COMP220: SOFTWARE DEVELOPMENT TOOLS

COMP285: COMPUTER AIDED SOFTWARE DEVELOPMENT

Sebastian Coope
coopes@liverpool.ac.uk
www.csc.liv.ac.uk/~coopes/comp220/
www.csc.liv.ac.uk/~coopes/comp285/

© University of Liverpool

Delivery

- **Lectures**
  - **3 Hours/week**
- **Tutorials**
  - **1 Hour week**

Assessments

- COMP220
  - 80% written examination
  - 20% continuous assessment:
  - Class Test (1 hour during a lecture)
  - Lab Test (2 hours: one lab session + 1 additional hour),
- COMP285
  - 50% Lab Test (same as for COMP220)
  - 50% Practical Assignment (only for COMP285)

Module aims

- To explore software development tools and new methodologies of Software Development

- To examine the techniques for implementing some Extreme Programming practices such as
- Automated Testing,
- Continuous Integration and
- Test Driven Programming

- To provide an insight into Eclipse (Integrated Development Environment)

Contents

- Methodology of Extreme Programming
  - (brief, methodological part)
- Software testing theory and practice
- Introduction to software tools
- Source code control
- Java Development with Eclipse
- Ant (the main tool considered in this course)
- Junit (this is the main, technical part of the course)

Recommended texts

- Java Development with Ant – E. Hatcher & S.Loughran. Manning Publications, 2003 (ISBN:1-930110-58-8) – THE MAIN BOOK
- Ant in Action, Second Edition of Java Development with Ant Steve Loughran and Erik Hatcher July, 2007 (ISBN: 1-932394-80-X)
- Eclipse in Action, A Guide for Java Developers, – D.Gallardo, E.Burnette, & R.McGovern. Manning Publications, 2003 (ISBN:1-930110-96-0)
- JUnit in Action, Second edition(!! For New version of JUnit (4); Old version of JUnit 3 will not be considered.), P. Tahchiev, F. Leme, V. Massol, G. Gregory, Manning Publications, 2011. (ISBN: 9781935182023)

Extra reading

- Java Tools for Extreme Programming – R.Hightower & N.Lesiecki. Wiley, 2002 (ISBN:0-471-20708-X)
- Professional Java Tools for Extreme Programming – R.Hightower et al. Wiley, 2004 (ISBN:0-7645-5617-7)
- Test Driven Development – K.Beck. Addison-Wesley, 2003 (ISBN:0-321-14653-0)

Lab sessions

- Labs will be devoted to simple exercises helping to really understand how the software development tools (discussed on lectures) work in practice
- Inseparable from the lectures
- The best and easiest way to prepare to the Exam and Class Test (for COMP220) during the semester
- Without doing labs well you will be unable to do and pass Lab Test (for both COMP220 and COMP285).
- COMP285 will have additional Practical Assignment and no exam. Thus, labs are invaluable for you

Websites

- **www.csc.liv.ac.uk/~coopes/comp220/**
- **www.csc.liv.ac.uk/~coopes/comp285/**

- Contain:
  - General course information and useful links
  - Course slides
  - Lab sessions description
  - Practical assignment for COMP285

Schedule

- Week 1
  - Introduction to Software Tools and Agile and test driven development
- Week 2
  - Introduction to Eclipse and
- Week 3
  - Testing theory and practise
- Week 4
  - Eclipse and JUnit

Schedule

- Week 5, 6, 7 and 8
  - Ant and Junit
- Week 9
  - Issues tracking
- Week 10
  - Revision

