Retooling Software Engineering

Recommendations of the Internal CASE Initiative

Revision 1.0

January 24, 1994

Acknowledgements

These recommendations are the result of an extended team effort. The following people participated in that effort.

Without the voluntary help and written contributions of these individuals, this document would not have been possible.

Leader and Editor	Chip Nylander
Reuse	Loren Konkus
Windows IDE	Eric Moore John Campbell, Ward Clark, Tom Lowell, Richard Wells
OSF/1 IDE	Mikael Rolfhamre Daryl Black, Solange Karsenty, Glenn Lupton
Portable GUI APIs	Al Simons Leo Treggiari, Jake VanNoy, Steve Zalewski
Base Class Libraries	Shawn Woods Mike Vogl, Thierry Pudet
C++ Language Use	Greg Nelson, Alan Martin
Interface Definition	Steve DiPirro Michael Gangnet, Paul Patrick

Code Checking	Joe Wild
	Don Carignan, Dave Evans, John Guttag, Jim Horning, Peter Van Roy, Andreas Podelski, Walter Van Roggen
Testing	Bill Mckeeman
	Tony Dellaferra, Andrew Payne, Herve Touati
Debugging	Rudy Bazelmans
	Dave Husson
Performance Analysis	Bob Morgan
Configuration Management	John Yates
Business and Legal Advice	Jim Evans

Engineering Requirements

David Baird	Console/Firmware
Jon Campbell	Pathworks
Scott Davidson	NIPG
Neil Davies	VMS
Meg Dumont	USG
Mike Feldman	ISE Western Hemisphere
Gary Feldman	SDT
Dick Gumbel	NOS
Ted Hess	Workgroup
Peter Lieberwirth	NT Engineering
Tim Quinn	Systems Engineering

Dave Snow	DECWest
Bruce Taylor	Production Systems
Hiro Yoshioka	ISE Asia

Many other individuals offered consulting, advice, and suggestions. Any serious omissions from the above list are the responsibility of the Editor.

Table of Contents

Purpose Of This Document

The purpose of this document is to:

- Recommend (within the constraints of the goals, scope, and evaluation criteria of this effort) a set of software development technologies and tools to be deployed widely across Digital Software Engineering.
- Document the rationale for these recommendations
- Document the actions required to deploy the recommended tools and technologies.
- To a first order of approximation, document the costs to deploy the recommended tools and technologies.

Is is explicitly *not* a purpose of this document to:

- Address all perceived Software Engineering problems.
- Address software development process improvements.
- Provide a complete Project Plan for deploying the recommended tools and technologies.
- Establish goals for implementing these recommendations in individual projects and organizations.

Executive Summary

Charter

Recommend how to retool Digital Software Engineering to provide a contemporary software development environment (tools and technology) to:

- Ensure that Digital software engineering technology reflects (and is competitive with) industry practice.
- Provide for manageability and reusability of Digital's source code base..
- Address needs of Digital's software engineers.
- Contribute to engineering cycle time reduction goals.

Requirements

- Windows/NT, as well as OSF and Windows/DOS, host development desktops.
- OSF and NT development servers.
- VMS servers for VMS software development.
- OSF, VMS, and NT target platforms.
- Cross development.
- Distributed development (local and wide area) by groups sharing code and process.
- Excellent support for use of C and C++.
- Support reuse

- Support use of O-O development technologies.
- Scalability of the software development environment across a range of projects (in terms of people, code, and complexity).
- Other desirable attributes (which can be traded off in favor of other goals) include:
  - Support for DOS and Windows/DOS targets.
  - Support for Macintosh targets.
  - Common technology across heterogeneous development desktops and servers.

