Factors such as increasing software complexity and the need for faster time-to-market are driving the adoption of integrated development environments (IDEs). OSDL's Bill Weinberg and MontaVista Software's Jacob Lehrbaum report how these graphical interfaces are increasingly replacing command-line tools

The ideals of IDEs

Over the last decade, both recently graduated and currently employed software developers have increasingly adopted graphical integrated development environments (IDEs) as a replacement for traditional command-line tools. This trend encompasses all types of programming, from the enterprise to embedded development. The forces driving this trend include: ever-increasing size and complexity of application code; the need to automate and accelerate the build cycle to accelerate time-to-market; the requirement to streamline training and make developers more productive more quickly; and the near-ubiquitous availability and use of products like Microsoft Visual Studio for Windows application development and the emerging Open Source Eclipse environment for Linux and other platform development.

THE RACE TO MARKET

In response to forces that include increasing global competition, commoditisation of IT products, and demands for greater returns on engineering investment, time-to-market pressure on software developers climbs year over year. Projects delivered in 18-24 months in 1994 must now leap from concept to delivery in just 9-12 months. Even as development pace has redoubled, software content has grown more precipitously - software accounted for a mere 10 per cent of cell phone development efforts ten years ago, but today can top 90 per cent of handset development budgets, with millions of lines of code being crafted by one to two hundred software engineers for successful feature-phones and smart handsets.

Projects delivered in 18-24 months in 1994 must now leap from concept to delivery in just 9-12 months.

As complexity grows, software developers are expected to become even more efficient and to work in collaboration with increasingly large and distributed teams.

Quantitative arguments for IDE usage also arise from application complexity:

- A 5,000 to 10,000-line application comprising a few dozen files is easily managed by one to five developers with a minimum of specialised tools.
- Applications with 10,000-100,000 lines of code composed of hundreds of source files, with a team of a dozen developers, greatly benefit from source code control and build tools like CVS and make. Individual developers and their manager also benefit from project/code visualisation tools, and IDEs simplify code contribution and building.
- For projects with over 100,000 lines of code, up into the millions of lines, built from thousands of source files, revision and source control are essential; hundreds of engineering team members greatly benefit from a common repository and build engine. IDEs become essential even to perform simple programming tasks.

A BRIEF HISTORY OF TOOLS

There exists a core metric in software engineering - a tenet which states that programmers can produce 7¬±2 lines of maintainable code per day. The whole history of software engineering has focused on making that creative act more efficient by letting each line of code accomplish more work, through greater abstraction.

At the dawn of IT, computers were programmed by physical insertion of plugs and patch cords. Von Neumann's revelation - that programs could be stored in memory and treated like common data - revolutionised the field by letting programmers differentiate software from the hardware it ran on, and build programs that processed other programs.

The simplest programmer's tools have not changed in half a century: the humble assembler still consumes human-readable, low-level assembly code and emits executable machine code (binary). As a productivity aid in the 1960s, high-level languages like FORTRAN, BASIC and ALGOL were born, offering greater abstraction but requiring compilers and interpreters to translate and execute programs on computer hardware.

In the 1970s and '80s, block structured languages like Pascal, stack languages like FORTH, and the ubiquitous high-/low-level C language rose to dominance. And, in the last decade, object-oriented programming in C++ and Java has become the norm for many types of development.

