Industrial Programming

Hans-Wolfgang Loidl
hwloidl@macs.hw.ac.uk
http://www.macs.hw.ac.uk/~hwloidl

Lecture 1: Course Overview

Welcome to Heriot-Watt

Hans-Wolfgang Loidl

F21SC: Course Contents

- This course is about the ***programming skills***
- We will cover:
  - Systems languages: C#
  - Scripting languages: Python
- It assumes ***solid prior knowledge of an object-oriented*** language, eg. Java.
- It is about quickly picking up a new language of a familiar paradigm.
- It is ***not*** a gentle introduction to programming.

Learning Outcomes

- Appreciation of role of different *programming paradigms* in configuring/managing systems:
  - Object-oriented: good at structuring large code
  - Imperative: good at performance
  - Functional: good at abstraction
  - Logic: good at reasoning
- Autonomous problem analysis/solution:
  - Really understand the problem to pick the right paradigm/approach for producing a solution
- Understanding of core characteristics of contemporary operating systems: make good use of available resources

Learning Outcomes

- Appreciation of role of "***language as glue wear***" in configuring/maintaining systems:
  - Scripting languages combine existing code
- Knowledge of key abstractions across programming languages:
  - Write reusable and maintainable code
- Technical proficiency in advanced techniques in different programming paradigms:
  - Learn the Best of all Worlds

Topics to cover

- Overview & Linux introduction (1 week)
- Core C# programming (3 weeks)
- Advanced C# programming (3 weeks)
- Python programming (4 weeks)
- Revision (1 week)
- Planned: Guest lecture on testing

Lecture Plan

- Week 1: Overview, Linux Introduction (with shell scripting)
- Week 2: .Net and C# Introduction, C# Fundamentals
- Week 3: C# Objects & Classes, C# Concurrency
- Week 4: C# Data Manipulation, Database access in C# and LINQ, C# GUI development
- Week 5: Threading in C#, C# Systems Programming
- Week 6: Advanced C# Features, C# Revision

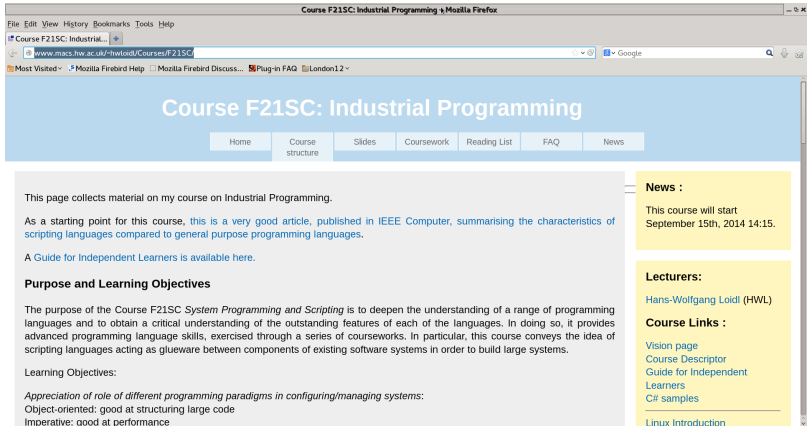
Lecture Plan (cont'd)

- Week 7: Parallel Programming in C#
- Week 8: Python Introduction and Data Types
- Week 9: Python Control Structures and Functions
- Week 10: Python Classes and Advanced Language Constructs
- Week 11: Python Libraries and Tools
- Week 12: Revision

Course material is available via the Vision system:
`http://vision.hw.ac.uk/`

`Main course information page:`
`http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/`

Main Course Information Page


Course F21SC: Industrial Programming - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Course F21SC: Industrial...
www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/
Google
Most Visited
Mozilla Firebird Help
Mozilla Firebird Discuss...
Plug-in FAQ
London12
Home
Course structure
Slides
Coursework
Reading List
FAQ
News

Course F21SC: Industrial Programming

This page collects material on my course on Industrial Programming.

As a starting point for this course, this is a very good article, published in IEEE Computer, summarising the characteristics of scripting languages compared to general purpose programming languages.

A Guide for Independent Learners is available here.

Purpose and Learning Objectives

The purpose of the Course F21SC *System Programming and Scripting* is to deepen the understanding of a range of programming languages and to obtain a critical understanding of the outstanding features of each of the languages. In doing so, it provides advanced programming language skills, exercised through a series of courseworks. In particular, this course conveys the idea of scripting languages acting as glueware between components of existing software systems in order to build large systems.

Learning Objectives:

Appreciation of role of different programming paradigms in configuring/managing systems:
Object-oriented: good at structuring large code
Imperative: good at performance

News :

This course will start September 15th, 2014 14:15.