Recommendations

- Establish an ongoing central function within Engineering that will continue to identify, evaluate, acquire, and support use of contemporary software development technology across Digital Software Engineering.
- World Wide Web technology and client tools to create a Digital Software Engineering infrastructure for discovery and reuse.
- Establish a Reuse Library Support Center.
- Microsoft Visual C++
- Premia Codewright
- FUSE / OSF
- Microsoft Windows API and Microsoft Foundation Class Library (MFC), using Wind/U from Bristol Technologies on VMS and OSF.
- Unified" COM / OLE / CORBA distributed object model coupled with 3rd party base class library.
- Fix SDL for current users, and to help dependent groups migrate to C and C++.
- DEC C -check option as standard filter whenever DEC C compiler is used for compilation.
- Gimpel Software's Flexelint to check C and C++ sources during development.
- ClearCase distributed configuration management and system building technology from Atria Software.
- Establish a Software Testing Expertise Center.
- tcl as the single portable test scripting language for all platforms.
- tclREX for random testing.
- Jig integrated with tcl for component testing.
- Variety of Debug technology is necessary.
- Performance Analysis:
  - PCA on VMS.

- PCA PC-sampling collector (port to OSF and NT).
- ATOM/OM for cycle-counter collector (port to NT).
- Portable analyzer in C/C++ with tcl-based user interface.
- tcl/tk to develop portable performance presentation programs.

Introduction

At the 1993 Engineering Senior Technical Leaders Forum, one of the key Engineering issues identified was that Digital Software Engineering did not consistently utilize (and did not have available) contemporary software development tools and technology.

In order to succeed and survive as a software technology provider, software engineering in Digital needs improvements in:

1. Software development productivity
2. Software time-to-market
3. Software defect prevention
4. Software defect time-to-repair

Digital also needs to provide a software engineering environment that provides it's software developers with contemporary tools and skills,and that will attract outside technical talent to Digital.

This is partially a management issue, resulting from planning,process, and operational issues that cannot be effectively addressed with software development technology.

However, it is partially a software development technology issue. Much software development within Digital Engineering is being done using the same tools and methodologies that were used ten years ago.

In order to improve Digital's software engineering performance and standards, we must provide the infrastructure and tools to advance Digital's software engineering practice.

The result of these observations was to charter an effort to understand the opportunities to improve the technology available to Digital Software Engineering, and make recommendations reflecting the following purposes:

1. Identify candidate technologies, evaluate these technologies, and recommend contemporary software development tools and technology for Digital Software Engineering.

2. Gain Engineering commitment to invest in and implement these improvements to the software development environment.

3. Enable identification of the appropriate pilot projects or groups who will be early adopters in FY94.

4. Provide the data necessary for budget planning and for deployment planning, including the data necessary to plan

    - Capital Equipment
    - 3rd Party Software
    - Training
    - Technology Development
    - Support

5. Provide the data necessary to plan the introduction of and transition to the recommended tools and technologies by groups and projects.

Implementation of these recommendations should begin with pilot projects FY94, and serious transition to the new environment in FY95.

This effort *must* be the first phase of ongoing improvement of Digital's software development environment: additional work will be needed to address software development activities beyond the scope of this effort, and to respond to discoveries made after this effort is completed.

Goals

- Provide a software development environment that meets the needs of Engineering today, and which looks to the future.

- Reflect Industry Practice
    - Utilize platforms and technology for software development that are representative of the environment to which customers, ISVs, and the industry are moving.
    - Move Digital Software Engineering into the developmental mainstream.
    - Be positioned to ride and exploit the development technology innovation curve.

- For Digital's Software Code Base
    - Help protect and maintain the security integrity of the source code pool.
    - Improve code quality, reliability, maintainability, and supportability.
    - Increase software reuse.
    - Improve the management and sharing of source code.

- For Digital's Software Engineers
    - Provide a contemporary technical environment that supports keeping skills current, to which the technical literature being published is relevant, and from which Digital's software engineers get professional satisfaction.
    - Attract talented engineers from outside Digital.
    - Retain talented engineers.
    - Make sure that Digital Software Engineers have the tools they need to get their job done.

- Contribute to Engineering Cycle Time Reduction. Achieve modest Phase 2 improvement, and additional cycle time reduction by overall productivity across the software lifecycle:
    - Improve software development productivity, time-to-market, defect rate, and time-to-repair.
    - Attract and retain superior people.
    - Reuse
    - Less maintenance effort; for example:
        - Improved quality (improved code, reuse, testing, performance)
        - Improved debugging
        - Improved configuration management (reconstruct sources for complex software configuration to diagnose system engineering problems)
        - Etc.

