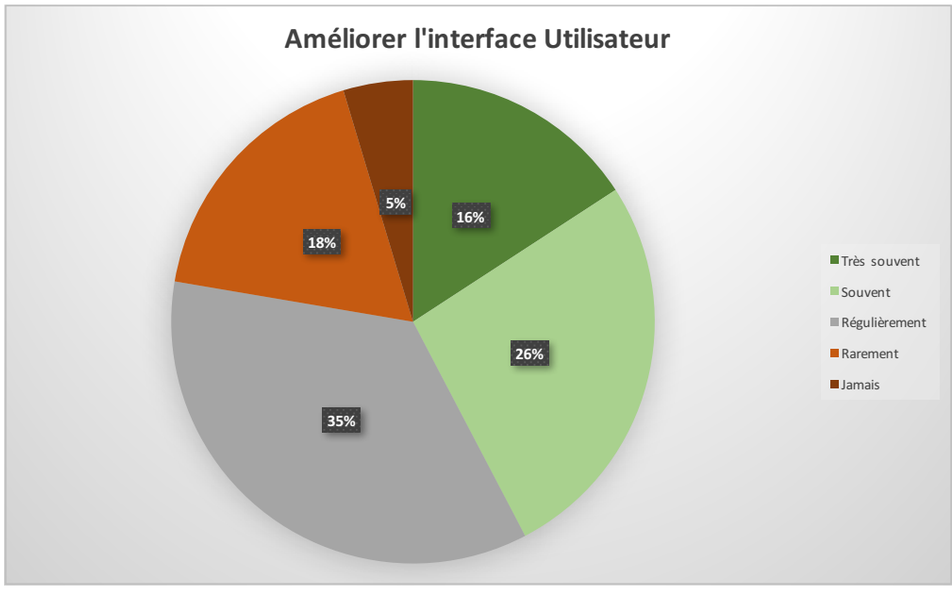

Fixer des bugs
48%
33%
15%
3 1%
Très souvent
Souvent
Régulièrement
Rarement
Jamais

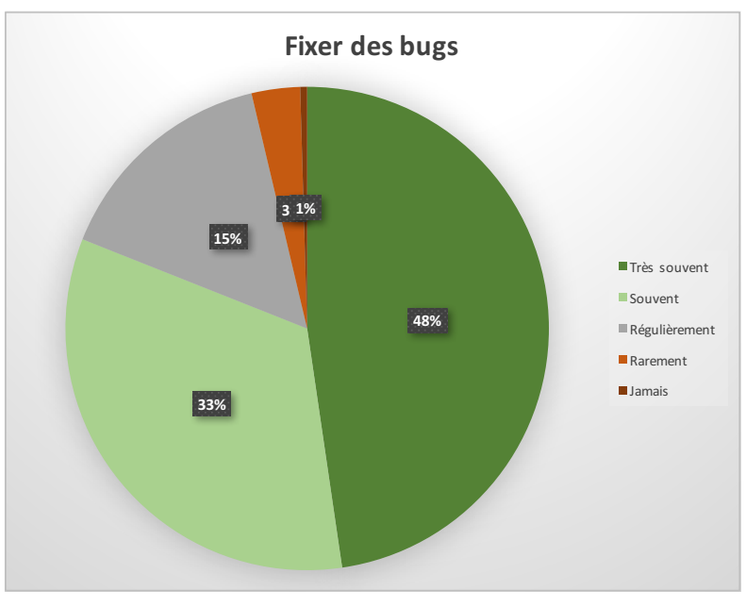

Ajouter de nouvelles fonctionnalités
46%
41%
12%
0%
Très souvent
Souvent
Régulièrement
Rarement
Jamais

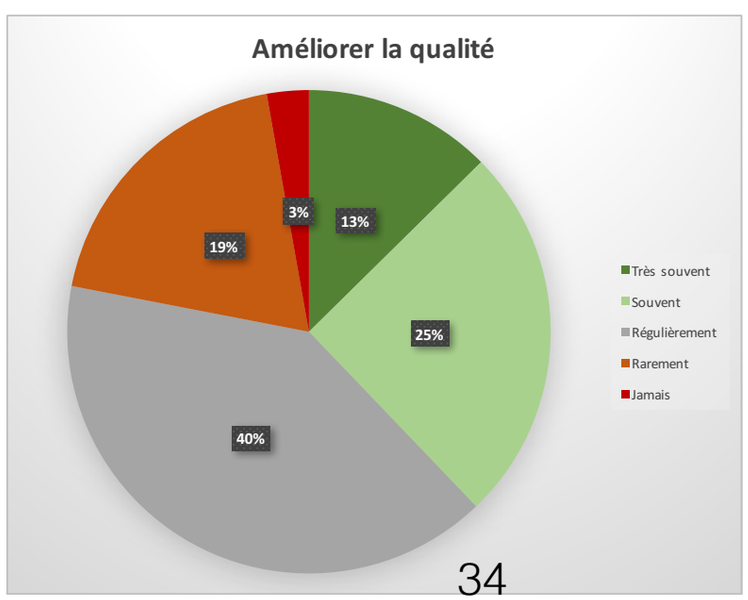

Supporter de nouvelles plateformes
2%
6%
15%
61%
16%
Très souvent
Souvent
Régulièrement
Rarement
Jamais


Améliorer les performances
1%
9%
25%
39%
26%
Très souvent
Souvent
Régulièrement
Rarement
Jamais


Améliorer la sécurité
9%
4%
14%
47%
26%
Très souvent
Souvent
Régulièrement
Rarement
Jamais


Améliorer l'interface Utilisateur
5%
16%
18%
26%
35%
Très souvent
Souvent
Régulièrement
Rarement
Jamais

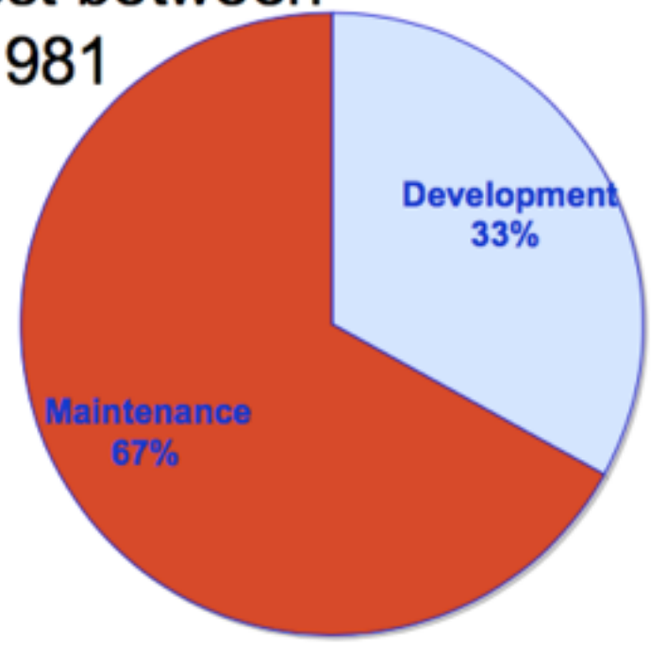

Fixer des bugs
3 1%
15%
48%
33%
Très souvent
Souvent
Régulièrement
Rarement
Jamais


Améliorer la qualité
3%
13%
19%
25%
40%
Très souvent
Souvent
Régulièrement
Rarement
Jamais

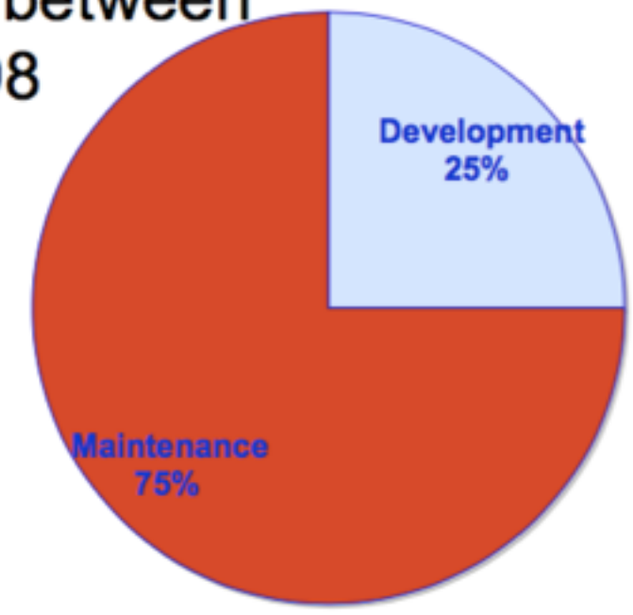
Changing software is costly

(Schach, 2008): Most of the effort and cost is spent on post-delivery maintenance based on various data sources

Average cost between 1976 and 1981



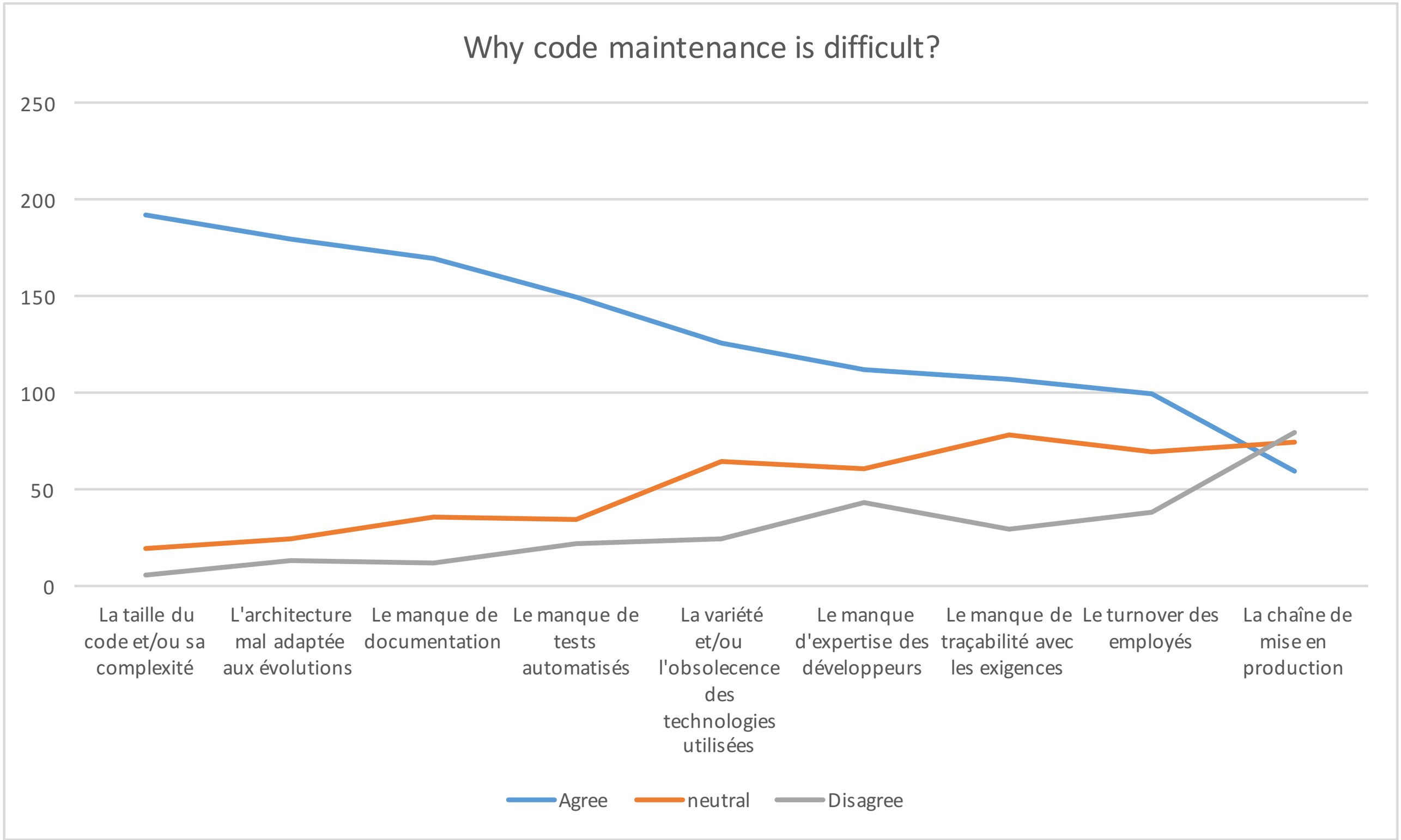
Average cost between 1992 and 1998



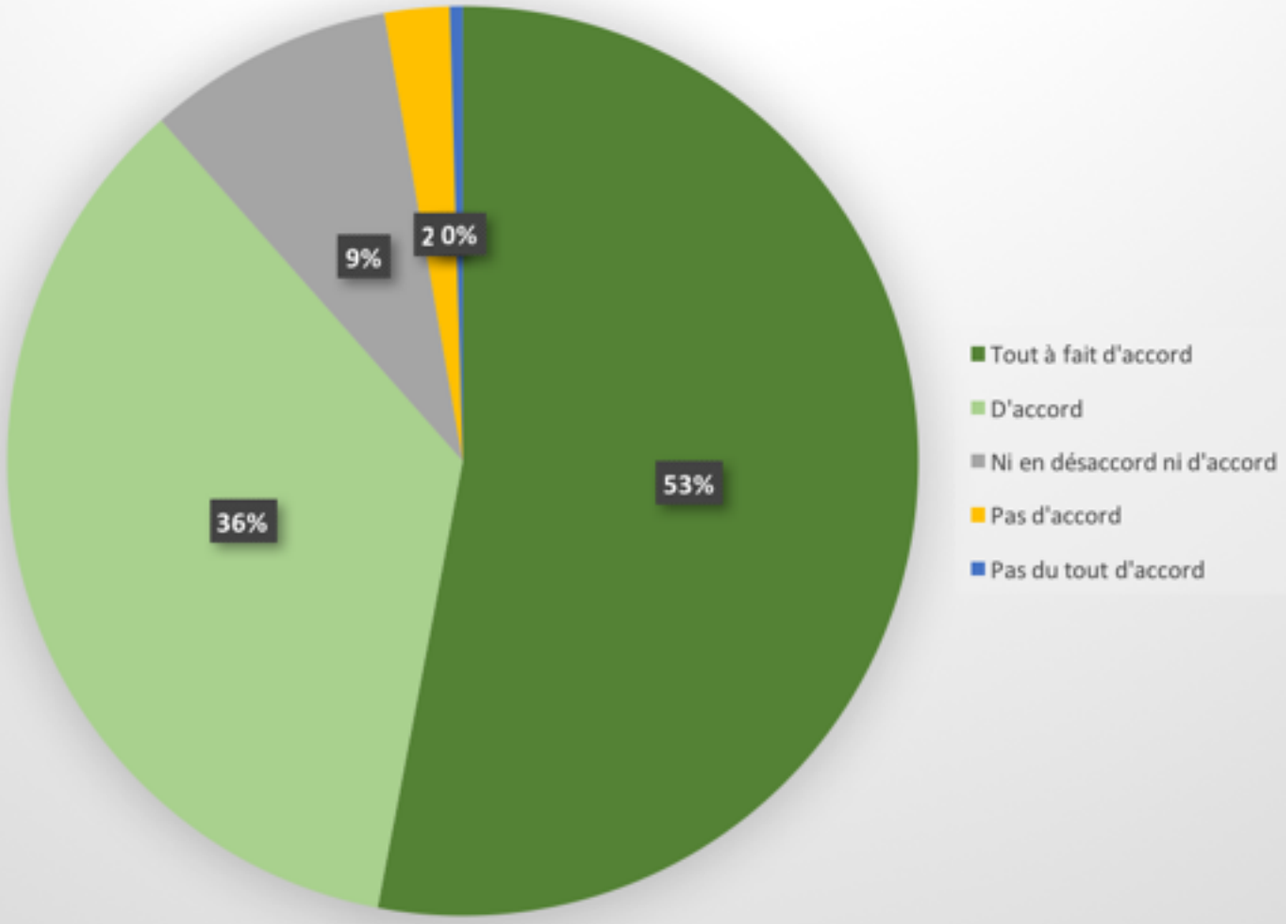
(Erlikh, 2000, IT Pro) "Leveraging legacy system dollars for E-business"
more than 90% of companies resources dedicated to software maintenance

http://www.slideshare.net/tommens/future-research-challenges-in-software-evolution?from_action=save