Lecturers:

Hans-Wolfgang Loidl (HWL)

Course Links :

Vision page
Course Descriptor
Guide for Independent Learners
C# samples

Linux Introduction

Assessment

- Assessed Coursework: 100%
- Demonstration (mandatory) of the coursework
- There is no exam for this module

Coursework:

- Project 1: C# programming project (50%)
- Project 2: Python programming project (50%)

Skills Tested in the Coursework

- Composing bigger applications out of existing components
- Rapid prototyping
- Resource conscious programming
- GUI programming
- Concurrency

Software Infrastructure

- Visual Studio 2015 with C# (Windows)
- Alternatively, stand-alone C# compiler with libraries needed for GUI etc programming
- sh or bash scripting languages (Unix)
- Python interpreter
- Overall: heavy use of libraries!

References (C#)

- Douglas Bell, Mike Parr, “C# for Students”, Addison Wesley, 2009.
- Jesse Liberty, Brian MacDonald, “Learning C# 3.0”, O'Reilly, 2009.
- Joseph Albahari, Ben Albahari, “C# 5.0 in a Nutshell: The Definitive Reference”, O'Reilly, 2012. *C# 6.0 due Nov 2015!*
- Kurt Normark, “Object-oriented Programming in C# for C and Java Programmers”, 2010. http://www.cs.aau.dk/~normark/oop-csharp/html/notes/theme-index.html
- Eric Gunnerson, “A programmer’s Introduction to C# 5.0, Springer, 2012
- Andrew Birrell, “An Introduction to programming with C# Threads”, Microsoft, 2005.
- Arnold Robbins, “Classic Shell Scripting: Hidden Commands that Unlock the Power of Unix”, O'Reilly, 2005.

Characteristics of Systems Lang

- Build algorithms and data structures from scratch
- Use strong typing to help manage complexity of large pieces of software
- Focus is often on speed of execution
- Easy access to low-level operating system is crucial
- Examples: C, C#

Characteristics of Scripting Lang.

- Their main purpose is to ***glue*** software together
- Focus is on rapid-prototyping
- Safety aspects are of a lesser concern
- Thus, scripting languages are often type-less
- Modern scripting languages incorporate features of general purpose programming languages, especially object-oriented (o-o) features, higher-order functions
- Easier to learn for casual programming
- Examples: sh, php, python, perl, ruby, lua

A Short History of System Lang

- Developed as an abstraction over assembler programs
- They are ***higher-level*** by introducing abstraction mechanisms to manage large pieces of code.
- They are safe by using ***strong typing*** to more easily detect mistakes in the program
- They delegate some control of the underlying machine to libraries and operating system
- Together this drastically increases programmer productivity

Classifying Systems Lang

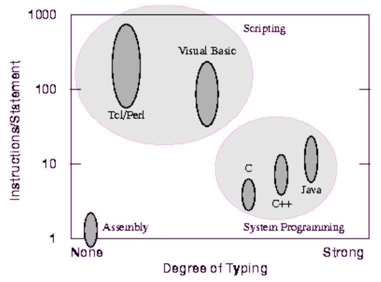


Figure 1. A comparison of various programming languages based on their level (higher level languages execute more machine instructions for each language statement) and their degree of typing. System programming languages like C tend to be strongly typed and medium level (5-10 instructions/statement). Scripting languages like Tcl tend to be weakly typed and very high level (100-1000 instructions/statement).

A Short History of Scripting Lang.

- ***First Generation***: simple composition of command-line jobs (espec. Unix systems); also called batch-languages
- Very little language abstraction
- Slightly different syntax in different languages
- Rich libraries for low-level coordination with the operating-system (OS)
- Examples: sh, bash, tcsh ...

History (cont'd)

- ***Second Generation***: Trying to combine many different language features into one language
- Addresses the problem that different batch languages do the same thing slightly differently
- Thus, the language becomes huge
- The mixture of concepts makes it hard to read third-party code.
- Also, extended support for graphical user interfaces (GUIs)
- Examples: perl, tcl

History (cont'd)

- ***Third Generation***: increasingly use modern programming language abstractions to make programming simpler
- In particular, heavy use of o-o concepts
- Also, concepts from other programming paradigms such as higher-order functions and polymorphism
- Examples: php, python, ruby, lua, go, dart

Relevance of Scripting Langs

- Increasing speed of processor makes the application of interpreted languages viable
- Existence of large libraries makes the development of new software from scratch less common-place
- Heterogeneous environment make a write-once run-everywhere approach appealing
- New technologies, such as the internet, make the issue of composing services even more important

Common Features of Scripting Languages

- Scripting languages are usually ***typeless***: no (type) restrictions on the use of the input/output to/from existing components are imposed
- This enhances the flexibility of the language but reduces the safety
- Example from Unix shells: pipeline mechanism:

  select | grep scripting | wc
- This reads the text currently selected in a window, passes it to a search for the word “scripting” and counts the number of lines in the output

