No Silver Bullet

Mike Swaim

April 17, 2012
Introduction

A number of different programming techniques have been put forward over the years as being the “silver bullet” that would slay the software development beast. This talk is to give a decent high level overview of some common techniques, their advantages and disadvantages.
Chapter 16. No Silver Bullet—Essence and Accident in Software Engineering
High Level Languages- Advantages

• A single programmer can only write so many lines of code per day

• Studies have shown that productivity decreases exponentially with program size (measured in lines of code)

• If you can get the same work done with fewer lines of code, the program will be finished sooner, and have fewer bugs
High Level Languages- Advantages

• The right high level language can produce a 3-5x increase in programmer productivity. (Involves suitability to task and familiarity.)
References

• Mythical Man Month- Chapter 8- Calling the Shot (Fredrick Brooks)
• Code Complete: Chapter 4, section 1- Choice of Programming Language (Stephen McConnell)
• http://stackoverflow.com/questions/966800/mythical-man-month-10-lines-per-developer-day-how-close-on-large-projects
High Level Languages - Disadvantages

• Some optimizations cannot be done in a high level language- The C++ version of Quake ran at ~ 70% of the speed of the version that shipped. The shipped version interleaved fp and integer math, running both math units in parallel.

• http://www.vertigosoftware.com/Quake2.htm

• http://groups.google.com/group/microsoft.public.dotnet.framework.performance/browse_thread/thread/e22a2c941e518428/990e084440d47623?hl=en&ie=UTF&q=quake+.net+assembly
High Level Languages - Disadvantages

• May not be able to take advantage of processor features- SSE on intel/AMD hardware or ALTiVEC on PowerPC (Xbox, Wii, PS3)
• A DirectShow implementation had a 25x performance increase by using SSE instructions
High Level Languages - Disadvantages

• High level languages may have inappropriate runtime requirements

• On NT when servicing some interrupts, the memory manager may be inactive, so any page faults would hang the OS, limiting the use of some languages and runtimes
Object Oriented Programming

- public struct IntPair
  - {
    - public IntPair(int x, int y)
      - {
        - Val1 = x;
        - Val2 = y;
      }
  - public int Val1;
  - public int Val2;
  - }
Object Oriented Programming

• Examples: C++, Java, C#, SmallTalk, Objective C
• Dynamic Dispatch
• Encapsulation
• Inheritance
• Polymorphism
Object Oriented Programming
Advantages

• Data Encapsulation
• Modularity
• Promotes Code Reuse
Disadvantages

• “Everything is an object” often fails in practice.
• You end up with static classes which are simply collections of methods.
• Modeling your problem with objects can be tricky.
• Class Library/Framework Explosion
Dependency Graph 1
Dependency Graph 2
Links


• [http://thedailywtf.com/Articles/Enterprise-Dependency-The-Next-Generation.aspx](http://thedailywtf.com/Articles/Enterprise-Dependency-The-Next-Generation.aspx)

• [http://thedailywtf.com/Articles/The-Enterprise-Dependency.aspx](http://thedailywtf.com/Articles/The-Enterprise-Dependency.aspx)

• [http://thedailywtf.com/Articles/Enterprise-Dependency-Big-Ball-of-Yarn.aspx](http://thedailywtf.com/Articles/Enterprise-Dependency-Big-Ball-of-Yarn.aspx)
Declaritive Languages

select e.event_id, e.locked, eg.event_group_id, eg.locked, 
    fv1.value, fv2.value, fv1.flags, fv2.flags 
from event e join event_group eg on e.event_group_id = 
    eg.event_group_id 
left outer join float_value fv1 on e.event_id = fv1.event_id 
left outer join float_value fv2 on e.event_id = fv2.event_id 
where e.attribute_group_id = @ag and 
    eg.data_form_id = @form and 
    fv1.attribute_id = @a1 and 
    fv2.attribute_id = @a2
Declarative Languages

• Examples: Prolog, SQL
• Pro: The programmer is able to describe what they want without having to describe how it is to be done.
• Con: Performance can be problematic
(define factorial
  (lambda (n)
    (if (= n 0) 1 (* n (factorial (- n 1))))))
Functional Programming

