Quelques exemples de MAUVAISE conception

Problèmes ?

Lisibilité ?

Adéquation aux spécifications ?

Maintenabilité ?

Testabilité ?

Extension pour couvrir les spécifications ?

Dette technique !

La règle des 3 en génie logiciel

Lisibilité

Lisibilité

Lisibilité

Pourquoi ?

- Parce que le **cerveau** assimile par **deux canaux d'entrée** principaux, visuel et verbal
- Chaque canal d'entrée possède une **capacité limitée d'assimilation**

- **La mémoire de travail** permet de conserver **plus ou moins 7 informations** de manière **simultanée**
- La zone tampon est extrêmement limitée dans le temps (30 secondes environ)

Et un développeur ?

*« Our study finds that on average developers spend **~58%** of their time on program comprehension activities »*

Measuring Program Comprehension: A Large-Scale Field Study with Professionals Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, Shanping Li . IEEE Transactions on Software Engineering, 2017 July, Volume PP, Issue 99, Pages 1-26

Reading vs Writing Code

“Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”

— Robert C. Martin, **Clean Code: A Handbook of Agile Software Craftsmanship**

Qu’est-ce qui pourrait empêcher du code d’être peu compréhensible ?

Root Cause	# Sessions
No comments or insufficient comments	92 (46%)
Meaningless classes/methods/variables names	75 (38%)
Large number of LOC in a class/method	63 (32%)
Unconsistent coding styles	42 (21%)
Navigating inheritance hierarchies	
Query refinement	83 (42%)
Query Refinement, and browsing a number of search results/links	42 (21%)
Lack of documents, and ambiguous/incomplete docu- ment content	79 (40%)
Searching for the relavent documents	12 (6%)
Unfamiliarity with business logic	NA

Measuring Program Comprehension: A Large-Scale Field Study with Professionals Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, Shanping Li . IEEE Transactions on Software Engineering, 2017 July, Volume PP, Issue 99, Pages 1-26

La Pierre de la Sagesse

UN LIVRE DONT

VOUS ETES LE HEROS

```java
public static boolean isBrelan(Hand main){
    //Récupérer les cartes de la main et les enregistrées dans la liste
    List<Carte> cartes1;

    cartes1 = new ArrayList<>();
    for (int j = 0; j < 5; j++){
        cartes1.add(main.getMesCartes().get(j));
    }
    int compteur=1;
    //Prendre les cartes dans l'ordre des saisis une à une
    for(int i=0;i<5;i++){
        // Les comparer aux cartes suivantes
        for (int p=i+1;p<5;p++){
            //Si on a la même valeur incrémenter le compteur
            if (cartes1.get(i).getValue().equals(cartes1.get(p).getValue())){
                compteur+=1;
            }

        }
        //Si on a assez de cartes pour former le brelan renvoyer True sinon réintialiser le compteur
        if (compteur==3){
            return true;
        }
        // Si on a un carré il faut le renvoyer à la première identification, sinon à cause de la méthode de
        // la méthode de parcours on risque de transformer un carré en brelan.
        if (compteur==4){
            return false;
        }
        else{
            compteur=1;
        }
    }
    return false;
}
```

```
public static boolean isDoublePaire(Hand main){
    int[] carte1 = new int[5];
    for (int j = 0; j < 5; j++){
        if (main.getMesCartes().get(j).getValue() == "V") {carte1[j]=11;}
        else if (main.getMesCartes().get(j).getValue() == "D") {carte1[j]=12;}
        else if (main.getMesCartes().get(j).getValue() == "R") {carte1[j]=13;}
        else if (main.getMesCartes().get(j).getValue() == "A") {carte1[j]=14;}
        else {carte1[j]= Integer.parseInt(main.getMesCartes().get(j).getValue());}
    }
    int i = 1;
    for (i = 1; i <4;i++){
        if (carte1[0]==carte1[i]){
            carte1=remove(0,i,carte1);
            i=5;
            if (carte1[0]==carte1[1] || carte1[1] == carte1[2] || carte1[0]==carte1[2]){
                return true;
            }
            else {
                return false;
            }
        }

    }
```

