

Réaliser, Maintenir & Modéliser du logiciel

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels

Sébastien Mosser
MGL7460 - Cours #1 - H20



1

Créredits

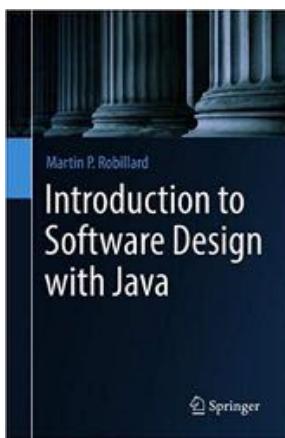


M. Blay-Fornarino



P. Collet

UNIVERSITÉ **CÔTE D'AZUR**



 McGill

3

Bienvenue dans ce cours de Génie Logiciel !



Michalis Famelis @MFamelis · 12 oct.

Most obvious proof that Devs are better than Wizards: Devs are often asked to perform magic. Wizards are never asked to write software.

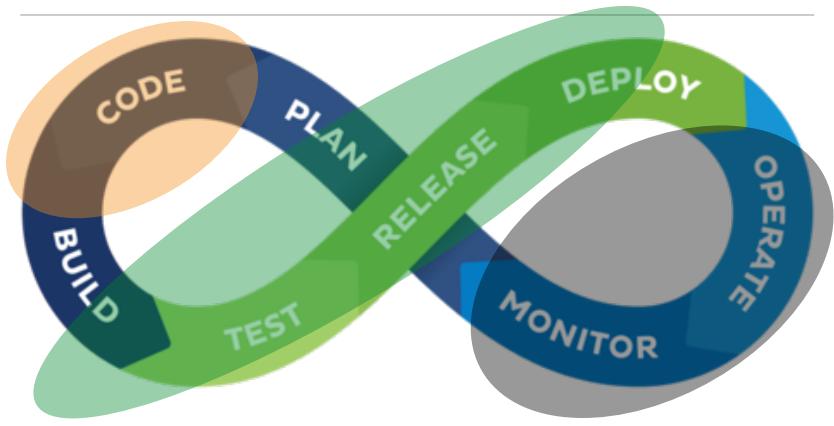
2

Leçon introductory

On va survoler le contenu du cours, et tenter de se donner une vision globale de ce qui va se passer pendant les 14 prochaines semaines.

4

Périmètre du cours sur la boucle DevOps



Pré-requis

MGL7460-H20

Hors-sujet (mais intéressant)

5

Encore un cours de chercheur déconnecté de la vraie vie ...

La suite va vous étonner ... 😊

6

Enseignements & Recherche Appliquées

- A l'UQAM (depuis 2019)
 - Direction adjointe "Génie Log" pour le BIGL (APP)
- En France (2012-2018)
 - Directeur du programme de Maîtrise en Architecture Logicielle
 - 50% des étudiants en partenariat industriels (~ "COOP")
 - Responsable des enseignements de GL pour Poly
 - Équipe de 17 enseignants, dont 14 industriels
- Contrats de recherche principalement industriels (2010-...)
 - *Recherche en GL ≠ s'inventer des problèmes*



Sébastien Mosser

"geek, snowboarder & composition-driven guy"

• 19-...: Professeur, UQAM



• 12-18: Maître de Conférences, UCA



• 11-12: Chargé de Recherche, SINTEF



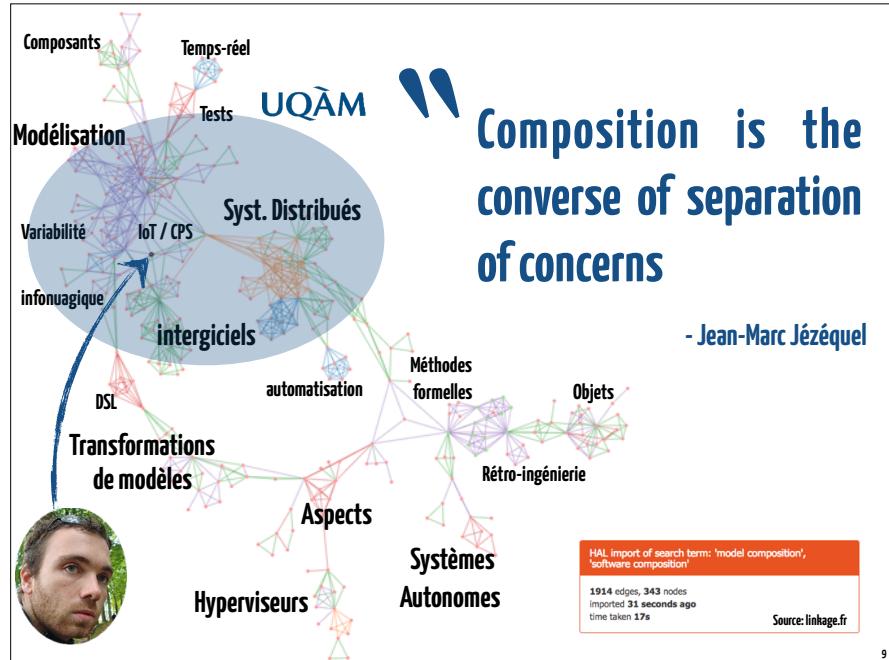
• 10-11: Postdoc, Inria Lille Nord-Europe



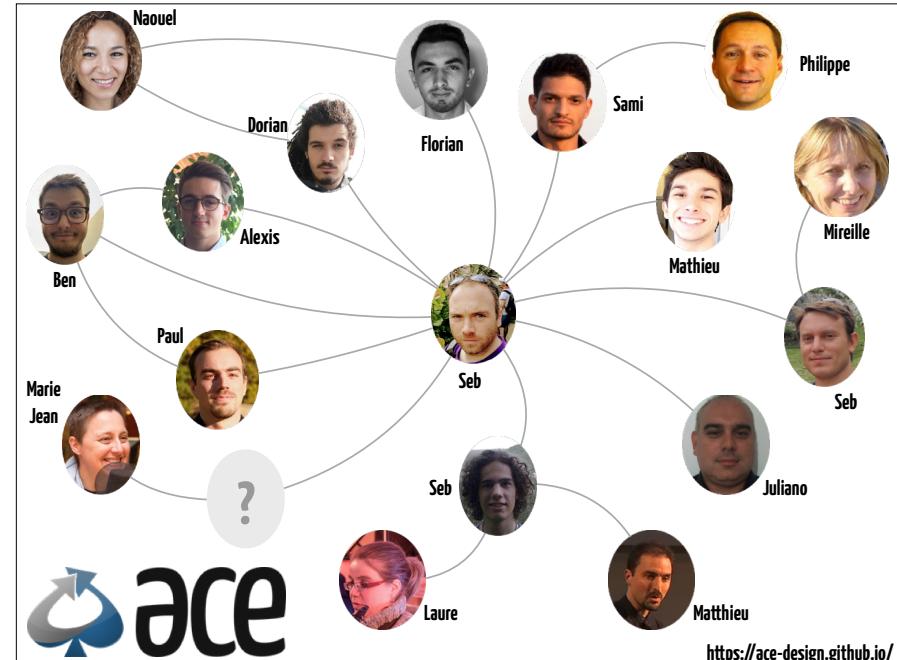
• 07-10: PhD, Composition Logicielle

7

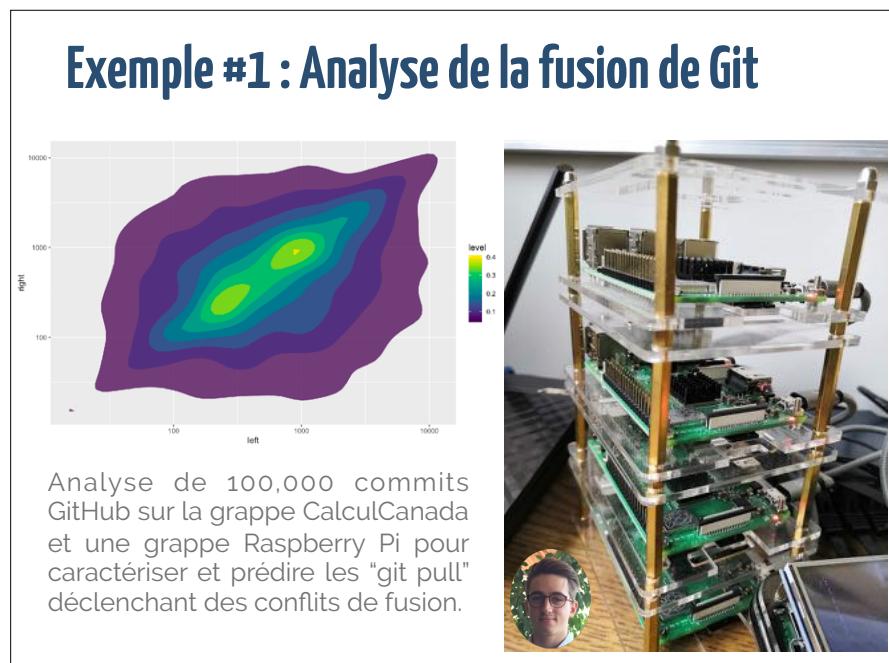
8



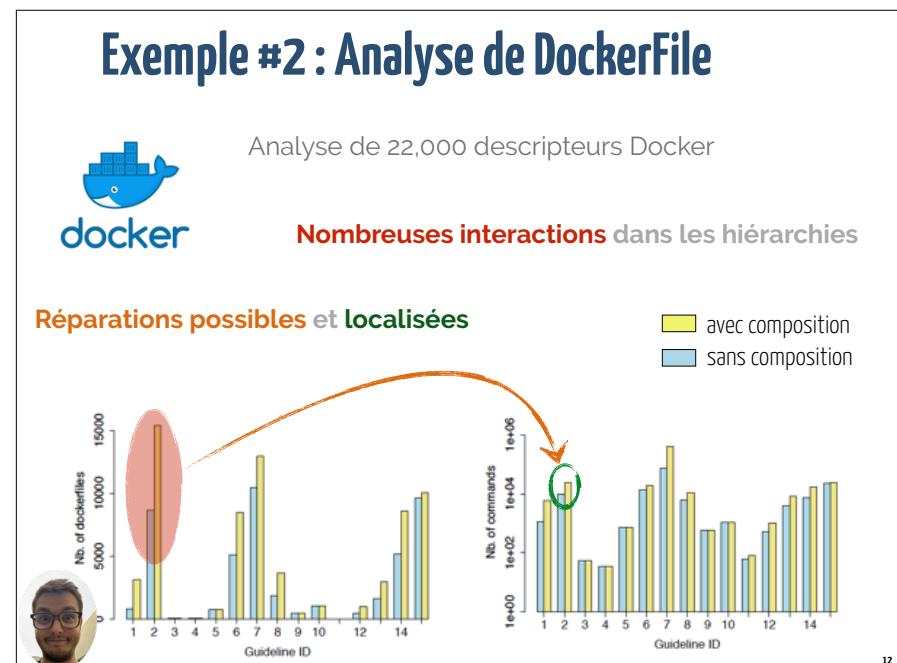
9



10

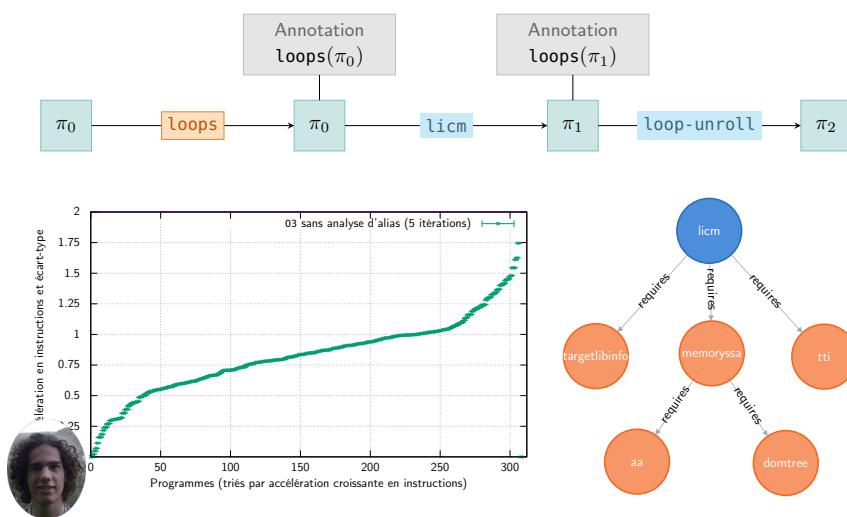


11



12

Exemple #3 : Cartographie de compilateur



13

Publicité outrancière 😊



14

De la recherche en Génie logiciel ?



15

- 1 Logistique du cours**
- 2 Réaliser ?**
- 3 Maintenir ?**
- 4 Modéliser ?**
- 5 Projet Technique (réalisation)**
- 6 Projet Individuel (maintenance)**

16

Logistique du cours

1

17

Contact : N'hésitez pas. Vraiment ...