• Examples: Lisp, Scheme, ML, OCaml, Haskell, Erlang, F#, APL, J
• First Class Functions
• Pure Functions
• Recursion
• Lazy Evaluation
Functional Programming Advantages

- Stateless
- Thread Safe (when using immutable objects)
- Pure functions are easy to test, since they have no side effects
Functional Programming
Disadvantages

• Longer Argument Lists
• You may want side effects/state
• More Object Creation/Destruction overhead
References

- Real-World Functional Programming with Examples in F# and C# (Tomas Petricek)
- http://stackoverflow.com/questions/1012573/how-to-learn-haskell/1016986#1016986
Data Oriented Design

• OO: class Ball
  { Point pos; Color color; double radius; void draw(); }

vector<Ball> balls;

• DOD: class Balls
  {
    vector<Point> pos;
    vector<Color> color;
    vector<double> radius;
    void draw();
  }
Data Oriented Design

• I am not aware of any languages that directly support Data Oriented Design

• The goal of DOD is to minimize cache misses by grouping data in space that will be processed together in time.

• This violated OO because you’re grouping data based on how you’re going to process it, rather than how it’s logically related.
Data Oriented Design
Data Oriented Design Advantages

- Speed! (Up to 35%)
- Parallelization is often simpler
Data Oriented Design Disadvantages

• Breaks the OO model
• Requires care allocating memory
• Data items must be processed sequentially
Data Oriented Design Gotcha

• In Java/C# collections of objects (including arrays) are collections of pointers. The objects are stored on the heap.

• In C#, arrays of structs ARE stored contiguously. Structs behave subtlety differently than objects, especially when passed as function arguments.
Links

Links

Metaprogramming

template <class myType> myType GetMax (myType a, myType b) {
  return (a>b?a:b);
}
Metaprogramming

• Examples: C++ Templates, C# Generics, Java Generics, Lisp Macros, Yacc, Bison, .net T4
• Code that generates code
• Useful when you want to do similar things repeatedly, with only some specifics changing.
Metaprogramming

Advantages
• Less code to maintain
• Simpler code

Disadvantages
• Conceptually harder
• Often harder to debug (specifically in C++)
Metaprogramming Links

• Modern C++ Design: Generic Programming and Design Patterns Applied (Andrei Alexandrescu)
Static Typing

- Int x = foo()
- Found in most compiled languages.
- Data types are explicit, and can’t change.
- Functions and variables are bound at compile time.
Static Type Advantages

• Find errors at compile time
• Many modern IDEs can use type information for name completion
Static Type Disadvantages

• Inflexible: Frameworks often expect objects to have a specific base class or interface to work correctly.
Dynamic Typing

- movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
- Type is inferred at runtime.
- Functions and variables are bound at runtime.
- Sometimes referred to as “Duck Typing”
Duck Typing

• If it walks like a duck, and quacks like a duck, it’s probably a duck.
• If a class exports the methods that you want to call, then it’s probably the type that you want.
Advantages

• The language is simpler
• Easier to test, because you don’t have to worry about creating/faking interfaces
• Source code is often more like pseudo-code
• Easier to pass objects between modules because you don’t have to worry about linker definitions.
• Metaprogramming is easier
Disadvantages

• Slower generated code
• Many coding errors won’t be caught until runtime
Links

• http://stackoverflow.com/questions/125367/dynamic-type-languages-versus-static-type-languages
• http://www.tbray.org/ongoing/When/201x/2011/12/27/Type-Systems
static int Foo()
{
    Bar b = new Bar();
    return b.Qux();
}
Garbage Collection

- VB, Java (JVM based languages), C# (CLR based languages), Python, Lisp
- Programmer doesn’t have to track memory usage
- Programmer often cannot access memory directly
- Programmer often cannot directly destroy objects
Advantages

• Programmer doesn’t have to worry about tracking object lifetime manually
• Memory leaks are greatly reduced, if not completely eliminated
• Heap fragmentation is often greatly reduced
Disadvantages