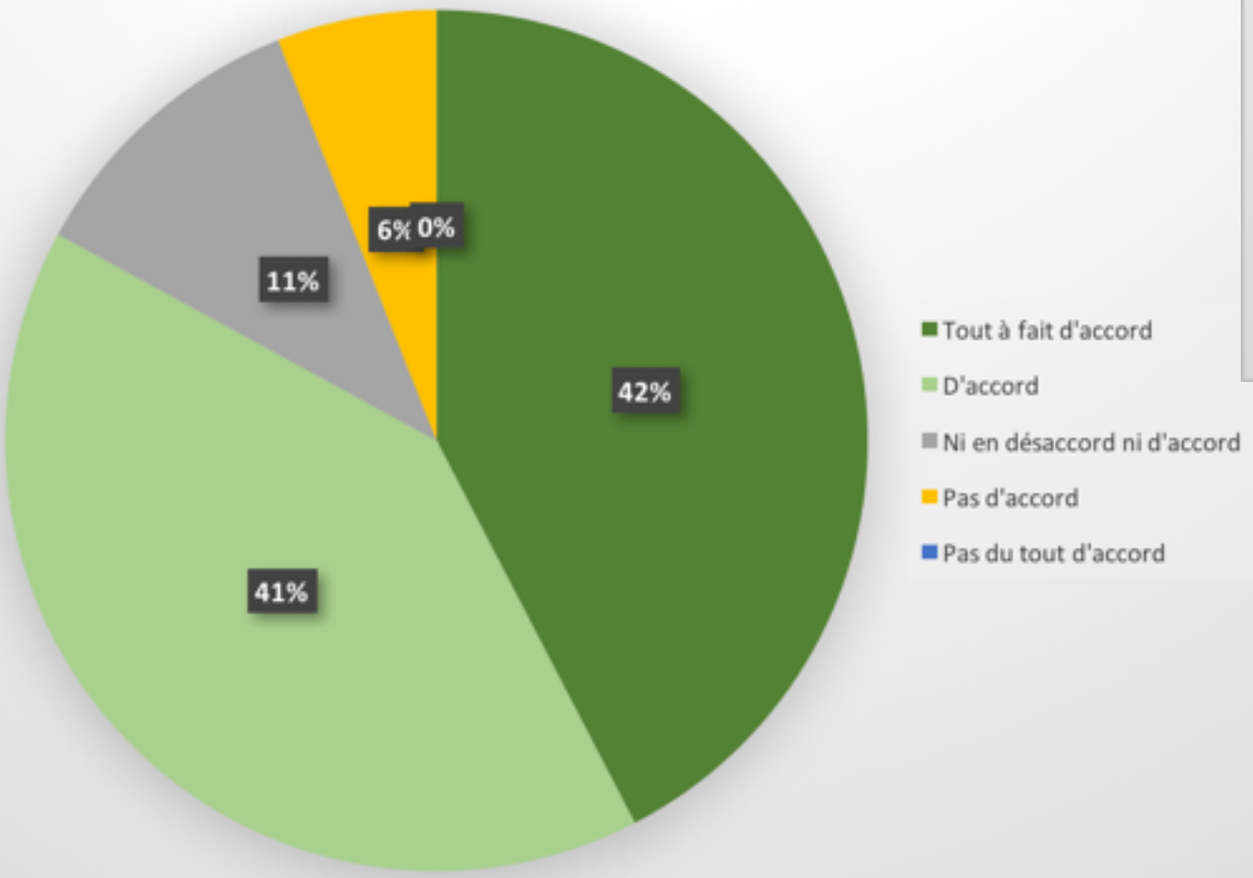
Pourquoi est-ce difficile ?

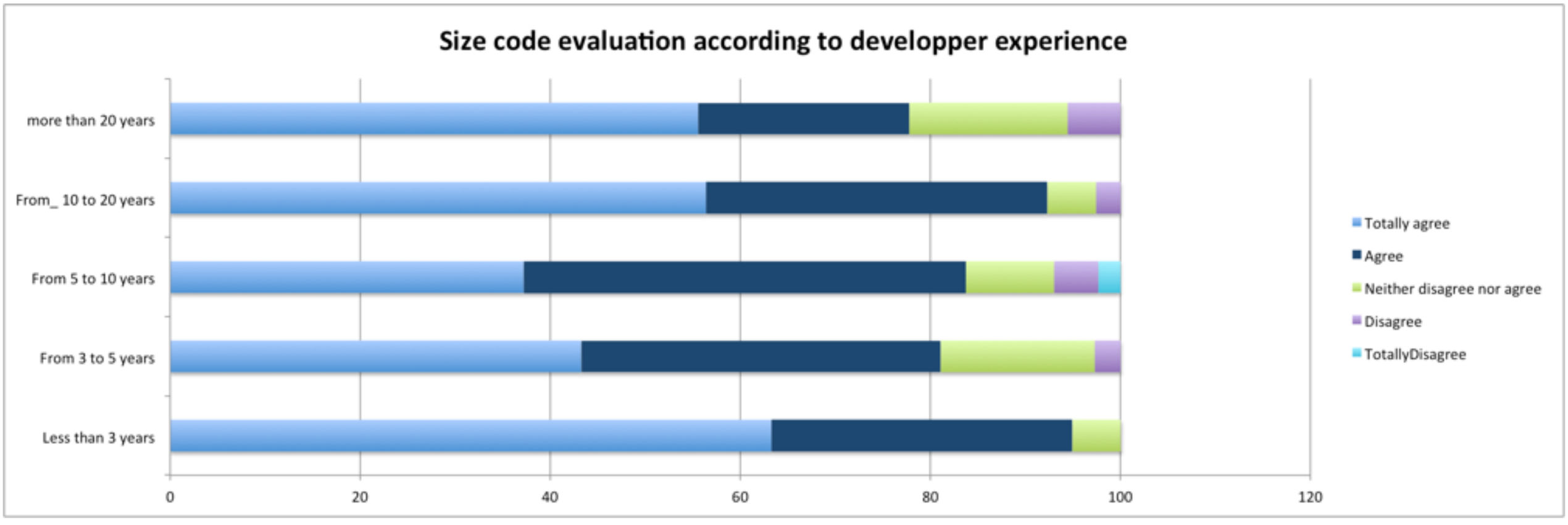

Why code maintenance is difficult?
250
200
150
100
50
0
La taille du code et/ou sa complexité
L'architecture mal adaptée aux évolutions
Le manque de documentation
Le manque de tests automatisés
La variété et/ou l'obsolecence des technologies utilisées
Le manque d'expertise des développeurs
Le manque de traçabilité avec les exigences
Le turnover des employés
La chaîne de mise en production
Agree
neutral
Disagree

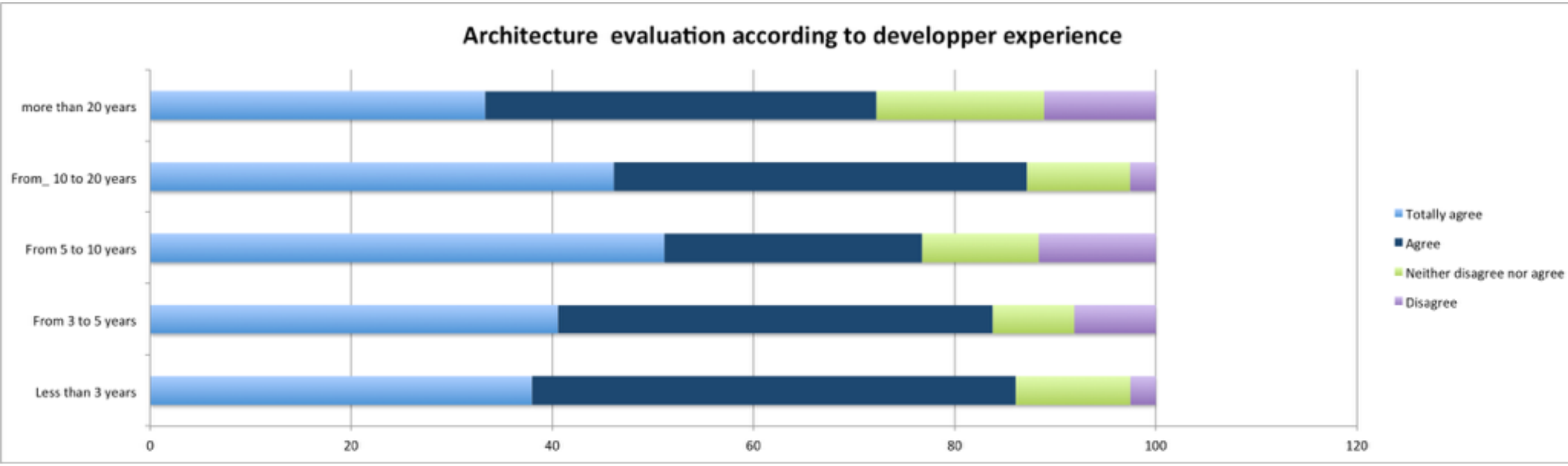
La taille du code et/ou sa complexité



L'architecture mal adaptée...

