CSEP504: Advanced topics in software systems

- Catching up on some things – interspersed
  - Structured reports
  - State-of-the-research reports
  - Topic #3
- Tonight: first of three lectures on software tools and environments
  - Main roles of tools and environments – absolutely incomplete
  - Historical overview – absolutely incomplete
  - Opportunities – absolutely incomplete

Last week

- Many thanks to Yuriy for lecturing
- Visited Georgia Tech, College of Computing
  - Member of dissertation committee
  - Colloquium

What is a tool?

lowes.com (1/31/2010) lists 4622 tools
(power tools, hand tools, electrical tools, etc.)

- "**1.** a device or implement, typically hand-held, used to carry out a particular function.
  2. a thing used to help perform a job • a person exploited by another."
- "tool *n.*" *The Concise Oxford English Dictionary*, Twelfth edition . Ed. Catherine Soanes and Angus Stevenson. Oxford University Press, 2008. *Oxford Reference Online*. Oxford University Press. University of Washington. 31 January 2010 <http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t23.e59094>

✓ We'll focus on software tools – that is, programs that help programmers develop programs

✗ Programs that help non-programmers (or programmers) develop non-programming artifacts –word processors for writing a book, …

✗ Other tools that help programmers develop programs – pencil & paper, whiteboards, information hiding, …

What environments and tools do you use?

Wikipedia: computer programming tools

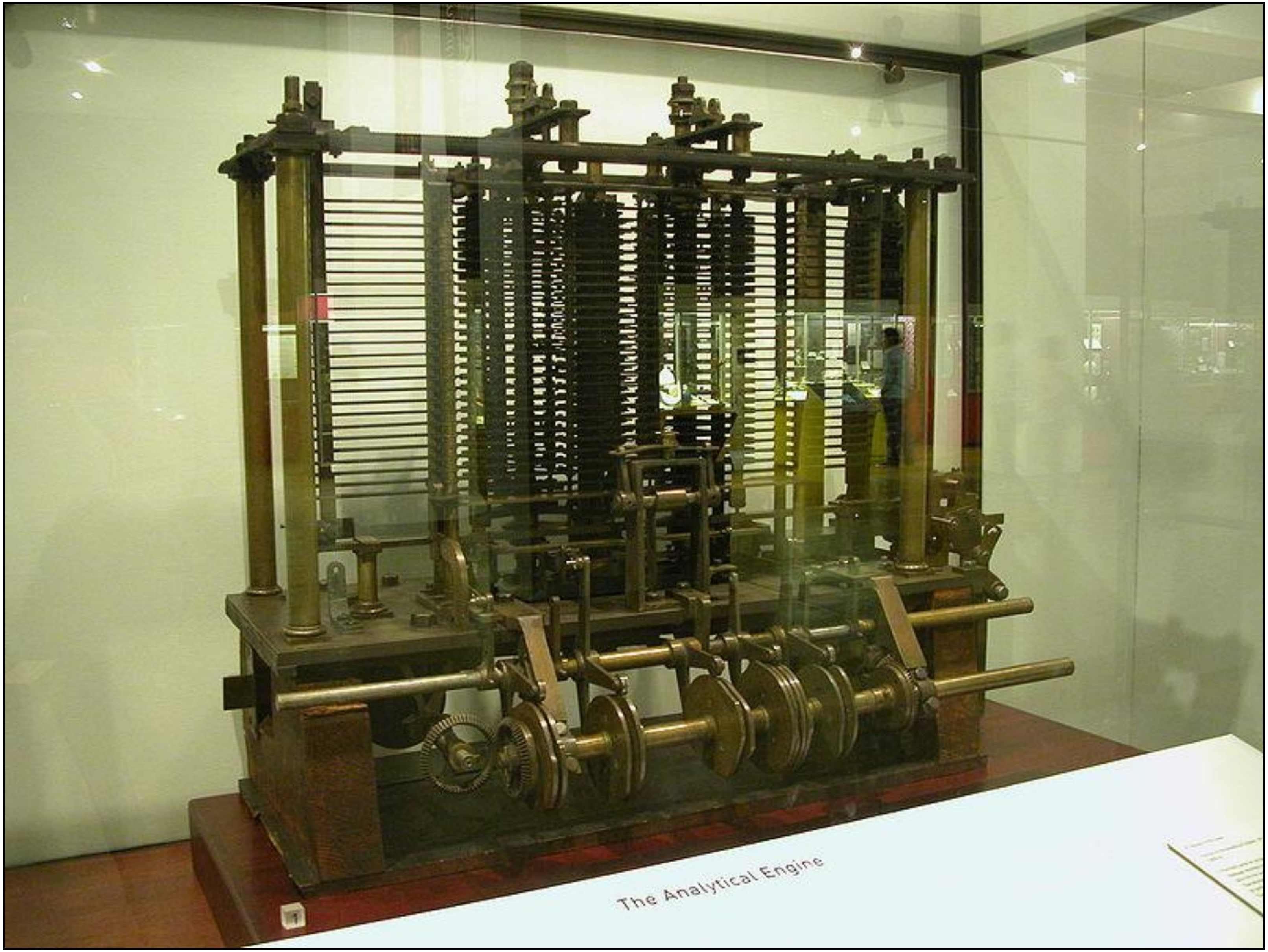
29 subcategories (1/31/2010)

- Bug and issue tracking software (42 pages)
- Build automation (37)
- Code navigation tools (1)
- Code search engines (11)
- Compilers (65)
- Computer-aided software engineering tools (15)
- Debuggers (2)
- Disassemblers (4)
- Discontinued development tools (5)
- Documentation generators (17)
- EXE packers (2)
- Formal methods tools (17)
- Free computer programming tools (25)
- GUI automation (37)
- Integrated development environments (143)
- Java development tools (67)
- Microsoft development tools (44)
- Profilers (17)
- Program testing tools (50)
- Programming language implementation (58)
- Revision control systems (27)
- Static code analysis (58)
- Text editors (38)
- UML tools (26)
- Unix programming tools (58)
- User interface builders (16)
- Web Services tools (2)
- Web development software (57)
- Computer programming tool stubs (124)

What was the first computer?

Analytical Engine(s)

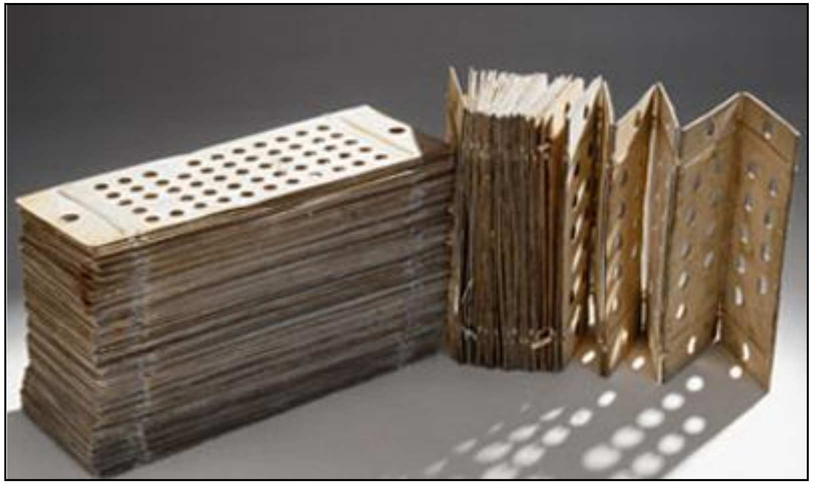
- [Material and photographs from the Computer History Museum web site and Wikipedia]
- Charles Babbage’s (1791-1871) Analytical Engine is “much more than a calculator and marks the progression from the mechanized arithmetic of calculation to fully-fledged general-purpose computation.”
- The logical structure had a separation of the memory (the 'Store') from the central processor (the 'Mill'), serial operation using a 'fetch-execute cycle', and facilities for inputting and outputting data and instructions
- 1,000 numbers of 50 decimal digits each



http://commons.wikimedia.org/wiki/File:AnalyticalMachine_Babbage_London.jpg {{Information |Description = Machine Analytique de Charles Babbage, exposée au Science Museum de Londres Date = 5 Mai 2009 |Auteur = Bruno Barral |Permission = CC-BY-SA-2.5}}

Programming the Analytical Engine

- Programmable using punched cards
- Idea borrowed from the Jacquard loom used for weaving complex patterns in textiles
- The programming language had loops and conditional branching
- Was Turing-equivalent (before Turing)



- Three different types of punch cards used: arithmetic operations, numerical constants, load and store operations

