UnrealScript: A Domain-Specific Language

Lecture 43

Presented by Aaron Staley

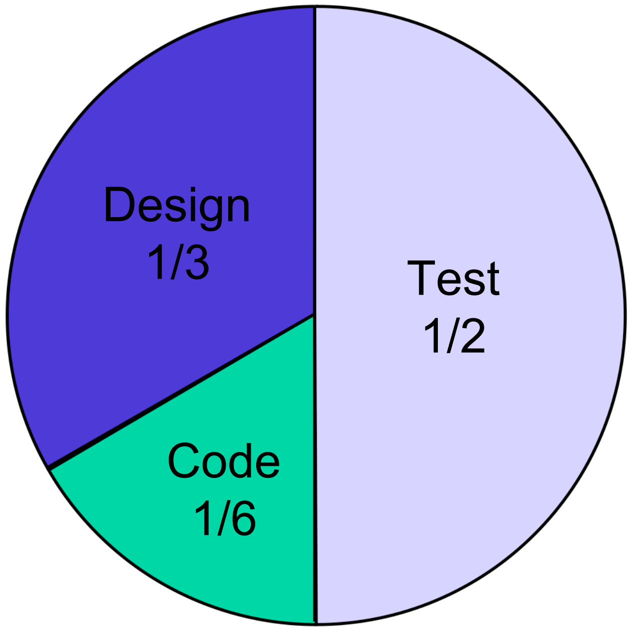
Some slides by Dave Mandelin

Announcements

- Your Project is due tonight at 11:59:59pm
- Review session for the final will be held Tuesday, May 13 at 8pm in 306 Soda
- The final will be held somewhere at 12:30pm on Saturday, May 17.
- HKN surveys next Monday in class!

Time Spent on Development

From *The Mythical Man-Month* by Fred Brooks



- Can we do more error checking and less testing?
- Better yet, can we avoid writing bugs?

Software Maintenance

- Maintenance is
  - Fixing bugs
  - Enhancing functionality & performance
  - Refactoring

- **60/60 Rule**
  - Project Cost: **60% is maintenance**
  - Maintenance
    - 60% is enhancements, 40% fixes
    - **30% is reading code**
  - From *Facts and Fallacies of Software Engineering* by Robert Glass