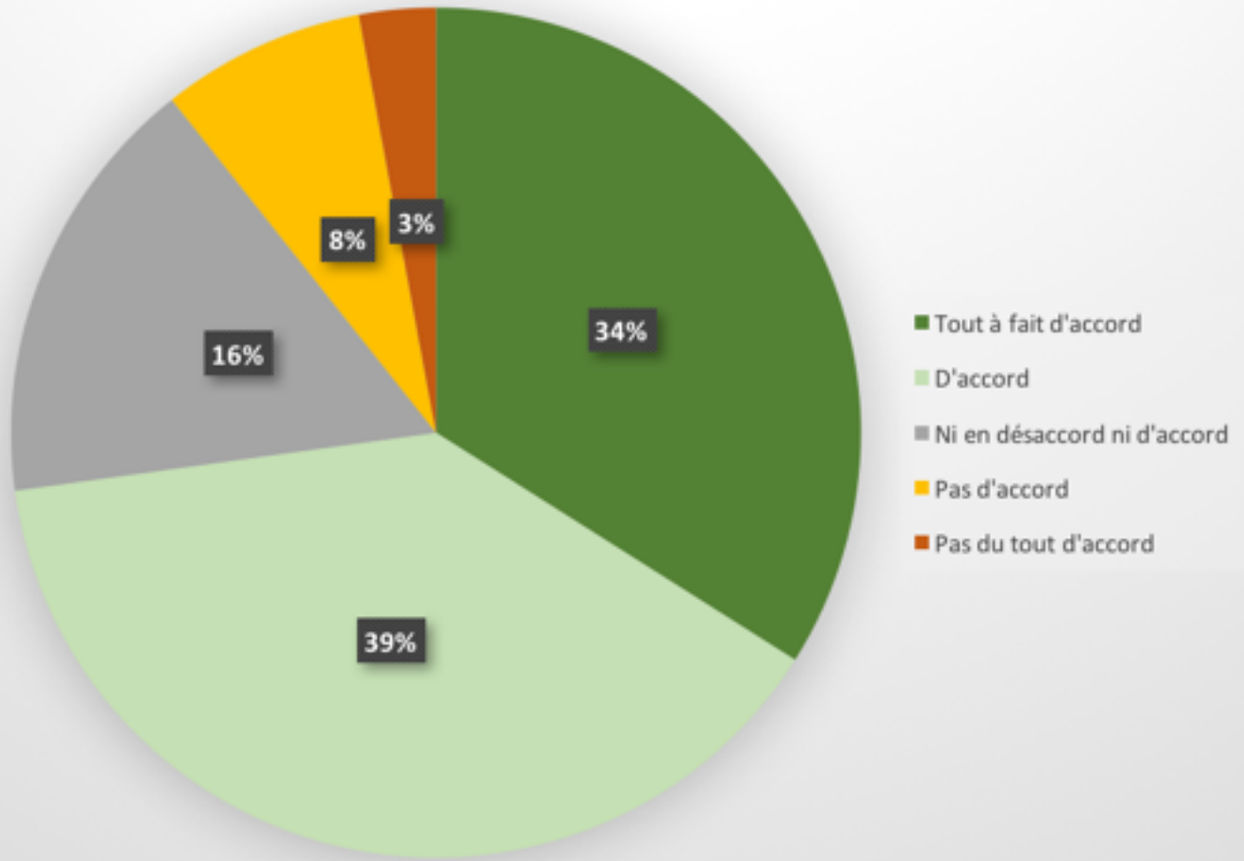


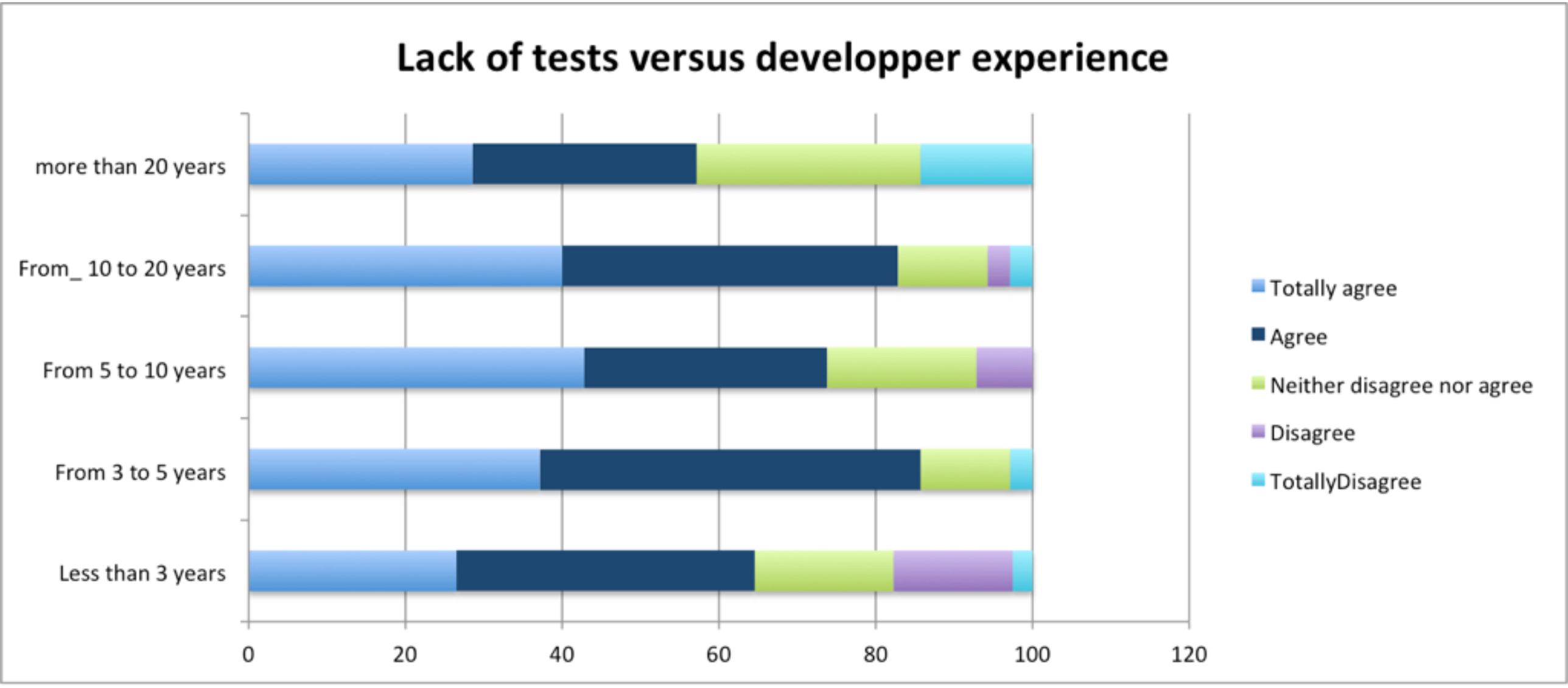

Size code evaluation according to developper experience
more than 20 years
From_ 10 to 20 years
From 5 to 10 years
From 3 to 5 years
Less than 3 years
0
20
40
60
80
100
120
Totally agree
Agree
Neither disagree nor agree
Disagree
TotallyDisagree

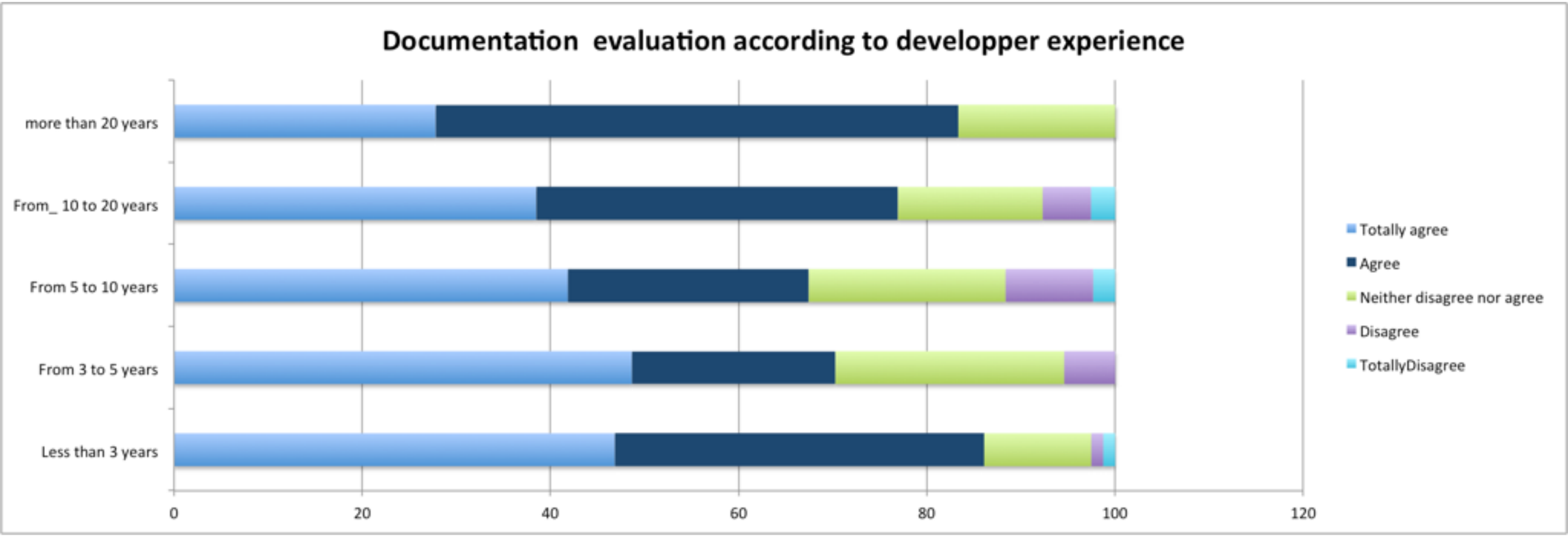

Architecture evaluation according to developper experience
more than 20 years
From_ 10 to 20 years
From 5 to 10 years
From 3 to 5 years
Less than 3 years
0
20
40
60
80
100
120
Totally agree
Agree
Neither disagree nor agree
Disagree

Le manque de documentation

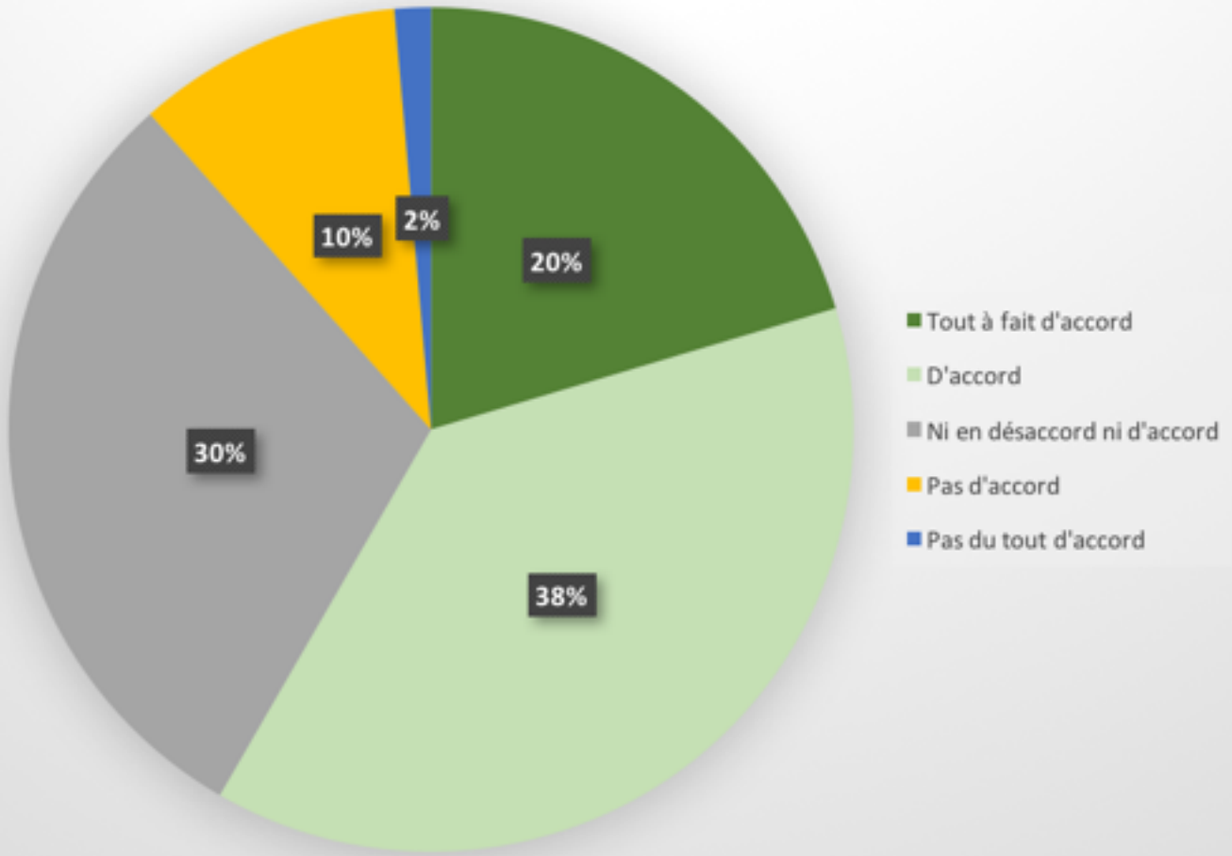
Le manque de tests automatisés



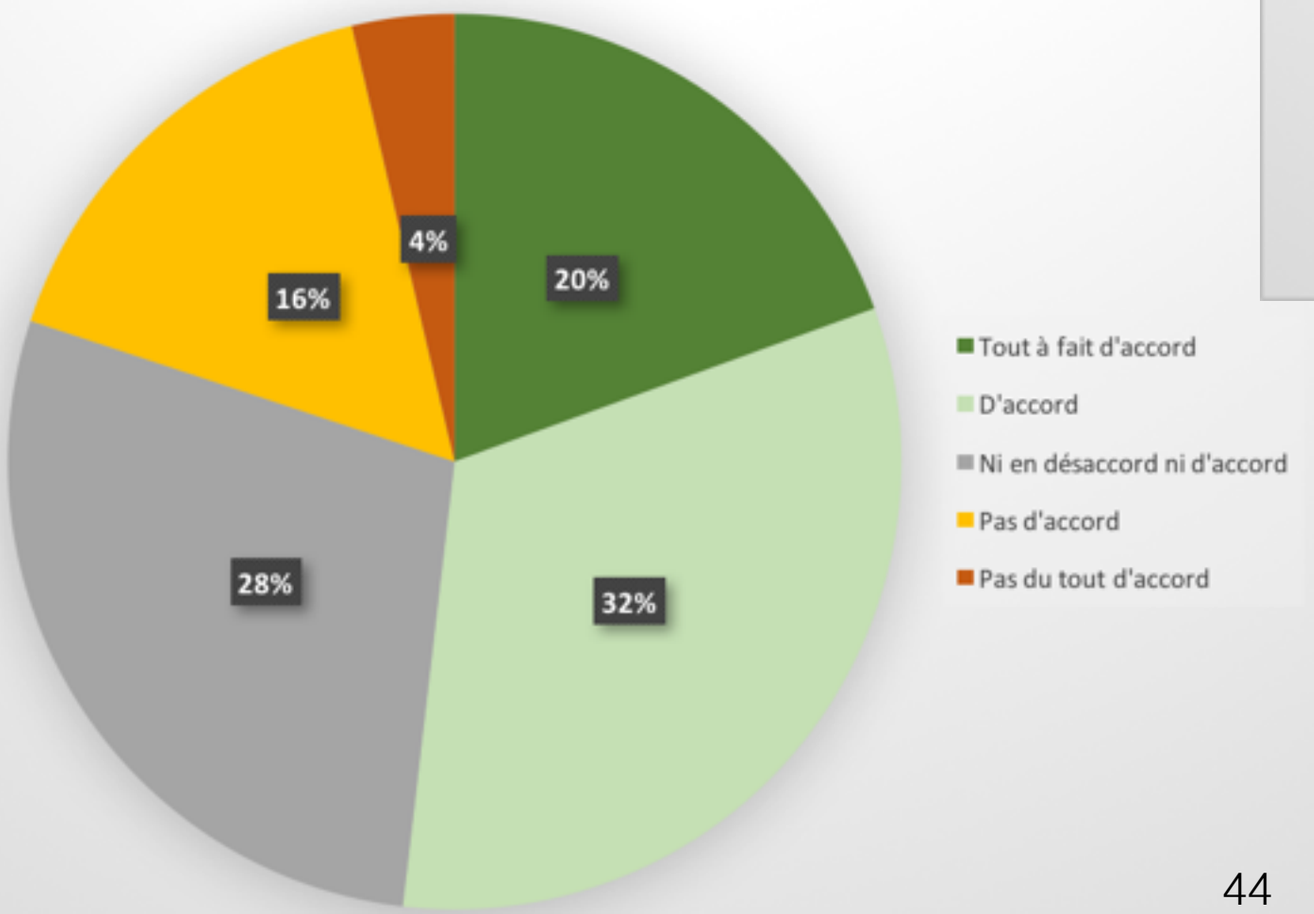

Lack of tests versus developper experience
more than 20 years
From_ 10 to 20 years
From 5 to 10 years
From 3 to 5 years
Less than 3 years
0
20
40
60
80
100
120
Totally agree
Agree
Neither disagree nor agree
Disagree
TotallyDisagree


Documentation evaluation according to developper experience
more than 20 years
From_ 10 to 20 years
From 5 to 10 years
From 3 to 5 years
Less than 3 years
0
20
40
60
80
100
120
Totally agree
Agree
Neither disagree nor agree
Disagree
TotallyDisagree

