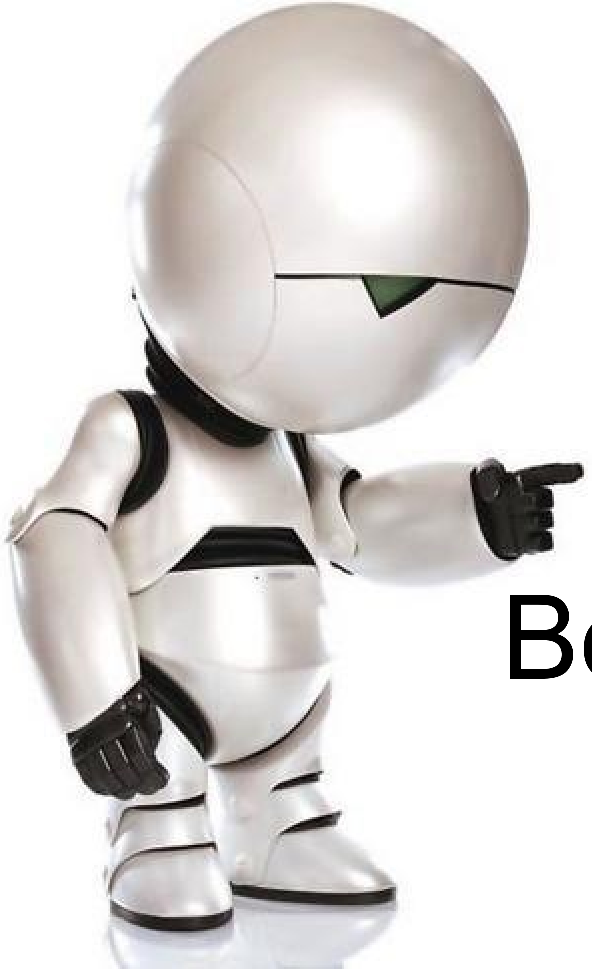
42: A language for paranoid programmers



Freedom is slavery!

Be an offensive programmer!

Programming at scale

- One programmer for a few days, no planned maintenance.
  Usually the customer is the single programmer.
  security?

- A single team/company for a few months, maintenance plan.
  Usually the customer is a wealthy individual or a small company.
  Security!

- A community effort for many years, maintaining IS developing.
  Usually there is no clear 'customer', but a large user base.
  Libraries, frameworks, OS, big open source projects..
  SECURITY !

Programming at scale

- Small scale: programming is mostly to express behavior.
- Larger scale: programming is mostly to restrict behavior.

Freedom is slavery

- Consider a multiplayer shooter or similar game
- New update: yet another kind of weapon

- Attack: cool, now I can do yet another thing; more freedom, more expressive power!
  - Still, I can only keep up to 4 weapons in my inventory
- Defense: Oh, no, now the enemy can kill me in yet another way
  - There are many enemies and they may have any weapon at all. How to defend against all possible weapons?

Freedom is slavery

- Consider a programming language
- New update: yet another kind of statement/expression
- Initial development: cool, now I can do yet another thing; more freedom, more expressive power!
  - Still, I can keep a few features in my head at once (finite brain).
- Maintenance/Usability: Oh, no, now the users/coworkers can interact with my code in yet another way.
  - Given enough time, all possible ways to interact with my code become relevant. How to make sure my code is correct in all those scenarios?

Cooperative or PvP

- Two kinds of gameplay: Cooperative and PvP
- Cooperative:
  - Python/Ruby: a user of you code can always break/dismantle everything if they want
  - Minecraft: building together to build amazing. Many players = high risk of griefing
- PvP:
  - PvP programming???
  - PvP Mincraft servers are basically giant wastelands

The C model delusion

- Ultimately everything is broken: just be sensible, practical and try to cooperate.
  - True in the old C/Shell/Lowlevel-OS programming.
- In Java/C#/Haskell a bunch of things are actually guaranteed by the language.
  - Java has 'security bugs/fix' because it has a security model: some properties that they promise.
  - C/C++: no such thing as a security fix, is nothing actually promised by the language.

Freedom is Slavery

Modular Offensive code: internal constraints are preserved even in an adversarial context. Possible in Java/C#/Haskell

- very unpractical in the general case
- Internal constraints offers no protection from actions (I/O etc)
- False if you load broken C code

Even then, if arbitrary adversarial code enters in the system

A real PvP scenario = The desolate wasteland

Modular consistency

- To have any hope to get modular correctness/security code must be able to make sound assumption of its own internal constraints:

  The module M can be placed in any context C and the internal invariant of M will never be violated

- In Java/C#/Haskell it is hard but possible
- In C/C++/Python is simply hopeless:
  - Python: user code can always disassemble and tweak any other code
  - C/C++: user code can always send the whole system into undefined behavior, after that the code of M can do anything.