Sébastien Mosser

@petitroll

Parfois on regarde les choses telles qu'elles sont en se demandant pourquoi. Parfois, on les regarde telles qu'elles pourraient être en se disant pourquoi pas.

✉ Montréal, Québec
✉ mosser.github.io

@petitroll

Courriel : **ne pas utiliser**

(~150 courriels/jour, souvent beaucoup de latence)



(voir invitation)

19

Les diapos ne sont pas un polycopié ...

Prenez des notes!

Posez des questions!

Discutez, râlez, échangez, ...

Les diapos sont uniquement un support

18

“We are going on an adventure”

Attention, MGL7460 (et le génie logiciel dans l'absolu) est un cours difficile, qui demande un travail continu et régulier durant la session.



20

Qui demande un travail
continu
et régulier
durant la session.

21

Cours : Plan de match



A partir de la semaine #5, du temps est prévu dans les séances pour les travaux en équipe.



F. Bordeleau
(ÉTS)



X. Blanc
(Univ. Bordeaux)

Environnement Technologique



JUnit



22

#Semaine	Cours (UQAM)	Atelier (SFL)
2	Leçon introductory: Réaliser, Maintenir et Modéliser du logiciel ? (Lab optionnel)	--
3	Gestion de versions, Démarche de tests	--
4	--	Git, GitLab, GitLab CI
5	Exigences, Scénarios d'acceptations	--
6	Déploiement continu, <i>Test-driven development</i>	--
7	--	BDD, TDD
8	Analyse de code	SonarQube
9	Semaine de relâche	--
10	Principes de développement (<i>CleanCode</i> , SOLID)	--
11	Métriques Logicielles & Visualisation pour la maintenance	--
12	Présentation intermédiaire du Projet Technique	idem
13	Cours de renforcement / Étude de cas	Suivi projet technique
14	Cours invité (Pr Xavier Blanc, Univ. Bordeaux)	Suivi projet technique
15	Cours invité (Pr Francis Bordeleau, ÉTS)	Suivi projet technique
16	Rencontres Projet Individuel	Suivi projet technique
17	Rencontres Projet Individuel	Suivi projet technique

23

Les labos, en lien avec le projet technique, sont fait en collaboration avec Savoir Faire Linux

24

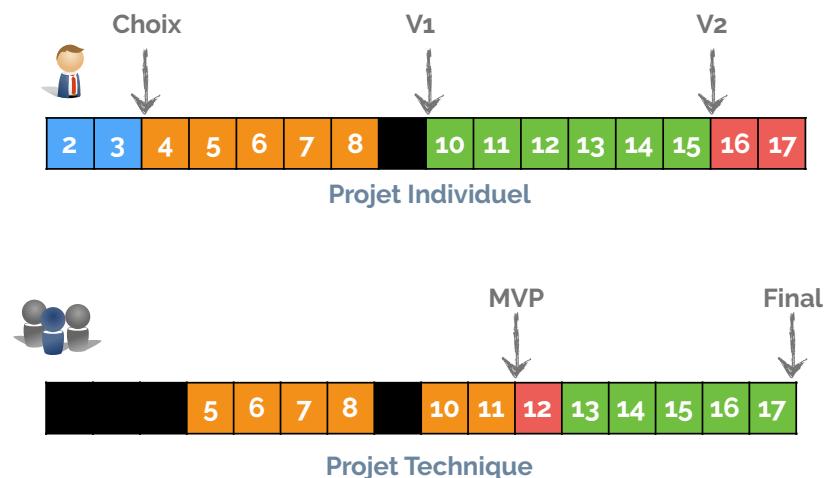
Projet Technique (Binômes) : Réalisation

- Réaliser une application dans un domaine fonctionnel simple
 - Vente de produits bancaires personnalisables à des clients
- Mettre en place un environnement complet
 - Gestion de version
 - Analyse de qualité
 - Tests (unitaire / intégration / acceptation)
 - Intégration Continue
 - Déploiement Continu

Évaluation : Application & Environnement

25

Organisation des projets de session & Jalons



27

Projet Individuel : Maintenance

- Analyse de la *maintenabilité* d'un logiciel
- Choisir un projet Open-source d'ampleur
- Proposer une analyse de ce logiciel
 - En étudiant l'équipe de développement
 - En étudiant le code
 - En étudiant son déploiement ...

Ce projet fait office d'examen, et demande un travail (i) conséquent, (ii) régulier et (iii) rigoureux.

26

Entente d'Évaluation

Date(s)	Travail à rendre	Poids
19.01.20	Choix du cas d'études individuel	0%
01.03.20	Projet Individuel - V1	20%
15.03.20	Projet Technique - MVP	10%
12.04.20	Projet Individuel - V2	40%
26.04.20	Projet Technique - Final	20%

Examen = Projet Individuel ("takeaway exam")

28

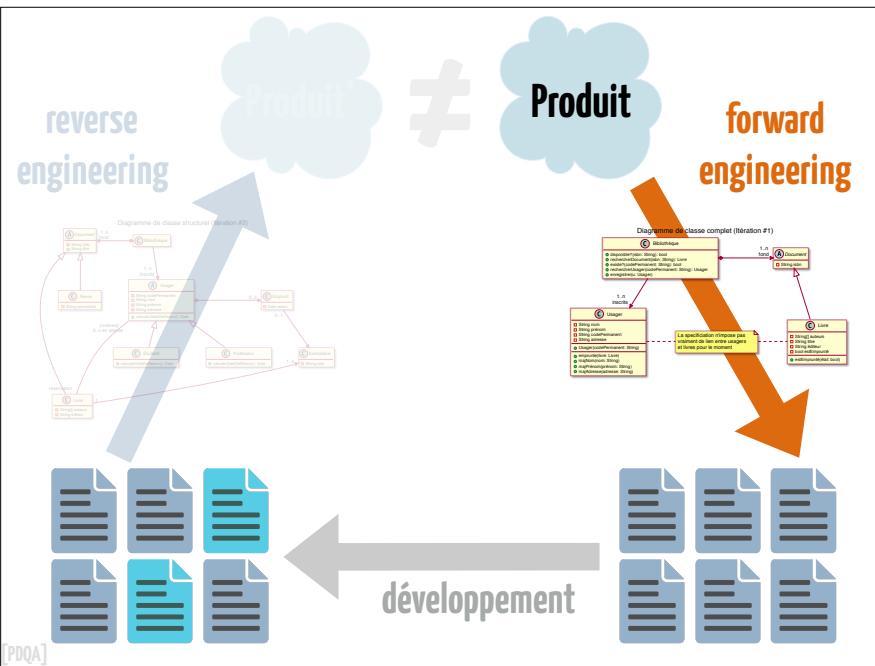
Réaliser ?



29

Hypothèse :

Vous savez coder



31

30

Définition du “Génie Logiciel”



Le génie logiciel est une **science de génie** industriel qui étudie les **méthodes de travail** et les **bonnes pratiques** des ingénieurs qui **développent** des logiciels.

“On time, on specs, on budget.”

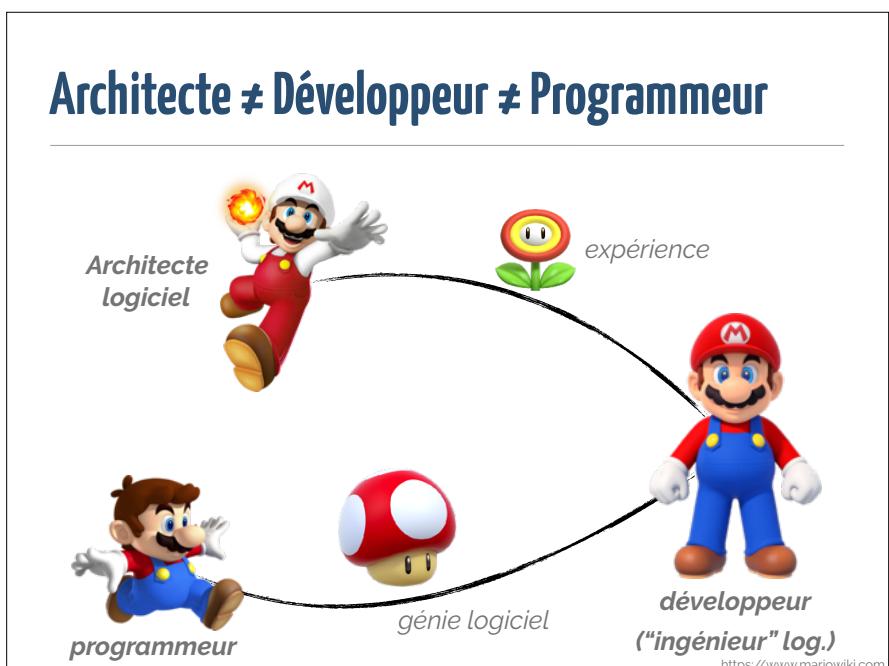
32



33



34



35



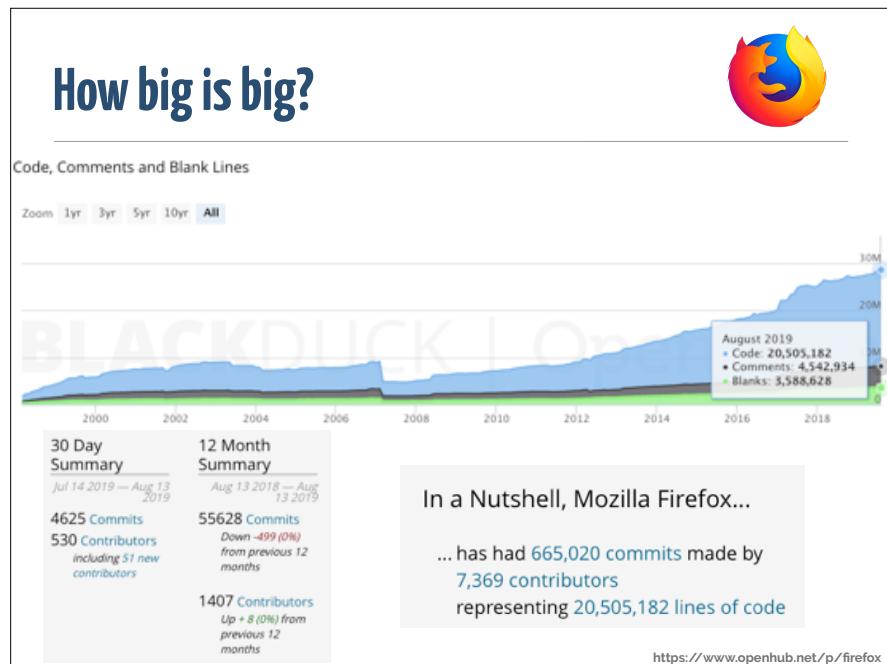
36

Définition du “Génie Logiciel”



Le génie logiciel s'intéresse en particulier aux **procédures systématiques** qui permettent d'arriver à ce que des **logiciels de grande taille** correspondent aux **attentes du client**, soient **fiables**, aient un **coût d'entretien réduit** et de **bonnes performances** tout en respectant les **délais et les coûts de construction**.

37



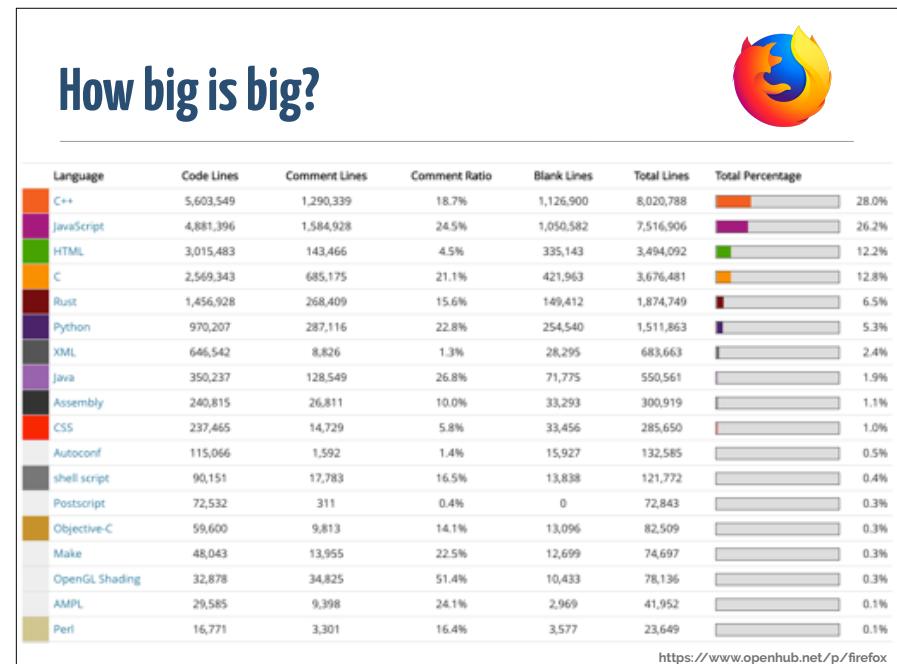
39

How big is big?