Common Features (cont'd)

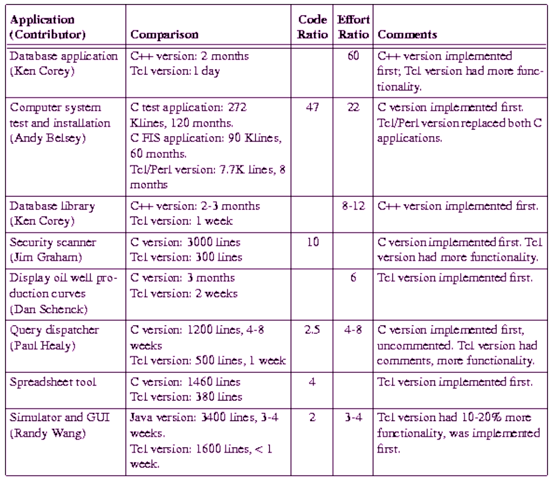
- Modern scripting languages provide a limited amount of type information to re-gain type safety
- To avoid frequent conversion functions between types, class hierarchies and implicit type conversions are used
- In contrast to systems languages, some type checks are performed at run-time rather than compile-time (dynamic typing)

Common Features (cont'd)

- Scripting languages are usually ***interpreted*** rather than compiled
- This gains rapid turnaround time in writing and testing code
- It increases flexibility, since the program can generate strings that are in themselves programs
- It loses performance compared to executing compiled code; but typically the amount of code in the scripting language is small and performance is dominated by the code in the components

Common Features (cont'd)

- Scripting languages are often higher-level than system languages, espec. for the latest generation
- For example many scripting languages have powerful, built-in mechanisms for regular expression substitution
- In the latest generation high-level concepts such as class hierarchies are included, too.



Application (Contributor)	Comparison	Code Ratio	Effort Ratio	Comments
Database application (Ken Corey)	C++ version: 2 months Tcl version: 1 day		60	C++ version implemented first; Tcl version had more functionality.
Computer system test and installation (Andy Belsey)	C test application: 272 Klines, 120 months. C FIS application: 90 Klines, 60 months. Tcl/Perl version: 7.7K lines, 8 months	47	22	C version implemented first. Tcl/Perl version replaced both C applications.
Database library (Ken Corey)	C++ version: 2-3 months Tcl version: 1 week		8-12	C++ version implemented first.
Security scanner (Jim Graham)	C version: 3000 lines Tcl version: 300 lines	10		C version implemented first. Tcl version had more functionality.
Display oil well production curves (Dan Schenck)	C version: 3 months Tcl version: 2 weeks		6	Tcl version implemented first.
Query dispatcher (Paul Healy)	C version: 1200 lines, 4-8 weeks Tcl version: 500 lines, 1 week	2.5	4-8	C version implemented first, uncommented. Tcl version had comments, more functionality.
Spreadsheet tool	C version: 1460 lines Tcl version: 380 lines	4		Tcl version implemented first.
Simulator and GUI (Randy Wang)	Java version: 3400 lines, 3-4 weeks. Tcl version: 1600 lines, < 1 week.	2	3-4	Tcl version had 10-20% more functionality, was implemented first.

Table 1. Each row of the table describes an application that was implemented twice, once with a system programming language such as C or Java and once with a scripting language such as Tcl. The **Code Ratio** column gives the ratio of lines of code for the two implementations (>1 means the system programming language required more lines); the **Effort Ratio** column gives the ratio of development times. In most cases the two versions were implemented by different people. The information in the table was provided by various Tcl developers in response to an article posted on the comp.lang.tcl newsgroup; see [7] for details.

When to use Scripting Lang

- Is the application's main task to connect pre-existing components?
- Will the application manipulate a variety of different kinds of things?
- Does the application involve a GUI?
- Does the application do a lot of string manipulation?
- Will the application's functions evolve rapidly over time?
- Does the application need to be extensible?

When to use Systems Lang

- Does the application implement complex algorithms or data structures?
- Does the application manipulate large data sets?
- Are the application's functions well-defined and changing slowly?

Application Domains for Scripting

- Graphical User Interfaces
  - Fundamentally “gluing” nature
  - Large percentage of code in modern apps
- Internet
  - Main role: connecting a huge number of existing computations and data (see success of perl)
  - Web services as the next level of gluing
- Component Frameworks
  - A flexible method of assembling components into applications

Summary

- Be aware of the *characteristics* of systems and scripting languages
- Decide early on in a project which class of language to use
- Today's trends in programming languages will be tomorrow's features in scripting languages
- Main reference:
  - “Scripting: Higher Level Programming in the 21st Century”, John K. Ousterhout, IEEE Computer, March 1998. URL: `http://home.pacbell.net/ouster/scripting.html`