• Typically only tracks memory (not file/OS object handles, ports)
• Lulls programmer into false sense of security (see above)
• GC time is more or less proportional to number of “live” objects in the system
• The system controls when a GC is done
• Garbage collection often invalidates the cache
Disadvantages

• Application cannot have custom memory manager (Although host can. Ex. SQL Server)
• Example of a custom memory manager: The current Delphi memory manager is based on the fastMM project hosted on SourceForge
Links


• http://www.artima.com/insidejvm/ed2/gcP.html


• http://sourceforge.net/projects/fastmm/
Allocating on the Stack

{
    File f("/some/path/name");
    // use f

}
The Stack

• The stack is used by the current thread to hold temporary data. Memory is allocated and deallocated in linear blocks. Memory is typically allocated when a function is called, and freed when the function returns.

• Functions store local variables (and function return addresses) on the stack.
Stack Based Languages

• Forth, Postscript are explicitly stack based languages
• C, C++, Lisp, Java, Python, Ruby
• The first version of Fortran did not use the stack (or support recursion)
Advantages

• Faster than allocating memory on the heap
• Freeing memory is faster than using the Heap (either GC or manual memory management)
• Don’t have to worry about fragmentation
• Objects are automatically (via compiler generated code) freed when a function returns
Disadvantages

• The stack is MUCH smaller than the heap
• Stack space is allocated with each function call, so deeply nested functions can have much less memory to work with.
• Object lifetimes are limited to the current function call.
C# Note

- Classes are always allocated on the heap
- Structs are allocated on the stack
- Structs that have a lifetime longer than the current function have to be “boxed” inside an object
- Almost all iterators are structs
Links

- http://stackoverflow.com/questions/161053/c-which-is-faster-stack-allocation-or-heap-allocation
Tail Call

function bar(data)
{
    if ( a(data) )
    {
        return b(data);
    }
    return c(data);
}
Tail Call

• If the compiler is smart enough, when the last statement in a function is another function call, it’ll reuse the stack space of the calling function in the new function
Tail Call

**Advantages**
- Stack space is used much more efficiently
- Used by almost all Lisp variants

**Disadvantages**
- Lose debugging information as stack frames are reused
Links

• http://en.wikipedia.org/wiki/Tail_recursion
Resource Acquisition is Initialization

• In C++, objects created on the stack are destroyed when the current function returns.
• Why not use this as a primitive, yet fast garbage collector?
• This is how smart pointers came to be in C++.
Resource Acquisition is Initialization

- C# has something similar for IDisposable objects with the using() keyword.
- Using (Foo foo = new Foo(bar) {})
- C# Structs behave like C++ objects allocated on the stack. C# structs cannot have a destructor.
Links

• http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization
Multiprocessing
Symmetric Multithreading

• All processors are considered equivalent
• Memory is shared by all processors
• Used by programs that create threads, use thread pools, use task parallel libraries
Advantages

• Conceptually simple
• Tools are plentiful
• Only requires a single compiler/runtime
Disadvantages

• Race conditions
• Locking/synchronization overhead for memory shared between threads
• Limited memory bandwidth. With more processors, the greater the chance of saturating the memory bus and processors being starved for memory.
• If two processors are accessing the same page of memory, you can get cache thrashing
Links

Heterogeneous Multitasking

• All processors may or may not share the same memory
• Processors may or may not share the same architecture
• Typically, one processor acts as a controller, directing the others
Heterogeneous Multitasking

- IBM Cell Architecture (PS3)
- Clusters
- Tibco Rendezvous
- Applications using special purpose processors (like graphics engines)
Advantages

• Performance scales better than SMT. (Often nearly linearly)

• For tasks where they’re suited, special purpose processors are often much faster than general purpose processors.
Disadvantages

• More overhead for communication between processors
• May need more than one compiler (OpenCL makes this easier)
• Debugging tasks on a SPE (special purpose engine) may be difficult
Links

- [http://www.khronos.org/opencl/](http://www.khronos.org/opencl/)
Questions?