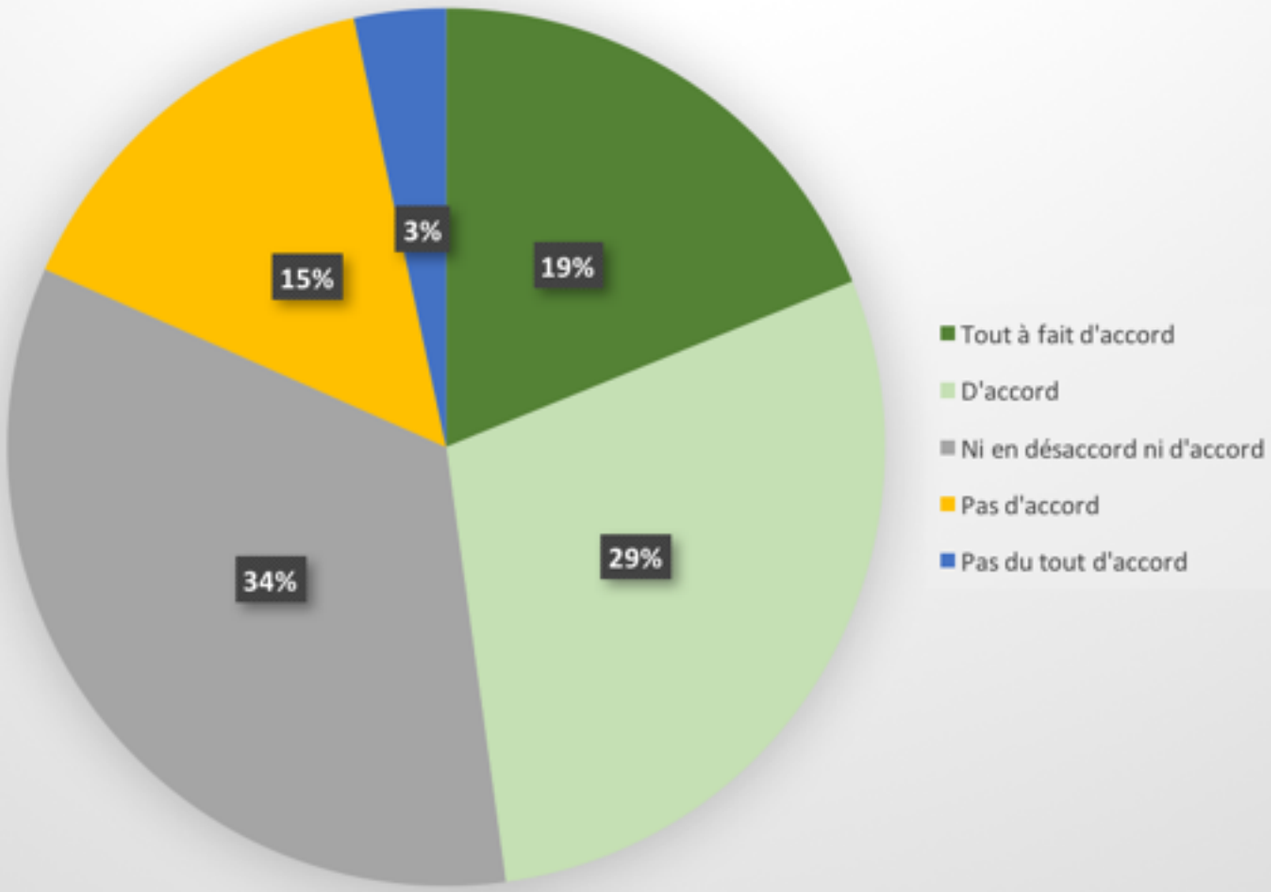
La variété et ou l'obsciescence ...



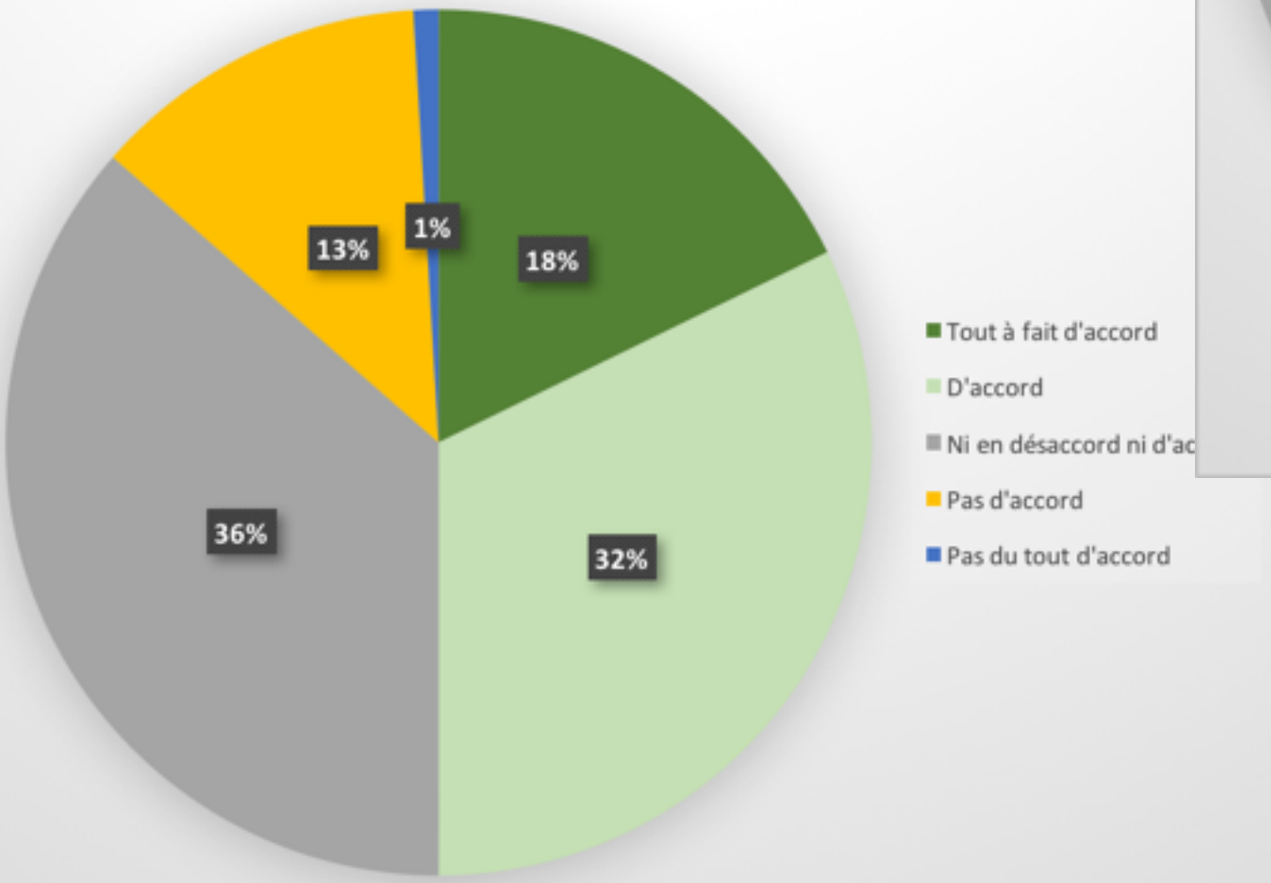
Le manque d'expertise des développeurs



Le turnover des développeurs



Le manque de tracabilité des exigences



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

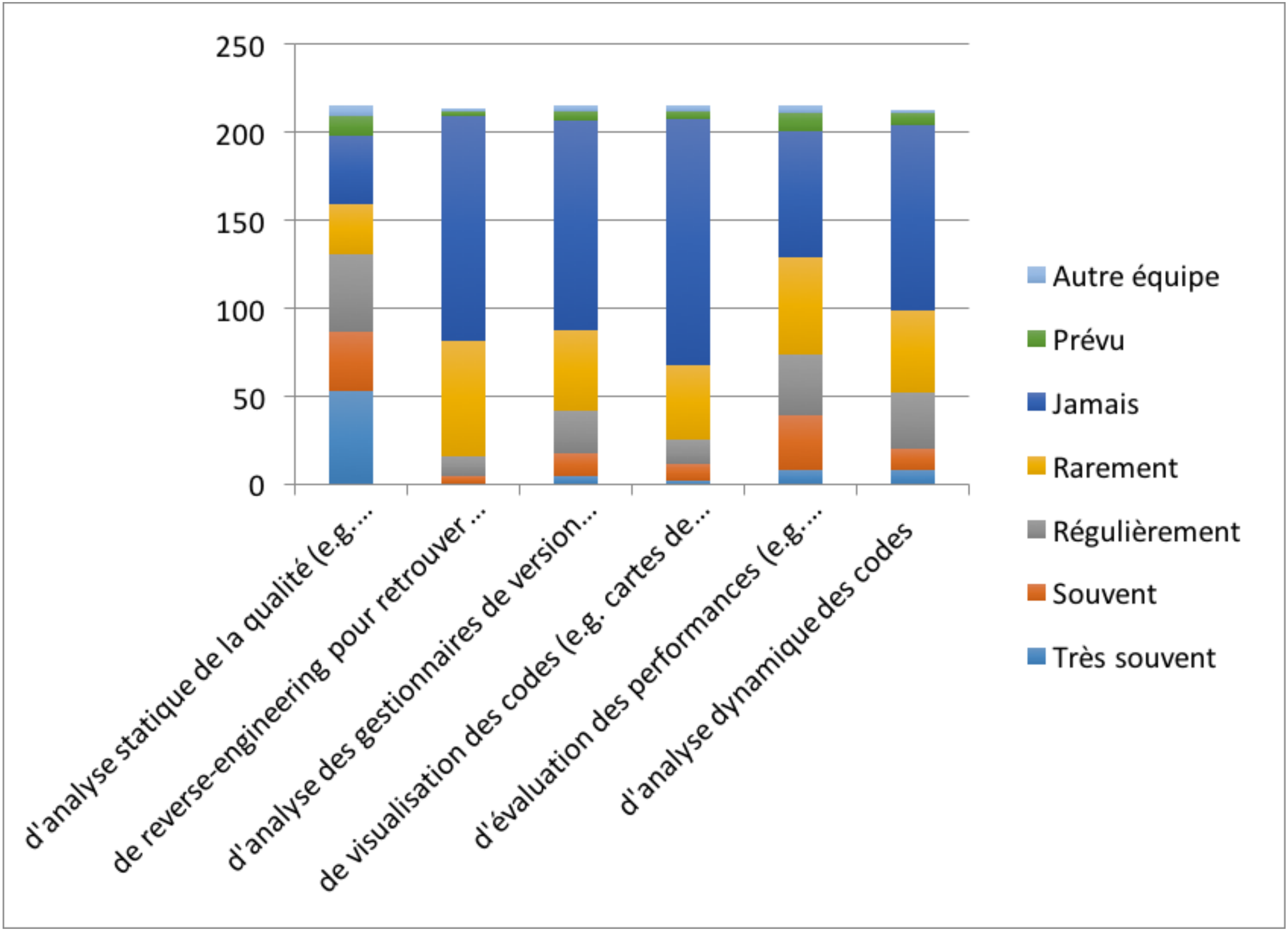
Architectural Problems

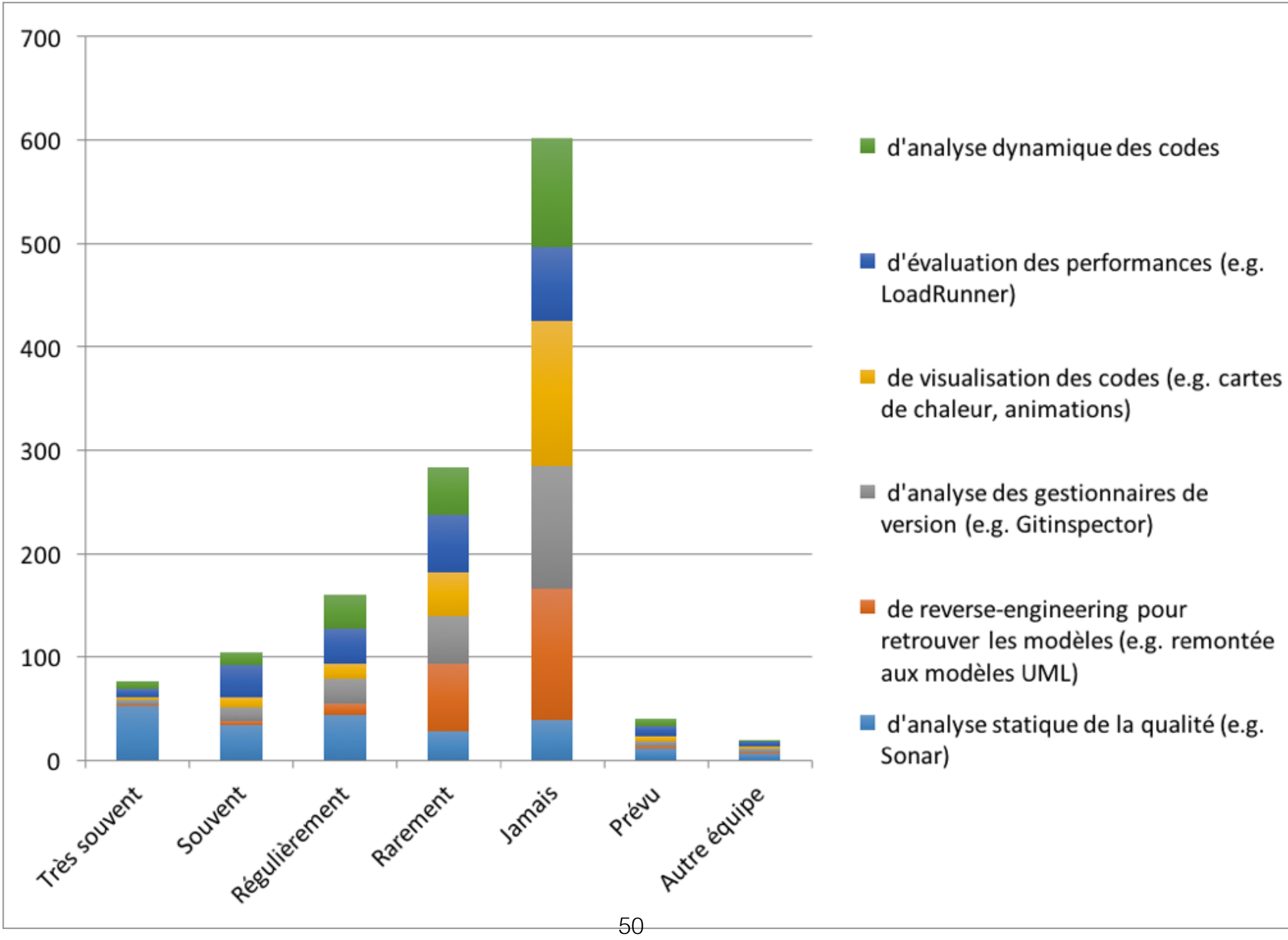
- insufficient *documentation*
  = non-existent or out-of-date
- improper *layering*
  = too few or too many layers
- lack of *modularity*
  = strong coupling
- *duplicated code*
  = copy, paste & edit code
- duplicated *functionality*
  = similar functionality
  by separate teams