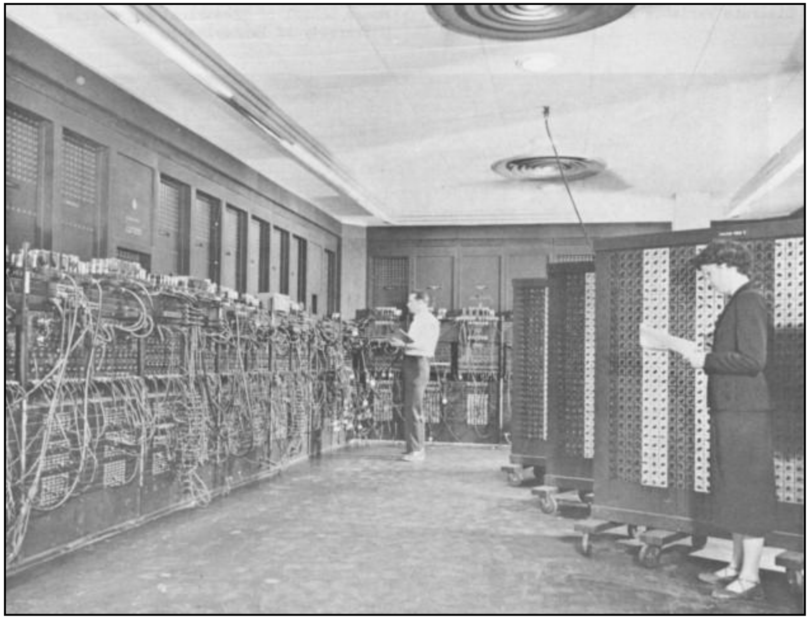
Ada Lovelace: the 1st programmer

- She wrote a method for calculating a sequence of Bernoulli numbers
- It would have run correctly (had the Engine ever been built)
- Babbage called her the "Enchantress of Numbers"
- Died in 1852 at 37 years-old

"C" is for …? Why?

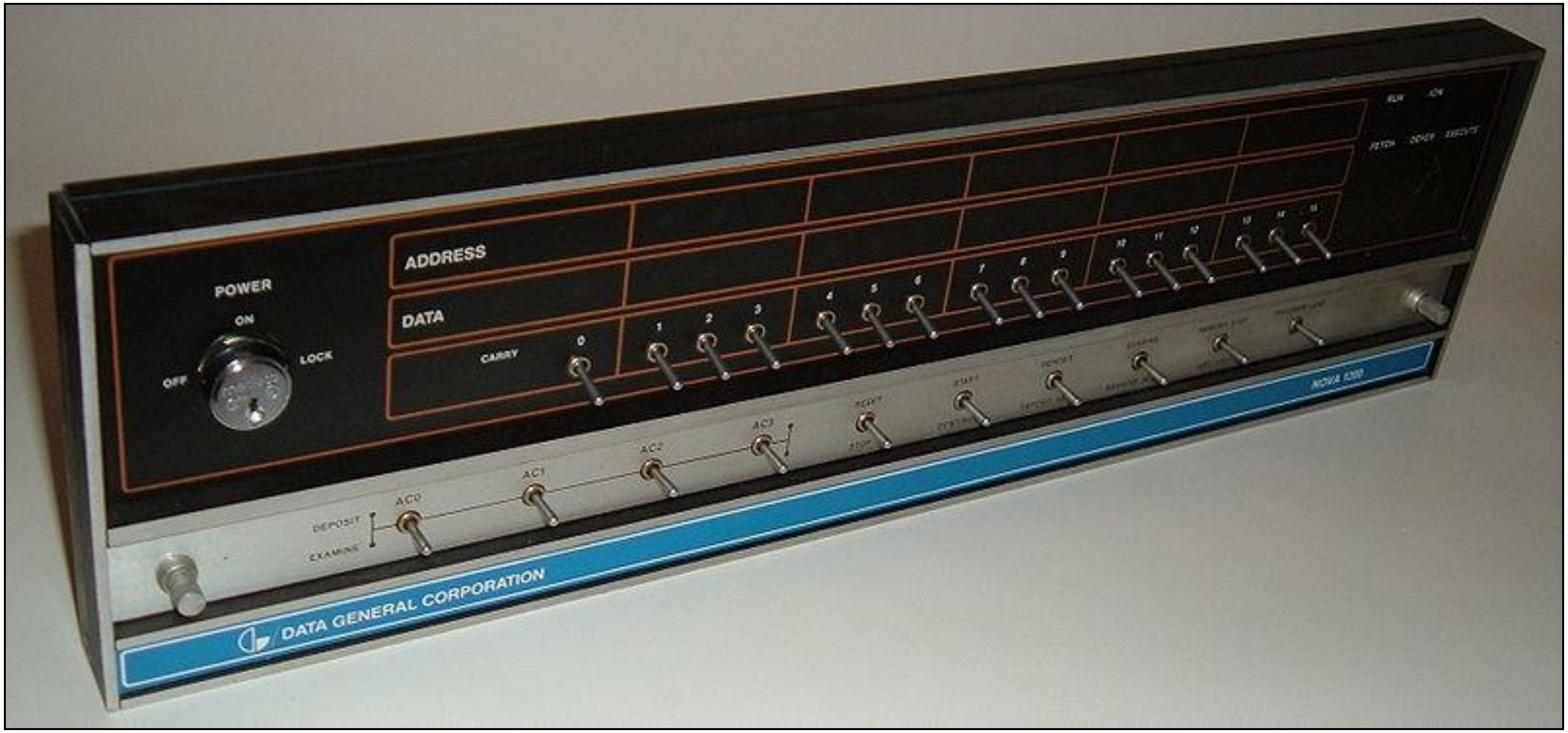
ENIAC [http://www.computersciencelab.com]

- Mauchly-Eckert at Penn in 1943-45
- Circ := 3.14 * diam
  - Rearrange many patch cords, locate and set three specific knobs to 3, 1, 4



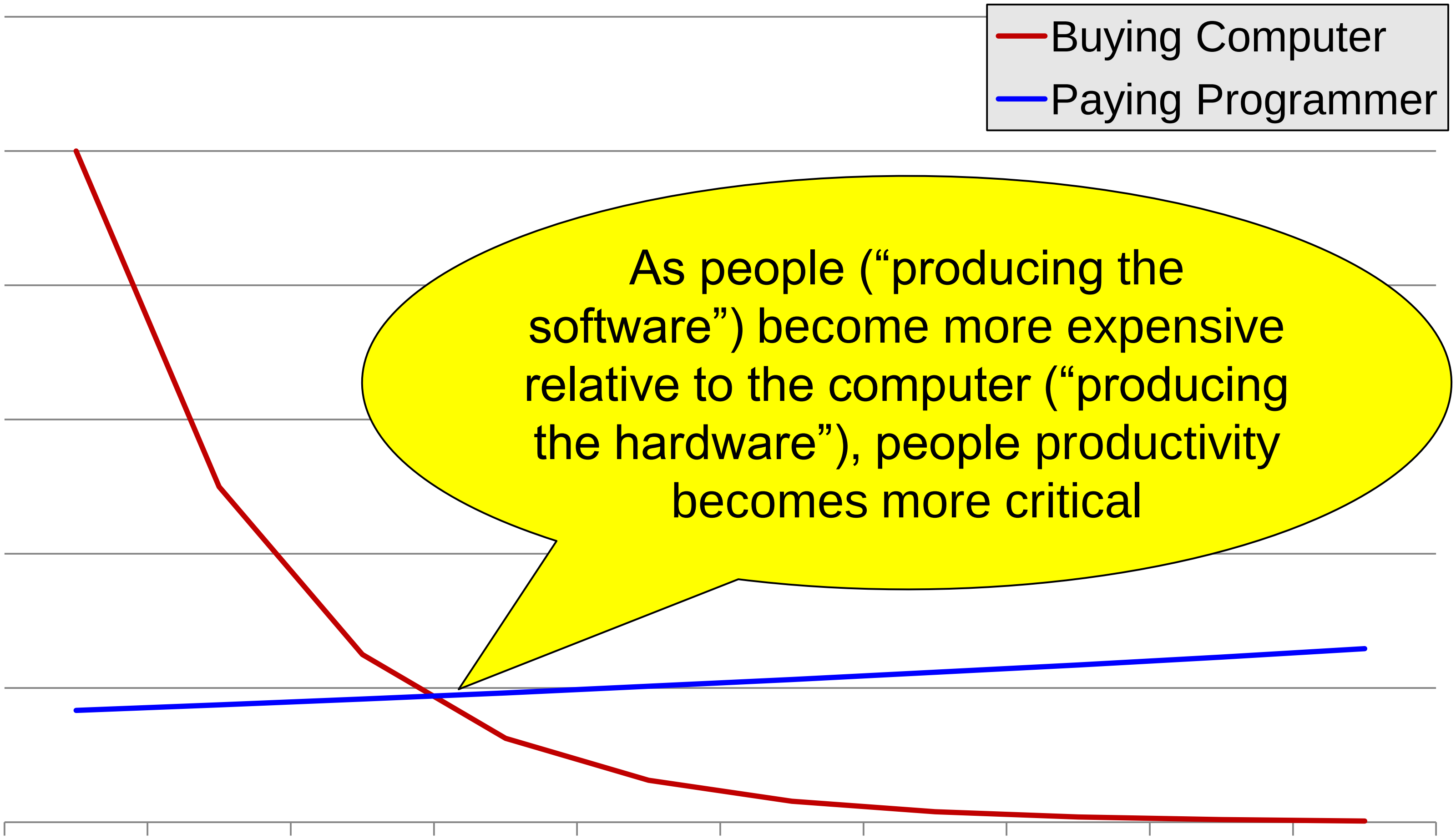
- 20'x40', 30 tons, more than 18K vacuum tubes
- Held 20 numbers
- 0.0028th second for a multiply, 100K cycles/sec
- The first ENIAC program remains classified

Data General Nova



http://commons.wikimedia.org/wiki/File:Nova1200.agr.jpg

The point: what’s expensive?



