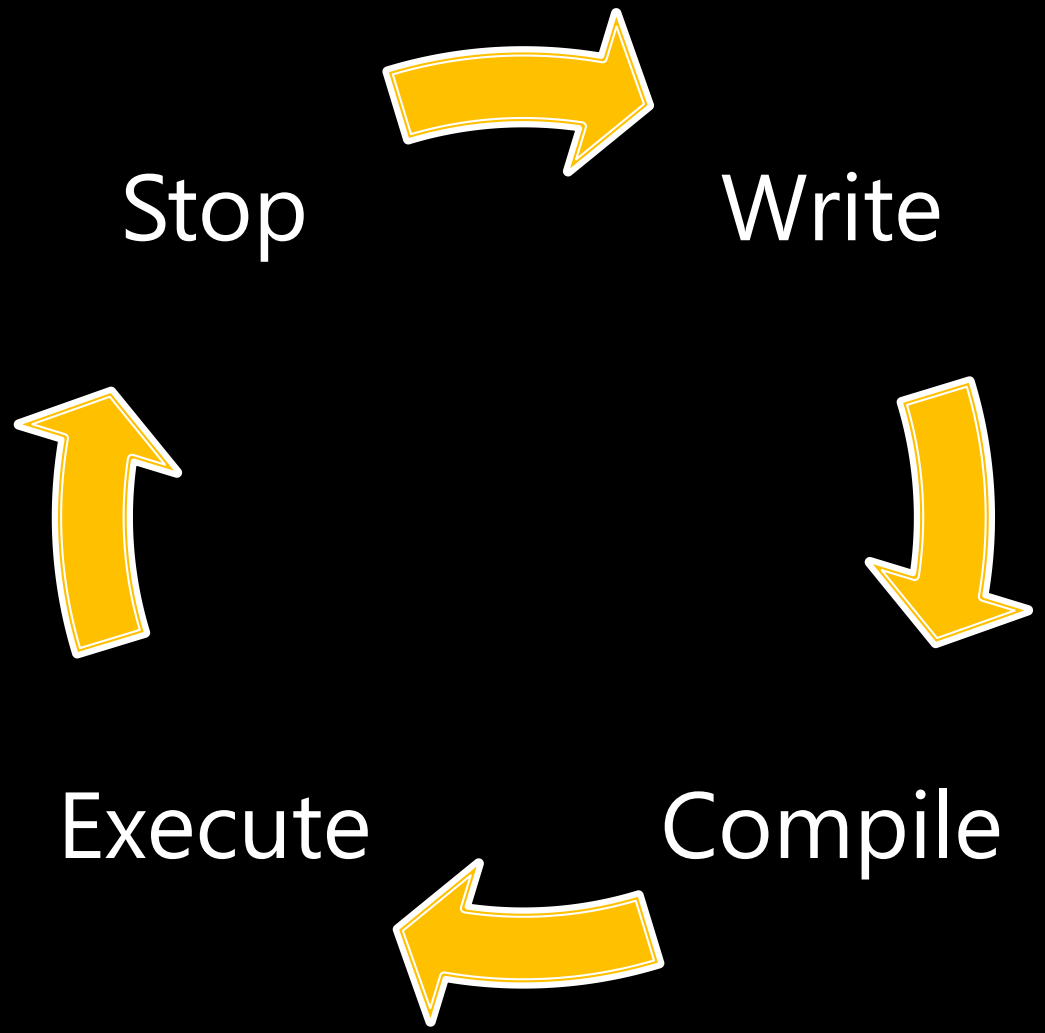
Dynamic Prototyping: Development Without Recompilation

Dmitri Nesteruk
ActiveMesa
@dnesteruk

Typical development Cycle



What's the Problem?

- Cognitive context switch: jumping between program and IDE
- Significant time delay (e.g., when writing a plugin)
- Recompilation is tedious and happens at a large scale (project, not individual file)
- Jumping into IDE makes sense for complicated, large-scale changes; I just want to change one line of code!
- Ideal situation is where the developed program is never stopped

The Plan

- We want to design a system where small, incremental changes can be made as the program is running
- Learn to recompile and substitute types
- Instantiate updated types at runtime
- Create a centralized, REPL-able container where types can be substituted
- Ensure container registration is persistent

Tools of the Trade

- `InProcessCompiler`: uses a `CodeDomProvider` (Roslyn?) to compile C# as the program is running
  - Generates a temporary DLL with just one type
  - Any subsequent in-process compilation references that DLL
- Centralized dependency injection container
  - Keeps a dependency graph
  - Uses property injection
  - Overwrites all instances of a particular component

InProcessCompiler

- Takes source code as text; compiles it to a DLL
- The newly created type is incompatible with the old
  - Even with same interface or identical content!
- Solution: interfaces
  - Define interface IFoo
  - Implement IFoo in type Foo
  - Foo is changed; a new version is compiled at runtime
  - Old Foo and new Foo are incompatible; but
  - Both can be assigned to an IFoo reference!

A More Public Registry

- Keep a list of available services
- Each service has an associated file with its own source code
  - One class per file rule
- Container can offer a list of editable types at runtime
- After editing a type, it gets updated in the container

DynamicContainer

- Dependency injection helps us satisfy necessary dependencies
- But what if one component changes?
- We don't want a service locator
  - `services.Get<IMyService>()`
- Instead, we want container to iterate types it already created and update all their injected members
  - Property injection with attributes (intrusive ☹, CoC?)
  - Keep a `WeakReference` to everything container creates
  - When a component changes, redo property injection
- REPLability: creating new types

TODO

- Persist the state of recreated components
  - What to do if the structure changes?
- Get REPL (csi) to detect generated assemblies and #r them
- Automatically add generated files to project
- Detect circular dependencies

That's It!

- Questions?
- Answers?
- Hate mail?
- @dnesteruk