La suite !

```
    if (i==4){
        for (int k = 2; k<5;k++){
            if (carte1[1]==carte1[k]){
                carte1 = remove(1,k,carte1);
                k=5;
                if (carte1[1]==carte1[2]){
                    return true;
                }
                else {
                    return false;
                }
            }
            else{
                return false;
            }
        }
    }
    return false;
}
```

```java
public void jouer(){
    boolean ok1 = false;
    boolean ok2 = false;
    Scanner scanner = new Scanner(System.in);

    while(!ok1) {
        ok1 = traitementMain(1, scanner);
    }

    // ne demander la saisie de la deuxième main que si on a bien saisie la 1ère
    if(ok1){
        while(!ok2) {
            ok2 = traitementMain(2, scanner);
        }
    }
```

```
// si le saisie des deux mains est bien fait
if(ok1 & ok2)
{
    calculeForceMain(main1);
    calculeForceMain(main2);

    // si les deux cartes possédent le même critére (carré, brelan ...)
    // il faut appeler les méthodes de calcule pour décider qui est la main gagnante
    if (compareForce(main1,main2) == null){
        int force = main1.getForce();
        switch(force){
            case 0:
                mainGagnante=Hand.calculeMainLaPlusForte(main1,main2);
                break;
            case 1:
                mainGagnante=Hand.calculePair(main1,main2);
                break;
            case 2:
                mainGagnante=Hand.calculeDoublePaire(main1,main2);
                break;
            case 3:
                mainGagnante=Hand.calculeBrelan(main1,main2);
                break;
            case 4:
                mainGagnante=Hand.calculeStraight(main1,main2);
                break;
            case 5:
                mainGagnante=Hand.calculeColor(main1,main2);
                break;
            case 6:
                mainGagnante=Hand.calculeFull(main1,main2);
                break;
            case 7:
                mainGagnante=Hand.calculeCarre(main1,main2);
                break;
            case 8:
                mainGagnante=Hand.calculeQuinte(main1,main2);
                break;
        }
    }
```

```
    }
    else{
        mainGagnante=compareForce(main1,main2);
    }
    if(mainGagnante==null){
        System.out.println("Egalité !");
    }
    else {
        int i = 2;
        if(mainGagnante.equals(main1)) i = 1;

        // faire l'affichage
        switch(mainGagnante.getForce()){
            case 0:
                printMainLaPlusForte(i, mainGagnante);
                break;
            case 1:
                printPair(i, mainGagnante);
                break;
            case 2:
                printDoublePaire(i, main1, main2);
                break;
            case 3:
                printBrelan(i, mainGagnante);
                break;
            case 4:
                printStraight(i, mainGagnante);
                break;
            case 5:
                printColor(i, mainGagnante);
                break;
```

```java
public boolean traitementMain(int i, Scanner scanner){
    boolean ok = true;
    String chaineSaisie;
    // récupérer la saisie des mains
    if(i == 1){
        System.out.println("Bonjour, veuillez saisir la 1ere main");
    }
    else
    {
        System.out.println("veuillez saisir la 2ème main");
    }

    chaineSaisie = scanner.nextLine();

    // diviser la chaine en sous chaines et vérifier si les cartes sont bien saisies
    List<String> sousChaines = new ArrayList<>();
    StringTokenizer tok = new StringTokenizer(chaineSaisie," ");

    while (tok.hasMoreTokens()) {
        String sousChaine = tok.nextToken();
        sousChaines.add(sousChaine);
    }

    // tester le nombre des chaines saisies
    if(sousChaines.size() == 5)
    {
```

```
// en testant si la carte existe déjà dans le jeu ou pas
for(String s : sousChaines){
    if(s.length() >= 3 && s.length() <= 4)
    {
        String valeurS = (String) s.subSequence(0, s.length()-2);
        String couleurS = (String) s.subSequence(s.length()-2, s.length());

        //tester si la carte existe dans le jeu ou non
        if(jeu.trouveCarte(valeurS, couleurS) != null){
            if(i == 1){
                main1.addCarte(jeu.trouveCarte(valeurS, couleurS));
            }
            else
            {