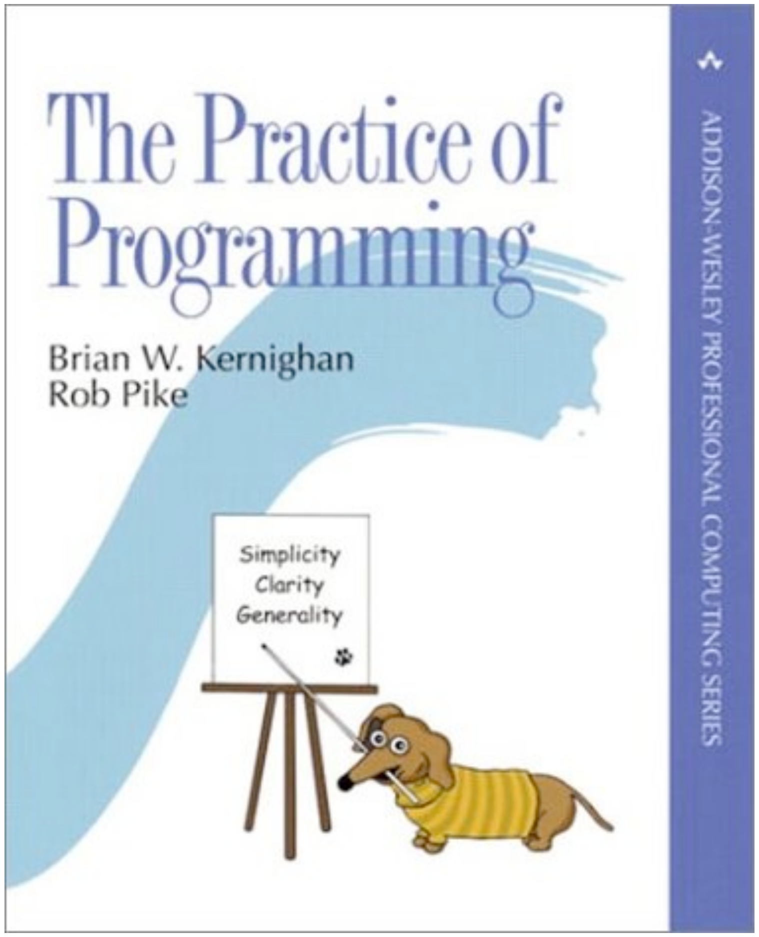
Lessons from Real Life

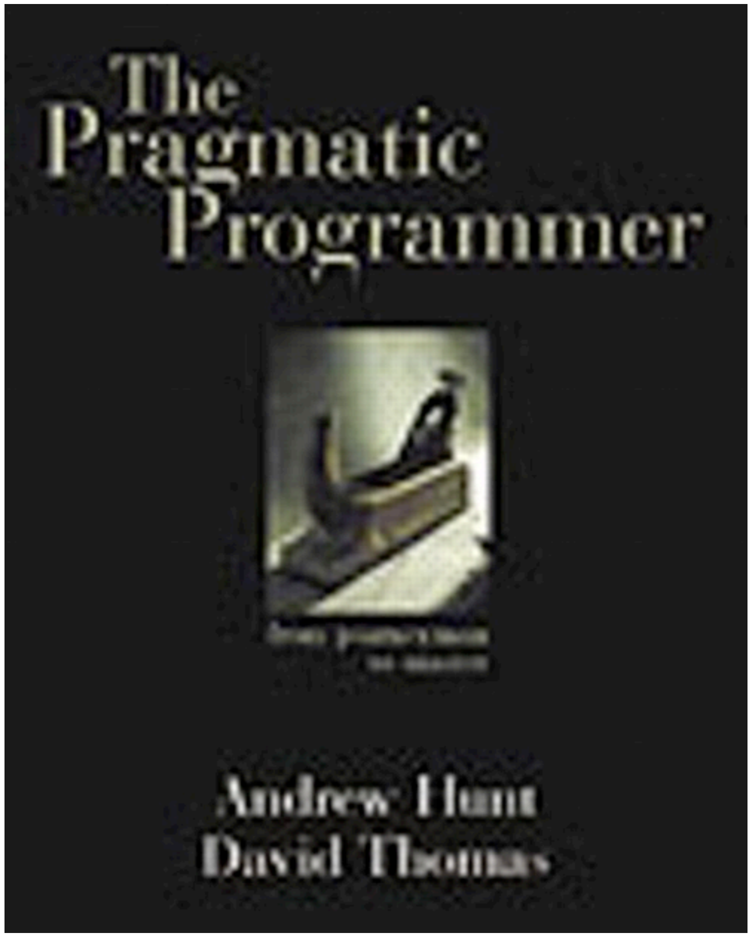
- Software needs to be
  - Reliable
  - Maintainable
  - Understandable
  - (only if it's intended to be good :)

Solutions for Real Life

- How can we write reliable, maintainable, understandable software?
- **Design a new language!**
  - A language specially designed for ***your*** problem – a *domain-specific language*
- Benefits
  - **Makes the program short, focused on functionality**
  - "Junk" implementation details (plumbing) hidden
    - And maintainable in one place
  - Error checking
  - Error avoidance
- Costs
  - Time to develop said language

Some books on this


The Practice of Programming
Brian W. Kernighan
Rob Pike
Simplicity
Clarity
Generality
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



Case Study: UnrealScript

Screenshot from Operation: Na Pali, a modification for Unreal Tournament (Unreal Engine 1 – released in 1999)