Exemple “simple” : un navigateur internet comme Firefox

38



40



<https://xkcd.com/202/>

41

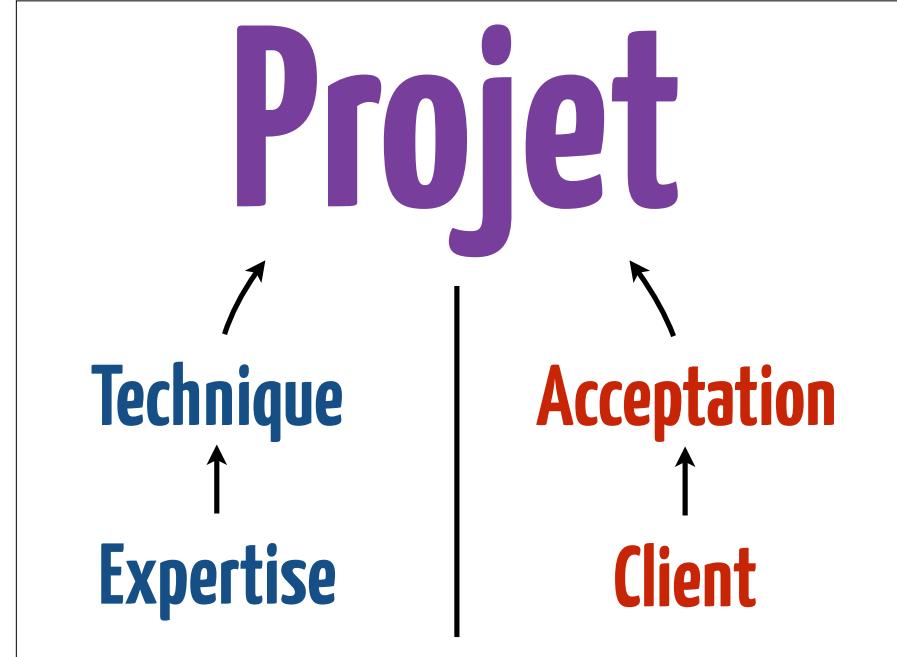
Bikeshedding

Passer beaucoup de temps à **discuter de détails inutiles** (le garage à vélo)

Ne plus avoir le temps de se concentrer
SUR ce qui est important (la centrale nucléaire)

Loi de la trivialité (Parkinson)

Berkeley Software Distribution (aka BSD)



42

Maximize Value, not Output

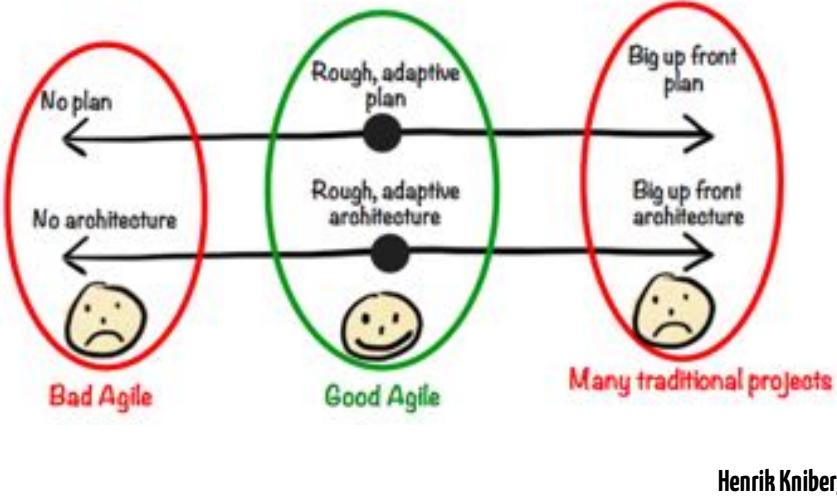


Henrik Kniberg

43

44

Don't go overboard with Agile!



Henrik Kniberg

45

Une **équipe** qui s'**améliore** et s'**organise** dans un **environnement complexe** pour **déliver en continu** de la **valeur** à ses **utilisateurs**, en toute **simplicité** ...

— Romain Couturier

47

Tests Unitaires



Scott W. Ambler
@scottwambler

Suivre

A good programmer looks both ways before crossing a one-way street.

Traduire le Tweet

02:28 - 21 juin 2019

6 Retweets 10 J'aime



Commenter Partager Aimer Envoyer

“Du code qui compile pas, ça a jamais tué personne.
Alors que du code qui compile par contre ...”

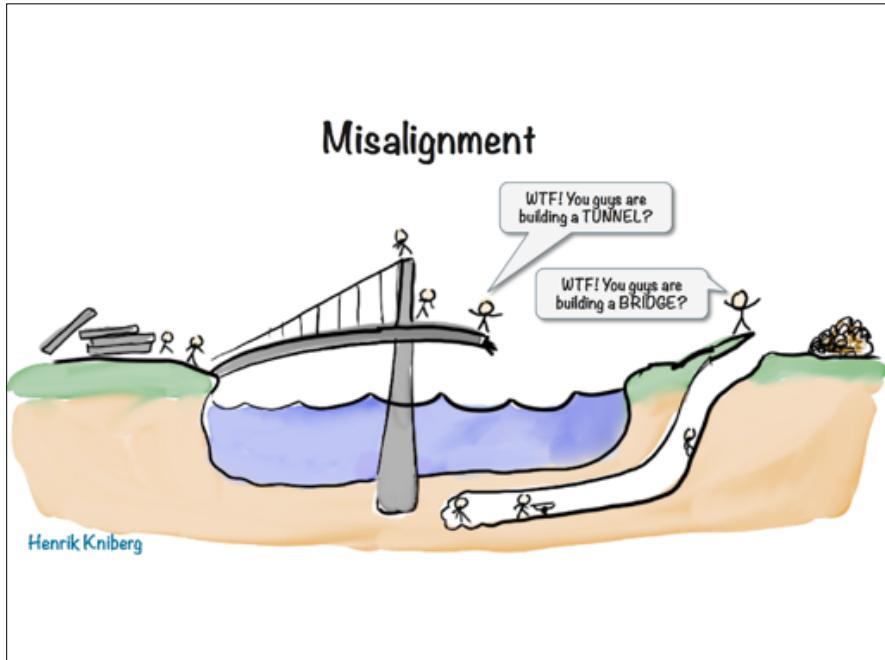
46

Simplicité

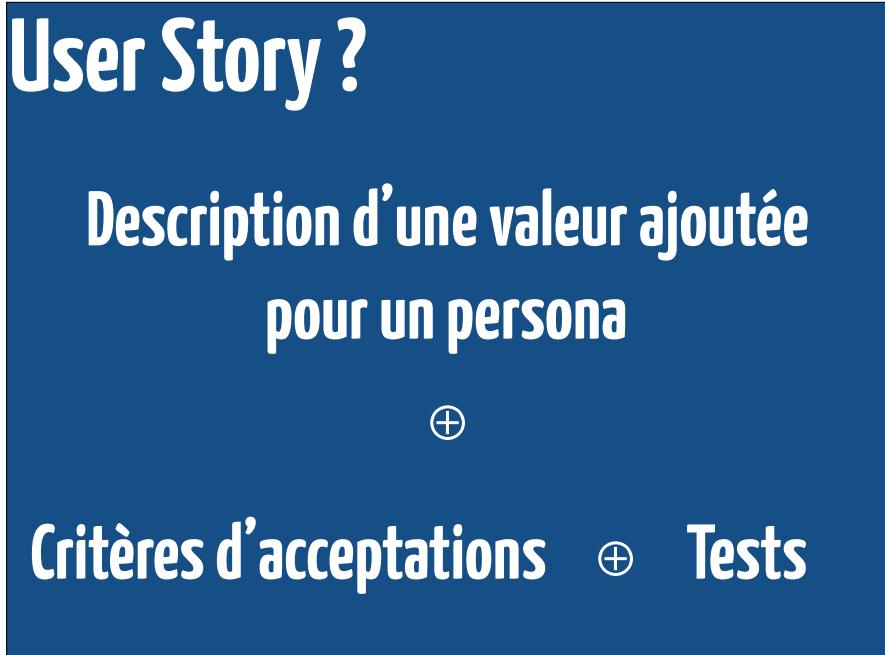
11

Art de **minimiser** la quantité de **travail inutile**

48



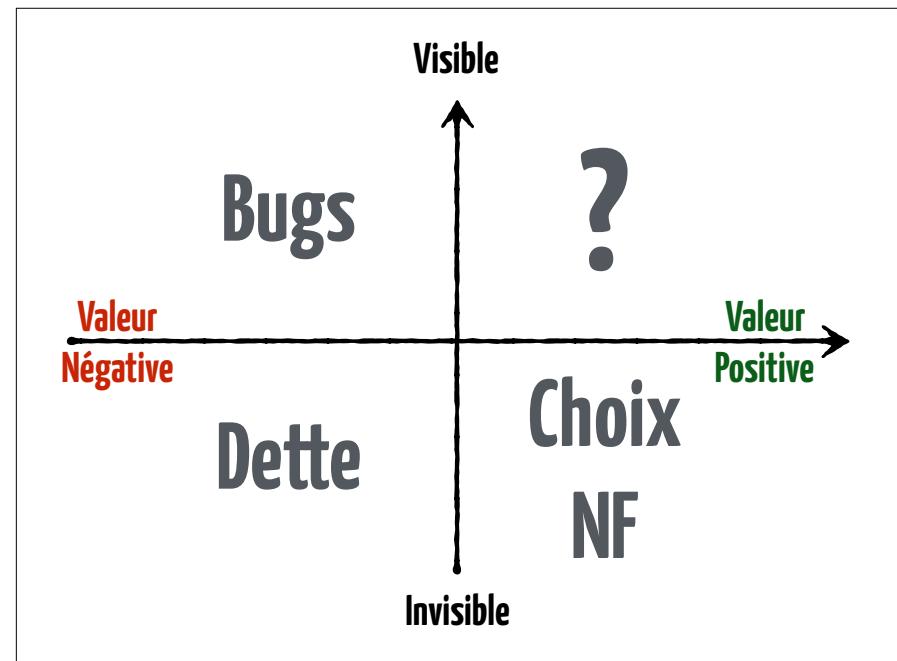
49



51



50



52

“Signal d’alarme”-driven development



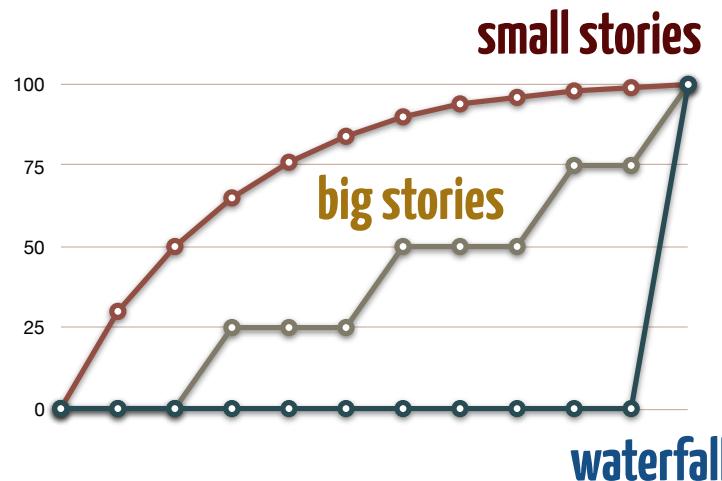
Attendre jusqu'à la **catastrophe**
pour faire plier les **exigences de qualité**

Permet de **rogner sur les tests, la documentation, ...**

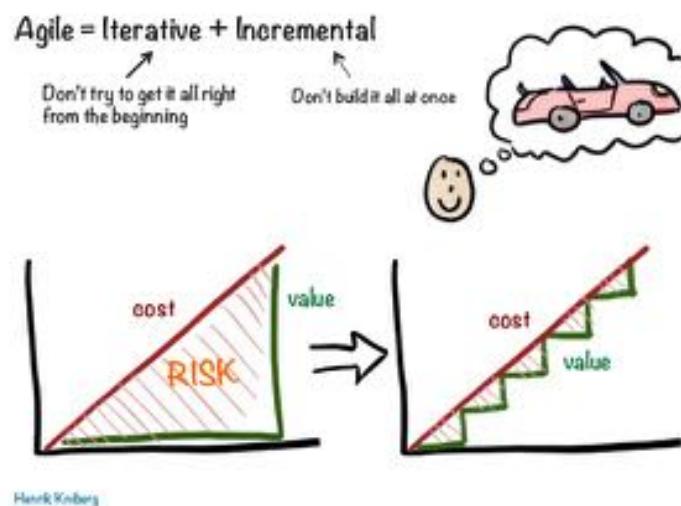
“Si vous attendez la dernière minute, ça ne prend
qu'une minute à faire” (Loi de Parkinson)

53

Cumulative value

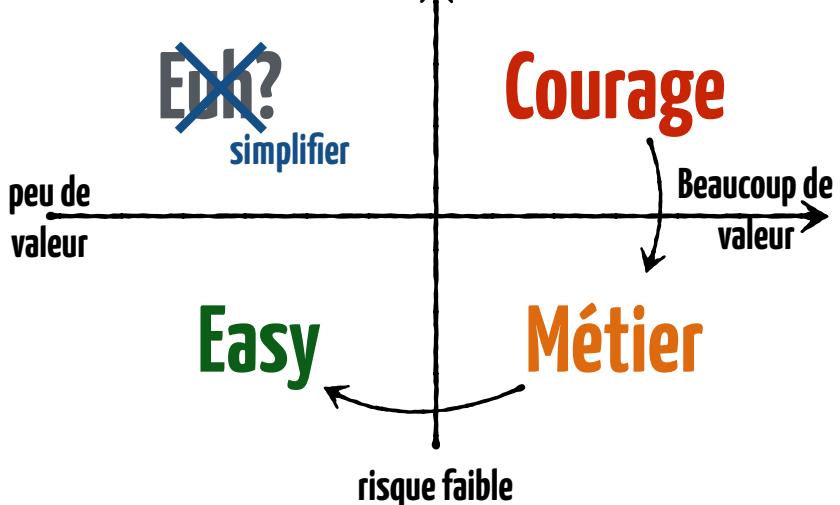


54



55

Planification? risque élevé



56

Le rôle des exigences est primordial !