Requirements

The overall requirements guiding these recommendations include:

- Windows/NT, as well as OSF and Windows/DOS, host development desktops.
- OSF and NT development servers.

- VMS servers for VMS software development.
- OSF, VMS, and NT target platforms.
- Cross development.
- Distributed development (local and wide area) by groups sharing code and process.
- Excellent support for use of C and C++.
- Support reuse
- Support use of O-O development technologies.
- Scalability of the software development environment across a range of projects (in terms of people, code, and complexity).
- Other desirable attributes (which can be traded off in favor of other goals) include:
  - Support for DOS and Windows/DOS targets.
  - Support for Macintosh targets.
  - Common technology across heterogeneous development desktops and servers.

These requirements, combined with a certain amount pragmatism, resulted in the following Goal Matrix for supporting various target platforms from various developer desktops:

Developer Desktop	--------------------------- **Target for Development** ---------------------------				
	NT/AXP	**NT/Intel**	**Win/DOS**	**OSF/AXP**	**VMS/AXP**
NT/AXP	Yes	Off the Shelf	Off the Shelf	Yes	Yes
NT/Intel	Yes	Off the Shelf	Off the Shelf	Yes	Yes
Win/DOS	No	Off the Shelf	Off the Shelf	No	No
OSF/AXP	No	No	No	Yes	No
VMS/AXP	No	No	No	No	No

Developer Desktop =	The desktop platform with which the engineer directly interacts.
Target for Development =	Platform for which software is being developed. The target system must host the technology required for system build, testing, debugging, and performance collection, independent of the desktop used by the engineer.

Yes = A Goal of these Recommendations

No = Not a Goal of these Recommendations

Off the Shelf = Goal, but recommendations assume only the use of commecial technology available off the shelf for the development software required on the target platform (debuggers, compilers, etc.).
It is a non-goal for Digital to develop or port target-required technology for these targets.

Scope

The scope of this effort will be the desktop and server tools and technology supporting the following software development activities:

- Code Preparation
- Code management
- Code sharing and reuse
- Component and System build
- Component and System test
- Component and System debugging

This sort of activity *must* be ongoing in support of Software Engineering at Digital, and future work should address other opportunities in a wider scope (e.g. design, problem tracking).

The technologies evaluated and recommended in support of the above development activities are:

- Software Reuse Technology
- Desktop Interactive Development Environment (IDE)
- Platform Independent GUI Development
- Base Class Library(s)
- C++ Language Use / Subsetting
- Data and Interface Definition
- Source Code Checking / Filtering
- Source Code Management, Configuration Management, System Build
- Testing Technology
- Debugging Technology
- Performance Analysis

Non-Goals

- Address all perceived Software Engineering problems

  There are many issues in Software Engineering, having to do with requirements, strategy, decision-making, communication, education, etc.

  These recommendations are purposefully and exclusively focussed on the tools and technology used by Digital software engineers in the code/build/test/debug cycle.

- Address software development process improvements

  It is not a goal of these recommendations to define new software development processes or improvements, or to address application of development process methods including Contextual Inquiry, QFD, VCA, SGIA, Six-Sigma, SEI Assessment, Demming Methods, Concept Engineering, TQM, Formal Inspections, Meetings-that-Work, Change Management, Voice of the Customer, etc.

  There are a number of parallel efforts addressing software development process within Software Engineering. This effort is complementary.

  We expect that these recommendations will be consistent with the implementation of a well-defined and repeatable software development process.

  In the absence of a plan to widely implement software development process changes or improvements, these recommendations can address their stated goals without direct reference to such process changes or improvements.

- Provide a complete Project Plan for deploying the recommended tools and technologies.

  Rather, the purpose is to provide the information necessary to write such a Project Plan.

- Establish goals for implementing these recommendations in individual projects and organizations.

  Rather, the purpose is to provide the information necessary for individual projects and organizations to plan and commit implementation of these recommendations, as appropriate to each project and organization.

  Each project and organization has specific attributes and constraints that will make implementation of these recommendations a highly individual exercise.

- Provide a Change Management Plan for introducing new tools and technologies.

  Such a plan will be a function of both the general Project Plan for deploying these recommendations, and the specific plan to implement them in each organization.