As people (“producing the software”) become more expensive relative to the computer (“producing the hardware”), people productivity becomes more critical

My view

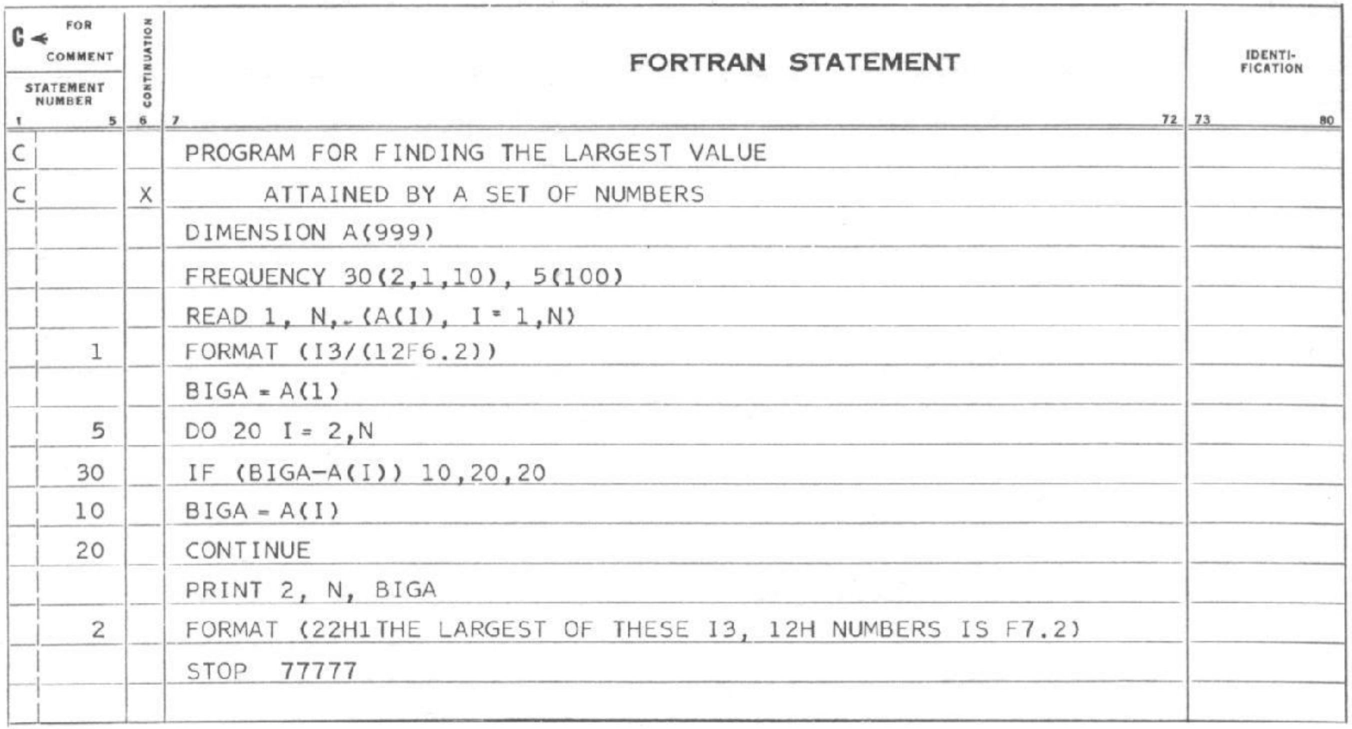
- Successful software tools are designed with an excellent understanding of what computers do well and what people do well
    - Unsuccessful tools generally make a mistake in at least one of these dimensions
- The “what computers do well” dimension surely shifts over time
    - Faster, bigger and cheaper computers
    - Better algorithms, interfaces, ideas, etc.
- The “what people do well” dimension shifts much less slowly over time
    - However, as the cost of quality people increases, the ratio changes as well

Interlude

- Structured reports – questions, concerns?
  - Yes, Virginia, there is a discussion board
- State-of-the-research reports
  - We'll announce turn-in and commenting mechanism soon
  - If there has been one confusion, it has been on how arbitrary the topic can be

Non-interactive programming

- Interactive programming would have been a poor early use of computer time
- Programming 1960’s mainframes was often done by programmers filling in coding forms, with keypunch operators then entering the code – programmers cheaper than computers, keypunchers cheaper than programmers



C ← FOR COMMENT

STATEMENT NUMBER 1 5

CONTINUATION 6

FORTRAN STATEMENT 7 72

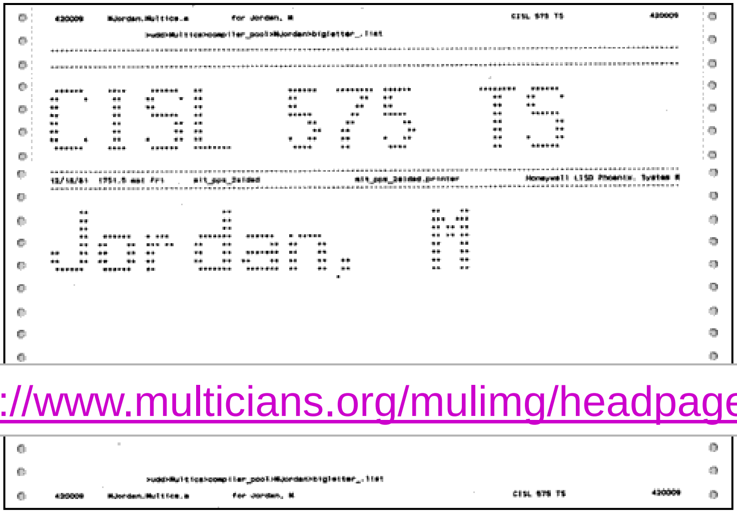
IDENTI-FICATION 73 80

```
C           PROGRAM FOR FINDING THE LARGEST VALUE
C          X    ATTAINED BY A SET OF NUMBERS
            DIMENSION A(999)
            FREQUENCY 30(2,1,10), 5(100)
            READ 1, N,.(A(I), I = 1,N)
      1     FORMAT (I3/(12F6.2))
            BIGA = A(1)
      5     DO 20 I = 2,N
     30     IF (BIGA-A(I)) 10,20,20
     10     BIGA = A(I)
     20     CONTINUE
            PRINT 2, N, BIGA
      2     FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)
            STOP 77777
```

http://www.fortran.com/ibm3.jpg

Keypunch

- IBM 029 – 1960’s and 1970’s
- “Ah, yes, but the real trick is fixing a variable name when you've left out a letter. You could jam one card in place, and either type or space to move the other, then continue duplicating.”
  - Conversations from the Old Programmers' Home http://www.firelily.com/humor/programming/old.prog.html
- Debugging from listings
  - No modifying structure
  - No search
  - No stepping through code
  - …
  - Counterpoint: more thinking



```
420009   MJordan.Multics.a      for Jordan, M                    CISL 575 TS      420009
              >udd>Multics>compiler_pool>MJordan>bigletter_.list

CISL 575 TS

12/18/81  1751.5 mst Fri   alt_pps_3sided     alt_pps_3sided.printer     Honeywell LISD Phoenix, System M

Jordan, M

              >udd>Multics>compiler_pool>MJordan>bigletter_.list
420009   MJordan.Multics.a      for Jordan, M                    CISL 575 TS      420009
```

http://www.multicians.org/mulimg/headpage.gif

High-level languages and compilers

- High-level languages were questionable because of difficulties in developing compilers
- There was a lack of theory, a lack of experience, limited memory capacity, and no lessening of performance demands
- As porting software to new architectures became more frequent, assembly languages became increasingly limiting – technology was developed that enabled compilers
  - Nice feedback loop between new architectures, high-level languages, compiling techniques, bootstrapping, etc.