                main2.addCarte(jeu.trouveCarte(valeurS, couleurS));
            }
            jeu.removeCarte(valeurS, couleurS);
        }
        else
        {
            System.out.println("La carte "+s+" est inconnue ou dupliquée");

            if(i == 1){
                for(Carte c : main1.getMesCartes()){
                    jeu.addCarte(c);
                }
            }
            else
            {
                for(Carte c : main2.getMesCartes()){
                    jeu.addCarte(c);
                }
            }

            return false;
        }
    }
```

```
public void calculeForceMain(Hand main){
    if(Hand.isQuinte(main)){
        main.setForce(8);
        return ;
    }
    if(Hand.isCarre(main)){
        main.setForce(7);
        return ;
    }
    if(Hand.isFull(main)){
        main.setForce(6);
        return ;
    }
    if(Hand.isColor(main)){
        main.setForce(5);
        return ;
    }
    if(Hand.isStraight(main)){
        main.setForce(4);
        return ;
    }
    if(Hand.isBrelan(main)){
        main.setForce(3);
        return ;
    }
```

```
public void printDoublePaire(int l, Hand main1, Hand main2) {
    if (Hand.isDoublePaire(main1) && Hand.isDoublePaire(main2)) {
        int[] carte1 = new int[5];
        int[] carte2 = new int[5];
        int[] ordreMain1 = new int[3];
        int[] ordreMain2 = new int[3];
        for (int j = 0; j < 5; j++) {
            if (main1.getMesCartes().get(j).getValue() == "V") carte1[j] = 11;
            else if (main1.getMesCartes().get(j).getValue() == "D") carte1[j] = 12;
            else if (main1.getMesCartes().get(j).getValue() == "R") carte1[j] = 13;
            else if (main1.getMesCartes().get(j).getValue() == "A") carte1[j] = 14;
            else carte1[j] = Integer.parseInt(main1.getMesCartes().get(j).getValue());
        }
        for (int i = 1; i < 5; i++) {
            if (carte1[0] == carte1[i]) {
                ordreMain1[0] = carte1[0];
                carte1 = remove(0, i, carte1);//remove c0 et ci

                if (carte1[0] == carte1[1]) {
                    if (carte1[0] <= ordreMain1[0]) {
                        ordreMain1[1] = ordreMain1[0];
                        ordreMain1[0] = carte1[0];
                    } else {
                        ordreMain1[1] = carte1[0];
                    }
                    carte1 = remove(0, 1, carte1);
                    //[double1,double2,last one]
                    ordreMain1[2] = carte1[0];
                    i = 5;
                } else if (carte1[1] == carte1[2]) {

                    if (carte1[1] <= ordreMain1[0]) {
                        ordreMain1[1] = ordreMain1[0];
                        ordreMain1[0] = carte1[1];
                    } else {
                        ordreMain1[1] = carte1[1];
                    }
                    carte1 = remove(1, 2, carte1);
                    //add c0(last one) to ordreMain in last [double1,double2,last one]
                    ordreMain1[2] = carte1[0];
                    i = 5;
```

```
            } else if (carte1[0] == carte1[2]) {
                if (carte1[0] <= ordreMain1[0]) {
                    ordreMain1[1] = ordreMain1[0];
                    ordreMain1[0] = carte1[0];
                } else {
                    ordreMain1[1] = carte1[0];
                }
                carte1 = remove(0, 2, carte1);
                //[double1,double2,last one]
                ordreMain1[2] = carte1[0];
                i = 5;
            }
        }
    }
    if (carte1.length == 5) {
        for (int k = 2; k < 5; k++) {
            if (carte1[1] == carte1[k]) {
                //add c1 to ordreMain in ordre
                ordreMain1[0] = carte1[1];
                carte1 = remove(1, k, carte1);
                if (carte1[1] <= ordreMain1[0]) {
                    ordreMain1[1] = ordreMain1[0];
                    ordreMain1[0] = carte1[1];//remove c1 et ci
                    k = 5;
                    ordreMain1[2] = carte1[0];
                } else {
                    ordreMain1[1] = carte1[1];
                    k = 5;
                    ordreMain1[2] = carte1[0];
                }

            }

        }
    }
```

```
for (int j = 0; j < 5; j++) {
    if (main2.getMesCartes().get(j).getValue() == "V") {
        carte2[j] = 11;