Refactoring opportunities

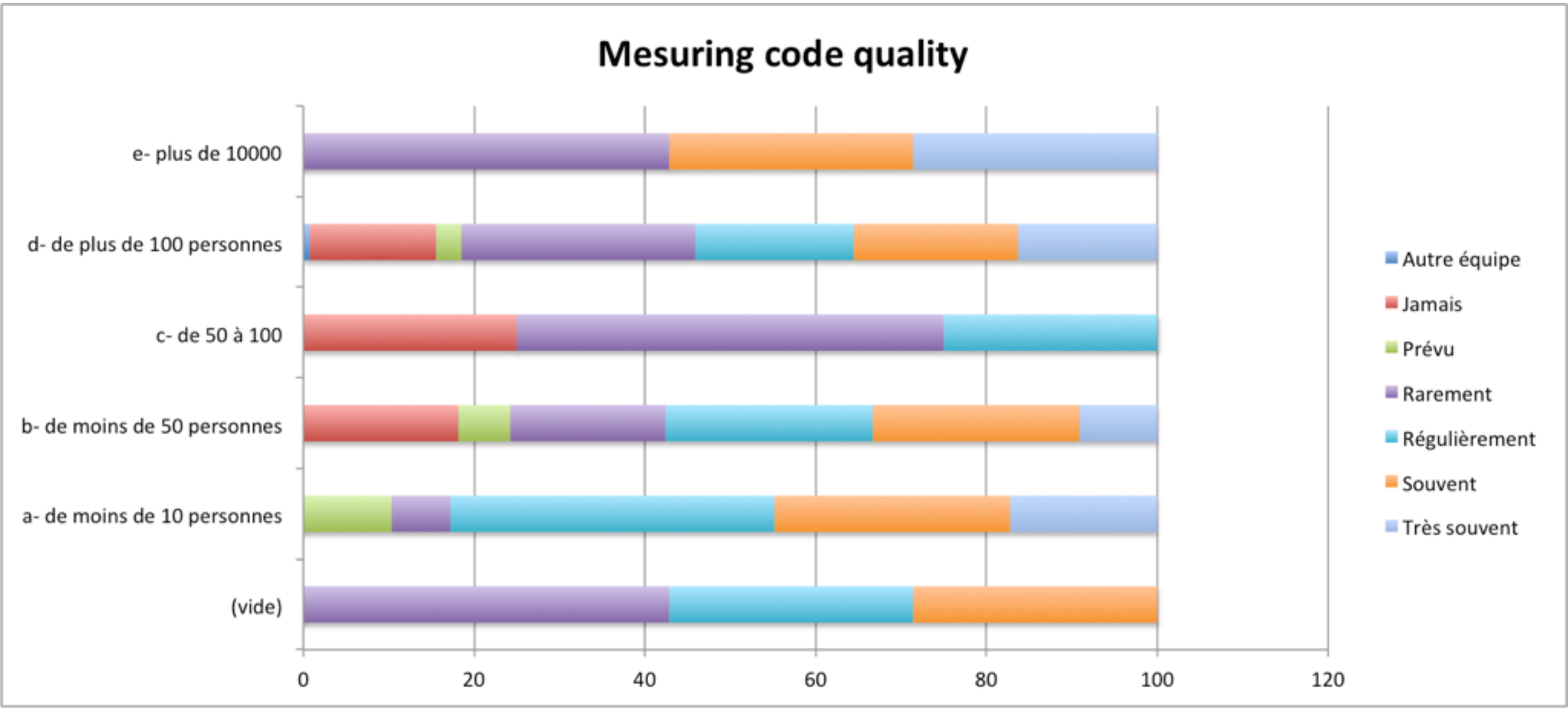
- *misuse* of inheritance
  = code reuse vs polymorphism
- *missing* inheritance
  = duplication, case-statements
- *misplaced* operations
  = operations outside classes
- *violation* of encapsulation
  = type-casting; C++ "friends"
- *class abuse*
  = classes as namespaces

Les outils que vous utilisez, servent à :


250
200
150
100
50
0
Autre équipe
Prévu
Jamais
Rarement
Régulièrement
Souvent
Très souvent
d'analyse statique de la qualité (e.g....
de reverse-engineering pour retrouver ...
d'analyse des gestionnaires de version...
de visualisation des codes (e.g. cartes de...
d'évaluation des performances (e.g....
d'analyse dynamique des codes

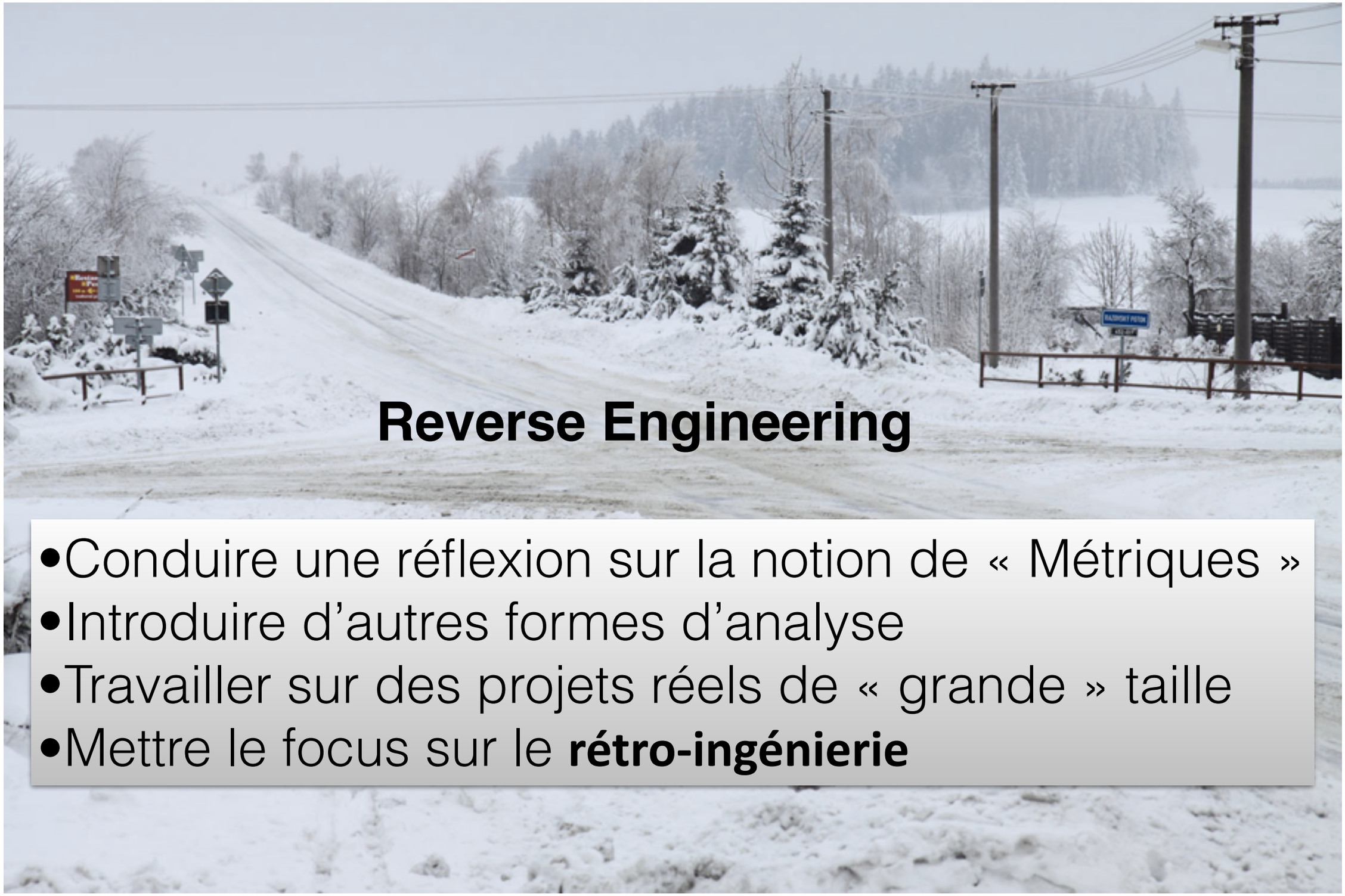

700
600
500
400
300
200
100
0
Très souvent
Souvent
Régulièrement
Rarement
Jamais
Prévu
Autre équipe
d'analyse dynamique des codes
d'évaluation des performances (e.g. LoadRunner)
de visualisation des codes (e.g. cartes de chaleur, animations)
d'analyse des gestionnaires de version (e.g. Gitinspector)
de reverse-engineering pour retrouver les modèles (e.g. remontée aux modèles UML)
d'analyse statique de la qualité (e.g. Sonar)

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.



Tacking a step back

Le module

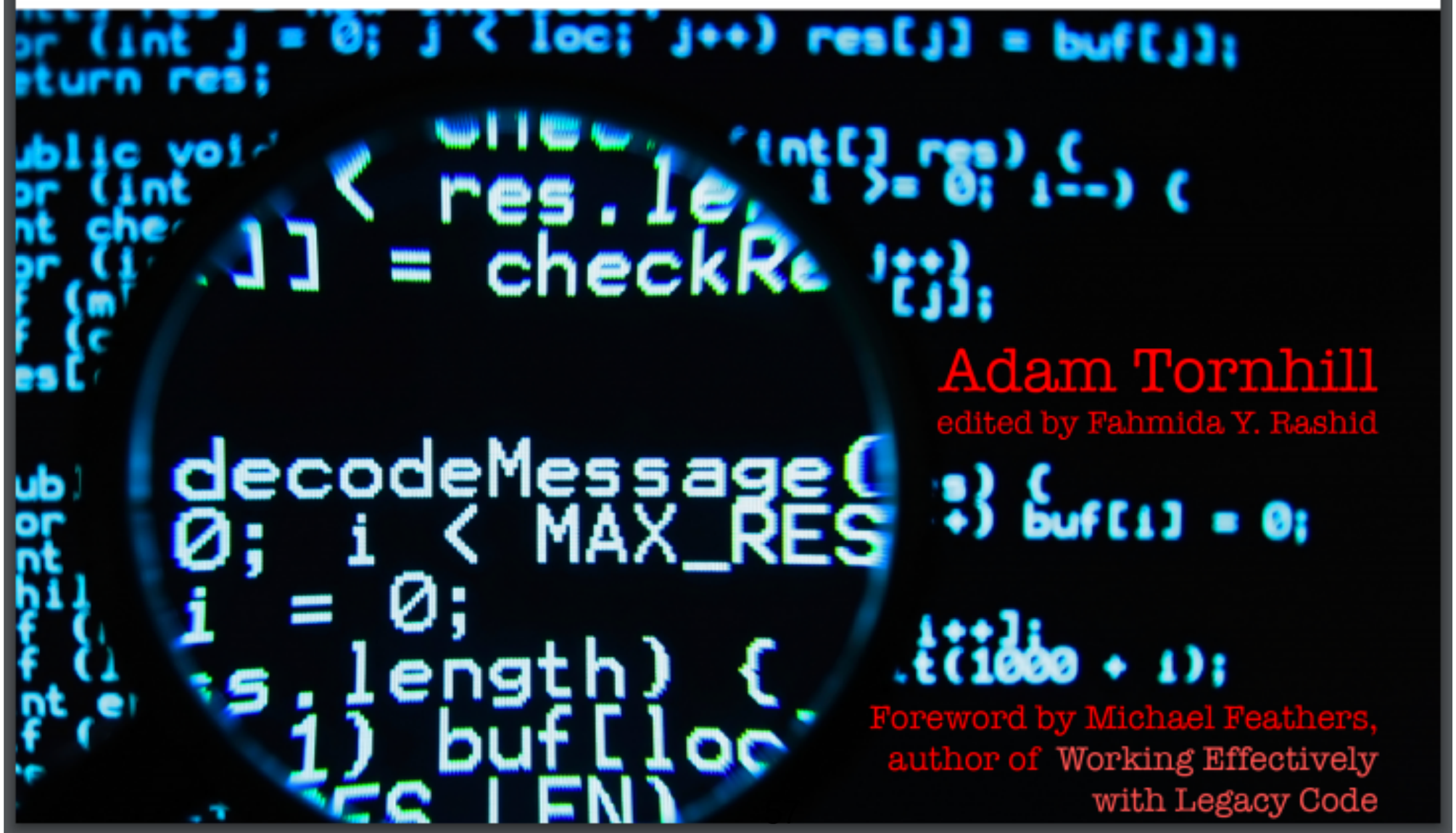
Plan du module

1. mar.19 déc. 2017
   - *08:00 – 10:00* : Cours - Introduction *(MBF)*
   - *10:00 - 11h00* : Autonomie
   - *11:15 – 12:15* : TD - Choix et caractérisation du sujet d'étude *(MBF,SM,PC)*
   - lundi 8 janv. 2018 à 15h au plus tard **Livrable L.1**
2. mar.9 janv. 2018
   - *8:00 – 9:00* : TD - Compléments sur le sujet en mode “coaching” *(MBF,SM,PC)*
   - *9:00 – 10:00* : TD - Compléments sur le sujet en mode “coaching” *(MBF)*
   - *10:00 – 12:15* : Autonomie
3. mar.16 janv. 2018
   - *08:00 – 09:30* : Cours - Comprendre un logiciel en regardant son histoire *Xavier Blanc(XB)*
   - *09:45 – 12:15* : TD - Validations Métrics/KPI *(MBF,XB)*
4. mar.23 janv. 2018
   - *08:00 – 11:00* : Oral (10mn exposé + 10mn questions) (MBF,PC) **Exposé E.1**
5. mar.30 janv. 2018
   - *09:45 – 12:15* : Autonomie
6. mar.6 févr. 2018
   - *08:00 – 10:00* : TD - Travail sur les articles sélectionnés par les étudiants, démarche & Métrics/KPI *(SM,PC)*
   - *09:45 – 12:15* : Autonomie
   - lundi 12 février 18h **Livrable L.2**
7. mar.13 févr. 2018
   - *08:00 – 9:00* : Intervention d'un industriel
8. mar.20 févr. 2018
   - Annonce de la sélection des articles utilisés pour l'examen
   - *08:00 – 11:00* : Autonomie
9. mar.27 févr. 2018
   - *08:00 – 11:00* : Examen **Livrable L.3 & Livrable 4**

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.