Interactive editing: line editors

- No window or screen display of content
- Short commands that minimized typing
- Search and (usually) random-access to lines
- Localized modifications (e.g., fix typos)
- Repeated modifications (e.g., change variable name)
- Search/replace initially based on simple matching, later on regular expressions
- Significant improvement on keypunches

Interactive editing: screen editors

- Video screens enabled screen editors
  - Indeed, screen editors and resulting productivity arguably increased the market for video screens
- Provided visible context – not only a cursor, but a cursor in context of a sequence of lines of data
- **`vi`** – screen version of **`ed`**, still in use
- Customizable/programmable (e.g., macros) – **`TECO`**, **`emacs`**, …

- But compiling, running, etc. was still a batch process

Emacs: MIT AI Lab (from emacswiki)

- 1972: display (screen) editing added to **`TECO`**
- 1974: Stallman added macros to **`TECO`**
- 1976: Stallman combined scads of macros into a single command set and an extensibility mechanism – “Editor MACrosS”
- 1978: Bernard Greenberg wrote **`MulticsEmacs`** (at Honeywell) in **`MacLisp`**, including user extensions
- 1981: Gosling **`Emacs`**, the first variant to run on Unix, written in C with **`MockLisp`** for extension
- 1984: Stallman started GNU **`Emacs`**

Emacs feature: extensibility

- Add libraries/packages/extensions
- Extensions turn **`Emacs`** into an environment in which programmers edit, compile, and debug code using a single interface
- And many more not specifically related to programming – calculators, calendars, document processing support (**`TeX`**, **`LaTeX`**), web browsing, email, chat clients, online help, spreadsheets, etc.

Emacs feature: customizability

- Simple setting of common customization variables such as the color scheme
- Defining keyboard macros
- **`.emacs`** startup files that set variables, set key bindings, etc.
  - Many people have huge **`.emacs`** files with wildly idiosyncratic customizations

Emacs feature: editing modes

- “Major modes” to provide convenience for text, for most programming languages, for **`HTML`**, for **`TeX/LaTeX`**, etc.
  - Syntax highlighting using different fonts or colors
  - Automatic indentation
  - “Electric” features such as automatic insertion of elements such as spaces, newlines, and parentheses
  - Special editing commands like commands to jump to the beginning and the end of a function
- “Minor modes” to support (for example) different indentation styles within the C “major mode”

Emacs features: other

- Performance
- Portability
- Self-documenting
- Manuals
- …

Editors: discussion

Interpreters

- Interactive access to computers enabled interpreters
- Tend to reduce time-to-execute (-test)
- Tend to make finding bugs and fixing them easier than for compiled languages
- Also expanded the number of programmers and the variation in their background (and possibly their ability)

`BASIC` (Kemeny/Kurtz ‘83) principles

- Easy for beginners (students, non-science and non-math focus)
- General-purpose programming language
- Allow advanced features to be added for experts
- Interactive
- Good error messages
- Quick response
- Allow programmers to remain unaware of hardware and OS

Debuggers

- In 1945, a moth was found trapped in and then removed from Harvard’s Mark II Aiken Relay Calculator
- The operator entry said this was the “first actual case of bug being found” and that they had successfully "debugged" the problem
- Features of debuggers include breakpoints, tracing, watchpoints, examining and set variables, examining the callstack, …

Interactive Fortran (circa 1969)

N. Rawlings (©1995-2007 Computer History Museum)

- FDEBUG
  - modifications to compiler (to keep the symbol table)
  - modifications to the operating system (to insert illegal instructions, give control to the debugger/user, etc.)
  - Programmers could set breakpoints, conditional breakpoints, change variable values, etc.
- "This was brilliant and it was a huge breakthrough in the ease of debugging Fortran programs. No one else matched it for years! Worldwide! Before then, the program had to change in order to debug it – and that change often moved the bug one was trying to isolate."

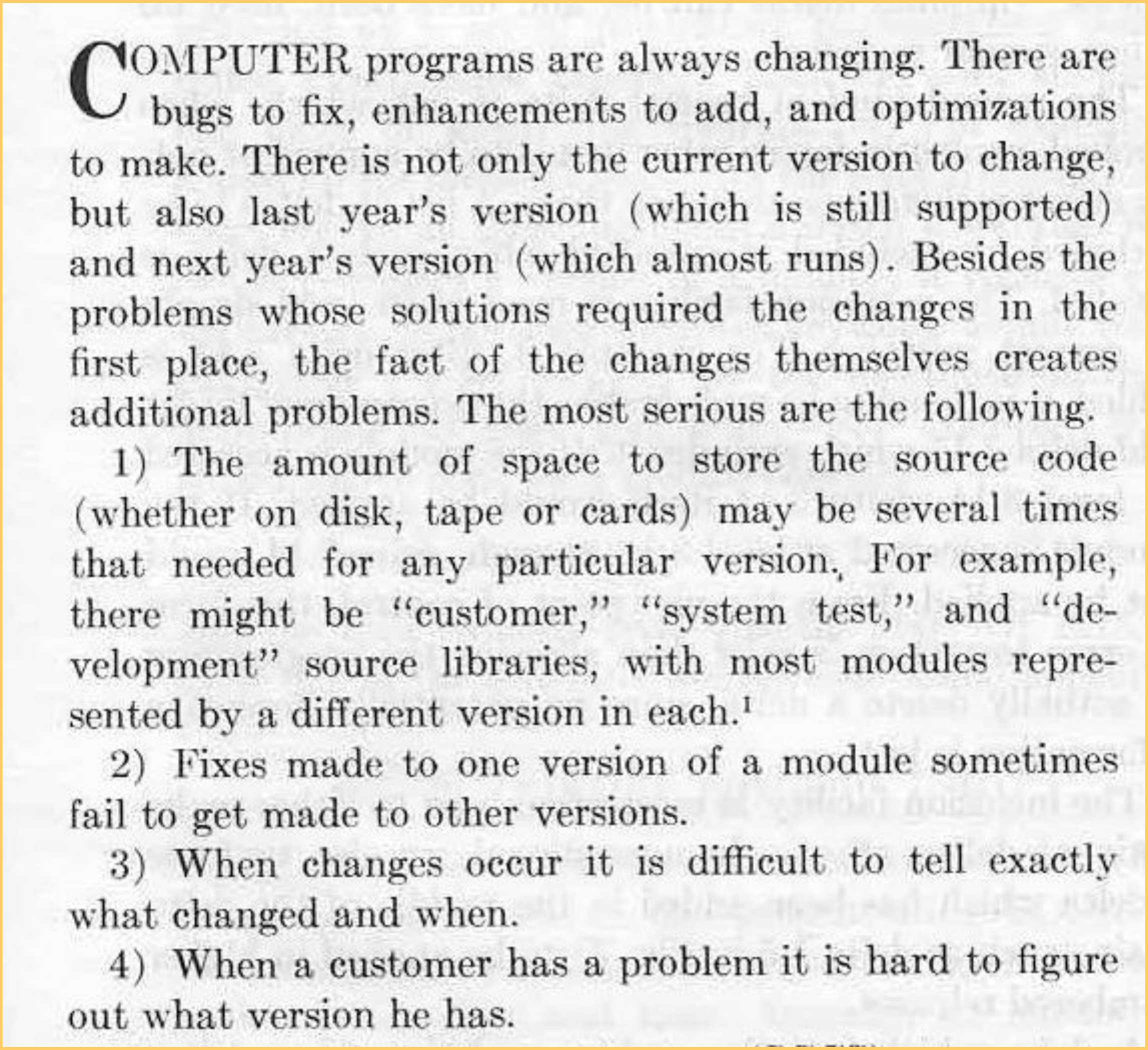
Sequence of Unix debuggers

- **`adb`** – absolute debugger, essentially symbolic access at the assembly language level
- **`dbx`** (1981, Linton) – symbolic, multiple languages supported, etc.
- **`gdb`**, **`DDD`** – “modern” **`dbx`** and screen-based debuggers

Windows debuggers?

Source code control tools

- AKA revision control, version control, source code management, …
- SCCS, Marc Rochkind, *IEEE Transactions on Software Engineering* SE-1:4 (Dec. 1975). – first ICSE N-10 Most Influential Paper recipient



COMPUTER programs are always changing. There are bugs to fix, enhancements to add, and optimizations to make. There is not only the current version to change, but also last year's version (which is still supported) and next year's version (which almost runs). Besides the problems whose solutions required the changes in the first place, the fact of the changes themselves creates additional problems. The most serious are the following.