    } else if (main2.getMesCartes().get(j).getValue() == "D") {
        carte2[j] = 12;
    } else if (main2.getMesCartes().get(j).getValue() == "R") {
        carte2[j] = 13;
    } else if (main2.getMesCartes().get(j).getValue() == "A") {
        carte2[j] = 14;
    } else {
        carte2[j] = Integer.parseInt(main2.getMesCartes().get(j).getValue());
    }
}
for (int i = 1; i < 5; i++) {
    if (carte2[0] == carte2[i]) {
        ordreMain2[0] = carte2[0];
        carte2 = remove(0, i, carte2);
        if (carte2[0] == carte2[1]) {
            if (ordreMain2[0] >= carte2[0]) {
                ordreMain2[1] = ordreMain2[0];
                ordreMain2[0] = carte2[0];
            } else {
                ordreMain2[1] = carte2[0];
            }
            carte2 = remove(0, 1, carte2);
            //[double1,double2,last one]
            ordreMain2[2] = carte2[0];
            i = 5;
        } else if (carte2[1] == carte2[2]) {
            if (carte2[1] <= ordreMain2[0]) {
                ordreMain2[1] = ordreMain2[0];
                ordreMain2[0] = carte2[1];
            } else {
                ordreMain2[1] = carte2[1];
            }
            carte2 = remove(1, 2, carte2);
            ordreMain2[2] = carte2[0];
            i = 5;
        } else if (carte2[0] == carte2[2]) {
            if (carte2[0] <= ordreMain2[0]) {
                ordreMain2[1] = ordreMain2[0];
                ordreMain2[0] = carte2[0];
            } else {
                ordreMain2[1] = carte2[0];
            }
            carte2 = remove(0, 2, carte2);
            //add c0(last one) to ordreMain in last [double1,double2,last one]
            ordreMain2[2] = carte2[0];
            i = 5;
        }
    }
```

```
            if (carte2.length == 5) {
                for (int k = 2; k < 5; k++) {
                    if (carte2[1] == carte2[k]) {
                        ordreMain2[2] = carte2[0];
                        ordreMain2[0] = carte2[1];
                        carte2 = remove(1, k, carte2);
                        if (carte2[1] <= ordreMain2[0]) {
                            ordreMain2[1] = ordreMain2[0];
                            ordreMain2[0] = carte2[1];
                            ordreMain2[2] = carte2[0];
                            k = 5;
                        } else {
                            ordreMain2[1] = carte2[1];
                            k = 5;
                            ordreMain2[2] = carte2[0];
                        }
                    }
                }
            }
            if (compareForce(main1, main2) == null) {
                if (ordreMain1[1] < ordreMain2[1]) {
                    System.out.println("La main " + l + " gagne grace à une double paire avec la paire " + ordreMain2[1]);
                } else if (ordreMain1[1] > ordreMain2[1]) {
                    System.out.println("La main " + l + " gagne grace à une double paire avec la paire " + ordreMain1[1]);
                } else {
                    if (ordreMain1[0] < ordreMain2[0]) {
                        System.out.println("La main " + l + " gagne grace à une double paire avec la paire " + ordreMain2[0]);
                    } else if (ordreMain1[0] > ordreMain2[0]) {
                        System.out.println("La main " + l + " gagne grace à une double paire avec la paire " + ordreMain1[0]);
                    } else {
                        if (ordreMain1[2] < ordreMain2[2]) {
                            System.out.println("La main " + l + " gagne grace à une double paire avec la carte la plus forte " + ordreMain2[2]);
                        } else if (ordreMain1[2] > ordreMain2[2]) {
                            System.out.println("La main " + l + " gagne grace à une double paire avec la carte la plus forte " + ordreMain1[2]);
                        } else {
                            System.out.println("Egalite !");
                        }
                    }
                }
            } else System.out.println("La main " + l + " gagne avec une double paire");
        }
    }
}
```

```
public void printQuinte(int l,Hand main) {
    Carte carte = Hand.calculeCarteLaPlusForte(main);
    Hand.conversionValeurEnLettre(carte);
    System.out.println("La main " + l + " gagne avec une quinte de couleur " + carte.getColor() + " avec carte la plus élevée : " + carte);
}
```