Lecture 1

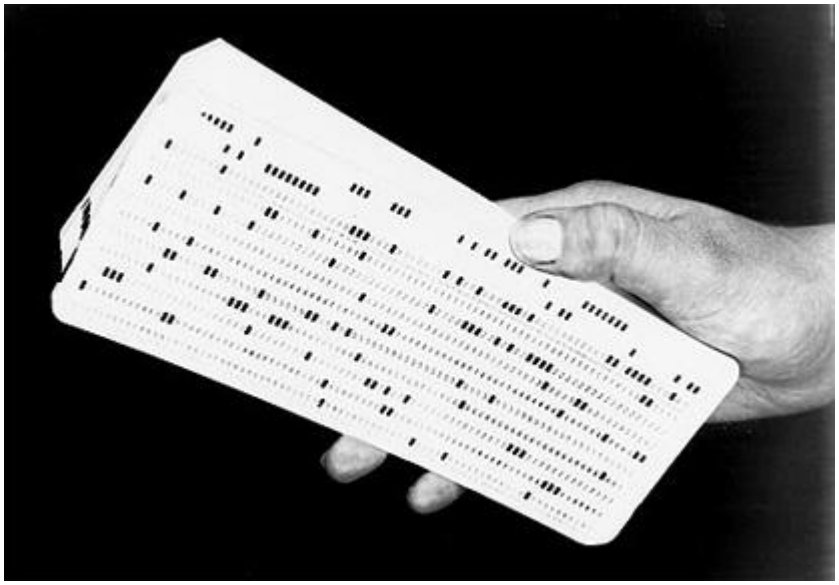
- Introduction to CASE tools
- Why bother?
- What types?
- How they work in practise?

Computer Aided Software Engineering

- Well what would be Software Engineering without computers?

Early days

- Software Engineering Stone Age style
  - All programs verified by hand



CASE definition

- "The use of tools (usually built from software) which make it easier to build high quality software" **S Coope**
- **CASE tools can be**
  - **General purpose, example build/test tools (language neutral)**
  - **Specialist e.g. Load testing/DoS tool like Low Orbit Ion Cannon**
  - **Built yourself (test harness)**

Are these tools

- High Level Language Compilers/Interpreters
- Object Relational Management software
  - Example Hibernate
- Profilers
- Debuggers
- Code analysers

Why tools?

- Would you hire a building company who used?
  - No power tools (productivity cost)
  - No measuring instruments (accuracy)
  - No spirit levels or electrical testing tools (quality)
  - Proper scaffolding/access control (safety)
- With software, these issues also matter
  - Productivity, accuracy, quality, safety

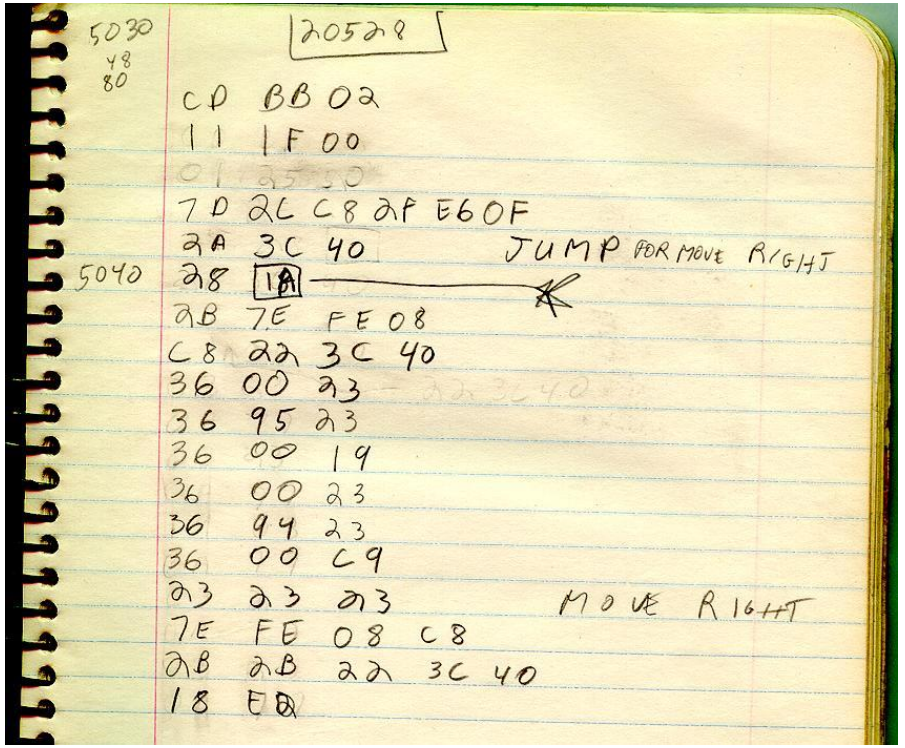
Why use CASE?