The Unreal Engine

- The **Unreal engine** is the game engine which powered *Unreal*, and many more since.
  - Unreal, Unreal 2, UT, UT 2003, UT 2004, UT2007, Gears of War, Deus Ex, Deus Ex: Invisible War, Splinter Cell, Mass Effect, Bioshock, America's Army
- It features its own scripting language **UnrealScript**
  - Allows rapid development of games using the engine
  - Allows easy development of modifications

Customizing Games

- Games (especially first person shooters) are expected to be **customizable**
  - By customers, designers, not just C++ hackers
  - Same goes for Office, Mozilla, network clients, …
- Need direct support for game logic
  - **Independent actors** (person, airplane, dog)
    - Sounds like a class
    - Or it is a thread? And can we have 10k threads?
  - **Actor behavior depends on state**
    - Class or methods change over time? Could be hard!
  - **Events, duration, networking**

UnrealScript

- Design Goals
  - From http://unreal.epicgames.com/UnrealScript.htm
  - Directly support game concepts
    - Actors, events, duration, networking
  - High level of abstraction
    - Objects and interactions, not bits and pixels
  - Programming simplicity
    - OO, error checking, GC, sandboxing

UnrealScript

- Looks like Java
  - Java-like syntax
  - Classes, methods, inheritance
- Game-specific features
  - States, networking
- Runs in a framework
  - Game engine sends events to objects
  - Objects call game engine (library) for services

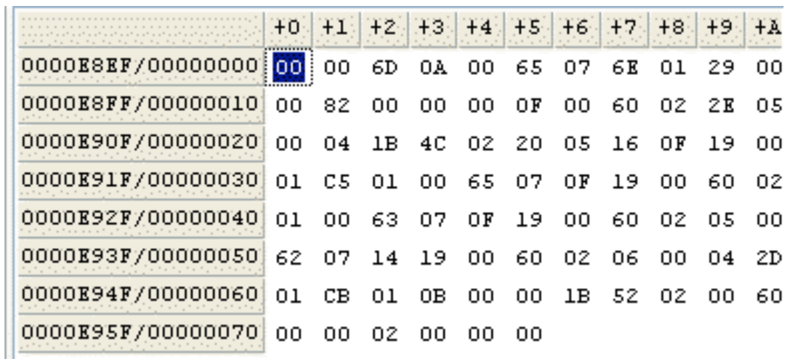
```
//code snippet
function
TranslatorHistoryList
Add(string newmessage){
   prev=Spawn (class,owner);
   prev.next=self;
   prev.message=newmessage;
   return prev;
}
```

Compilation

- Unrealscript is compiled to a bytecode that is executed at runtime
  - No JIT though!

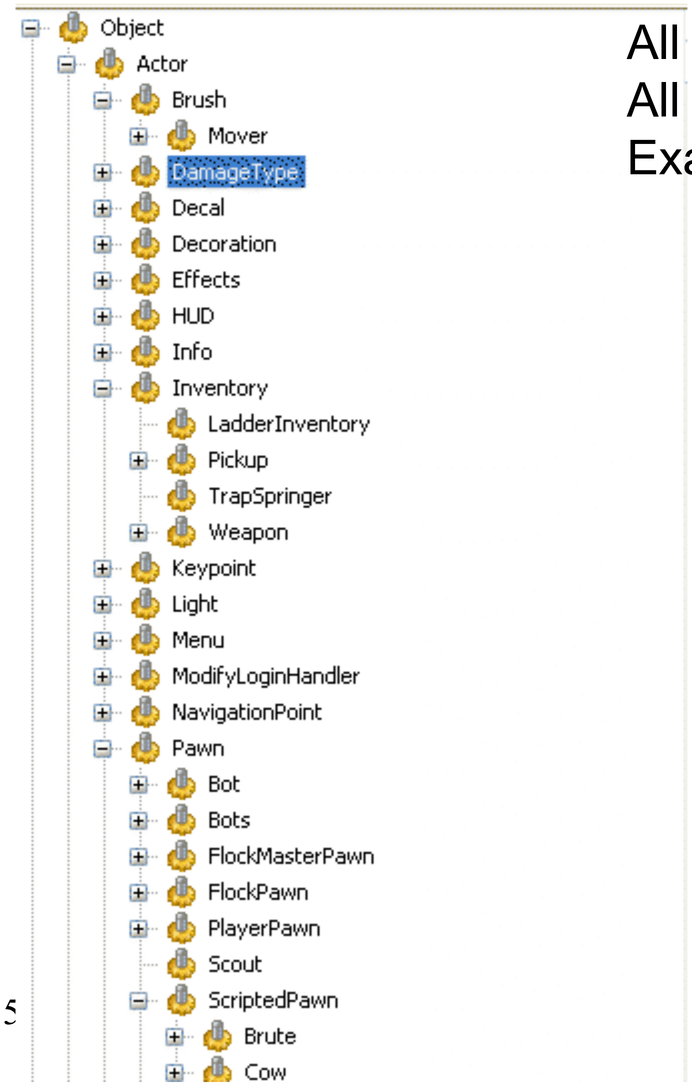
```
function AddSortedItem (string Value,
optional string Value2, optional int
SortWeight)
{
  local UDComboListItem i;

  i = UDComboListItem(Items.CreateItem(
          Class'UDComboListItem'));
  i.Value = Value;
  i.Value2 = Value2;
  i.SortWeight = SortWeight;
  i.Validated = True;
  Items.MoveItemSorted(i);
}
```