BRACE YOURSELVES
A NEW PLAYER HAS ENTERED THE GAME
memegenerator.net

```java
public class Combination {
    private int highestCard;
    private String highestCardValueString;
    private String highestCardColorString;
    private boolean isPair = false; //done
    private int pair;
    private String pairValueString;
    private boolean isTwoPair = false; //done
    private int twoPair;
    private String twoPairValueString;
    private boolean isThreeOfAKind = false; //done
    private int threeOfAKind;
    private String threeOfAKindValueString;
    private boolean isStraight = false; //done
    private int straight;
    private String straightValueString;
    private boolean isColor = false; //done
    private int color;
    private String colorValueString;
    private boolean isFull = false; //done
    private int full;
    private String fullValueString;
    private boolean isSquare = false; //done
    private int square;
    private String squareValueString;
    private boolean isStraightFlush = false; //done
    private int straightFlush;
    private String straightFlushValueString;


    public Combination(Hand hand) {
        this.setHighestCard(hand);
        this.setPair(hand);
        this.setTwoPairNThreeOfAKind(hand);
        this.setStraight(hand);
        this.setColor(hand);
        this.setFullNSquare(hand);
        this.setStraightFlush(hand);
    }
```

```
public class Score {
    private int score;
    private String winBy;

            public class Winner {

                private int numberOfCall=0;

                public Score setScoreNWinBy(Combination combination){
                    Score score = new Score();
                    if (combination.isPair()){
                        score.addScore(1);
                        score.setWinBy("paire de "+combination.getPairValueString());
                    }
                    if (combination.isTwoPair()){
                        score.addScore(2);
                        score.setWinBy("deux paires de "+combination.getTwoPairValueString());
                    }
                    if (combination.isThreeOfAKind()){
                        score.addScore(3);
                        score.setWinBy("brelan de "+combination.getThreeOfAKindValueString());
                    }
                    if (combination.isStraight()){
                        score.addScore(4);
                        score.setWinBy("suite de "+combination.getStraightValueString());
```

```java
public String setWinner(Combination combination1, Combination combination2){
    int score1 = setScoreNWinBy(combination1).getScore();
    int score2 = setScoreNWinBy(combination2).getScore();
    String winBy1 = setScoreNWinBy(combination1).getWinBy();
    String winBy2 = setScoreNWinBy(combination2).getWinBy();
    if (score1>score2){
        return "La main 1 gagne avec "+winBy1;
    }
    else if (score2>score1){
        return "La main 2 gagne avec "+winBy2;
    }
    else{
        if (score1==0){
            if (combination1.getHighestCard()>combination2.getHighestCard()){
                return "La main 1 gagne avec carte la plus élevée : "+combination1.getHighestCardValueString()+" de "+combination1.getHighestCardColorString
            }
            else if (combination2.getHighestCard()>combination1.getHighestCard()){
                return "La main 2 gagne avec carte la plus élevée : "+combination2.getHighestCardValueString()+" de "+combination2.getHighestCardColorString
            }
            else return "Egalite";
        }
        if (score1==1){
            if (combination1.getPair()>combination2.getPair()){
                return "La main 1 gagne avec "+winBy1;
            }
            else if (combination1.getPair()<combination2.getPair()){
                return "La main 2 gagne avec "+winBy2;
            }
            else return "Egalite";
        }
        else if (score1==2){
            if (combination1.getTwoPair()>combination2.getTwoPair()){
                return "La main 1 gagne avec "+winBy1;
            }
            else if (combination1.getTwoPair()<combination2.getTwoPair()){
                return "La main 2 gagne avec "+winBy2;
            }
            else return "Egalite";
        }
```