Modular consistency tradeoffs

- Easy to use
- Certainty of Behavioral constraints
- Simplicity to express Behavioral constraints

42:

- Near as easy to use as Java
- Behavioral constraints: Security Full certainty
- Expressive and intuitive Behavioral constraints

Designing new languages

- Can we make it **as easy as possible** to do simple tasks?
  - Misguided objective: Freedom is Slavery

Instead

- Make it **impossible** to do certain kinds of **errors**!
  - Classify and mathematically model various kinds of errors.
  - The language semantic prevents those errors.
  - **Cost**: this language is now **harder** to use.

Offensive programming

(A variation of Defensive programming)

- Monitor how your code is used.
- As soon as something goes out of the expected behavior, throw error!

```
/**
 * Invariant:
 * -the name is never empty
 */
class Person {
  String name;
  }
```

If the user tries to get a person with empty name,
they must fail with an error!

Offensive programming

```
class Person {
  String name;
  Person(String name){
    this.name=name;
    }
  }
class User{
  void user(){
    //how to prevent this?
    Person invalid=new Person("");
    }
  }
```

Offensive programming

```
class Person {
  String name;
  Person(String name){
    if(name.isEmpty()){ throw new Error(); }
    this.name=name;
    }

  }
class User{
  void user(){
    //now this will throw Error, and an invalid person
    //would never exists
    Person invalid=new Person("");
    }
  }
```

Offensive programming

```
class Person {
  String name;
  Person(String name){
    if(name.isEmpty()){ throw new Error(); }
    this.name=name;
    }

  }
class User{
  void user(){
    //
    //
    Person bob=new Person("Bob");
    bob.name="";//How to prevent this instead?
    }
  }
```

Invariant silently violated :-(
Our program is wrong but does not tell us!

Offensive programming

```
class Person {
  private String name;
  public String name(){return name;}
  public void name(String n){
    name=n;
    if(name.isEmpty()){ throw new Error(); }
    }
  Person(String name){
    this.name=name;
    if(name.isEmpty()){ throw new Error(); }
    }
  }
class User{
  void user(){
    Person bob=new Person("Bob");
    bob.name("");//this now throws error!
    }
  }
```

Offensive programming

```
class Person {
  private String name;
  public String name(){return name;}
  public void name(String n){
    name=n;
    if(name.isEmpty()){ throw new Error(); }
    }
  Person(String name){
    this.name=name;
    if(name.isEmpty()){ throw new Error(); }
    }
  }
class User{
  void user(){
    Person bob=new Person("Bob");
    try { bob.name(""); }
    catch(Throwable t){}
    //And here we can see an invalid 'bob' with empty name :-(
    }}
```

Invariant silently violated
:-(
Our program is wrong
but does not tell us!

Offensive programming

```java
class Person {
  private String name;
  public String name(){return name;}
  public void name(String n){
    if(n.isEmpty()){ throw new Error(); }
    name=n;
    }
  Person(String name){
    this.name=name;
    if(name.isEmpty()){ throw new Error(); }
    }
  }
class User{
  void user(){
    Person bob=new Person("Bob");
    try { bob.name(""); }
    catch(Throwable t){}
    //Here bob is still untouched… sometimes is much harder
    }}
```

Offensive programming

Alternative exercise, with a list of allergies

```
/**
 * Invariant:
 * -there are no more then 10 allergies
 */
class Person {
  private List<String> allergies;
  public List<String> allergies(){ return allergies; }
  public void allergies(List<String> a){
    if(a.size()>10){ throw new Error(); }
    allergies=List.copyOf(a);
    }
  Person(List<String> a){ allergies(a); }
  }
```

The good code is not the natural one!

- In Java, the more complex the invariant or the data structure, the harder it gets to write good code.
- More direct in 42!

```
Person = Data:{
  S name
  S.List allergies

  @Cache.Now class method Void invariant(S name, S.List allergies) = X[
    !name.isEmpty();
    allergies.size() <= 10I
    ]
  }
```

Website / YouTube

- See the 42 tutorial in video format at ...
  (Intro to 42 playlist)
  https://www.youtube.com/watch?v=9RZIkdg3YBU&list=PLWsQqjANQic8c5wG3LfSe-mMiBKfOtBFJ

  (https://Forty2.is)
  (https://www.youtube.com/MarcoServetto)