1) The amount of space to store the source code (whether on disk, tape or cards) may be several times that needed for any particular version. For example, there might be "customer," "system test," and "development" source libraries, with most modules represented by a different version in each.[1]

2) Fixes made to one version of a module sometimes fail to get made to other versions.

3) When changes occur it is difficult to tell exactly what changed and when.

4) When a customer has a problem it is hard to figure out what version he has.

Post-SCCS…

- RCS (~backward deltas)
- CVS (~client-server based RCS)
- SVN (~”improvement” on CVS)
- ClearCase (~versions supported below application level)
- Then came distributed revision control systems
  - Arch, Monotone, BitKeeper, Git, Bazaar, …

Distributed revision control

- Multiple “central” repositories permitted
- Most operations don’t require network access
- Allows different structures of authority, perhaps better suited for open-source projects
- Not reliant on a single machine
- Concepts are not as simple to grasp

Source code control: discussion

Other tools

- Testing tools
- Bug databases
- …
- …others?

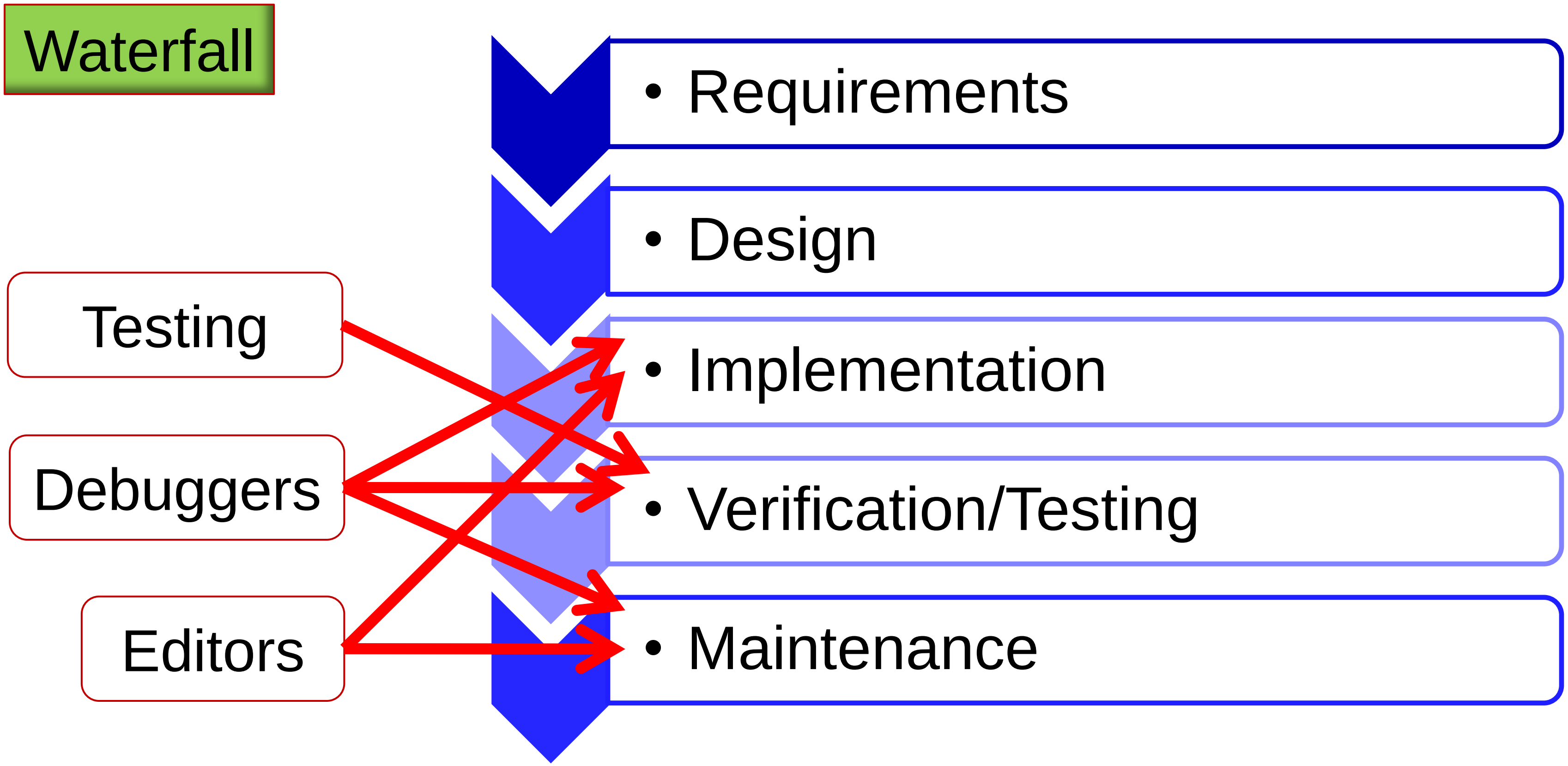
Final two lectures: suggestions

- Software testing
- Discussion of papers in groups
- Design patterns

- Here’s what I think I want
  - One lecture on software engineering economics [me]
  - One lecture used for presentations on your state-of-the-research reports [you]
    - But we don’t likely have time for every person or group to present (even 10 minutes/presentation lets us do at most 15 in one evening)
    - An alternative idea: an all-at-UW poster session evening, with people asking and answering questions on these papers
- Last suggestions by Wednesday, and then I’ll decide

Roles of tools

- What aspects of the software lifecycle do they address? Can they address? Should they address?



Waterfall

- Requirements
- Design
- Implementation
- Verification/Testing
- Maintenance

Testing

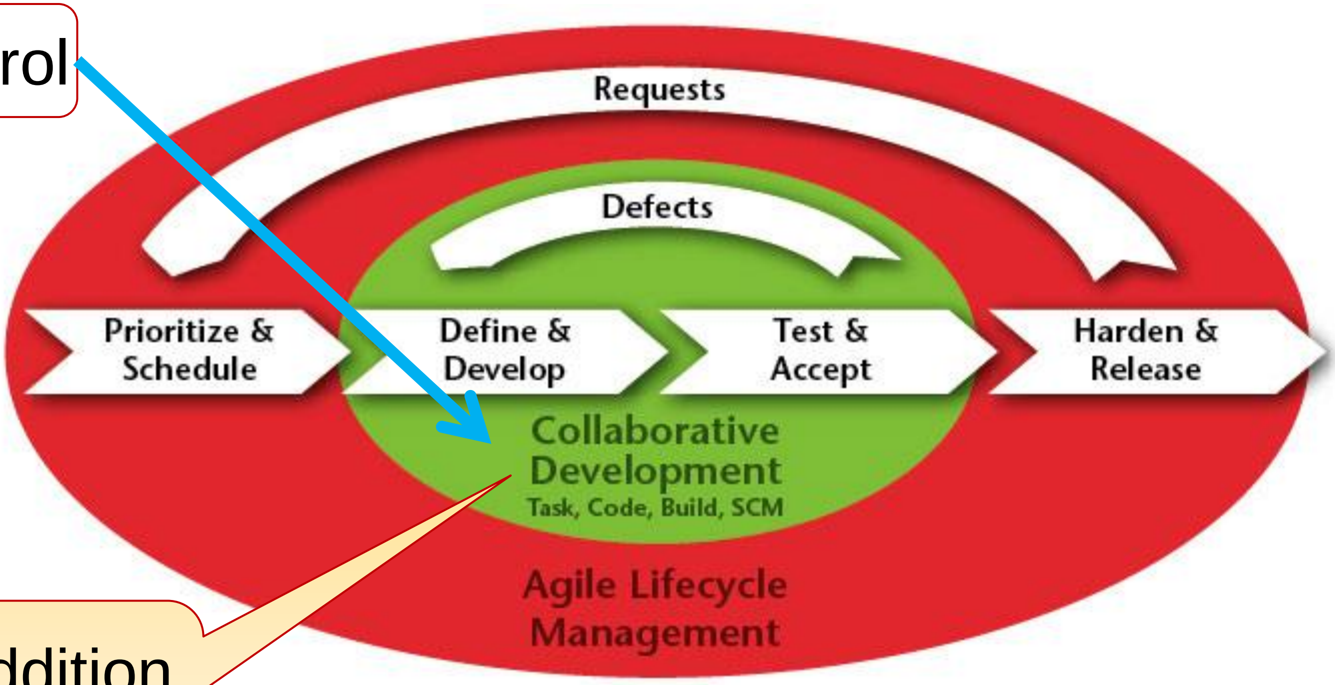
Debuggers

Editors

Roles of Tools

Version control

Agile



Major addition from waterfall

© 2010 Rally Software Development Corp
http://www.rallydev.com/agile_products/lifecycle_management/