=



```
                   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A
0000E8EF/00000000  00 00 6D 0A 00 65 07 6E 01 29 00
0000E8FF/00000010  00 82 00 00 00 0F 00 60 02 2E 05
0000E90F/00000020  00 04 1B 4C 02 20 05 16 0F 19 00
0000E91F/00000030  01 C5 01 00 65 07 0F 19 00 60 02
0000E92F/00000040  01 00 63 07 0F 19 00 60 02 05 00
0000E93F/00000050  62 07 14 19 00 60 02 06 00 04 2D
0000E94F/00000060  01 CB 01 0B 00 00 1B 52 02 00 60
0000E95F/00000070  00 00 02 00 00 00
```

Objects Represent World Entities



```
Object
  Actor
    Brush
      Mover
    DamageType
    Decal
    Decoration
    Effects
    HUD
    Info
    Inventory
      LadderInventory
      Pickup
      TrapSpringer
      Weapon
    Keypoint
    Light
    Menu
    ModifyLoginHandler
    NavigationPoint
    Pawn
      Bot
      Bots
      FlockMasterPawn
      FlockPawn
      PlayerPawn
      Scout
      ScriptedPawn
        Brute
        Cow
```

All inherits from object

All entities in the world inherit from actor

Examples:

- Inventory – items carried
- HUD – heads-up display
- Pawn – “Character” (AI or player controlled)
- ScriptedPawn – creature in world

Actor States as part of Language

Without States

```
void spokenTo(Speaker s) {
    if (state == ANGRY) {
          shootAt(s);
    } else {
          greet(s);
    }
}

void bumpsInto(Object obj) {
    backUp();
    say("Raaaaaaargh!!!");
    state = ANGRY;
}

// And what about inheritance?
```

With States

```
state angry {
begin:
    say("Raaaaaaargh!!!");

    void spokenTo(Speaker s) {
          shootAt(s);
    }
}

void bumpsInto(Object obj) {
    backUp();
    GotoState('angry');
}

void spokenTo(Speaker s) {
    greet(s);
}
```

Networking

- Unreal network architecture
  - Server “replicates” object information
  - Client simulates world to hide latency and conserve bandwidth
  - Server only sends client what cannot be predicted.
    - Once a client knows the starting location and orientation of a rocket, it can simulate movement
    - A client cannot accurately predict movement of human-controlled players.
- Language Support
  - Replication definition block
  - Simulated Keyword
    - Controls whether an event should be run on a client