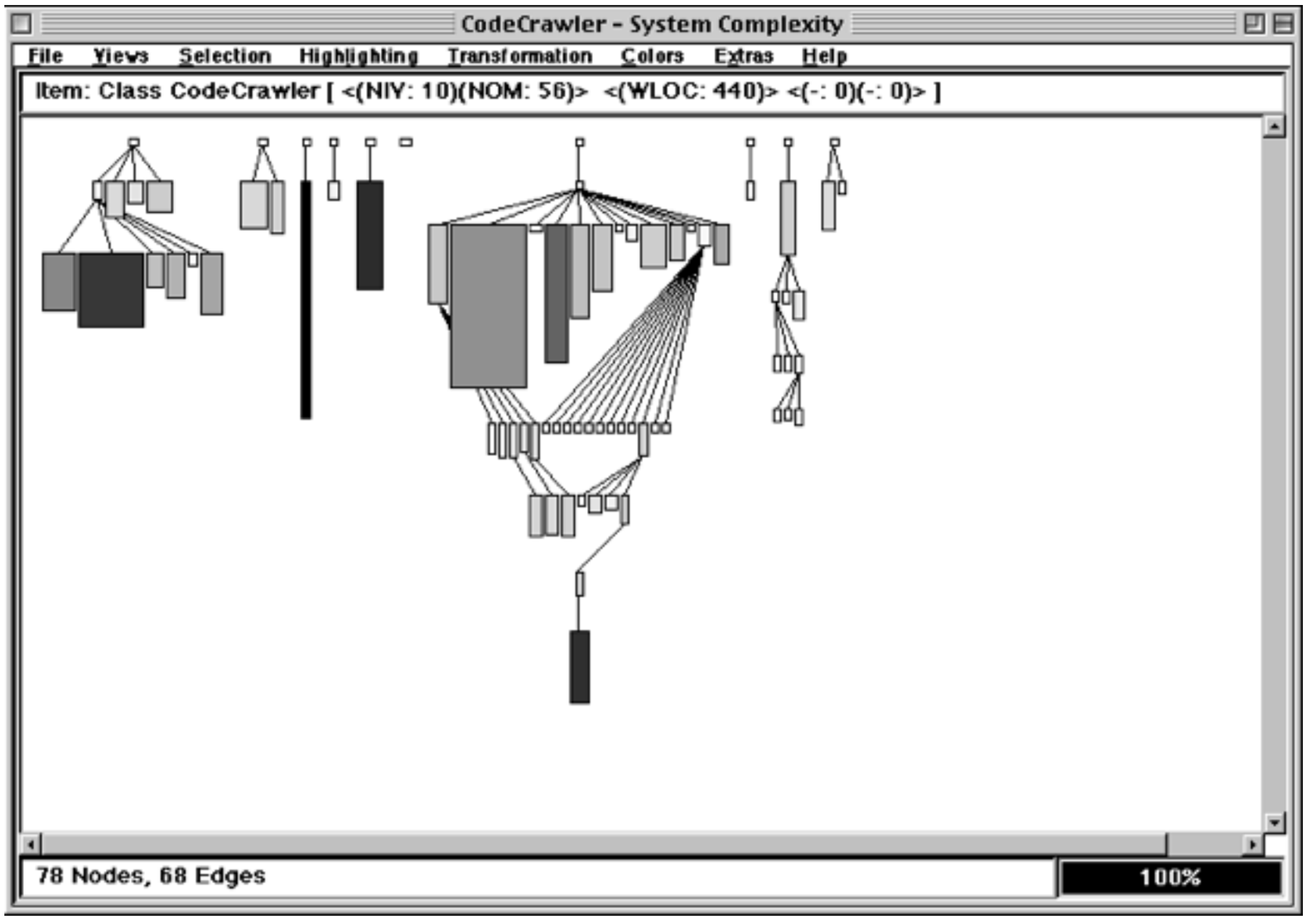
« Profiler » le code

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

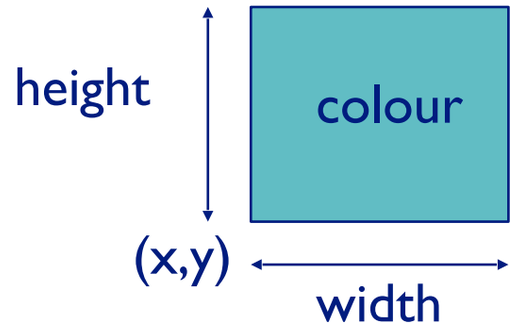
I took a course in speed reading and read "War and Peace" in twenty minutes. It's about Russia.

– Woody Allen

Visualizing Metrics



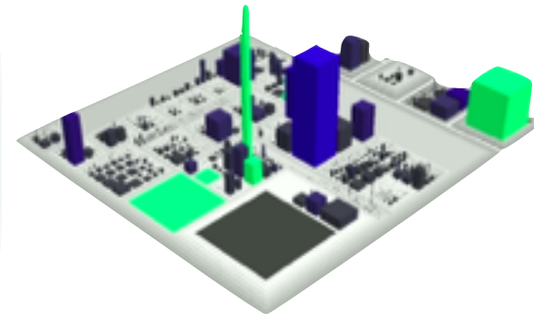
Use *simple* metrics and layout algorithms



Visualizes up to 5 metrics per node

« Profiler » le code

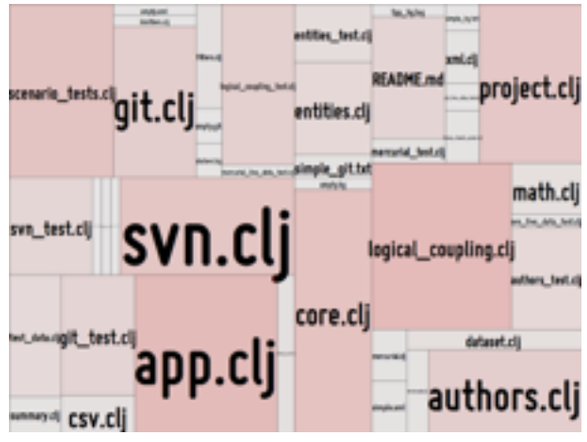
- Analyse « géographique » : Visualiser le code (codeCity)



- Analyse des changements : qui, quoi, quand (MetricsTreeMap) :

- Forte complexité et beaucoup de changements => un hotspot !

- Beaucoup de changements sont signes de problèmes (Hassan 09)