- Expertise de développement
 - Faire le produit de la bonne manière
- Expertise de spécification
 - Faire le bon produit
- Expertise de gestion de projet
 - Faire le produit dans les temps

Il est nécessaire pour un développeur de comprendre ces trois dimensions lors de la réalisation d'un logiciel

57

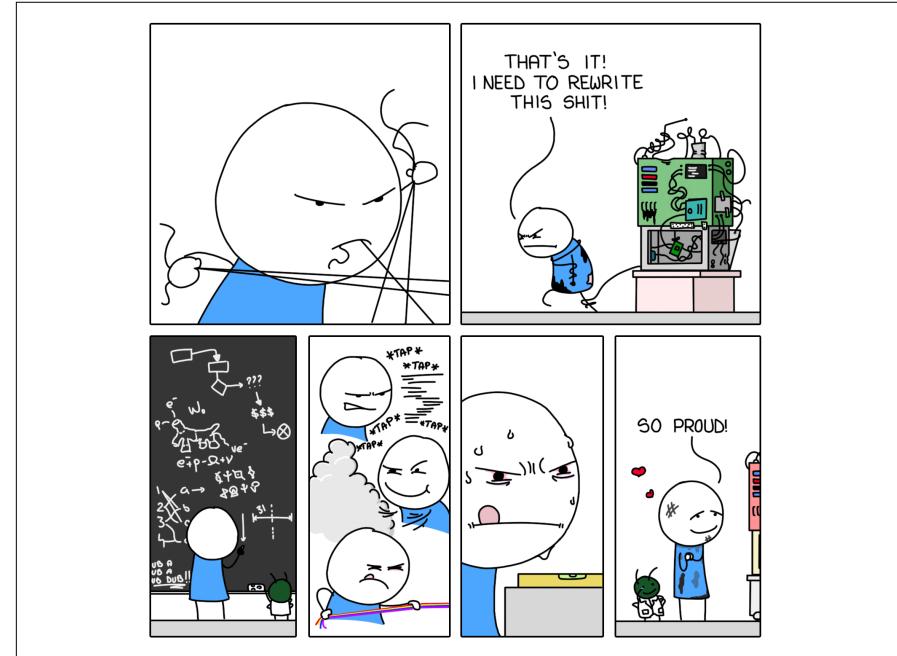
Maintenir ?



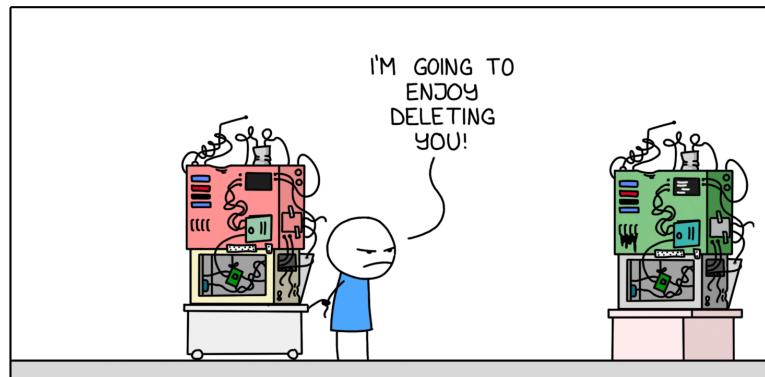
59

Le génie logiciel est la science des compromis.

58

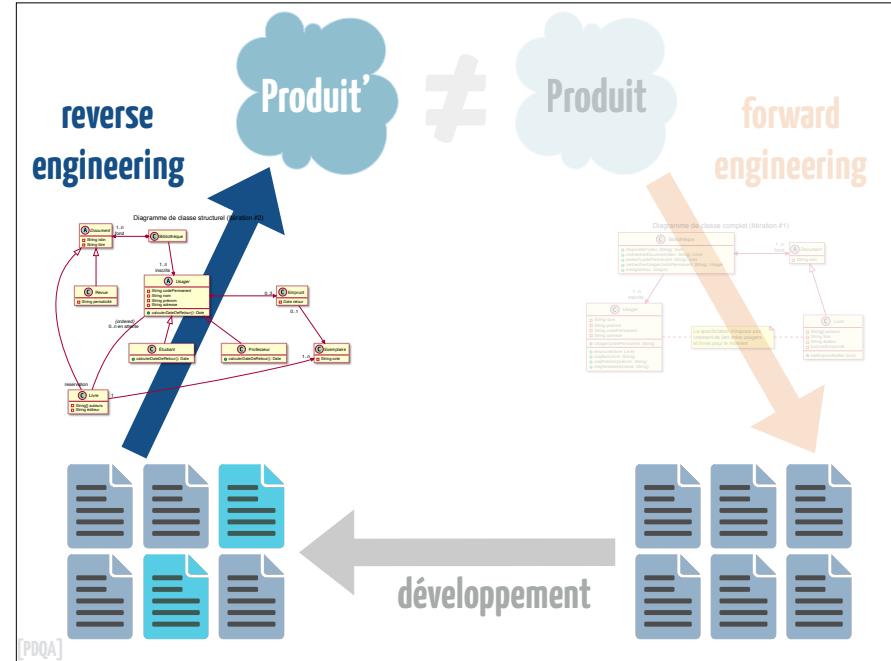


60



Maintenir la qualité, c'est compliqué

61



62

Enseigner la maintenance logicielle

“

C'est une discipline de l'informatique et du génie logiciel réputée "non enseignable", qui repose sur la pratique et l'expérience de longue durée, difficilement atteignable dans une session de cours.

On peut "gruger" en montrant artificiellement des situations, mais sur le long terme ces apprentissages sont inutiles.

Rien ne remplace une mise en pratique de longue durée.

Consensus enseignants / industriels

63

Un exemple ?

```
#include <stdio.h>
#include <string.h>
char *d =
"\"n'+,#/*{w+/w#cdnr+,{}r/*de}+,*{*,/w%+,/w#q#n+,#{l+,/n{n+,/+n+,/#\n
;#q#n+,/+k#+*,+'r :d'3,{w+K w'K:+}e#';dq#`l \
q#'+d#K!/+k#;#`r)eKK{n}#/;#q#n'){})w'{){{nl}'/+#n';d}rw' i;# \
){nl)!/n{n#'; r#w'r nc{n}#/;{l,+`K {rw' iK{{nl}'/w#q#n'wk nw' \
iwk{KK{nl}!/w'1##w#` i;:{nl}'/*{q#`ld;r'}{nlwbl/*de}'c \
;:{nl}'-{}rw' /+,}##`*){nc,' ,nw'}/+kd'+e}+,#'rdq#w! nr' /' )+}{rl#'{n' ' )# \
}+##(!!/";
char *s = "!ek;dc i@bK'(q)-[w]*n+r3#1,{:nuwloca-0;m .vpbks,fxntdCeghiry";
int exec(int t, int _, char *a)
{
    if (t < 0) { while (t++ < 0) { a = 1 + index(a, '/'); }; return exec(0, _, a); }
    if (t == 0) { while (*a != '/') { putchar(index(s, *a++)[31]); }; return 0; }
    if (t == 2) { exec(0, 0, d); exec(1_, 0, d); exec(-13, 0, d); } if (t < _) { exec(t+1, _, a); } exec(-27+t, 0, d); if (t == 2 & _ < 13) { return exec(2, _+1, ""); } return 0;
}
int main() { return exec(2, 2, ""); }
```

extrait du livre de Martin Robillard

64

```

int exec(int t, int _, char * a) {
    if (t < 0) {
        while (t++ < 0) {
            a = 1 + index(a, '/');
        }
        return exec(0, _, a);
    }
    if (t == 0) {
        while (* a != '/') {
            putchar(index(s, * a++)[31]);
        }
        return 0;
    }
    if (t == 2) {
        exec(0, 0, d);
        exec(1 - _, 0, d);
        exec(-13, 0, d);
    }
    if (t < _) {
        exec(t + 1, _, a);
    }
    exec(-27 + t, 0, d);
    if (t == 2 && _ < 13) {
        return exec(2, _ + 1, "");
    }
    return 0;
}

int main() {
    return exec(2, 2, "");
}

```

\$ gcc -o mystery mystery.c
\$./mystery

Respect des conventions de codage du langage C

C'est "opti"

Et en plus ça s'exécute comme il faut ... où est le problème ?

65

```

public class Mystery {

    private static String[] gifts = {
        "A partridge in a pear tree.", "Two turtle doves and",
        "Three french hens", "Four calling birds",
        "Five golden rings", "Six geese a-laying",
        "Seven swans a-swimming", "Eight maids a-milking",
        "Nine ladies dancing", "Ten lords a-leaping",
        "Eleven pipers piping", "Twelve drummers drumming",
        "And a partridge in a pear tree.", "Two turtle doves"
    };

    private static String[] days = {
        "first", "second", "third", "fourth", "fifth", "sixth", "seventh",
        "eighth", "ninth", "tenth", "eleventh", "Twelfth"
    };

    public static void main(String[] args) {
        for (int i = 0; i < days.length; i++) {
            System.out.printf("%nOn the %s day of Christmas%n", days[i]);
            System.out.println("My true love gave to me:");
            for (int j = i; j >= 0; j--) {
                System.out.println(gifts[i == 11 & j < 2 ? j + 12 : j]);
            }
        }
    }
}

```

Que pensez vous de cette version là ?

67

Twelve Days of Christmas

On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

...



66

Et de celle-ci ?

```

public static void main(String[] args) {
    List<Event> events = new ArrayList<>();
    Event first = new Event(new Gift("a partridge in a pear tree"));
    events.add(first);

    Event second = first.buildNext(new Gift("two turtle doves"));
    events.add(second);

    // ...
    events.forEach(System.out::println);
}

```

4 version ≠ produisant le même résultat !

68

Sécurité



Analyse multi-dimensionnelle

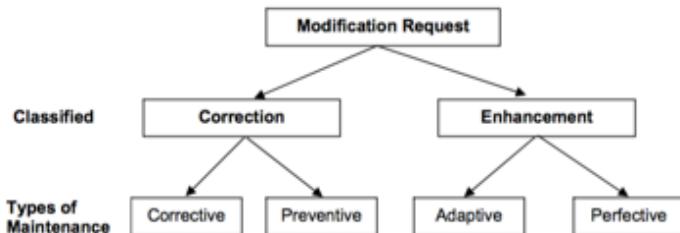
Maintenabilité

Performance

(en dimension N en vrai)

69

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

10

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

71

VOUS LE VOULEZ COMMENT VOTRE PROJET ?

(VOUS POUVEZ FAIRE JUSQU'A DEUX CHOIX)



"Gratuit" n'est pas une option.

70

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.