Networking

- Replication block:

```
replication{
        reliable if ( Role<ROLE_Authority )
                Password, bReadyToPlay; //some variables
        unreliable if( Role<ROLE_Authority )
                ServerMove  //client->server movement
        reliable if( Role<ROLE_Authority )
                Say;   //client wants to send a message
        reliable if( Role==ROLE_Authority )
                ClientChangeTeam; //provide client w/ team info
}
```

Role indicates who controls object

  On server an object is Role_Authority

"Unreliable" means no guarantee of transmission

Can replicate variables and functions

Variable Modifiers

- Want to make configuration very easy
- Can specify that variable is configurable by map designer with () after var.
  - `var(Movement) rotator Rotation;`
- Can specify that variable's state should be saved to a config file.
  - `var config bool bInvertMouse;`
- Defaultproperties block at end of code sets default values

```
defaultproperties {
  Mesh=LodMesh'Nalit'
  Health=160
```

Error checking in UnrealScript

- Statically typed language
- Traditional static checking
  - Name checking
  - Type checking
  - Pretty similar to PA2
- Runtime sandboxed as in Java
  - In theory, running any UnrealScript package cannot harm anything outside of Unreal install

Dynamic Error Handling: null

Null pointer dereference

- Unreal Tournament ('99) has 200,000 lines of script
  - Null dereference errors could be triggered by level designer error
- Don't want to crash program!
- Log error, return false/0/Null depending on type

Dynamic Error Handling: ∞

Infinite loops and infinite recursion

- Hard for game engine to recover from
  - Important for any plugin architecture
- singular function declaration
  - Means “don’t recur into me”
  - Declare bugs out of existence
- Engine also will detect infinite loops and gracefully exit

Performance

- Implementation
  - Compiles to VM bytecode (like Java)
- Performance
  - 20x slower than C++
    - Ugh! Today's Java is only 2-4x slower.
    - But wait…
  - Even with 100s of objects CPU spends only 5% time running UnrealScript
  - Graphics/physics engine does most of the work
  - UnrealScript doesn't need to be fast

What occurs where?

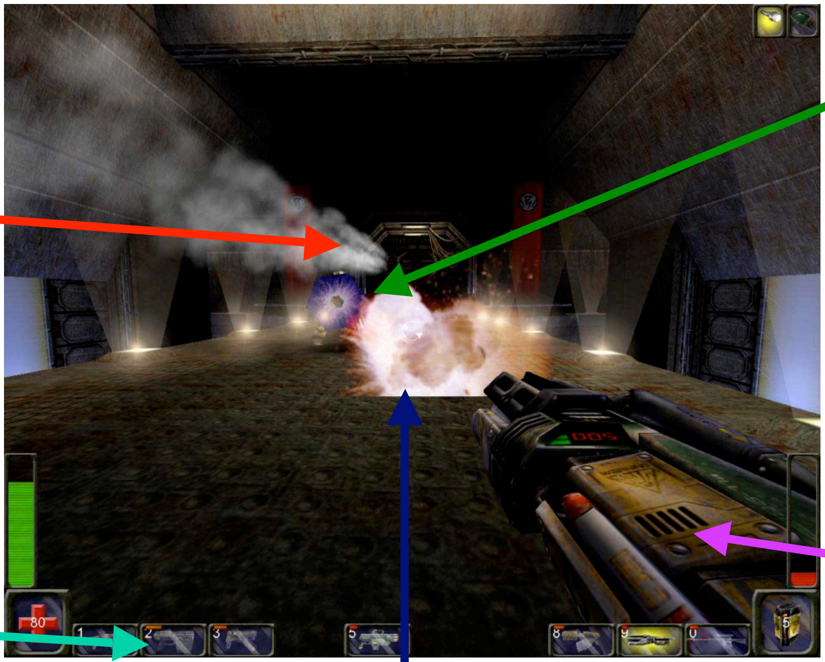
World is being rendered by engine (C++)

Most gameplay events (health tracking, ammo tracking) handled by UnrealScript

Creature's movement driven by C++ physics

Rocket's physics are controlled by C++

UnrealScript timer spawns smoke



Unrealscript controls targets, animations, attacks, defenses, etc.

