This Lecture

- .NET Lecture Review
  - The Framework
  - CLR and C#
  - Web Services
- Introduction to Software Engineering
  - The Software Crisis
  - Methodologies
- Course Summary

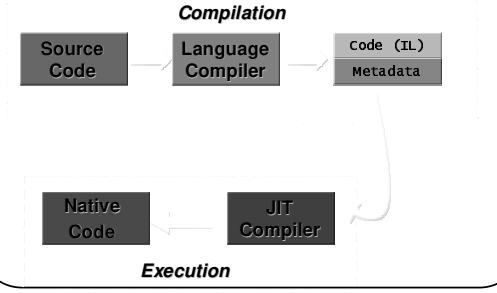
.NET Review
Intro to Software Engineering

David Talby

.NET Compilation Model

- 30+ languages compile to IL
  - Can be done in Java (Python, Eiffel)
  - IL defines which language features can run
  - C++, Eiffel, ML and others lose some features, and extend the language for other features
- Metadata
  - Reflection
  - Identification
  - Attributes
  - …

.NET Compilation & Execution Model



.NET Language Interoperability

- Language doesn't matter
  - Single stack trace, heap, threads, …
  - Polymorphism, exceptions, thread locks, garbage collection, singletons, …
  - Development tool – Cross-language debugging
- Shared libraries
  - Huge impact for "esoteric" languages
  - Same performance for all languages
  - Much easier reuse of older code

.NET Execution Model

- JIT Compilation
  - Compared to HotSpot in Java
  - Enables safe yet native code
- 10%-15% Slower than native
  - Also wastes more memory
  - Same in Java
  - Developers & Time cost more than hardware

CLR Compared to JVM II

- Managed vs. unmanaged code
  - C# has an unsafe keyword for "unsafe" sections
  - Pointers and direct access to OS are allowed
  - Enables both power and safety
- COM Inter-Op
  - Transparent use of gigantic COM code base
- Generics
  - Designed in advance for CLR, libraries and C#
- .NET is not forward-compatible

CLR compared to JVM

- Class loader
  - Dynamic loading, can be controlled
  - Security Manager + Code Signing
- Garbage collection
  - Can be deterministic in CLR
- Disassemblers
  - Can be done easily in IL or Java, also in C/C++/etc.
  - Obfuscation – partial solution, hinders reflection
  - Only real solution – hide the code
- Exceptions

C# Language Highlights II

- Reflection
  - Including generics, dynamic proxies, attributes
- Attributes
  - Added to Java in 1.5, but not to libraries
  - For the Compiler: Debug info, obfuscate, ...
  - For Libraries, by Reflection: Serialization, Security, GUI properties, Documentation, ...
  - For Aspect-Oriented Programming: XCSharp defines interfaces for code injection

C# Language Highlights

- Unified type system
- Value and reference types
- Explicit Polymorphism
  - virtual, override, new, class::method syntax
- Component Programming
  - Properties, events (delegates), indexers
- A lot of syntactic sugar
  - Boxing, Operator Overloading, Enums, Iterators

Evolution: Web Services

- Web Services (.NET / Java)
  - New standard protocols for interfaces, method calls, and object creation
  - Based on HTTP and XML
- "Share schema, not class"
  - Independent deployment and versioning
  - Heterogeneous platforms
- Strong security facilities in the standard
  - Authentication, Single sign-on, Encryption, ...

Evolution: Before Web Services

- The Issue: apps working together
- C APIs (System Calls, Win32 API, ...)
  - Application-specific protocols
  - Anything except data transfer must be coded
- COM / DCOM or RMI / CORBA
  - Binary standard for interfaces, method calls, types & marshalling, error handling, ...
  - Location + protocol + language independent
  - Services: Security, Deployment, Naming, Administration, Load Balancing, Failover, ...

Why Web Services are Important

- New WWW applications
  - Software and not humans navigate the web
  - Strong security -> economic transactions
- Simplifies integration between apps
  - Major issue facing large organizations today
  - Many systems, platforms, formats, upgrades, …
- A Real heterogeneous platform

Developing Web Services

- This (mostly) applies to both .NET and Java
- Developing a service
  - Write a normal class in your favorite language
  - Use attributes to define web methods / classes
  - Create a deployment file, and publish it to a server
- Developing a client
  - Choose "Add Web Reference" and write a URL
  - An interface in your favorite language is generated
  - Full debugging, type safety, metadata, intellisense, …
- All "plumbing" is transparent in both ways

The Software Crisis

- In Numbers
  - 84% of software projects are not on time
  - 31% of software projects never complete
  - ~60% of completed code is never used
  - ~200 Billion $ a year lost to software bugs
- In Words
  - Most software is buggy, unstable and insecure
  - A lot of software is totally unusable
  - Yet, software runs the world

Intro to Software Engineering

David Talby

State of the Software World

- Large Scale
  - Lack of repeatability, even for small projects
  - Inability to provide quality software
  - No standard definition of roles & products
  - No standard for requirements, design, tests, …
  - It's a "wild west" profession
- Small Scale
  - Developers don't produce working software
  - Developer tools are also far from perfect

What is Engineering

- Repeatability
  - Ability to do a similar project again well
  - Same time, budget, quality are expected
- Methodology
  - Well-defined roles: Architect, Engineer, …
  - Well-defined products: Designs, Specs, Code, …
  - Standard workflow of how things are done
- Legal Liability
  - Both Civil and Criminal
  - Certification required for life-critical issues
  - Methodology & Notation are laws

Rational Unified Process

- By Rational, see rational.com/rup
- Decompose large system to sub-systems
  - A team and development effort per system
  - Architects Team does overall design, sharing
- Five stages of each system's life cycle
  - Business modeling, Requirements, Analysis & Design, Implementation, Test
  - Many artifacts are not code or tests
- Iterative Development
- Highly managed, highly automated process

Development Methodologies

- A methodology describes
  - An entire life cycle of a software product
  - Roles, Products, Workflow
  - Best Practices
- eXtreme Programming
  - For small projects: up to 12 people, 100 stories
- Rational Unified Process
  - For large projects: a "heavy-weight" process
  - A commercial product

The XP Principles

- Develop by iterations of 1-3 weeks each:
  Plan (user stories) -> design (simplest!) ->
  test (unit tests) -> code (and refactor)
- Testing
  - Functional tests: in design phase
  - Unit tests as part of coding
- Continuous Integration
- Quality Work
  - Refactoring, 40-Hour Week

eXtreme Programming

- By Kent Beck, see XProgramming.com
- Embrace change
- Simplicity
- User involvement & rapid feedback
- Incremental pay-as-you-go design
- Test-first programming

Summary

- Writing Software ≠ Delivering Products
  - Requirements, Architecture, Design, Code, Integrate, Test, Deploy, Maintain, Update
- The Software Crisis
- Software Today
  - < 20% of existing code is Object-Oriented
  - > 90% of new code is Object-Oriented
  - Reuse: Libraries, Components, Web Services
  - Major Frameworks/Platforms: Java and .NET

The 12 XP Principles

- Planning Game
- Small Releases
- On-Site Customer
- Metaphor
- Simple Design
- 40-Hour Week
- Pair Programming
- Collective Ownership
- Testing
- Refactoring
- Continuous Integration
- Coding Standard