- Used properly
  - You will become a better software engineer
  - You will work better with others
  - Your code will be tested more
  - You will do things, you wouldn't normally bother to do (e.g. re-factoring code)
  - Your life as a software engineering will be slightly less stressful

Think of CASE as a series of developments

- Machine language (no real CASE)

Seb 1983 Apple II



Assembly language

- Simple case tool called an assembler translates to machine code, 1 to 1 relation to machine code
- No type check, complex data types, all abstraction in programmers head

```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32  PROC          ; procedure begins here
       CMP  AX,97    ; compare AX to 97
       JL   DONE     ; if less, jump to DONE
       CMP  AX,122   ; compare AX to 122
       JG   DONE     ; if greater, jump to DONE
       SUB  AX,32    ; subtract 32 from AX
DONE:  RET           ; return to main program
SUB32  ENDP          ; procedure ends here
```

High Level Language Compilers

- Translate from abstract (non machine language) to machine language of intermediary code
- Abstractions are constructed as part of the language, e.g. int, long, String, Person, BankManager
- Code is capable of
  - Being structured
  - Being tested as part of compilation

Interpreters

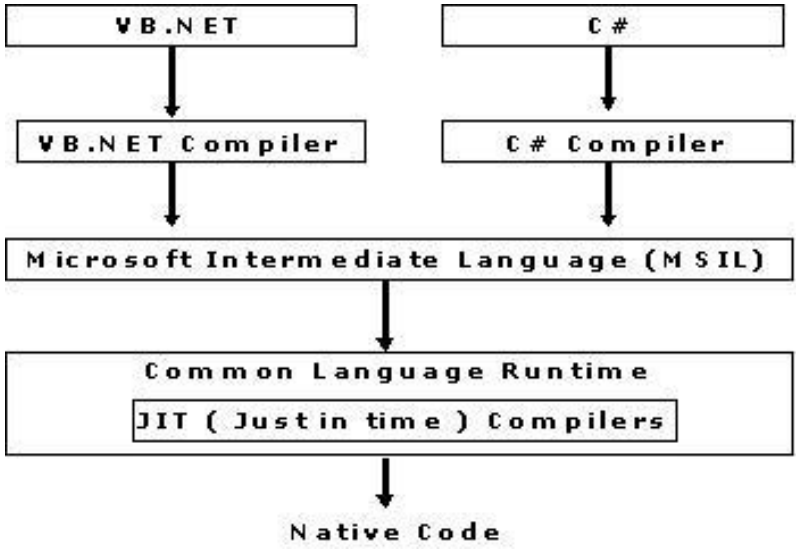
- Programs which run programs, like JVM for Java (Forth, Basic)
- Interpreters can be
  - Source code based (execute the code directly) e.g. Java script interpreters
    - Fast turnaround time, no-precompile check, only checks running code
  - Intermediary code based
    - e.g. Java byte code, slower turnaround, supports pre-compile check