UnrealScript controls what icons are drawn where;

Engine renders icons

C++ collision detection invokes Unrealscript event when projectile hits a wall

Weapon logic driven by unrealscript; script calls C++ library to render weapon

Event-driven Language

- No “main”. Engine spawns some objects initially – eventually yours is spawned
  - Your objects can also be placed in world by level designer.
- Actors receive various events from engine:
  - BeginPlay → Actor added to world
  - HitWall → Actor hit a wall
  - Touch → Actor was touched by a pawn
  - Timer → unrealscript sets when timers go off
  - Tick → Called every frame
  - PostRender → Called after world rendering to do additional drawing. HUD drawn here

Large Native Library

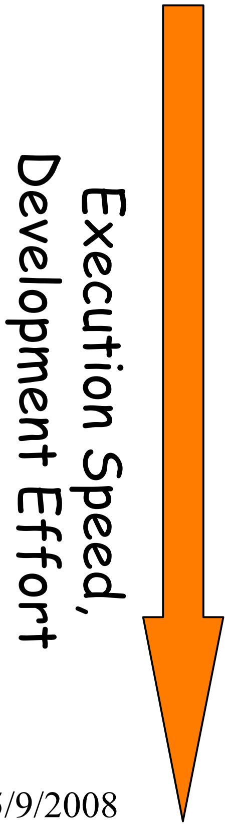
- Unrealscript can call functions in engine
  - `native static final operator vector +  ( vector A, vector B );`
  - `native final function SetSpeed (float newSpeed);`
- Especially needed for AI search, object drawing, collision tests
- Native side of things rather ugly:

```
void UDemoInterface::execSetSpeed (FFrame& Stack, RESULT_DECL){
   guard (UDemoInterface::execSetSpeed);
   P_GET_FLOAT(newSpeed);
   P_FINISH;
   DemoDriver->Speed = newSpeed;
   unguard;
}
IMPLEMENT_FUNCTION (UDemoInterface,-1,execSetSpeed);
```

Garbage Collection

- Generational Garbage Collector
- Added complication that actors in world have a destroy() function
  – Garbage collector also responsible for setting pointers to destroyed actors to NULL.

Implementation Quality



Interpreter

Google Calculator

Bytecode Interpreter

UnrealScript, Java 1.0

Basic Compiler

Project 3

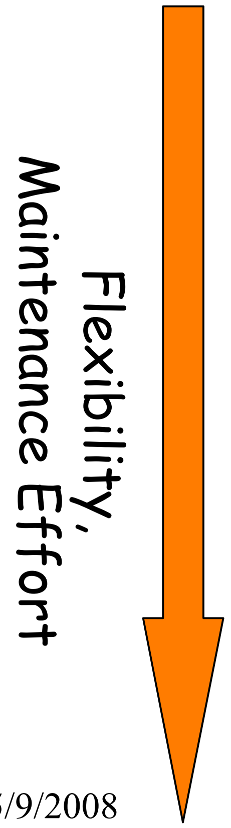
Simple Optimizing Compiler

Project 3 (hopefully!) ☺

Fancy Optimizing Compiler

Java 1.5 HotSpot VM (JIT), gcc

Language Flexibility



Little languages

make

Domain-specific languages (DSLs)

UnrealScript

General-purpose languages (GPL)

Python, C, Java

Aside: Many DSLs are at least Turing-Complete (such as UnrealScript), but often can't do important general tasks (like opening files)

Why UnrealScript Worked

- Why was it so successful?
  - Many reasons
- From a language point of view
  - Domain-specific concepts
    - Easy to use
  - Based on existing languages
    - Easy to learn
  - Runs slow
    - Easy to implement