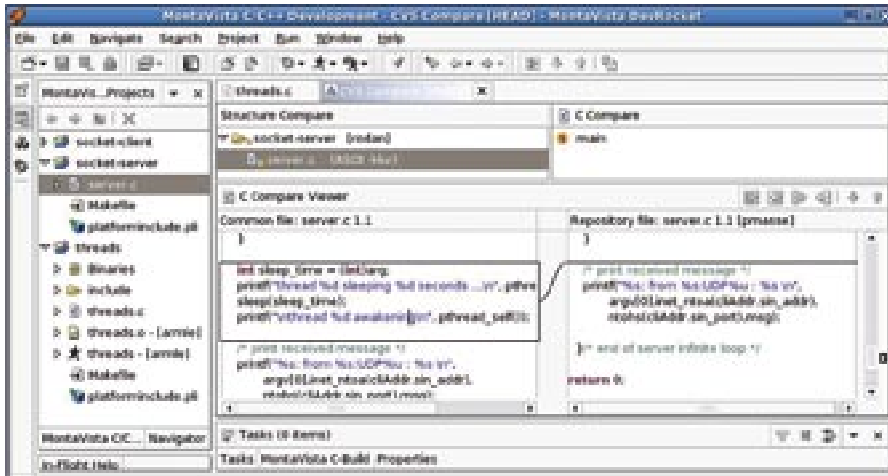
STREAMLINING WITH IDES

Developers engage in various adjunct activities to coding itself. They design the programs they intend to write; craft prototypes to prove their designs; compile or otherwise process code they have written; invoke or run programs to test them, often using controlled execution environments (for example, simulators and debuggers); integrate their programs with code from other developers; and package their programs for distribution and deployment.

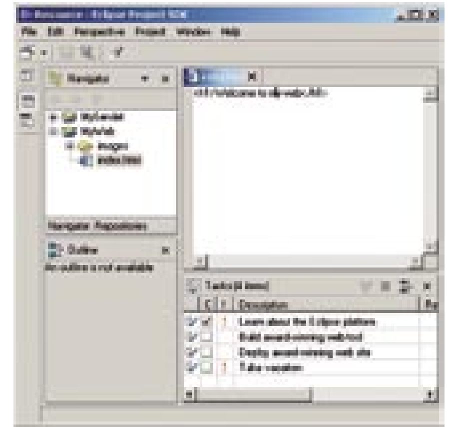
As code size and complexity increases, each of these tasks (and others) becomes more arduous, especially when using only command-line tools.

While numerous tools exist to integrate and accelerate the steps above (for example, build scripts and make files), the dominant paradigm today streamlines development by integrating the edit/compile/debug cycle (and other steps) through a point-and-click graphical user interface – an integrated development environment.

The Eclipse project started at IBM, but now has many contributors



Eclipse has now been extended by a number of vendors



Eclipse started as a general purpose IDE

Graphical IDEs encompass functions that include code browsing and editing, compilation, debugging, and basic project management capabilities. More advanced IDEs provide additional capabilities such as static analysis, performance monitoring, and other tools to enable developers to understand system bottlenecks and use of system resources. Another popular feature built into advanced IDEs is collaborative development support, fostering code sharing among development team members.

Despite the apparent ubiquity of their use, there are still command-line interface (CLI) die-hards who label IDE usage as a 'quiche-eating' activity...

IDE MARKET TRENDS

In terms of IDE technology trends, tools ISVs today fall into three major categories:

- Long-standing suppliers of proprietary development tools who have invested many man-years in building their own integrated environment (for example, GreenHills Multi, MetroWerks CodeWarrior, and of course Microsoft's ubiquitous VisualStudio product line).
- Companies that create customisations or plug-ins to Microsoft Visual Studio, to enable product or application-specific development (for example, LynuxWorks VisualLynx, Mentor Graphics code|lab and Texas Instruments CodeComposer Studio).
- The newest and most dynamic ISV IDE strategy is to build on a shared open source code-base like the Eclipse Project. Evolved from IBM's WebSphere Developer Studio, Eclipse combines code contributed by IBM and dozens of open source community members. The growing list of companies supporting, contributing to and leveraging Eclipse includes Ericsson, HP, IBM, Intel, MontaVista Software, QNX, Samsung, SAP, TimeSys, and many others.

Legacy tools suppliers are increasingly moving from their proprietary IDE bases to the poles of Visual Studio and Eclipse (just as the broader ISV and applications space gravitates towards Microsoft Windows or open source Linux). The best example of such migration is Wind River Systems legacy Tornado environment, which the Alameda-based company is replacing with an Eclipse-based offering.