Tools and environments

- Understanding
- Building
- Modifying
- Debugging
- Testing
- …

X

- Interaction and rich user interfaces
- Translation of high-level descriptions
- Maintaining consistency among large and complex representations
- Encoding knowledge about an activity, organization or process
- Broader availability of pertinent information
- Communication medium
- …

Environments vs. tools

- Degree of focus on a single task vs. a set of tasks
- Degree to which the user interacts in a uniform way
- Degree to which the user is responsible for managing consistency
- Degree to which there is a shared representation

Why environments? 30 years ago

- “During the past decade there has been a growing realization [that software tools] have by and large failed to reduce cost and improve quality. … [T]he essence of an environment is that it attempts to redress the failures of individual software tools through synergistic integration.” Osterweil, 1981
- “Current software development environments often help programmers solve their programming problems by supplying tools such as editors, compilers, and linkers, but rarely do these environments help projects solve their system composition or management problems.” Notkin & Habermann, 1979
- “A software development environment consists of a set of techniques to assist the developers of software systems, supported by some (possibly automated) tools, along with an organizational structure to manage the process of software development. Historically, these facilities have been poorly integrated.” Wasserman, 1981

Done deal

- There is little doubt that users prefer integrated tool sets
  - This can come from integrated development environments like VisualStudio or Eclipse – along with extensions
  - This can come from aggressive extensions of tools like Emacs
- Although the engineering and the “feel” may be different, there seems to be little question that blurring the edges of editor, compiler, debugger, source code control, etc. is generally preferred

Unix: toolkit-based environment

- Simple integration mechanism
  - Convenient user-level syntax for composition
- Standard shared representation
- Language-independent (although biased)
- Efficient for systems’ programming

Interlisp (Xerox PARC)

- Started in 1967 at BBN as BBN Lisp
- Moved to Xerox PARC with Bobrow, Teitelman and others and became Interlisp
- Lisp machines were developed and sold InterLisp-D.
- The environment was notable for the integration of interactive development tools – debugger, DWIM, etc.
- Very fast turnaround for code changes
- Monolithic address space
  - Environment, tools, application code commingled
- Code and data share common representation

Teitelman and Masinter, 1981

- “From its inception, the focus of the Interlisp project has been not so much on the programming language as on the programming environment. … The programmer's environment influences, and to a large extent determines, what sort of problems he can (and will want to) tackle, how far he can go, and how fast. If the environment is cooperative and helpful (the anthropomorphism is deliberate), the programmer can be more ambitious and productive. If not, he will spend most of his time and energy fighting a system that at times seems bent on frustrating his best efforts.”
- “The second major influence in Interlisp's development was a willingness to ‘let the machine do it.’ The developers were wiling to expend computer resources to save people resources because computer costs were expected to continue to drop.”

Features

- DWIM – Do What I mean
- The system detects errors attempts to guess what the user might have intended – a “pervasive philosophy of user interface design”
- The environment as an agent of the programmer – the programmer’s assistant

- MASTERSCOPE
- For analysis and cross-referencing
- Information about function calls and variable usage

Smalltalk-80 (Xerox PARC)

- Alan Kay and others at Xerox PARC
- Language and environment – largely indivisible
- Pure OO, dynamically-typed, reflective – metaclasses to make “everything an object”
- Environment structured around language features – class code browser/editor, protocols, inspectors, notifiers, etc.
- Rich libraries (data structures, UI, etc.)

Cedar (Xerox PARC)

- Intended to mix best features of Interlisp, Smalltalk-80, and Mesa
- Primarily was an improvement on Mesa
  - Language-centered environment
  - Abstract data type language
    - Strong language and environment support for interfaces
  - Key addition: garbage collection

Some illustrative research environments

Toolpack (Osterweil, 1983)

- Consider breadth of tools needed for software development (static analysis and testing tools, documentation, etc.)
- “Tool fragments” to support tight integration of tools into an environment
- Centralized tree-structured file system for sharing data
- Focus on mathematical software

Gandalf (Habermann, Notkin, et al.)

- Structure-based environments
  - Direct-manipulation of program objects – intent was to decrease the gap between the programmer and the compiler through sharing the abstract syntax tree
- Environment generation – to allow multiple language support
  - Integration through implicit invocation by active abstract syntax trees
  - Shared database structured on ASTs

Initial IPE: Medina-Mora & Feiler 81

- “The integration of the tools allows IPE [interactive programming environment] to present the programmer with a uniform view of the program in terms of its source form. The program is manipulated through a syntax-directed editor, and its execution is controlled by a language oriented debugger. The debugging actions are embedded in the supported language and are invoked by editing the program. … [The] tools and their representation are not visible to the user, the only (thus uniform) interface is the user interface of the editor.”

Mentor

- Donzeau-Gouge, Huet, Kahn, Lang, Levy, 1984
- Structure-based environment
- Users could define dynamic views
  - General-purpose tree manipulation language
- Formal basis for semantic definition
  - Led to Centaur generation system
  - Used natural semantics

Cornell

- Program Synthesizer [Teitelbaum & Reps, 1981]
  - Syntax-based editing environment
  - Text at expression-level
- Synthesizer Generator [Reps & Teitelbaum, 1984]
  - Generation of syntax-based editors
  - Based on definition of attribute grammars
  - Incremental attribute grammar update algorithm [Reps 1983]

Structure-oriented editors

- These failed because they
  - unnecessarily and overly constrained the users
    - one editor, one style
    - hard to integrate external tools
    - not bad for novice environments
  - tried to reduce the wrong cost
    - getting syntax (and static semantics) right isn’t such a big deal
  - costly to produce (even after tons of research)

Structure-oriented editors

- Some language-oriented features have flourished
- Every successful program editor now encodes knowledge about the programming language syntax and often semantics
- Language-knowledgeable refactoring is a part of editors now – this requires a richer representation than text alone
- Some commands – like “compile” – are largely disappearing (something Gandalf aspired to 30 years ago)

Arcadia (R. Kadia, 1992)

- Process-based environment
  - Process definition and execution
- Analysis and testing
- Measurement and evaluation
- UI development and management
- Event-based integration
- Typed object-base

Pecan (Reiss, 1984)

- Graphically-based environment
- Multiple, concurrent views
  - Data structures
  - Symbol table
  - Flowchart
  - Nassi-Shneiderman diagrams
- Many views read-only

Desert (Brown, Reiss)

- Single editor and lightweight data integration through fragments
- Ability to dynamically jump among logically connected files, and the ability to create and edit virtual files, files that contain information relevant to the current task extracted from all over the system.

Intentional Programming (http://intentsoft.com/)

- Charles Simonyi – first at Microsoft/MSR
- High-end domain-specific programming – exploit both knowledge and notation from specific domains
- Some domains are claimed as successes – some European financially-oriented efforts, for example
- “Traditional” programming is not, but then again the goal is to reduce or eliminate that, not to support it

Endeavors (UCI, Taylor & Redmiles)

- An open, distributed, extensible process execution environment
- Designed to improve coordination and managerial control of development teams by allowing flexible definition, modeling, and execution of typical workflow applications

Some myths: environments

- *Integration is job #1*
  - Integrating tools helps, but only if the tools are the “right” tools
  - That is, integration is a second order effect, not a first order effect
  - This also suggests that extensibility, replaceability, etc. are crucial because the “right” tools necessarily change over time
- *Graphics inherently dominate text* – “A picture is worth a thousand words”
  - Screen real estate
  - Sharing with other tools
  - Sharing with other people

Some myths: CASE & environments

- *Goal should be to change how software engineering is done*
  - No, it should be to enhance how people are doing software engineering
- *The tools can handle creative aspects of software engineering*
  - Tools frequently fail to be useful because they make poor judgments about what the human does well and what the computer does well

Modern IDEs

- “Eclipse started as a Java IDE, but has since grown to be much, much more. Eclipse projects now cover static and dynamic languages; thick-client, thin-client, and server-side frameworks; modeling and business reporting; embedded and mobile; and, yes, we still have the best Java IDE.” http://www.eclipse.org/projects/
- Microsoft Visual Studio is an IDE that supports development of programs for virtually all Microsoft platforms. Framework is language-independent, with plug-in structures to specialize the environment to given languages, source code control systems, etc.