General Game Scripting

- Why make your own language? It does take a lot of time.
- **Typical solution these days: GPL + library + engine**
  - A high level language, like Python, can be used as a scripting language with the engine implemented at lower level (C++)
  - Unfortunately, this loses the special benefits of an application-specific language
  - Let's see if we can get them back

UnrealPython

- Alternative scripting architecture:
  - Source Language: UnrealPython
    - Python + our extra stuff
  - Target Language: Python
- Goals
  - singular keyword
  - Survive null pointer errors **really** well

singular for UnrealPython

- Let's add the new keyword:

```
# @singular
def onGainedCash(self, amount):
   self.celebrate()
   self.gamble()       # Danger: can gain more cash!
   self.invest()       # Maybe here too
   self.buyMoreStuff()
```

Implementing singular

```
# @singular
def onGainedCash(self, amount):
   if hasattr(self.onGainedCash, ‘onStack’) \
      and self.onGainedCash.onStack = True:
        return
   self.onGainedCash.onStack = True

   self.celebrate()
   self.gamble()
   self.invest()
   self.buyMoreStuff()
   self.onGainedCash.onStack = False
```

Done? **No.**

What if gamble() raises an exception?

Implementing singular: correct

```
# @singular
def onGainedCash(self, amount):
   if hasattr(self.onGainedCash, ‘onStack’) \
   and  self.onGainedCash.onStack = True:
        return
   self.onGainedCash.onStack = True
   try:
        self.celebrate()
        self.gamble()
        self.invest()
        self.buyMoreStuff()
   finally:
        self.onGainedCash.onStack = False
```

Key benefits of language customization

- **Saves repetition and typos (onGainedCash)**
  - **Only need to figure out hard stuff once (exceptions)**

singular with decorators

```
# Return a singular version of ‘func’.
def singular(func):
   def singularVersionOfFunc(*args, **kw):
        if hasattr(func, ‘onStack’) and func.onStack = True:
                raise SingularException()
        func.onStack = True
        try:
                return func(*args, **kw)
        finally:
                func.onStack = False
   return singularVersionOfFunc

# Now Python’s decorator mechanism lets us can write
@singular
def onGainedCash(self, amount):
   …
```

Why use decorators?

- Adding a keyword is now easy!
  - At least if we can implement the keyword by ‘wrapping’ a function

- Other languages have related features
  - Java: AspectJ
  - .NET: Dynamic Code

Null pointer error protection

- UnrealScript catches null pointer errors

```
def doStuff(self, stuff, args):
    startStuff()
    self.progressBar.showPercent(20) # c/b None
    doSomeStuff()
    self.progressBar.showPercent(40) # c/b None
```

- ***A* missing progress bar shouldn't stop us!**

Squashing null pointer errors

- **Step 1: What transformation do we want?**
  - Source code

    ```
    self.progressBar.showPercent(20)
    ```
  - Target code
    - **Detect & silently catch null pointer errors**

```
try:
        self.progressBar.showPercent(20)
except AttributeError, e:
        if str(e) != "'NoneType' object " +
                        "has no attribute 'progressBar'":
                raise
```

Squashing null pointer errors (2)

- **Step 2: How do we do implement the transformation?**
    - Doesn't wrap: can't use decorators
  - Parse code to AST
  - Find attribute accesses
  - Replace with null-safe version
- Python will help us
  - **Recall: existing language ⇒ lots of stuff done for us**
  - See modules **parser**, **compiler**, **dis**(assembler)

Creating Your Own Language

- CS 164
  - Projects 1-3
  - You have more than enough skills!
- Hard part is language design
  - Requires experience
  - So create some languages!

Getting Started

- Language Design
  - **Factor out differences** from stereotypical code
  - **Base on existing languages**
  - **Extensibility** is good
- Implementation
  - Look for **parsers** and modification features (e.g. **decorators**)
  - **Interpreters** are easy to write
  - Compilers can make it faster
    - **Even compile to High-level language**: C, bytecode
- Libraries and Runtimes
  - An easy way to make common operations fast
  - Good libraries make a language popular
    - Java, .NET, Perl, Python