- Recommendations for any of the following development activities and technologies:
  - Requirements Analysis
  - Design
  - Rapid Prototyping
  - Project Management
  - Inter-product dependency management
  - Product Documentation
  - Installation Procedures
  - Software submission to manufacturing
  - Software manufacturing
  - Problem reporting and tracking
  - Metrics and measurement of productivity and reliability (such as tools to support application of QSM).

  These activities and technologies may be addressed by parallel efforts, or by future phases of this effort.

- Recommendations for Compilers

  Choice of compiler can be a critical decision for a software project or organization.

  However, the technical and sourcing issues around compilers generally require that a number of compilers be supported for internal use, and in addition there are parallel efforts within engineering to recommend compiler strategy for internal development.

  These recommendations have not assumed or recommended any particular compiler technology.

- Using what we build

  To the extent that Digital builds or has access to technology which evaluates as equal or superior to competitive technologies, choosing the Digital-built or Digital-available technology is the obvious choice.

However, biasing technology choices towards Digital-built or Digital-available technology is an explicit non-goal of these recommendations (versus recommending the best technology for the job).

- Provide a strategy for software development using VMS desktops.
- Immediate application to System Integration or Digital Information Systems.

Assumptions

Key assumptions that guided these recommendations included:

- In the future, most software engineering technology will execute on the software engineer's desktop, supported by shared servers.
- Non-desktop servers can be used for source pool management and sharing, builds, regression testing, and other batch procedures.
- C++ is the base language of choice for Digital software engineering.
- Object Oriented technology (in some form) can help increase productivity and reuse.
- 3rd party environments and tools are where much of the action is (especially on Windows desktops), and Digital currently has a severely limited ability to invest.
- The productivity of Digital's software development and maintenance can actually be significantly increased by application of technology to the coding / build / test / debug cycle described above.
- The capital investment is possible to provide the necessary desktop environment and server-based infrastructure to Digital's software engineers.
- This effort is the first phase of ongoing improvement of Digital's software development environment: additional work will be needed to address software development activities beyond the scope of this effort, and to respond to discoveries made after this effort.

Relationship to Development Processes

It is absolutely clear that, in principle, the most appropriate use of software development tools and technology is in support of a well-defined, documented, measurable, and repeatable software development processes.

Ideally, these recommendations would be made within the framework of such a defined process, and would support that process.

Although there are many efforts ongoing within software engineering to define and improve software development process, implementation plans for such process definition or improvements are not available today.

These recommendations are complementary to the various development process efforts underway, and we expect that these recommendations will either work as is, or can be easily adapted to, any development process definition or improvement implemented throughout Software Engineering.

Implementation

These recommendations should be implemented by the following program:

1. Establish a project, with qualified manager and technical staff.

   These recommendations need to be carried forward and deployed as a project, with a manager, technical staff, and plan. Subsequent to piloting and full deployment, this group must support internal tools and technology for Digital Engineering, acquire and distribute new tools and versions, and keep Digital Software Engineering on the development technology curve.

2. Initiate Pilots

   The next step in the process should be a set of Pilots, begun immediately and evaluated by the end of FY94. These projects should be identified as soon as possible (volunteers will be solicited), the appropriate technologies made available to them, the necessary support made available to them, and the presence established to collect and understand the lessons learned.

   The function of these pilots is to:

   - Test these recommendations in the context of real projects.
   - Provide the data necessary to modify these recommendations based on real experience.
   - Provide the data necessary for successful wide deployment.

3. Plan wide deployment

   Engineering must decide how and at what rate to introduce the recommended improvements to Engineering, decide the rate at which new desktop hardware and software will be capitalized, and decide the rate at which the recommendations are implemented in specific groups.

   An implication of these recommendations is that many VMS desktops will be replaced by NT (or OSF) desktops; this capitalization must be planned.

   There should not be a fixed timetable set for all of Engineering. These recommendations should roll out across Engineering over time, beginning in early FY95.

These recommendations should be planned for introduction into individual groups and projects at a rate and in a manner appropriate to each group (considering the current technology used by the group, the technology developed by the group, the group's current and future commitments and schedules, etc.)

4. Manage wide deployment

   Wide deployment must be managed, and will require both management and technical support.