ADDING VALUE

Legacy proprietary IDE suppliers add value through the unique features and functions they build into their products, and retain users through familiarity, ease-of-use, branding and synergy with companion offerings like OSes (Windows, MacOS, RTOSes), hardware-specificity (TI DSP support), or ties to a larger framework (Mentor Graphics CAE). How then do multiple Eclipse implementers differentiate their offerings around that open source IDE?

First they "do no harm". With market penetration and increasing developer use, programmers will come to expect certain core default behaviours from an Eclipse-based environment (as they do with Visual Studio). Eclipse-based ISVs do not "customise" the IDE; instead, they add value through highly-purposed and application-specific plug-ins supported as natural extensions to the common Eclipse look-and-feel and capability set.

For example, MontaVista Software offers its own Eclipse-base IDE, MontaVista DevRocket. MontaVista DevRocket integrates tools into its Eclipse base that enable rapid, automatic building of custom versions of the MontaVista Linux OS for deployment to embedded target systems. The Rational Software division of IBM offers Eclipse plug-ins that bridge to and integrate its ClearCase products.

Traditional ISVs like Oracle or IBM leverage Eclipse and other IDEs to enable building and deploying applications for use with their respective application server software.

IDE ADOPTION AND USE TRENDS

Integrating multiple development functions into a single environment is not a new idea. Short of full-blown IDEs, many single-function tools allow varying degrees of bridging multiple point-tools: smart, extensible code editors (GNU/EMACs, SlickEdit) support compiler and debugger invocation, and many debuggers support point-and-click transition to editing from their source-code displays. Despite the apparent ubiquity of their use, there are still command-line interface (CLI) die-hards who label IDE usage as a "quiche-eating" activity (along with programming in VisualBasic or "programming with pictures" paradigms).

The IDE-CLI divide correlates strongly with a number of other factors and trends:

- Recent college graduates (last 5-10 years) learned to program with IDEs and prefer their use.
- Enterprise application development today favours IDEs over CLIs.
- Almost 100 per cent of all Microsoft Windows application development occurs in Visual Studio and related IDEs.
- The majority of Java applications development is IDE-based, thanks to tools from IBM, Symantec, Borland, and others.
- Most modern Web programming development leverages IDEs (at least for page composition).
- In the embedded world, systems and firmware developers still prefer command-line interfaces, especially for embedded Linux development. IDEs are, however, preferred for SoC code development in conjunction with verification and co-design paradigms.
- Embedded applications-code developers, like their enterprise counterparts, tend to leverage IDEs.
- Linux and other open source developers still prefer CLI tools, probably from a decade of

The simplest programmer's tools have not changed in half a century: the humble assembler still consumes human-readable, low-level assembly code and emits executable machine code (binary).

CLI-focused history and, until the introduction of Eclipse, because of a fragmented IDE landscape.

- Tools and OS vendors like to market IDEs because they provide an attractive demonstration vehicle and a more palpable, visible asset for licensing than CLIs.
- Managers like IDEs better than line developers, probably because IDEs offer a neat vision of their team's development process; line engineers like CLIs because they afford more control.

IDE CHECKLIST

Not all integrated environments are created equal - integration for its own sake adds little to a positive developer experience. An effective IDE should meet or exceed the quality of the sum of its integrated parts.