WHAT ABOUT TESTS?
TEST
YOUR
STUPIDITY
INSERT
$100

```java
@Test
void brelanVsBrelanEgaliteSecondMaxCardTest(){

    Card card1 = new Card(1);
    Card card2 = new Card(1);
    Card card3 = new Card(1);
    Card card4 = new Card(5);
    Card card5 = new Card(3);
    Hand hand1 = new Hand(card1, card2, card3, card4, card5);
    assertEquals(CombinationType.BRELAN, hand1.getBestCombination().getType());

    Card card6 = new Card(1);
    Card card7 = new Card(1);
    Card card8 = new Card(1);
    Card card9 = new Card(5);
    Card card10 = new Card(2);
    Hand hand2 = new Hand(card6, card7, card8, card9, card10);

    assertTrue(hand1.compareTo(hand2) > 0);
}
@Test
void brelanVsBrelanEgaliteFirstMaxCardTest(){

    Card card1 = new Card(1);
    Card card2 = new Card(1);
    Card card3 = new Card(1);
    Card card4 = new Card(5);
    Card card5 = new Card(3);
    Hand hand1 = new Hand(card1, card2, card3, card4, card5);
    assertEquals(CombinationType.BRELAN, hand1.getBestCombination().getType());

    Card card6 = new Card(1);
    Card card7 = new Card(1);
    Card card8 = new Card(1);
    Card card9 = new Card(4);
    Card card10 = new Card(2);
    Hand hand2 = new Hand(card6, card7, card8, card9, card10);

    assertTrue(hand1.compareTo(hand2) > 0);
}
```

```
  static Hand hautFullHouse;
  static Hand hautFullHousePetitePaire;
  static Hand petitFullHousePetitePaire;
  static Hand petitFullHouse;

@BeforeAll
static void setup() {
        asTr = new Card(14, "tr");
        roiTr = new Card(13, "tr");
        reineTr = new Card(12, "tr");
        valetTr = new Card(11, "tr");
        hautFullHouse = new Hand(asTr, asCo, asCa, roiTr, roiCo);
        hautFullHousePetitePaire = new Hand(asTr, asCo, asCa, deuxTr, deuxCo);
        petitFullHousePetitePaire = new Hand(quatreTr, quatreCo, quatreCa, deuxTr, deuxCo);
        petitFullHouse = new Hand(quatreTr, quatreCo, quatreCa, deuxTr, deuxCo);
```

```
 @Test
 void fullHouseEvenPair() {
        assertTrue(hautFullHousePetitePaire.compareTo(petitFullHousePetitePaire) > 0);
 }

 @Test
 void fullHouseEvenTriple() {
        assertTrue(hautFullHouse.compareTo(hautFullHousePetitePaire) > 0);
 }
```