Interpreter factors

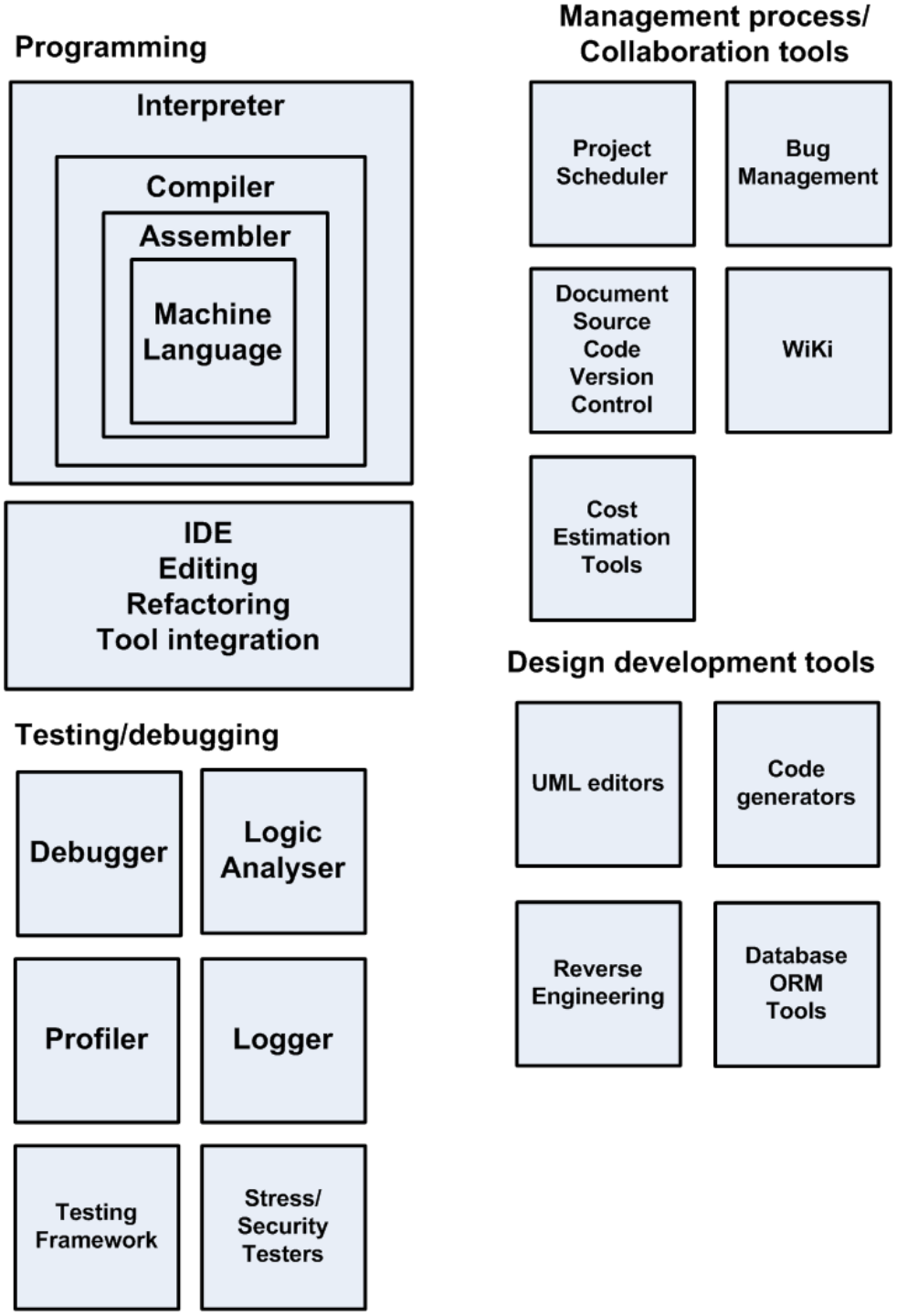
- Program can be debugged by facilities in JVM
- JVM can run time check code for
  - Errors (array out of bounds)
  - Type conversion issues
- Garbage collection (contentious, see iOS debate)
- JVM can be ported to different hardware, making object code portable

Intermediary code systems

- Allows for multiple languages to easily integrate, parts of code can be written in language of choice
- Good or bad thing?



CASE in perspective



Processes with/without case

- No CASE
  - Tester sends bug report to programmer via email, programmer looks at code, fixes code, sends new source back to tester
- Problems
  - No proper record of bug and fix
  - Visibility of process limited (team leader manager)
  - Easy for bugs to get lost
  - Hard to but priority on bug

Bug fix with CASE

- Tested finds bug and records on bug management tool (example Bugzilla) and assigns bug to programmer
- Bug management tool sends email to programmer
- Programmer logs into bug management tool and accepts/rejects bug and adds comment
- Programmer updates latest source code from source code control system
- Programmer fixes/tests bug using debugger
- Programmer re-commits code back to source code control with comment, which links back to bug id
- Programmer adds comment from source code control log and comment about how to test bug to bug control system
- Team leader downloads all source code and makes new build

Bug fixing with CASE support

- Looks a lot more work, but most of the CASE processes take v.little time
- Allows bugs to be properly assigned to staff
- Gives a clear view of the process to the management of the project
- One can look back and find out which bugs were fixed on which versions of files
- Makes sure that bug reports are filled in correctly

Summary

- Using CASE can make you
  - Quicker
  - Better team working
  - More controlled
- Many tools to choose from
- Using IDE will encourage best practise
- We will be looking in more detail at some of these tools in the next lecture