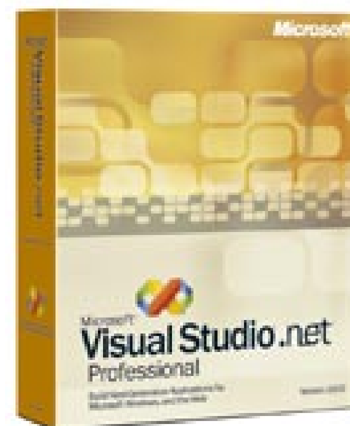
First-time IDE users or developers experienced with a particular IDE and making a transition should consider the following:

- Many IDEs are at least partially open - ensure you can customise key functions to suit your particular needs, such as substituting your favourite editor for the default, invoking a different compiler revision or an entirely different compiler without breaking integration of point-and-click error tracking, and integrating resident revision control tools.
- Some IDEs are host-specific (for example, Visual Studio runs only on Microsoft Windows workstations). If your team uses a mix of workstation types (Windows, Linux, Solaris) or even different versions of the same host (Windows2000 vs WindowsXP, Red Hat vs SuSE Linux, Red Hat 8.0 vs 9.0, etc), make sure the IDE supports appropriate activities on each type of workstation.
- No IDE is completely seamless and no IDE vendor is immortal. Ensure your chosen IDE lets your team members access project components using CLI tools and that project data and code itself can be exported to familiar formats and used with legacy "make" tools.
- Ensure a new IDE offers a clean migration path either from legacy CLI build paradigms or from formats employed by incumbent environments.
- Coming from a CLI environment, you may find some IDEs exert unexpected control over your development project. For example, some IDEs by default hide intermediate file types (for example, relocatable object files), forcing you either to find new ways to work or to go outside the IDE to preserve your prior work-flow.
- Impressive and attractive IDEs may accomplish the goals of a vendor giving a demo, but take pains to understand how they address your projects and practices. Also, IDEs present themselves as a collection of views, perspectives, or states; transition among these views is often non-intuitive. Built in documentation with real-use scenarios is essential

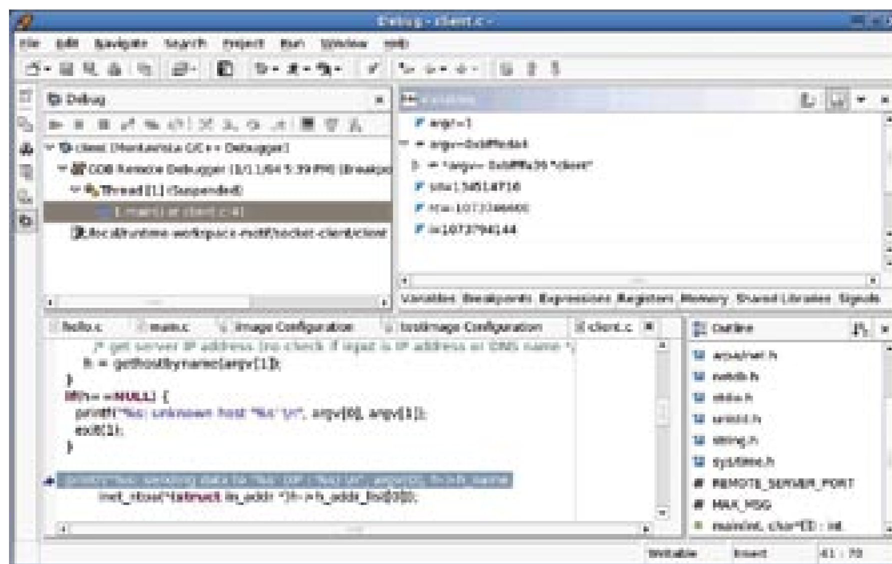
ALL TOGETHER NOW

Advances in microprocessor technology enable new and increased functionality, and so drive software content and complexity ever-upwards. Integration of once-disparate tools is today a prevalent response to taming some of this complexity. Integrated development environments, based on Eclipse and other frameworks, are delivering on the promise of greater efficiency and complexity management through abstraction and visualisation.

But beware the march of IDEs: not all IDEs are created equal, and adoption merits thoughtful evaluation of IDE capabilities and project requirements. More and more developers, however, are finding that IDEs are worth their salt, making it easier to organise coding efforts and source code management for projects of all sizes.



New developers are likely to have grown up with Visual Studio



Eclipse has now been extended by a number of vendors