SO WHAT?
SQUID
GAME
imgflip.com

Cohésion

Degré pour lequel les différentes données gérées dans une classe ou une fonction sont reliées entre elles. C'est le degré de corrélation des données entre elles

- **Augmenter la cohésion du code**

Des méthodes courtes qui touchent à un petit nombre d'attributs…

Couplage

Nombre de liens entre les données de classes ou fonctions différentes.

Métrique de couplage : le nombre de références ou d'appels fait à l'objet d'un autre type

- **Diminuer le couplage**

Distribuer les responsabilités dans les objets

S.O.L.I.D.

- Single responsibility principle (SRP) : une classe n'a qu'une seule responsabilité
- Open/closed principle (OCP) : un élément logiciel (classe ou méthode) doit être ouvert à l'extension mais fermé à la modification
- Liskov substitution principle (LSP) : les objets d'un programme doivent pouvoir être remplacés par des instances de leurs sous-types sans «casser» le programme
- Interface segregation principle (ISP) : il faut privilégier plusieurs interfaces spécifiques à des besoins clients
- Dependency inversion principle (DIP) : il faut dépendre des abstractions, pas des réalisations concrètes

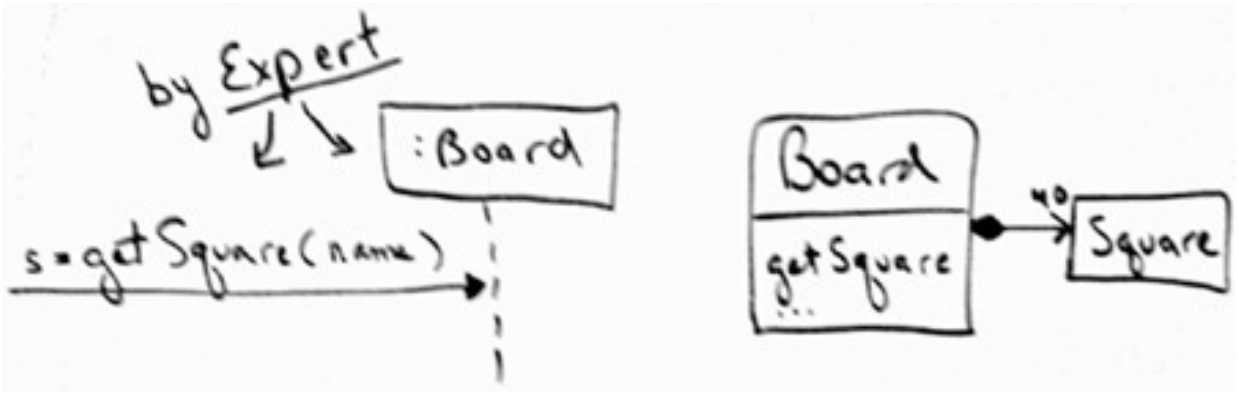
GRASP?

General
Responsibility
Assignment
Software
Pattern

Patrons logiciels généraux d'affectation des responsabilités

- *De **faire***
- *De **savoir***

GRASP 1. Spécialiste de l'information



- L'accomplissement d'une responsabilité nécessite souvent que l'information nécessaire soit répartie entre différents objets