Eclipse architecture

- Lightweight plug-in mechanism on top of a run-time system supports extensibility, integration and uniformity
- The IDE has an incremental Java compiler and a full model of the Java source files, enabling plug-ins for language syntax and semantics

Tools plug-ins (984)

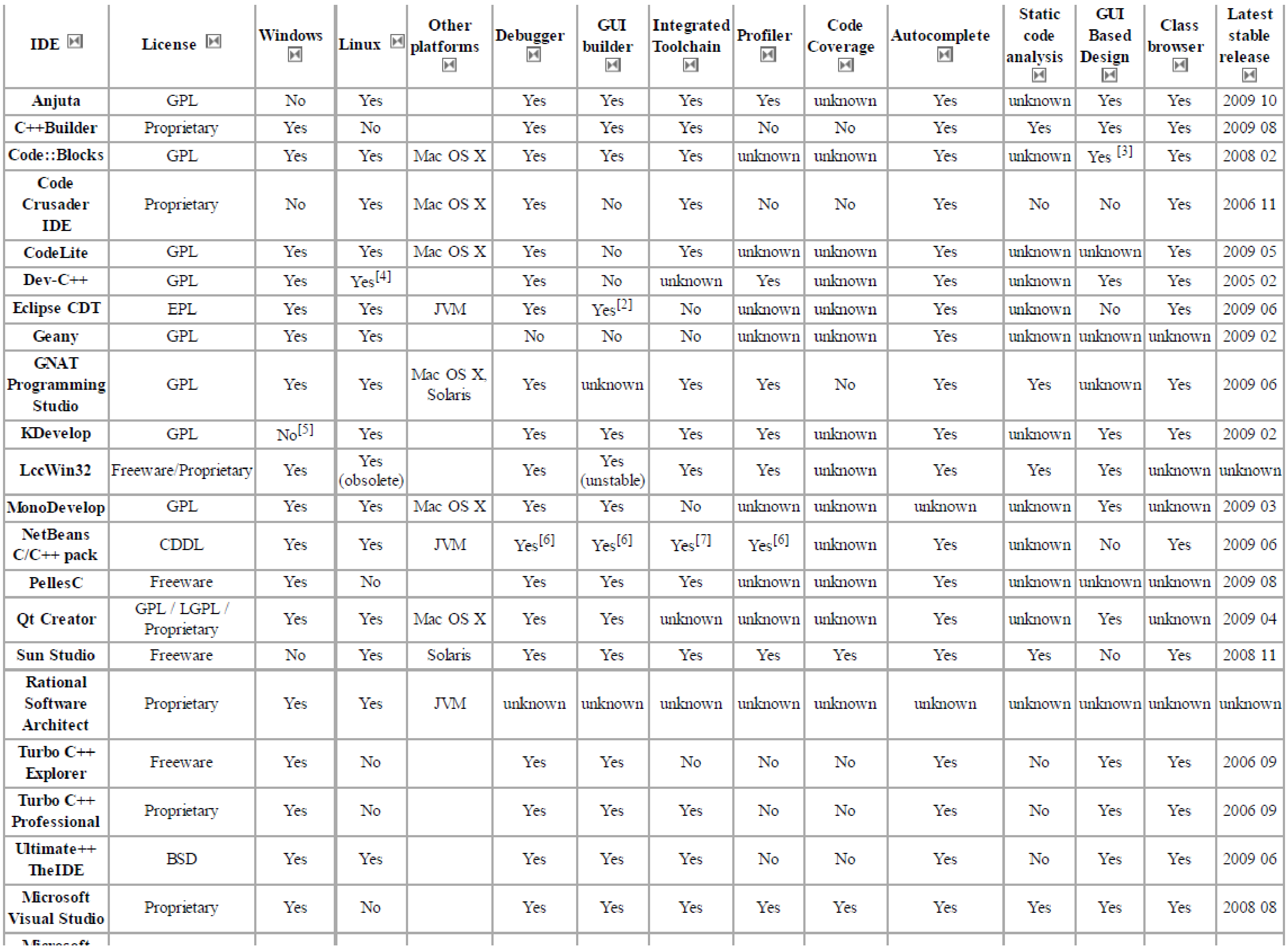
- Application Management (46)
- Application Server (17)
- Build and Deploy (54)
- Code Management (51)
- Database (32)
- Database Persistence (3)
- Documentation (27)
- Editor (90)
- Entertainment (9)
- Graphics (16)
- IDE (113)
- J2EE Development Platform (25)
- J2ME (11)
- Languages (62)
- Logging (4)
- Modeling Tools (81)
- Mylyn Connectors (8)
- Network (10)
- Other (41)
- Process (19)
- Profiling (14)
- Reporting (5)
- Rich Client Applications (12)
- SCM (13)
- Search (3)
- Source Code Analyzer (57)
- Systems Development (16)
- Team Development (52)
- Testing (61)
- Tools (215)
- UI (58)
- UML (37)
- Web (43)
- Web Services (16)
- XML (16)

Platform components

• Ant	Eclipse/Ant integration
• Core Platform	Runtime and resource management
• CVS Platform	CVS Integration
• Debug	Generic execution debug framework
• Releng	Release Engineering
• Search	Integrated search facility
• SWT	Standard Widget Toolkit
• Team/Compare	Generic Team & Compare support frameworks
• Text	Text editor framework
• User Assistance	Help system, initial user experience, …
• UI	Platform user interface
• Update	Dynamic Update/Install/Field Service

Wikipedia: truncated C/C++…

2/1/2010 Comparison of integrated developmen…



IDE	License	Windows	Linux	Other platforms	Debugger	GUI builder	Integrated Toolchain	Profiler	Code Coverage	Autocomplete	Static code analysis	GUI Based Design	Class browser	Latest stable release
Anjuta	GPL	No	Yes		Yes	Yes	Yes	Yes	unknown	Yes	unknown	Yes	Yes	2009 10
C++Builder	Proprietary	Yes	No		Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	2009 08
Code::Blocks	GPL	Yes	Yes	Mac OS X	Yes	Yes	Yes	unknown	unknown	Yes	unknown	Yes [3]	Yes	2008 02
Code Crusader IDE	Proprietary	No	Yes	Mac OS X	Yes	No	Yes	No	No	Yes	No	No	Yes	2006 11
CodeLite	GPL	Yes	Yes	Mac OS X	Yes	No	Yes	unknown	unknown	Yes	unknown	unknown	Yes	2009 05
Dev-C++	GPL	Yes	Yes[4]		Yes	No	unknown	Yes	unknown	Yes	unknown	Yes	Yes	2005 02
Eclipse CDT	EPL	Yes	Yes	JVM	Yes	Yes[2]	No	unknown	unknown	Yes	unknown	No	Yes	2009 06
Geany	GPL	Yes	Yes		No	No	No	unknown	unknown	Yes	unknown	unknown	unknown	2009 02
GNAT Programming Studio	GPL	Yes	Yes	Mac OS X, Solaris	Yes	unknown	Yes	Yes	No	Yes	Yes	unknown	Yes	2009 06
KDevelop	GPL	No[5]	Yes		Yes	Yes	Yes	Yes	unknown	Yes	unknown	Yes	Yes	2009 02
LccWin32	Freeware/Proprietary	Yes	Yes (obsolete)		Yes	Yes (unstable)	Yes	Yes	unknown	Yes	Yes	Yes	unknown	unknown
MonoDevelop	GPL	Yes	Yes	Mac OS X	Yes	Yes	No	unknown	unknown	unknown	unknown	Yes	unknown	2009 03
NetBeans C/C++ pack	CDDL	Yes	Yes	JVM	Yes[6]	Yes[6]	Yes[7]	Yes[6]	unknown	Yes	unknown	No	Yes	2009 06
PellesC	Freeware	Yes	No		Yes	Yes	Yes	unknown	unknown	Yes	unknown	unknown	unknown	2009 08
Qt Creator	GPL / LGPL / Proprietary	Yes	Yes	Mac OS X	Yes	Yes	unknown	unknown	unknown	Yes	unknown	Yes	unknown	2009 04
Sun Studio	Freeware	No	Yes	Solaris	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	2008 11
Rational Software Architect	Proprietary	Yes	Yes	JVM	unknown	unknown	unknown	unknown	unknown	unknown	unknown	unknown	unknown	unknown
Turbo C++ Explorer	Freeware	Yes	No		Yes	Yes	No	No	No	Yes	No	Yes	Yes	2006 09
Turbo C++ Professional	Proprietary	Yes	No		Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	2006 09
Ultimate++ TheIDE	BSD	Yes	Yes		Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	2009 06
Microsoft Visual Studio	Proprietary	Yes	No		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	2008 08
Microsoft														