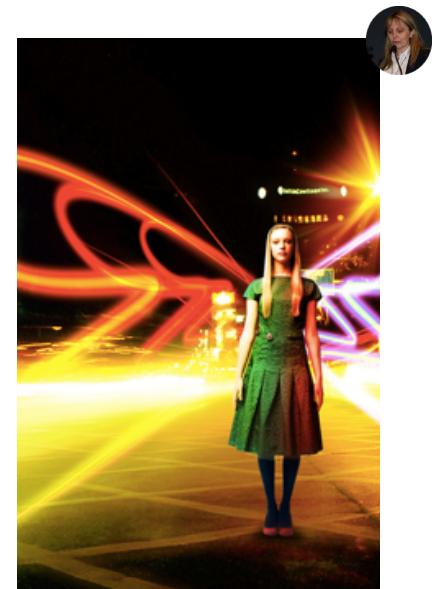


image : <http://tomcunniff.com/2012/03/old-vs-new-media-the-future-is-a-feedback-loop/>

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

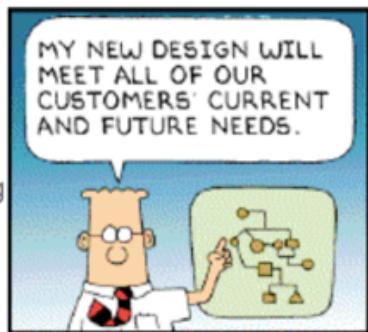
19

72

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

73

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



74



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

75

76

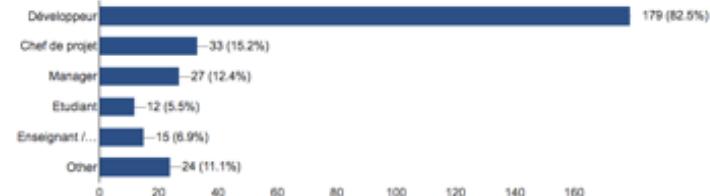
Maintenir ?



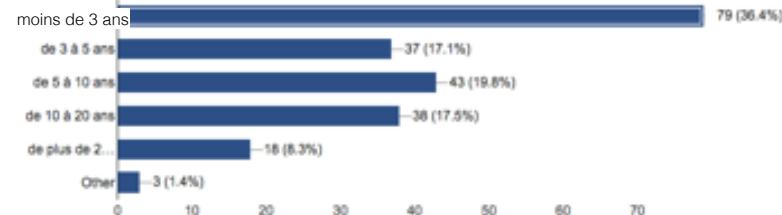
77

Vous êtes (217 réponses)

217 réponses



Vous avez une expérience professionnelle de (217 réponses)



79

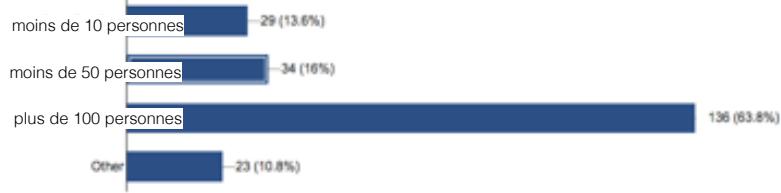
Que veut dire Maintenir en industrie ?

Sondage réalisé en 2018

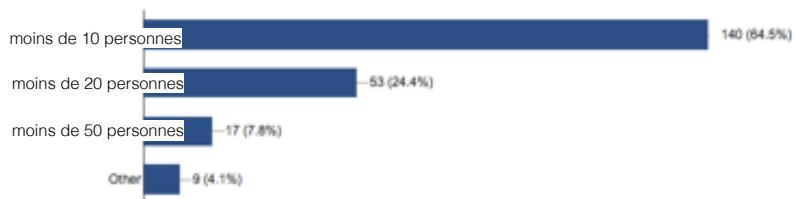


78

Vous travaillez pour une entreprise (213 responses)

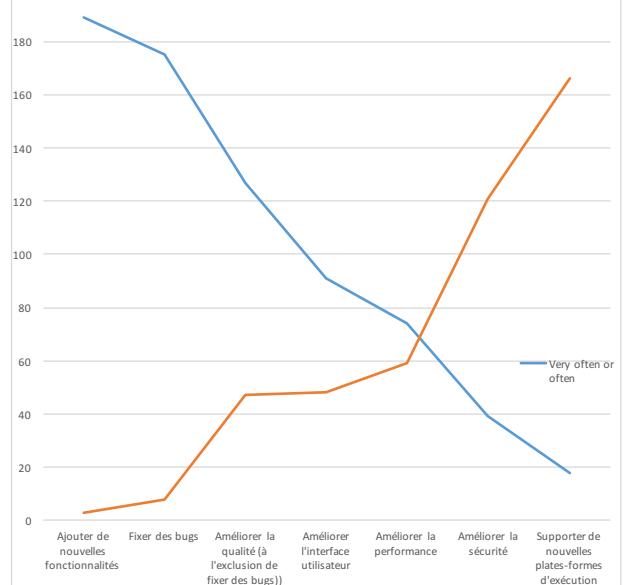


Vous travaillez dans une équipe (217 responses)



26

80

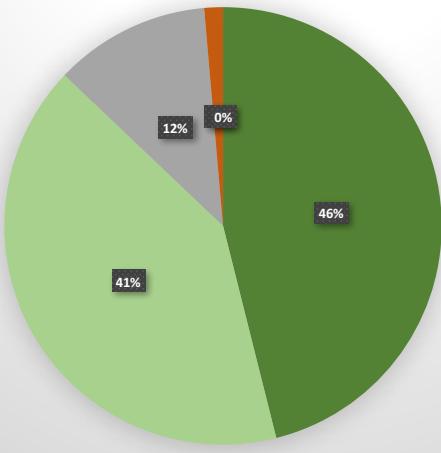


Vous modifiez votre code pour :

81

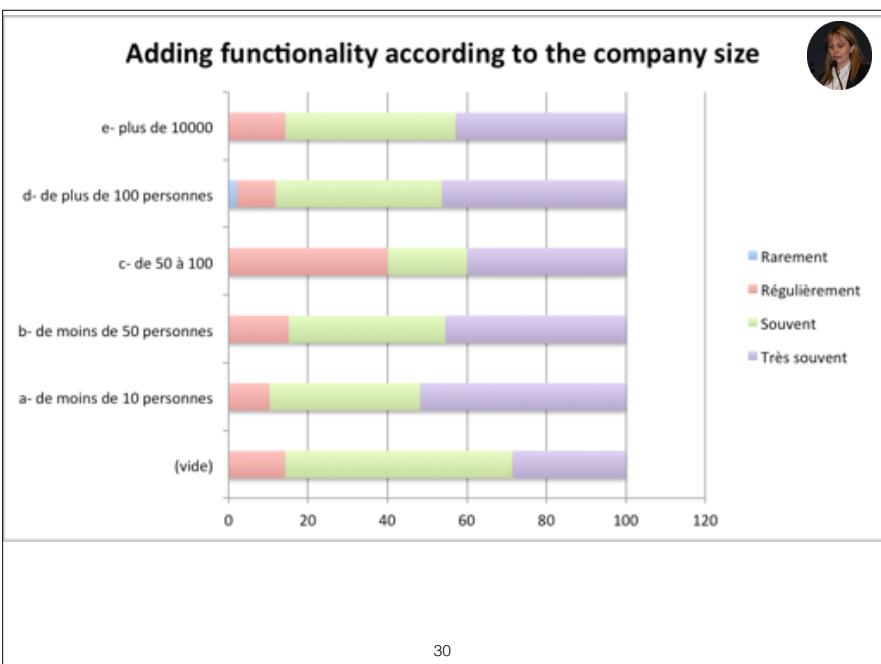


Ajouter de nouvelles fonctionnalités



« we will never be able TO PREDICT THE FUTURE »

29

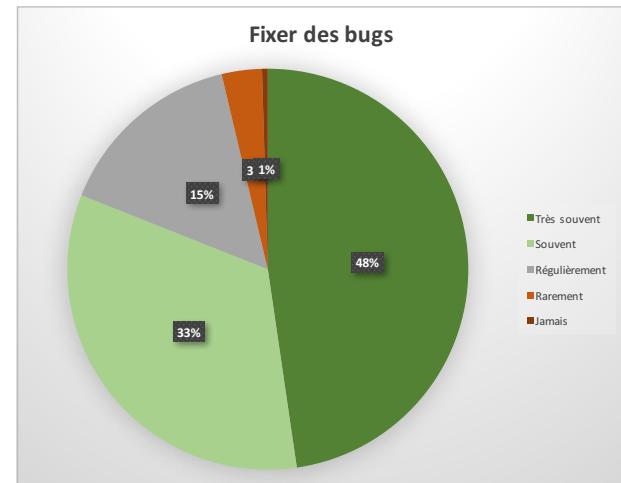


30

83

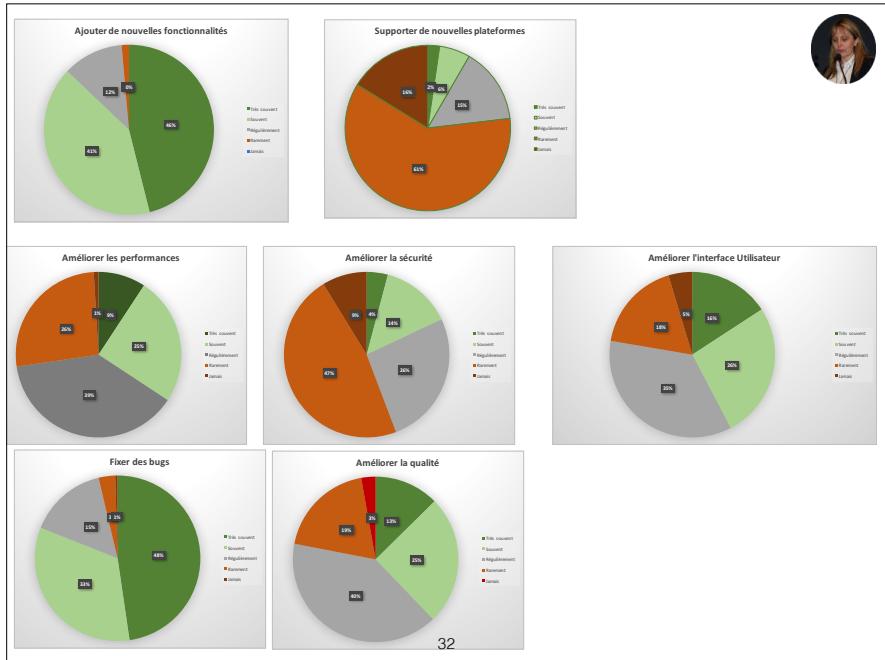


Fixer des bugs



31

84

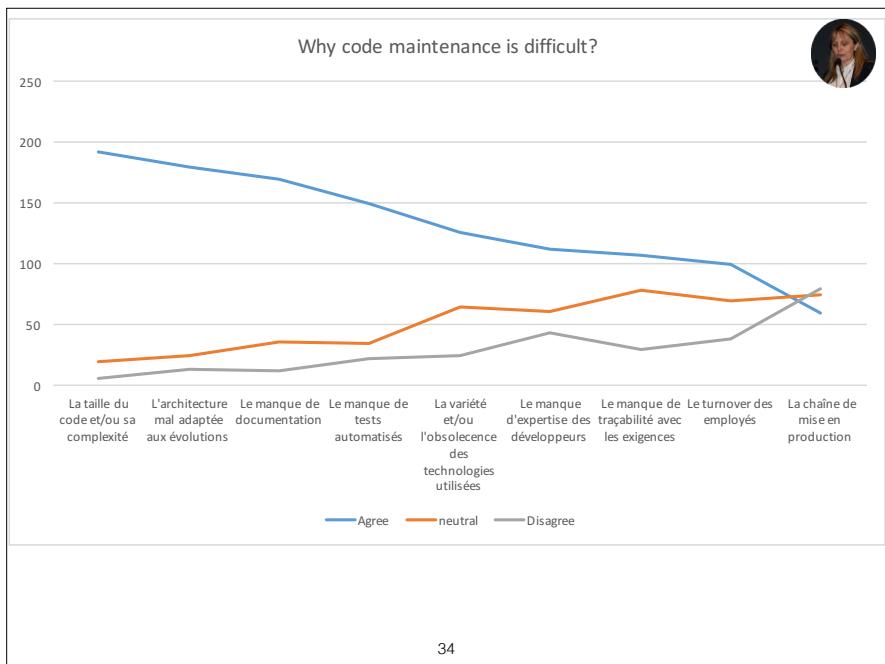


85

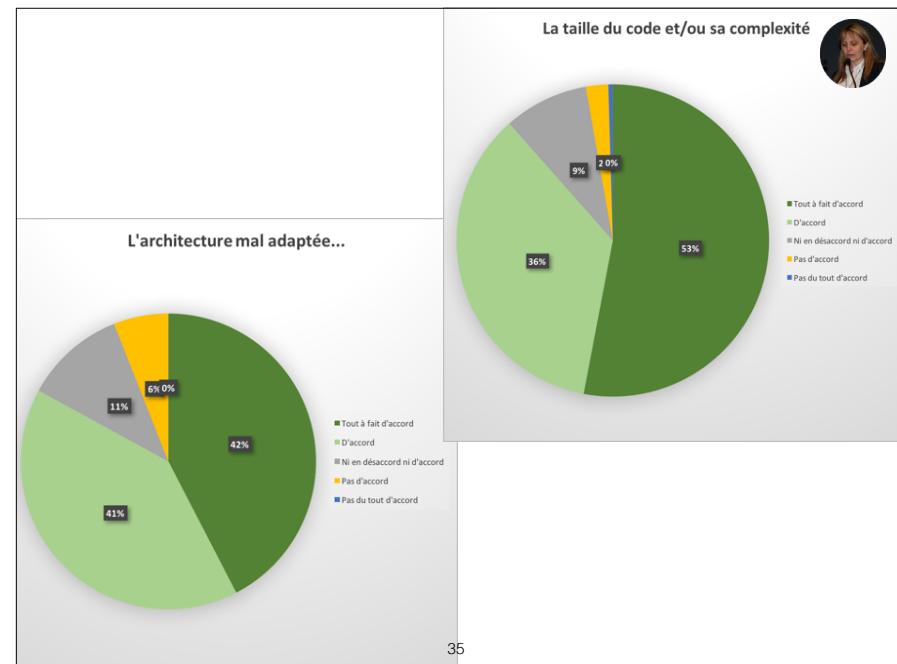
Quelles sont les difficultés ?

