STUPID – Untestability

- Jedná se o tzv. netestovatelnost
- Má souvislost s prováděním automatických testů
- Při návrhu třídy je nutno pamatovat na to, že bude testována a že bude možná testován i její potomek
- Problémy
  - Singleton – znemožnění Mock testing závislostí
  - Porušení SRP – dlouhé metody, nic nevrací (procedury)
  - Nutnost ručního testování – vysoké náklady
  - Znemožňuje modifikace

STUPID – Indescriptive naming

- Používání nicneříkajících názvů
- Týká se proměnných, metod, tříd …
- Proměnné podle toho, co obsahují
- Metody podle toho, co dělají (rozkazovací způsob)
- Kolekce v množném čísle
- Třídy podle toho, co reprezentují

STUPID – Duplication

- Duplicitní kód
- Už jsme o něm mluvili u DRY
- Kód ve stejné nebo ve velice podobné podobě
- … a to na více místech aplikace …
- ... je samozřejmě špatně a je hrozbou pro další vývoj
- Špatná orientace v kódu
- Změny nutno provádět na několika místech
- .. ale to jste už slyšeli u DRY a KISS

SOLID - Single responsibility principle

- Každá třída má právě jednu zodpovědnost.
- Tato zodpovědnost by měla být danou třídou či modulem plně pokryta
- Za zodpovědnost se zpravidla považuje nějaká rozumně jednoduchá a oddělená funkcionalita.
- Použití tohoto principu snižuje složitost systému a zvyšuje jeho soudržnost a pochopitelnost.

SOLID - Single responsibility principle

Porušení principu

```
class Book {
    private Name name;
    private Author author;
    private Content content;

    // ... getters
    // ... settters

    public void print() {
        // book printing code
    }

    public void read() {
        // book reading code
    }
}
```

Oprava podle principu

```
class Book {
    private Name name;
    private Author author;
    private Content content;

    // ... getters
    // ... settters
}
```

```
class Printer {
    public void print(Book book) {
        // book printing code
    }
}
```

```
class Reader {
    public void read(Book book) {
        // book reading code
    }
}
```

SOLID - Interface segregation principle

- Princip oddělení rozhraní
- Každé rozhraní by mělo být co nejmenší možné...
- …a třídy by neměly být nuceny používat rozhraní, která nepoužívají.
- Pokud nějaké rozhraní přesáhne rozumnou velikost, musí se rozdělit do několika dalších a užších rozhraní.
- Potom se touto změnou zasažené třídy přepracují tak, aby implementovaly jen minimální potřebnou podmnožinu původních rozhraní.

SOLID - Interface segregation principle

Porušení principu

```
interface Lifecycle {
    void start();
    void stop();
}
```

Oprava podle principu

```
interface Startable {
    void start();
}
```

```
interface Stoppable{
    void stop();
}
```