Applying UML and Patterns – Craig Larman

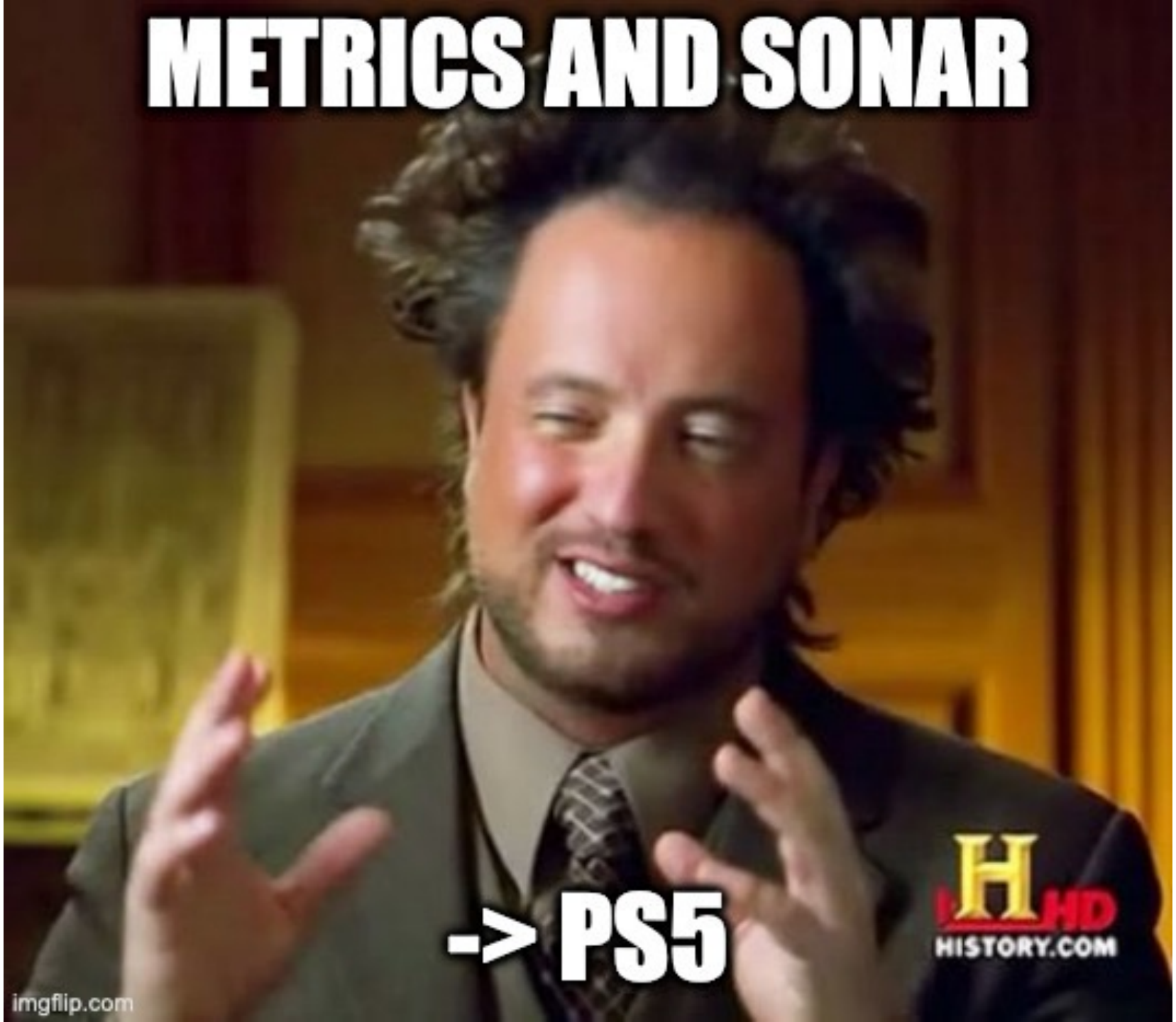
Dans la main de poker, qui a la responsabilité de contrôler la saisie des mains ? De les comparer ?

Qui a celle de déterminer ce qu'on contient une main ?


SOLID AND GRASP
LIGHTYEAR
-> POO
imgflip.com

```
void myMethod () {
  try {
    if (condition1) {                        // +1
      for (int i = 0; i < 10; i++) {         // +2 (nesting=1)
        while (condition2) { … }             // +3 (nesting=2)
      }
    }
  } catch (ExcepType1 | ExcepType2 e) {      // +1
    if (condition2) { … }                    // +2 (nesting=1)
  }
}                                            // Cognitive Complexity 9
```

SonarSource Cognitive Complexity – A new way of measuring understandability by G. Ann Campbell, SonarSource SA 2016


METRICS AND SONAR
-> PS5
HISTORY.COM
imgflip.com