Sondage réalisé en 2018

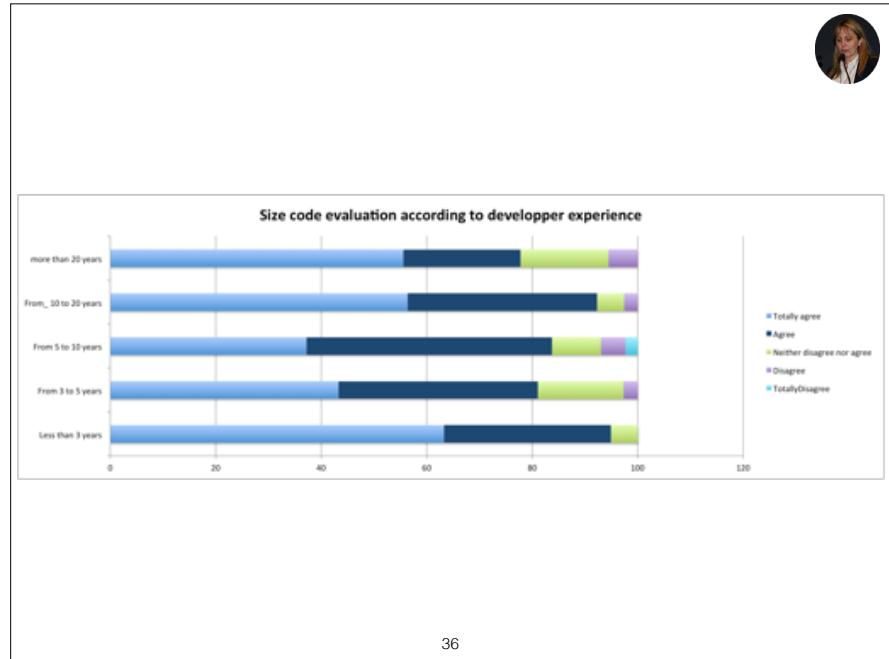
86



87

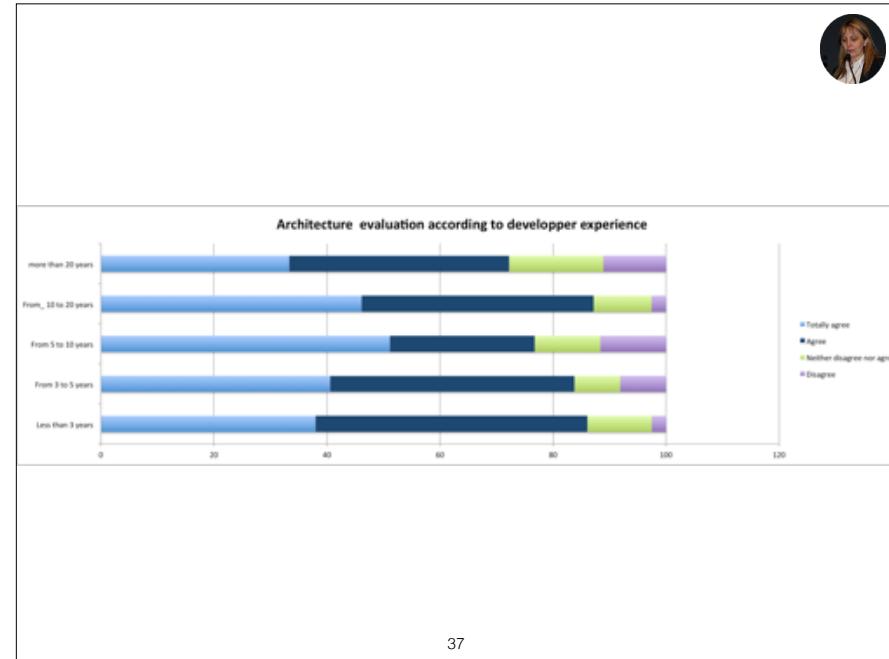


88



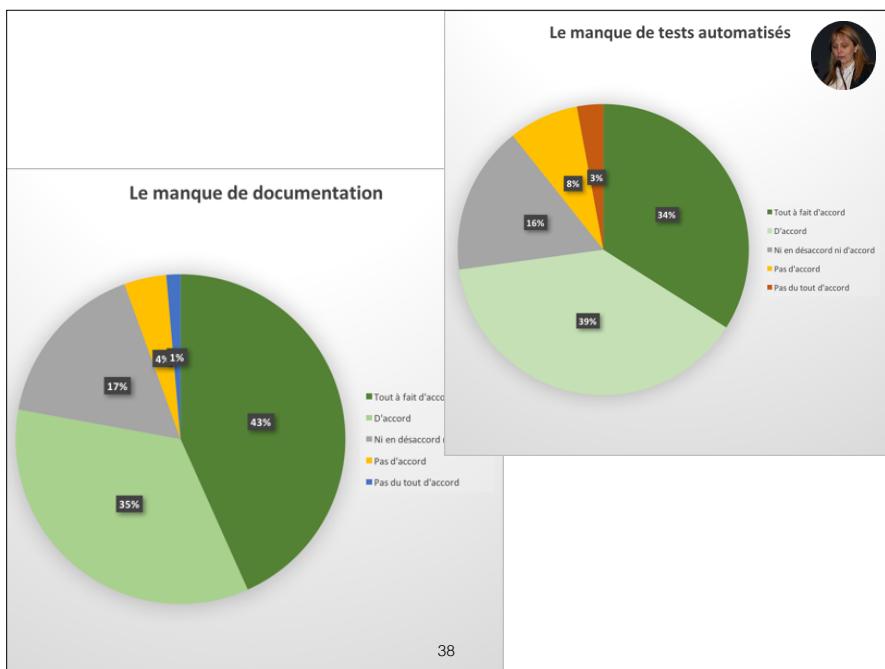
36

89



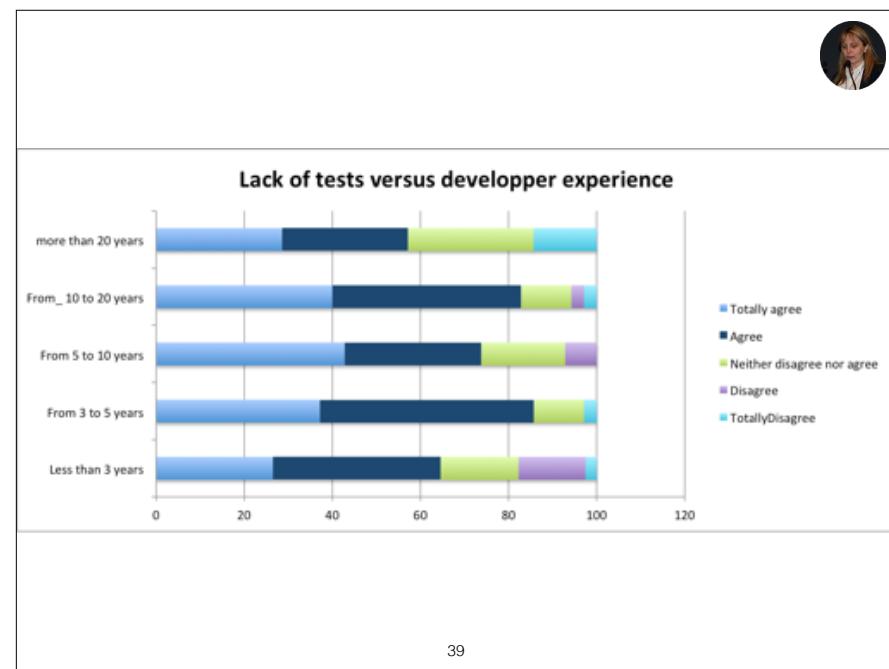
37

90



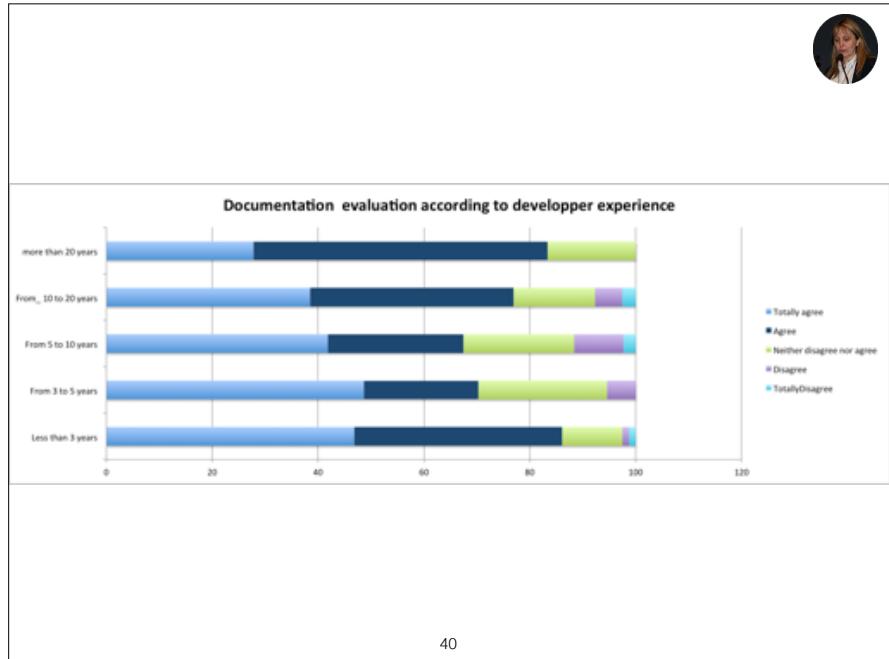
38

91

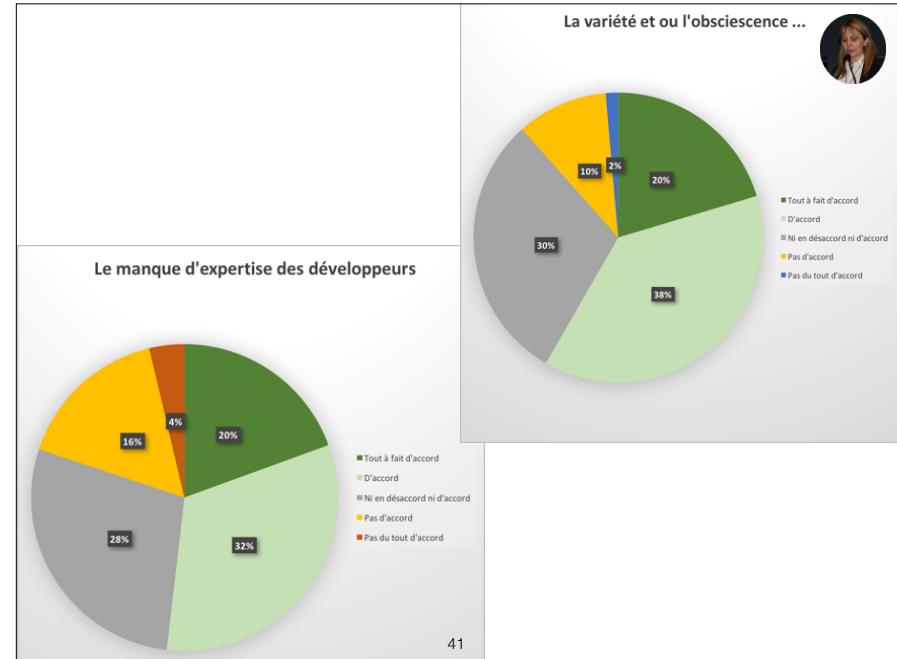


39

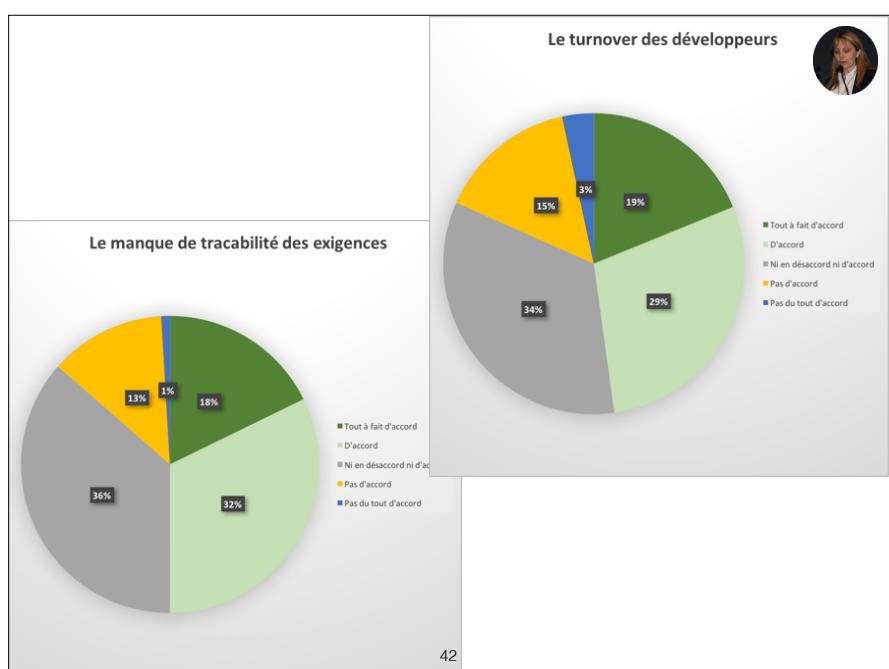
92



93

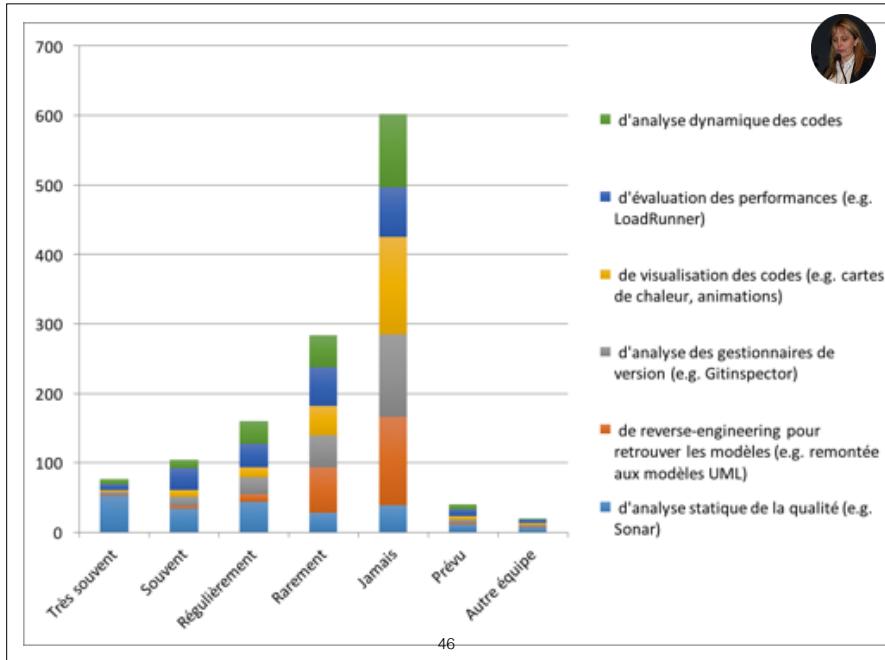


94

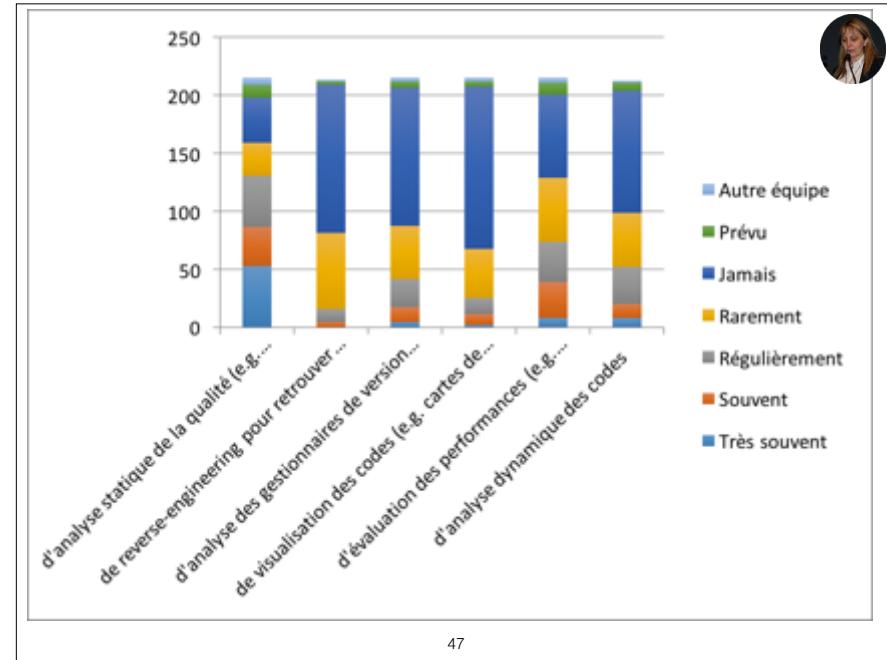


95

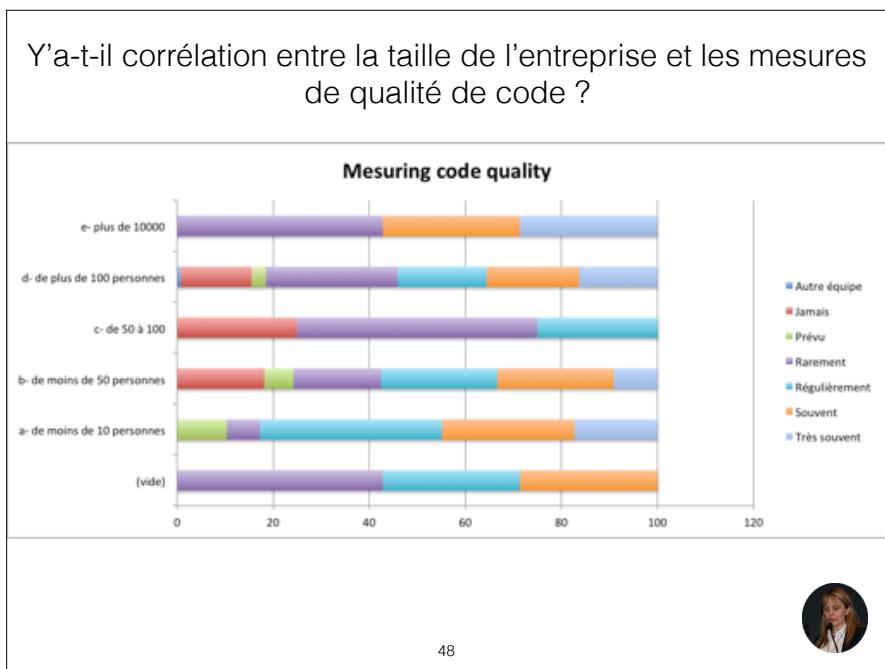




97



98



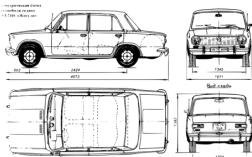
99

Modéliser ?



Pourquoi Modéliser ?

Spécifier la structure et le comportement d'un système



Aider à la construction d'un système



Visualiser un système



P. Collet

5

Documenter les décisions



101

D'où vient UML ?



- Dans les années 90 :
 - Pléthore de notations et de méthodes (OMT, Booch, ...)
- Personne n'y comprend plus rien !
 - Besoin d'un langage standard
- Création de l'entreprise Rationale
 - aka "La communauté de l'objet"
- Alliance des auteurs des méthodes existantes