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) == inexplore){
      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 arriere 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+") arrie`re");
            etat_a.put(a, "arcarriere");
          }
          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 a partir de s1: " + chemins);
    Chemin chemin = chemins.get(0);
```

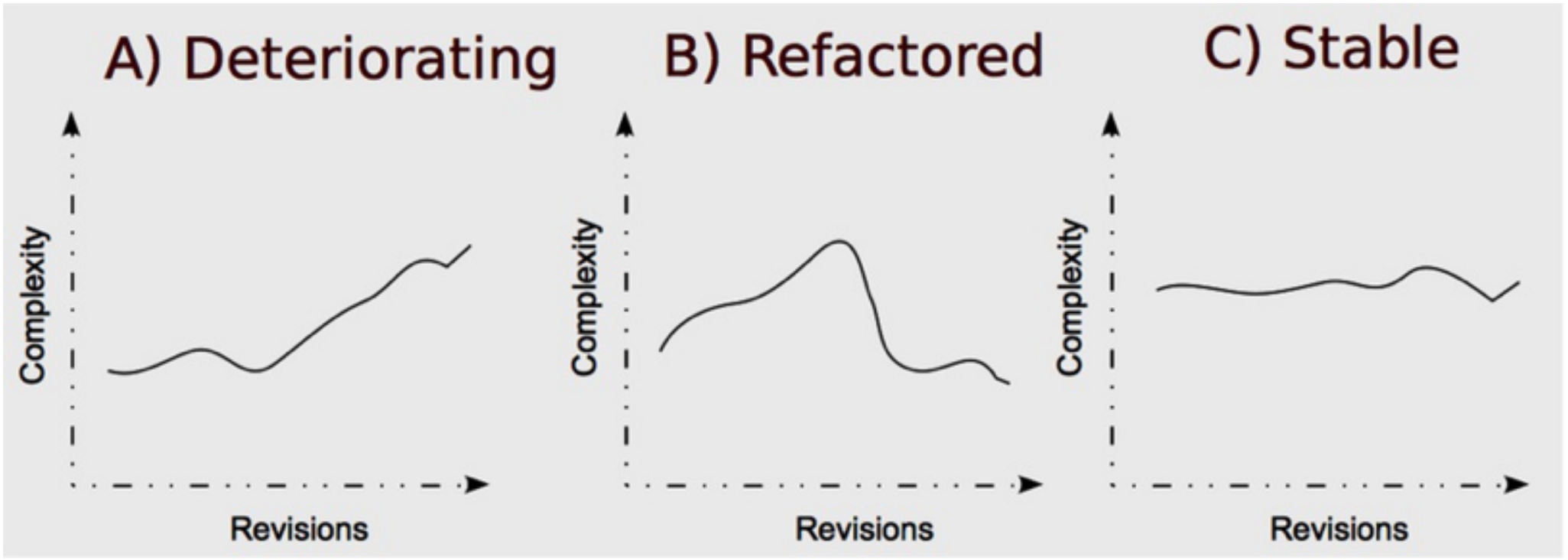
Code Complexity, d’un coup d’oeil?

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

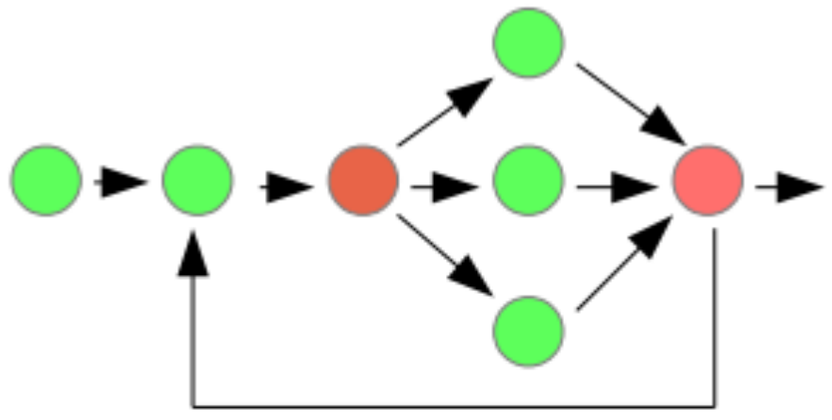
Evolution de la complexité du code

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

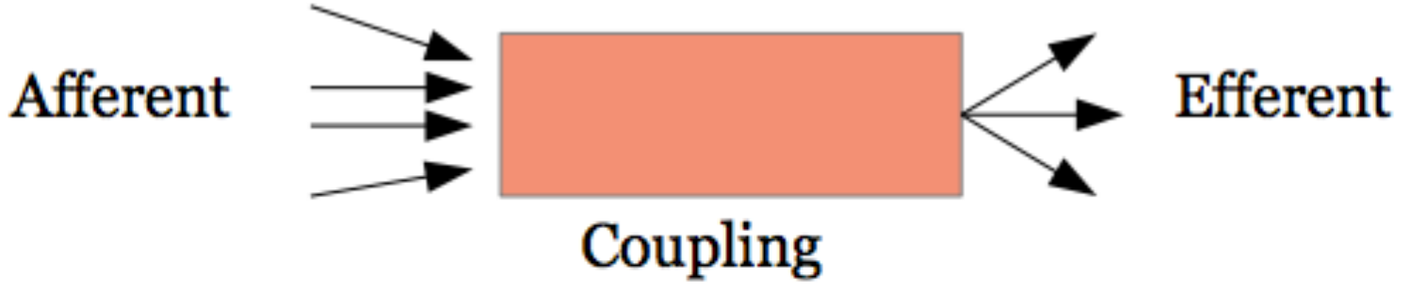
- ajout de code ou modification ? on calcule la moyenne et l'écart type : s'ils diminuent c'est plutôt bon signe.



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)

https://www.jfokus.se/jfokus14/preso/Code-as-a-crime-scene.pdf

Metrics ?

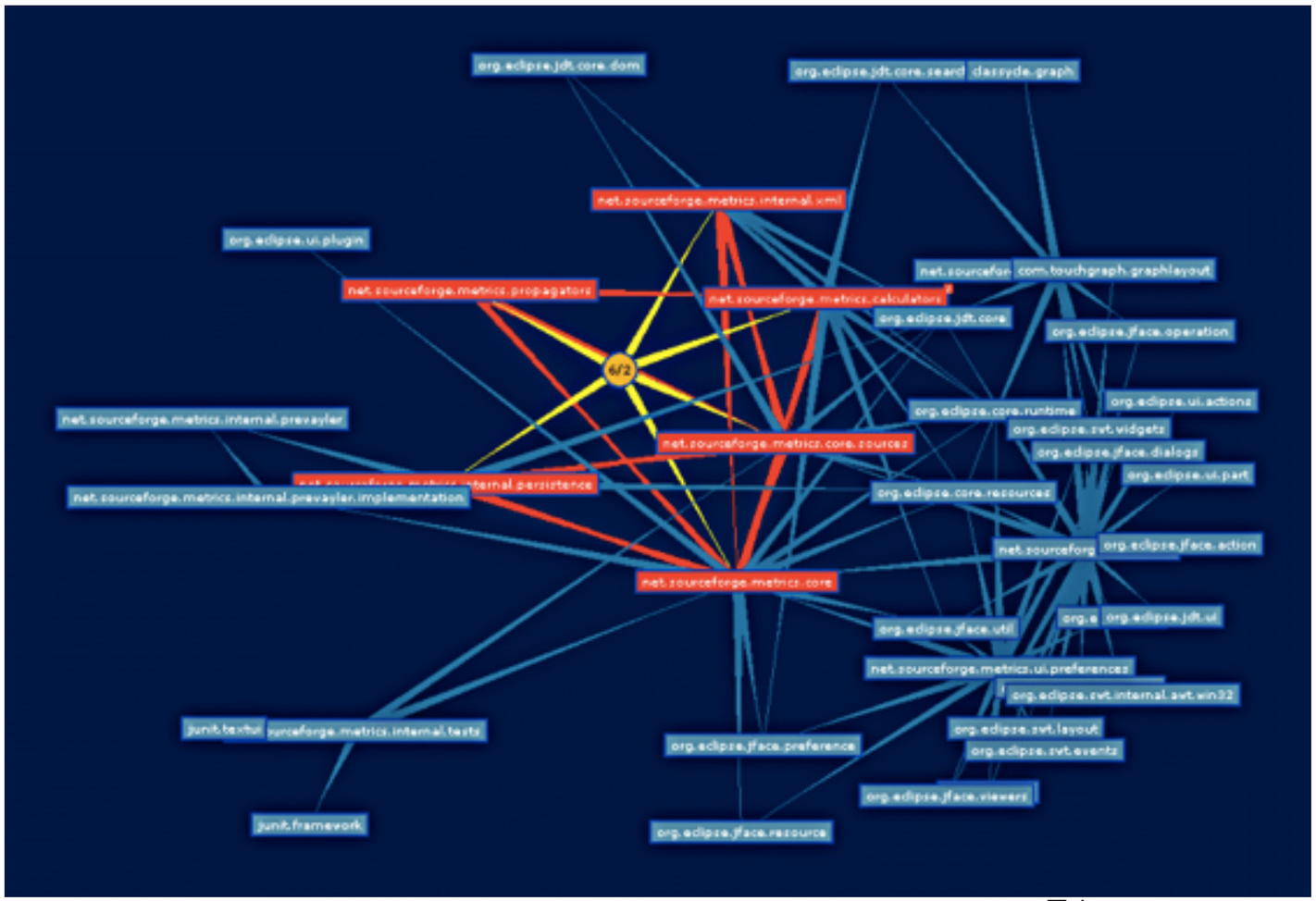
The average cyclomatic complexity is decreasing in Linux. This is due to the high rate of aggregation of small functions, that dilute the average value of the complexity.

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.

https://www.jfokus.se/jfokus14/preso/Code-as-a-crime-scene.pdf

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

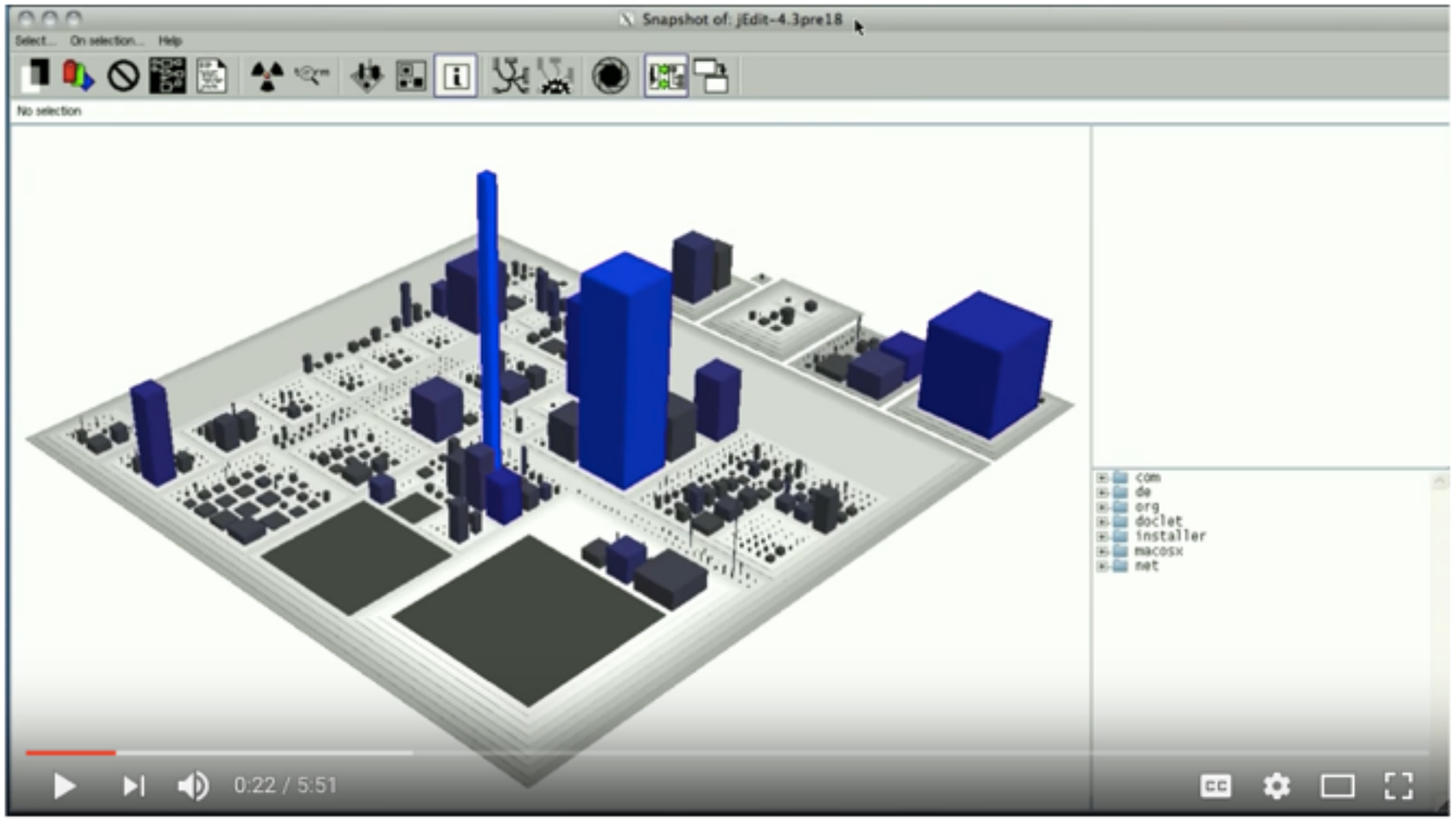
- Code dupliqué…
- Mauvaise encapsulation ou répartition des responsabilités
- Producteur & Consommateur

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 ?

Refactor to Understand

Problem: *How do you decipher cryptic code?*

Solution: *Refactor it till it makes sense*

- Goal (for now) is to *understand*, not to reengineer
- Work with a *copy* of the code
- Refactoring requires an adequate test base
  - ☞ If this is missing, *Write Tests to Understand*
- Hints:
  - ☞ Rename attributes to convey roles
  - ☞ Rename methods and classes to reveal intent
  - ☞ Remove duplicated code
  - ☞ Replace condition branches by methods

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 !

« Beauty is the absence of ugliness »

- « As you break the expectations of someone reading your code, you have introduced a cognitive cost. »

- « Since an architectural decision is by definition more important than local coding construct, breaking beauty in a high-level design is even worse ».

Comprendre l’architecture

a) Déterminer quels sont les composants principaux (ne pas se perdre dans le détail) (cela ne suit pas forcement la structuration en fichier)

b) Déterminer le processus/pattern attendu par exemple Pipe, MVC, etc.. (et donc les dépendances)

c) Regarder comment les codes sont couplés en les regardant évoluer et vérifier qu’il n’y a pas de dépendances inattendues.