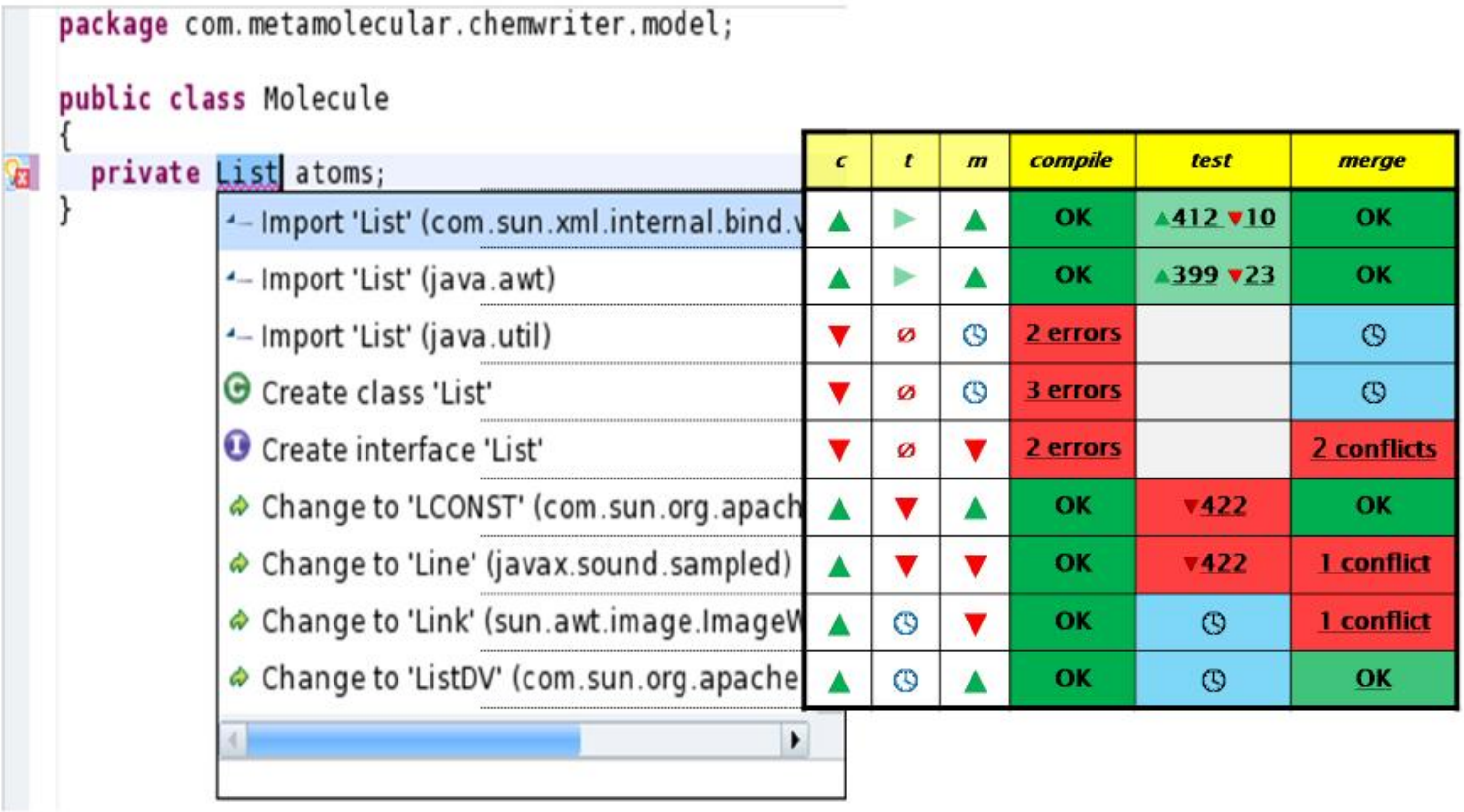
Visual Studio

- A code editor, with IntelliSense (a modern-day DWIM) and refactoring
- Integrated source- and machine-level debuggers
- Support for GUI, web and database design
- Can add plug-ins for domain-specific languages and tools for other aspects of the lifecycle

Distributed software development

- What tools are needed?

Speculation: ongoing research @ UW



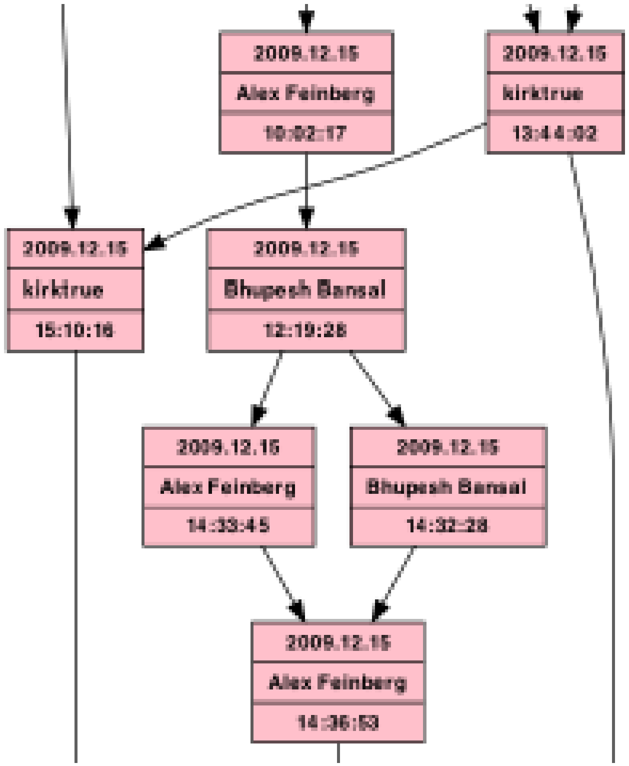
```
package com.metamolecular.chemwriter.model;

public class Molecule
{
  private List atoms;
}
```

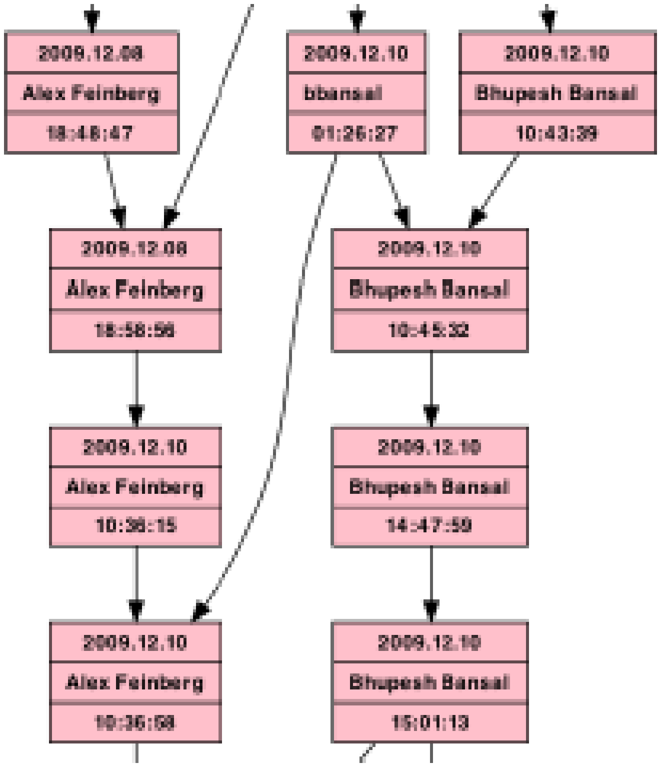
	c	t	m	compile	test	merge
Import 'List' (com.sun.xml.internal.bind.v	▲	▶	▲	OK	▲412 ▼10	OK
Import 'List' (java.awt)	▲	▶	▲	OK	▲399 ▼23	OK
Import 'List' (java.util)	▼	∅	◷	2 errors		◷
Create class 'List'	▼	∅	◷	3 errors		◷
Create interface 'List'	▼	∅	▼	2 errors		2 conflicts
Change to 'LCONST' (com.sun.org.apach	▲	▼	▲	OK	▼422	OK
Change to 'Line' (javax.sound.sampled)	▲	▼	▼	OK	▼422	1 conflict
Change to 'Link' (sun.awt.image.ImageW	▲	◷	▼	OK	◷	1 conflict
Change to 'ListDV' (com.sun.org.apache	▲	◷	▲	OK	◷	OK

Figure 1: Mockup of the user interface for displaying contingent validation results.

Modified from http://depth-first.com/articles/2008/01/11/my-favorite-eclipse-shortcut-quick-fix © 2006-2007 Richard L. Apodaca. Original content licensed under the Creative Commons Attribution-Share Alike 3.0 United States License.

Speculation over merging?





Next two lectures (no lecture 2/15)

- Next week
  - A few tools in some more depth: likely candidates include reflexion models, continuous testing, concolic testing, delta debugging, …
- February 22 (Reid Holmes): Future directions and areas for improvement
  - Rational behind the drive towards integration
    - Capturing latent knowledge
    - Task specificity / awareness
    - Supporting collaborative development

Questions?