Unified Modeling Language

103

Modéliser aide à Comprendre

Heliocentrism

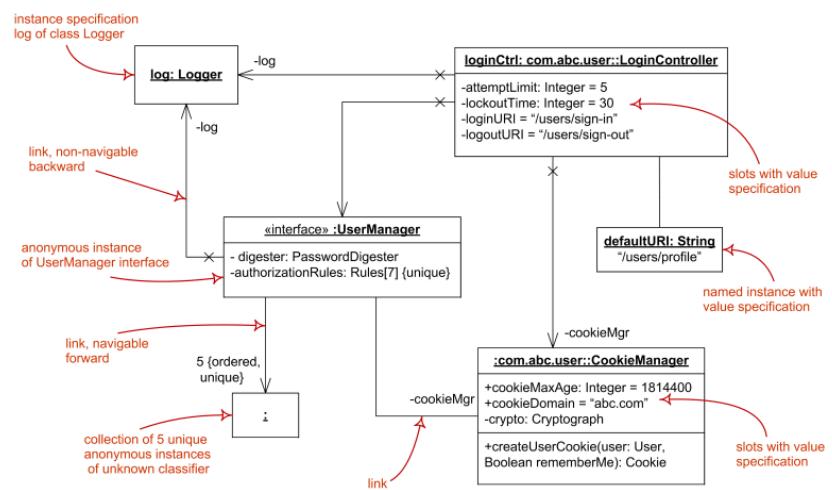


Geocentrism



102

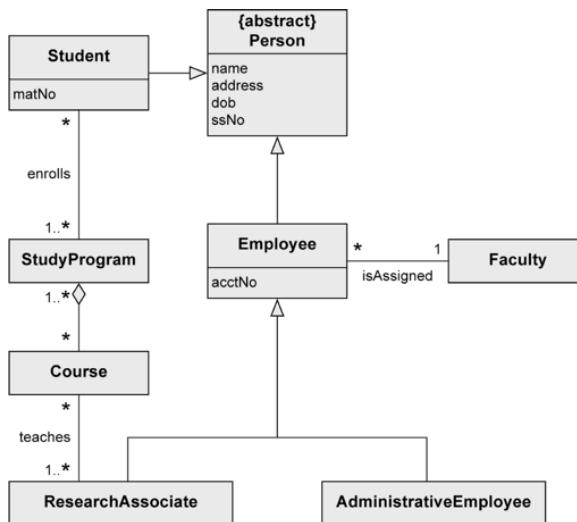
Objets



uml-diagrams.org

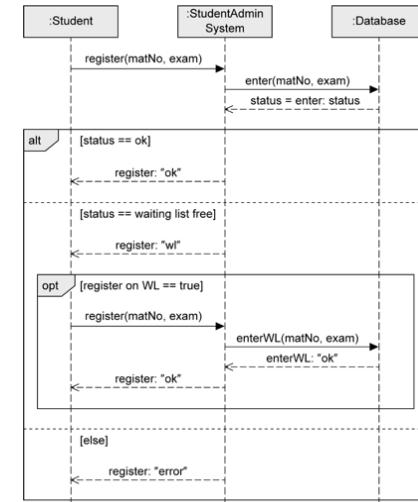
104

Classes



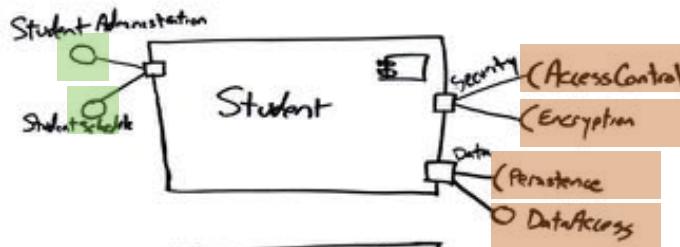
105

Séquence



106

Composants



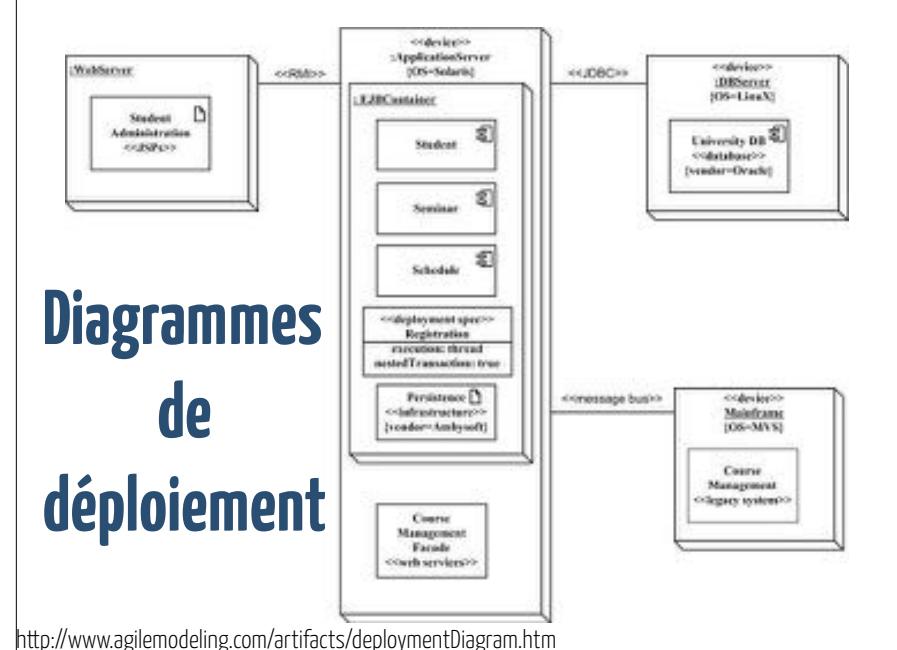
Interfaces fournies

Interfaces requises

<http://www.agilemodeling.com/artifacts/componentDiagram.htm>

107

Diagrammes de déploiement



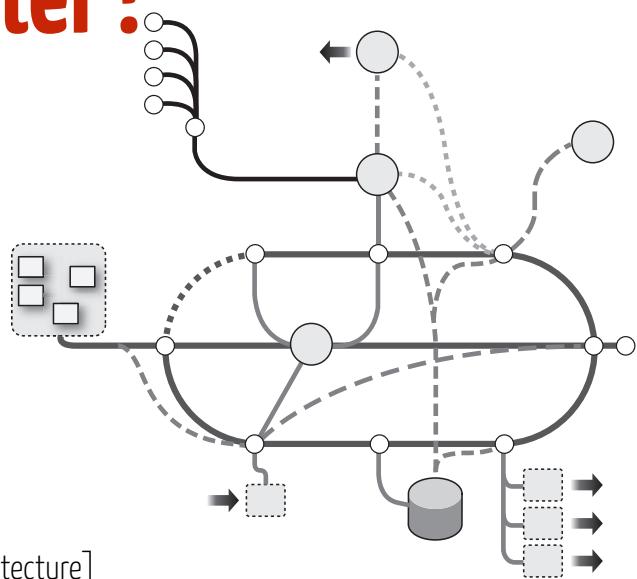
108

Modéliser ≠ UML

- Dans une certaine mesure, appliquer un patron de conception, c'est une forme de modélisation;
- Un dessin sur un tableau blanc, c'est aussi un modèle;
- Une représentation graphique d'un processus métier, c'est un modèle;
- Une visualisation, c'est un modèle;
- ...

109

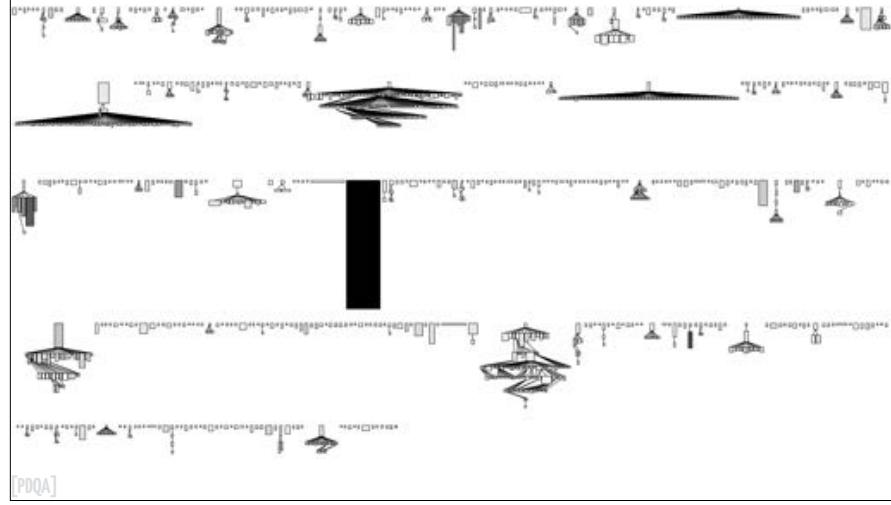
A éviter !



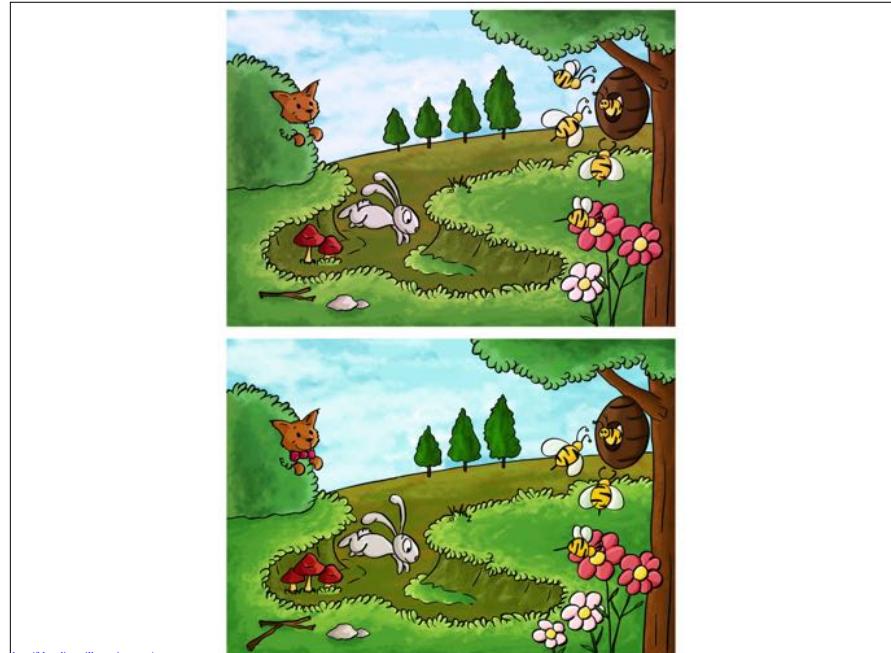
[Beautiful Architecture]

111

La visualisation rend la conception visible



110



112



113

Principe des patrons de conception

- Standardisation des concepts de modélisation ;
- Capturer l'expérience de conception ;
- Réutiliser des solutions élégantes & efficace pour des problèmes récurrents ;
- Améliorer la documentation ;
- Faciliter la maintenance.

**Le module de calcul de taux d'intérêt ?
C'est une "Stratégie"**

115



114

“The Code is the truth”

- Les modèles utilisés ne doivent pas être décoléré du code
- Le code doit respecter les principes fondamentaux du paradigme de programmation utilisé
- Par exemple en orienté-objet :
 - Patrons GRASP,
 - Principes SOLID
- Il existe des “bonnes pratiques” propres à chaque paradigme
 - Procédural, Objet, Fonctionnel, Logique, Réactif, ...

(attention aux mélanges)

116

Projet Technique (réalisation)



117

Domaine fonctionnel : produits bancaires

- Une application constituée de 3 modules
 - *Une application console pour les conseillers en agence*
 - *Une application console pour les clients*
 - *Une partie arrière commune pour gérer le SI de la banque*
- Ce n'est pas un cours d'architecture logicielle, ni de POO. Même si la qualité est évaluée, vous DEVEZ faire des choix de simplification pour uniquement démontrer ce qui à de la valeur
 - *Typiquement, pas de base de données, on mettra un bouchon*

119

Objectif : Mise en oeuvre d'un pipeline



L'application est "simple", mais il n'y a pas qu'elle à développer

<https://dzone.com/articles/observability-vs-monitoring>

118

Environnement de développement

- Mettre en place un environnement adapté, qui porte sur:
 - Gestion des exigences : *backlog traçable*
 - Gestion de version : modèle de branche / livraison
 - Stratégies de Tests : unitaires, intégration, acceptation
 - Intégration continue : pipeline d'integration
 - Déploiement continu : fabrication d'images prête à l'emploi

120

Critères d'évaluation

- Sur l'application : (*livraison : code*)
 - Respect des exigences
 - Couverture fonctionnelle
 - Qualité du code développé
- Sur l'environnement (*livraison : dépôt + rapport*)
 - Qualité des exigences / trace au code
 - Qualité des plans de tests / plans de construction
 - Analyse de synthèse des approches mises en oeuvre

Ne soyez pas dogmatique !

121

Objectifs

- Situation professionnelle :
 - analyse de viabilité d'une **tierce maintenance applicative**.
 - Répondre à la question : On prend le contrat, ou pas ?
- Voir de quels moyens dispose un développeur pour analyser un logiciel, sur différentes dimension (équipe, code, tests, architecture, déploiement, ...) ?
- Mener une telle analyse sur un projet de grande envergure

123

Projet Individuel (maintenance)



122

**C'est
difficile !**

(mais pas impossible)

124

Mais c'est une bonne manière d'approcher l'enseignement de la maintenance



Initiative
RIMEL
(~40 profs FR)



Enseigner la rétro-ingénierie, en s'interrogeant sur l'évolution du logiciel : retour d'expériences*

Mireille Blay-Formarno¹, Sébastien Mosser¹, et Xavier Blanc²

¹ Université Paris-Est, L'Atour

Laboratoire IRS

person.nobuaki.fr

² Université de Bordeaux,

L2S, bordeaux-inp.fr

Xavier.Blanc@l2s.fr

Résumé

Nous avons expérimenté cette année au niveau M2 un enseignement sur la rétro-ingénierie d'applications logicielles. Dans cet article, nous partageons les objectifs de cet enseignement, explications de la démarche mise en place et conclusion par une rétrospective.

1 Introduction

La maintenance du logiciel est connue pour être la phase la plus coûteuse de tout projet logiciel [9]. Quelles que soient les raisons de cette maintenance adaptative, prédictive ou préventive [8], elle requiert de « comprendre » le système à maintenir. Cette étape peut être facilitée en utilisant une approche de rétro-ingénierie (réécriture logicielle) qui vise à analyser les logiciels pour identifier leurs erreurs et à les corriger sans créer de risques supplémentaires. Un tel projet sera donc forcément orienté sur à des niveaux d'abstraction plus élevés [2, Chapitre 5]. Cette étape fondamentale repose non seulement sur l'étude des codes, mais également des artefacts adjacents comme la documentation, l'histoire du logiciel, l'expertise des développeurs et les tests unitaires. Cet article présente une première partie où nous décrivons la démarche mise en œuvre avec le développement des projets open source, la démonstration de plates-formes de développement courantes telles que GitHub, le développement d'approches empiriques et l'utilisation d'outils de fouilles de données (cf. [1] et sa web version [10]).

Dans ce contexte de mutation des méthodes de développement et de production du logiciel, devenue en question des règles de gestion de l'évolution du logiciel [7], de nouvelles approches de rétro-ingénierie [1] il a été décidé de proposer aux étudiants de M2 de la filière « Architectures Logicielle » un enseignement sur la rétro-ingénierie. Nous présentons dans cette partie les objectifs de cet enseignement d'une approche pratique, nous avons donc fait le choix d'un enseignement de la rétro-ingénierie dirigé par des études de cas et plus spécifiquement des questionnements autour de l'évolution du logiciel. Ce travail a été mené dans le cadre d'un cours de 20h. Il a été nécessaire à l'issue de ce cours de faire une rétrospective pour répondre à cette question : à quelle manière de rétro-ingénierie ?

Dans cet article, nous partageons cette expérience d'enseignement*. Par ce retour d'expérience, et les témoignages que nous souhaitons occasionner nous avons pour ambition d'améliorer notre pédagogie et de proposer des meilleures pratiques pour enseigner l'art de la rétro-ingénierie du logiciel.

Cet article présente une première partie où nous expliquons les objectifs, la démarche et la mise en œuvre de ce module d'enseignement, puis dans un deuxième temps nous faisons une rétrospective sur cette expérience.

*Merci à Philippe Collot, Nasred Mohs, Simon Uri, Yves Roudier pour leur aide dans la mise en œuvre de ce module.

1. Tous les artefacts associés à cet enseignement sont disponibles sur le site du cours [1].

125



127

Travail à Faire

- Choisir un projet open-source pour lequel vous avez de l'intérêt
 - Si vous êtes à sec d'idée, discutez avec moi en cours ou Slack!
- Pratiquez une analyse en détail de ce projet
 - En écrivant des programmes qui analysent le dépôt de code;
 - En utilisant des outils de l'état de la pratique;
 - En croisant les informations pour mener une réflexion de synthèse sur la "viabilité" de ce logiciel
- L'important est de bien justifier votre méthodologie d'analyse
 - Échangez régulièrement avec moi sur Slack!

126