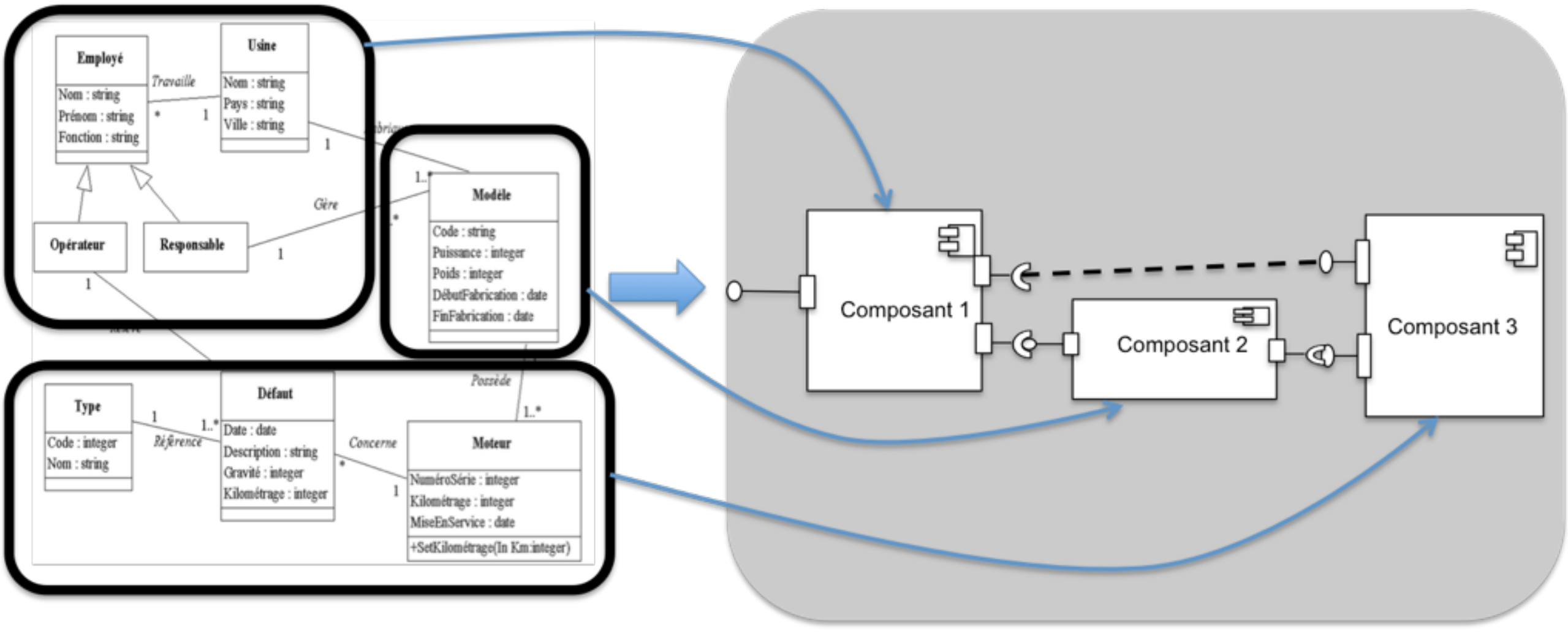
Comprendre l'architecture

➡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



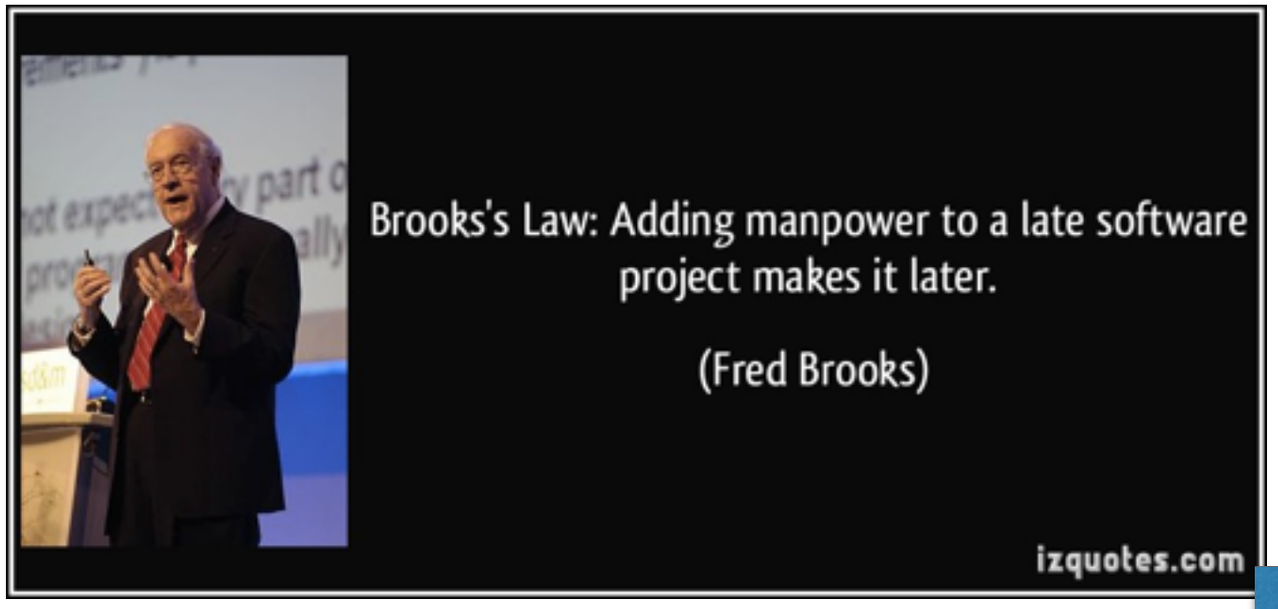
Importance de l’équipe

Pluralistic ignorance

Vérifier par des
questions et des faits

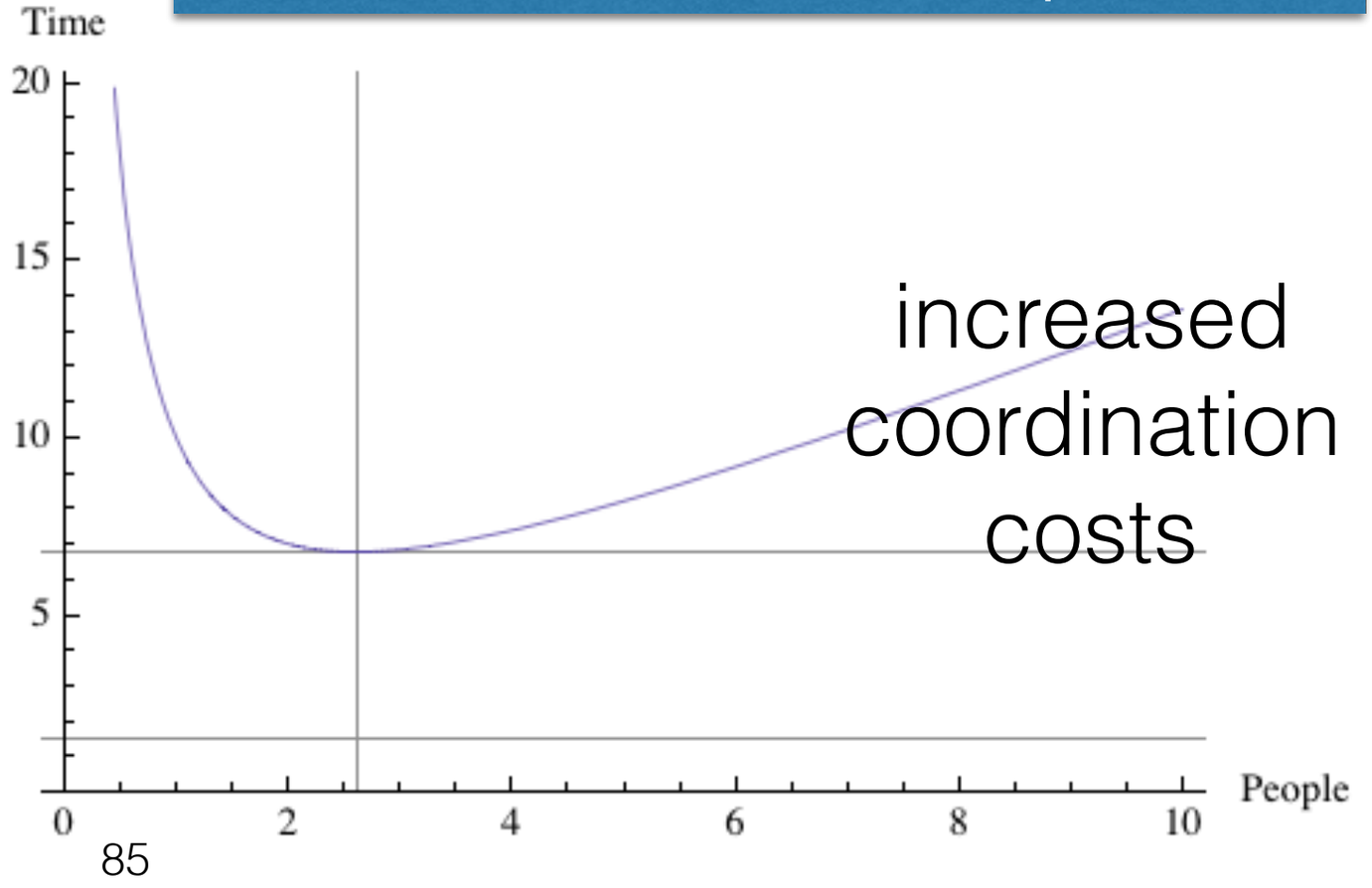
Commits

Point de départ pour une discussion



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

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



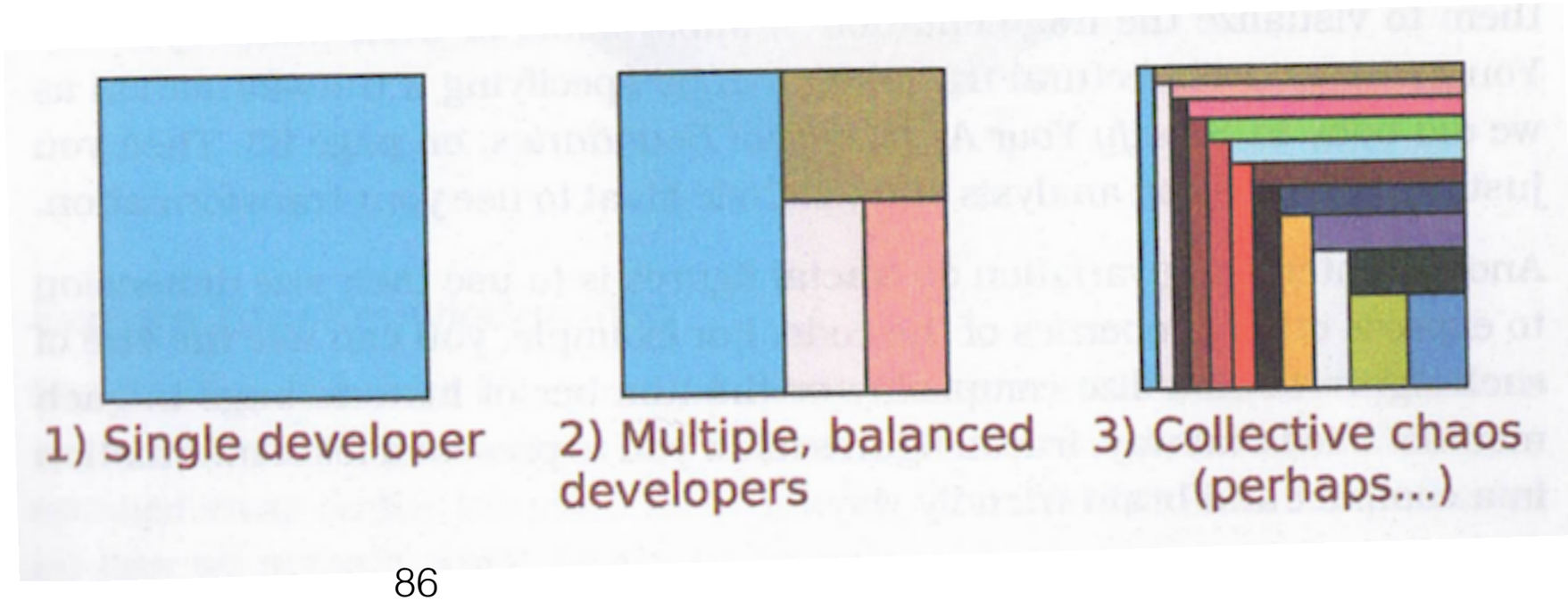
organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

— 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.*

MARTINFOWLER.COM

Evaluation du module

- Les rendus associés à un livre

➡ **Rendu L.1.:** “Voici mon sujet” Date limite : 8/01 à 15h.

➡ **Exposé E.1. :** Présentation du projet, 23/1

➡ **Rendu L.2. :** Synthèse d’articles, 12/2 à 18h

➡ **Rendu L.3 :** Contenu du livre

➡ **Rendu L.4 :** Codes/Résultats Brutes

➡ Examen

https://mireilleblayfornarino.i3s.unice.fr/doku.php?id=teaching:reverse:2017:evaluation

A book (Github book)

1) Introduction

Un chapitre par sous-groupe

Un chapitre de l'an dernier

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

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

- 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

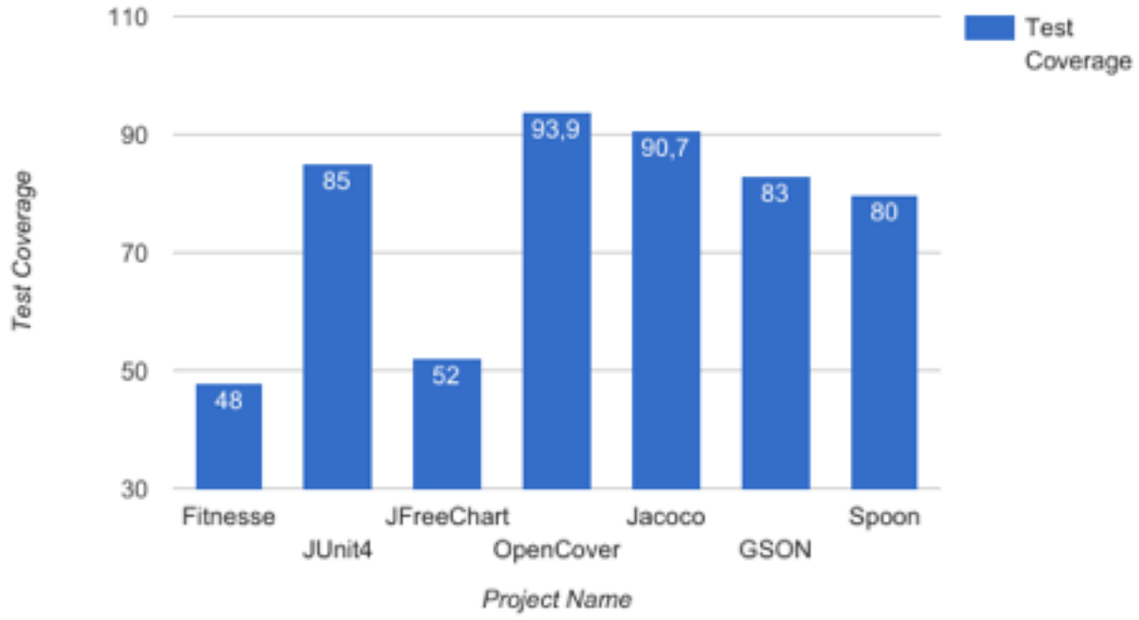
Google Gson …around 1,300 commits
JaCoCo …around 1,400 commits
Spoon ..around 1,800 commits

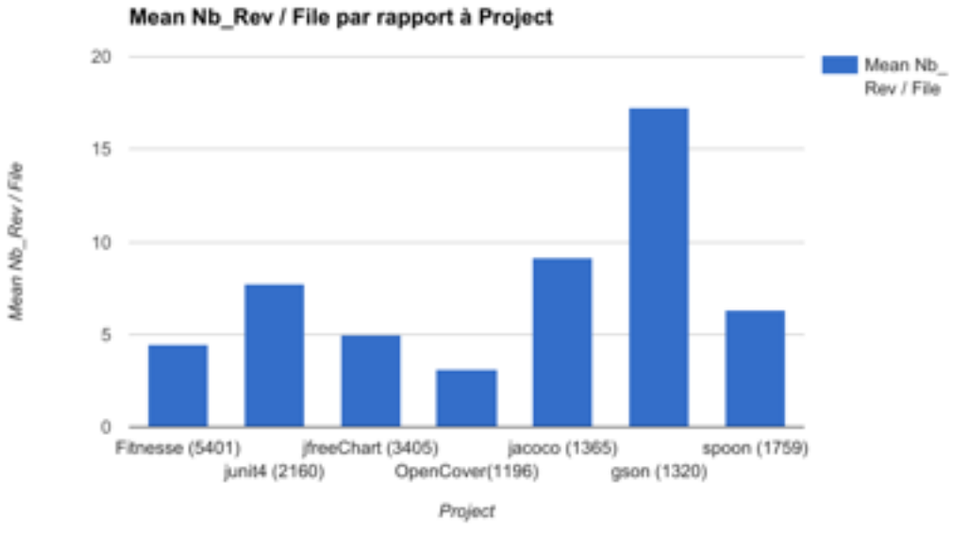
Metrics	Test-Driven Development				Test-Last Development		
	Fitnesse	JUnit4	JFreeChart	OpenCover	Spoon	GSON	JaCoCo
Code Coverage	48%	85%	45%	93.9%	90.7%	83%	80%
Sonar issues	1927	833	5039	286	2341	592	200
Complexity	8612	2061	19323	1568	7635	1945	1962
Code Age	48.3%	21.5%	18.1%	83.6%	9%	50%	35%
Average 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%

JaCoCo : couverture de tests

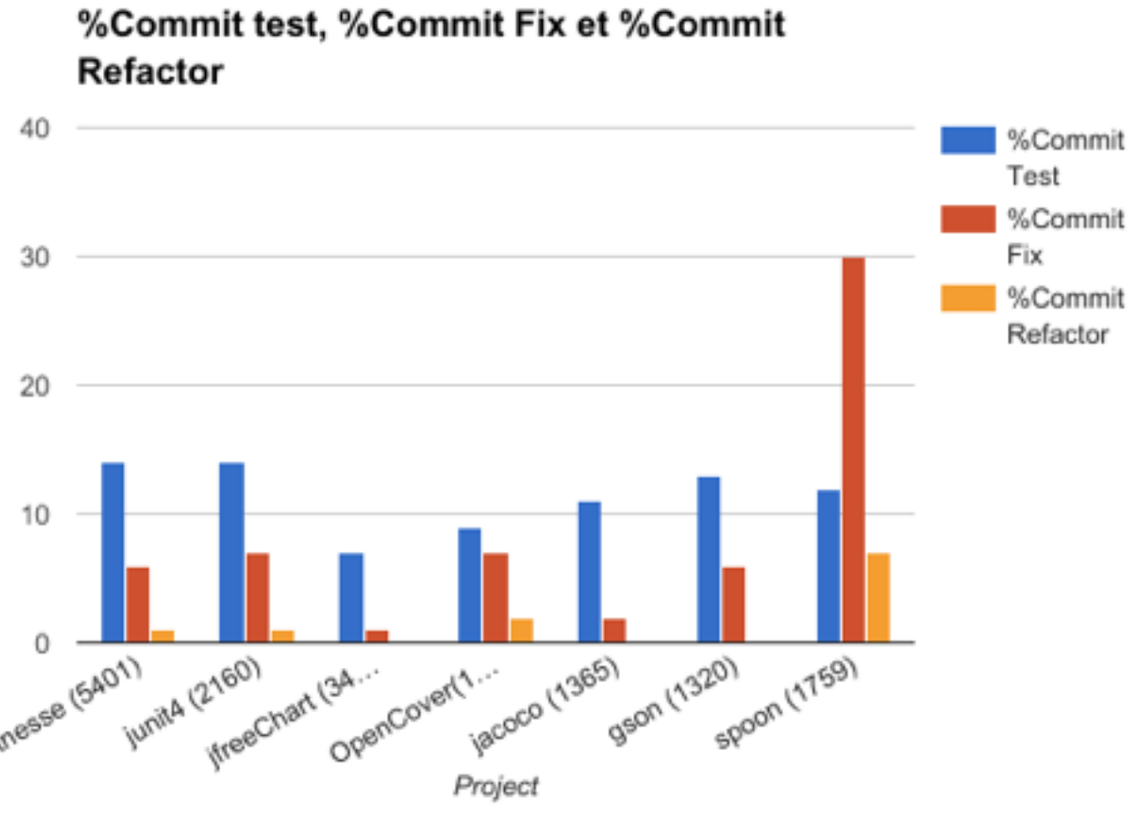
Sonar : issues, métriques et visualisation

Code maat : analyse de dépôts Git





=> 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



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

Visualisation : SoftVis3D

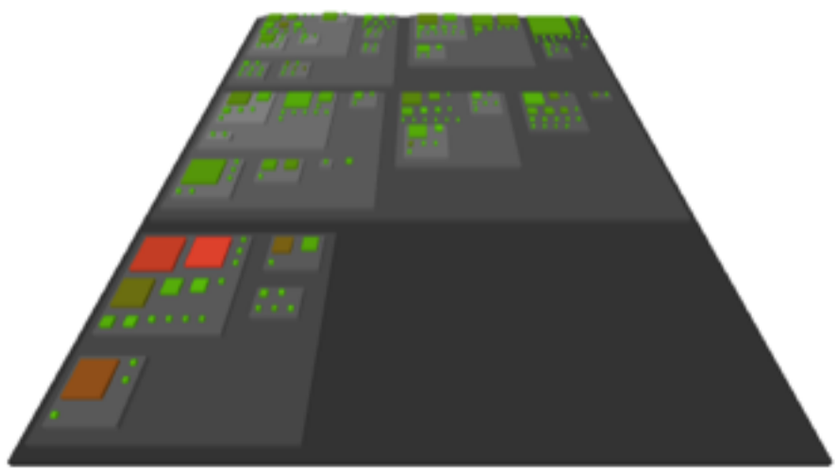


Figure 4 : SoftVis3D results for JUnit4

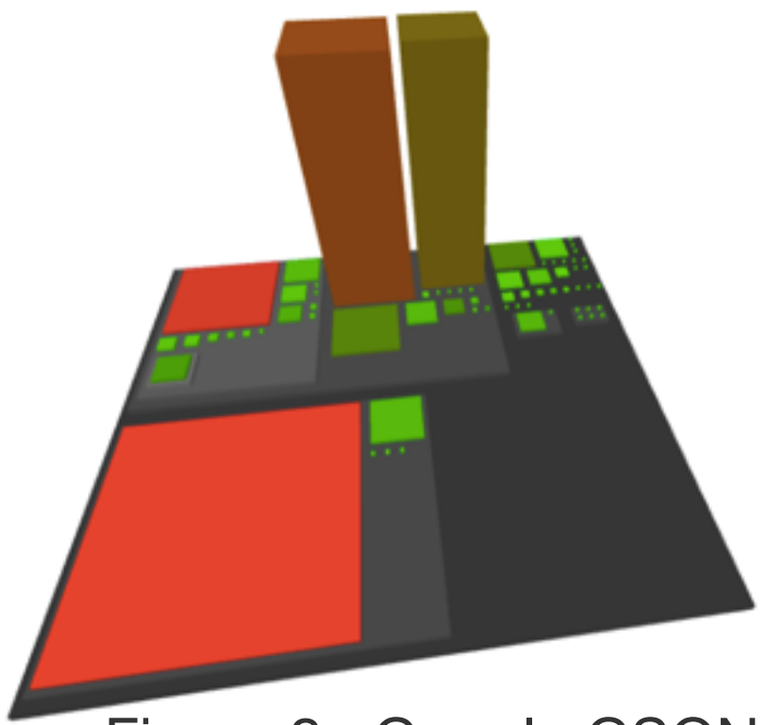


Figure 8 : Google GSON

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%.

A book Chapter: question examples

- Un projet d'état : PIX, que nous dit le logiciel?
    - e.g. Support d' Evolutions ? Agilité ? Données ?
- Bibliothèque d'algorithmes de "machine learning"(ML) : Apprendre.
    - e.g. Structure ? Complexité ? Apprendre du code.
- De Python 2 à Python 3 : Pourquoi ne pas migrer ?
- Multi-versions d'un "logiciel" ? JTS - Java Topology Suite
- Usages d'un logiciel? Docker


YEAH LETS
GO TEAM!!!!!!!!
memegen.com