SOLID

Uma introdução ao conjunto de princípios fundamentais para

O desenvolvimento de software saudável

Princípios solidos

Princípios sólidos

Organização

Objetivos do

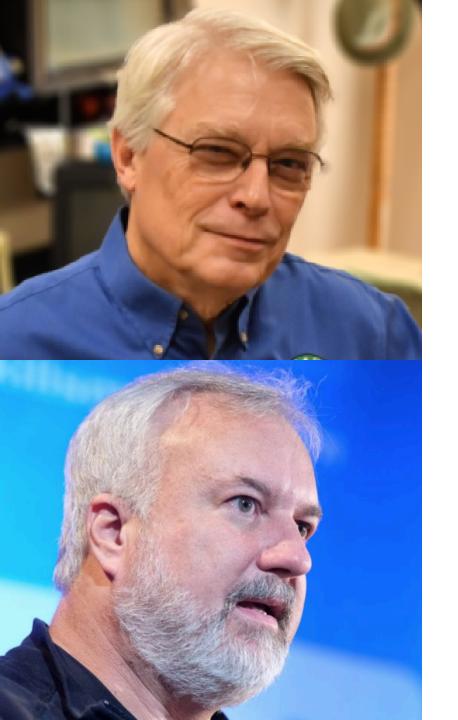
SOLID

- Tolerar mudanças
- Sejam faceis de entender
- Sejam a base de código que possa ser usada em muitos sistemas de software

Breve

SOLID

história



Robert C Martin (signatário do Manifesto Agil, autor do da série de livros codigo limpo e desenvolvedor desde a década de 70), vem escrevendo sobre qualidade de software a muito tempo e em 2000 já havia estabelecido um conjunto de princípios e práticas em seus trabalhos e publicações.

Foi ai que em 2004, Michael Feathers (um importante e antigo desenvolvedor da comunidade C++) percebeu que se reorganizasse os princípios, as primeiras letras de cada principio poderiam formar a palavra SOLID.

Assim nasceu os princípios SOLID.

SOLID

Single

Responsability

Principle

SRP

Principio

Responsabilidade

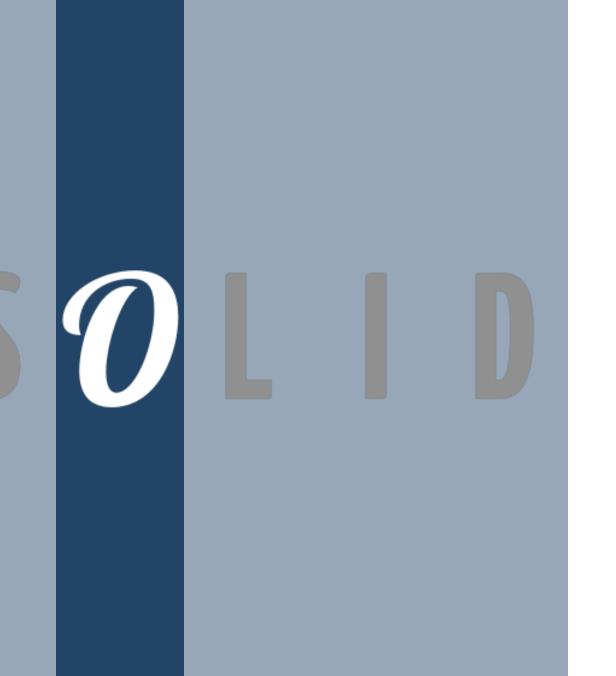
Unica

SRP

```
public class Robot {
   void cook() { ... }
   void text() { ... }
   void state() { ... }
   void image() { ... }
   void video() { ... }
   void youtube() { ... }
}
```

SRP

```
public class RobotChef {
 void cook() { ... }
public class RobotGarderner {
 void clean() { ... }
public class RobotPainter {
 void paint() { ... }
public class RobotDriver {
 void drive() { ... }
```



Open-Closed

Principle

Princípio

(1141) Aberto-Fechado

OCP

```
public class Robot {
 void exec() {
   this.skill.cut();
public class Robot {
 void exec() {
   this.skill.paint();
```

OCP

```
public class Robot {
   Robot(Array<Skil>l skills){
    this.skills = skills;
   }
   void exec() {
    for (Skill skill: this.skills){
       skill.exec();
    }
   }
}
```

OCP

```
public class CookSkill
  implements Skill {
  void exec() { ... }
public class PaintSkill
  implements Skill {
  void exec() { ... }
// ... more skills
// using
Robot robot = new Robot([
  new CookSkill(),
  new PaintSkill()
]);
```

Liskov

Substitution

Principle

Princípio

LSP

Substituição

Liskov

LSP

LSP

```
public class Sam {
  Coffee cook() { ... }
}

public class Eden extends Sam {
  Water cook() { ... }
}
```

LSP

```
public class Sam {
   Coffee cook() { ... }
}

public class Eden extends Sam {
   Coffee cook() {
        // cappucino is a Coffee type
        return cappucino;
   }
}
```

Interface

Segregation

Principle

Princípio

ISP Segregação

Interface

```
public interface Robot {
 void spin(); // girar
 void rotateArms(); // rodar braços
 void wiggleAntenas(); // mecher antenas
public class Eden implements Robot {
 Pair<Arm, Arm> arms;
 Pair<Antena, Antena> antenas;
 void spin() { ... }
 void rotateArms() { ... }
 void wiggleAntenas() { ... }
```

```
public class Sam
  implements Robot {

  Pair<Arm, Arm> arms;

  void spin() { ... }
  void rotateArms() { ... }

  // NÃO TENHO ANTENAS
  void wiggleAntenas() {
    throw new
      Exception("Não tenho antenas!");
  }
}
```

```
public interface SpinRobot {
 void spin();
public interface ArmsRobot {
 Pair<Arm, Arm> arms;
 void rotate();
public interface AntenasRobot {
 Pair<Antena, Antena> antenas;
 void wiggleAntenas();
```

```
public class Eden implements
  SpinRobot,
 ArmsRobot,
 AntenasRobot {
  Pair<Arm, Arm> arms;
  Pair<Antena, Antena> antenas;
  void spin() { ... }
 void rotateArms() { ... }
 void wiggleAntenas() { ... }
public class Sam implements
  SpinRobot,
 ArmsRobot {
  Pair<Arm, Arm> arms;
 void sping() { ... }
 void rotateArms() { ... }
```

Dependency

Inversion

Principle

Princípio

nn lersão

Dependência

DIP

```
public class Robot {
   CutterArm arm = new CutterArm();

   // cortar
   void cut(Pizza pizza) {
       arm.cut(pizza);
   }
}
```

DIP

```
public class Robot {
 ArmTool arm;
  // recebe a ferramenta
  Robot(ArmTool arm) {
    this.arm = arm;
  // cortar
  void cut(Pizza pizza) {
    arm.cut(pizza);
```

DIP

```
Robot robot = new Robot(
  new ArmKnifeTool()
);
Robot robot = new Robot(
  new ArmLaserTool()
);
Robot robot = new Robot(
  new ArmPexeraTool()
);
```