5. Improve and update on an ongoing basis

   This effort *must* be the first phase of ongoing improvement of Digital's software development environment: additional work will be needed to address software development activities beyond the scope of this effort, and to respond to discoveries made after this effort is completed.

   In addition, these recommendations include the creation of a small number of "expertise centers" to support use of testing technology, reuse technology, configuration management technology, etc. across Software Engineering. These expertise centers should be managed by the same person that manages the general Internal CASE project.

Summary of Costs

The following table summarizes the costs associated with implementing these recommendations. These costs are more fully explained and associated with specific tasks and actions in the detailed recommendation sections.

Some caveats apply to these costs:

- These are *only* estimates, and do not have the accuracy of a project plan.
- This table is intended to provide a "ballpark" view of costs across Software Engineering. In all cases, the detailed sections are the final authority, and should be consulted by the seriously interested reader.
- The costs in this table do not include capital hardware.
- Some costs are still to-be-determined, as noted below and in the detailed sections.
- The indicated software costs are probably *worst case,* and represent the "typical price"; using Digital's leverage for site or volume license agreements is likely to lower these software costs significantly.

		Labor Costs Across All Software Engineering						Other Cost
		Must Be Done			Should Be Done			
	Software Costs per developer	Before Pilots (person months)	Before Wide Utilization (person months)	Ongoing Infra-structure (persons)	Before Pilots (person months)	Before Wide Utilization (person months)	Ongoing Infra-structure (persons)	
General		1*		3 person				
Reuse			2	3 person		6		$120K/year
Windows IDE	$200-$350		4	0.5 person				
OSF/1 IDE			24	1 person	6	6		
GUI APIs	see details	18						
Base Class Library	$100-$500	3	3		24	6 1 task tbd	1 person	
Interface Definition			12					
Code Checking	$200-$400	1		0.5 person				
Testing	variable		18	4 person				$100K once
Debugging	see details	3	39 6 tasks tbd	see detail				
Performance		9	14					
Config. Mgt.	$1000? (see details)		1 1 task tbd			12	2 person	

* A manager / driver for implementing these recommendations must be funded and named. This 1 month of labor represents the startup time for this person.

Hardware Considerations

Analysis of the recommended development environment for Software Engineering did not reveal any startling new requirements.

With a few exceptions for certain special cases (noted in the detail sections), the recommended software development tools and technology can be supported by a properly configured software development workstation.

In 1994, such a workstation consists of:

- A fast processor (50 Mhz Intel or 166 Mhz Alpha, or better)
- 32 - 64 MBytes main memory (64 recommended for OSF)
- 800 or more MBytes local disk (e.g. two (2) RZ25 disks)
- Large color monitor (VRC16 minimum, 19" better)
- 3.5" floppy
- CD-ROM
- Ethernet board

One exception to this simple story is the network bandwidth between workstations and servers, and between servers.

If Digital Software Engineering achieves a true client/server environment, including distributed configuration management and parallel builds, the network bandwidth must be available to support this.

For some sites and groups, these recommendations may stress the network more than has previously been experienced.

In addition, effective multisite development (particularly development that includes overseas groups) requires high bandwidth long haul network connections.

Reuse Technology Summary

Problem Statement

According to the Booz-Allen study, Digital's software products are twice as expensive to build and maintain and are of lower quality than competitors' products. This gives companies like Hewlett-Packard a large competitive advantage. One of the reasons for their success is software reuse.

Software Reuse requires culture and process changes, which are currently underway at Digital in the form of several different reuse and process improvement initiatives. The initiative that is most closely allied with this Internal CASE technology recommendation is the Engineering Excellence Reuse Change program. An aspect that each of these initiatives shares is the need for a technology for discovering reusable resources.

Digital's current resource discovery process is chaotic. Engineers searching for an existing component, document, or even someone that has knowledge that they need must do all of the foot work:

- searching notes conferences
- asking other engineers they know or meet in the hallway
- looking through public directories for file names they recognize
- reading through a usenet newsgroup
- searching for a name archie recognizes
- searching through FTP file archives on the Internet

This process takes a long time, requires expertise with many different tools, and the results are uncertain. This leads to the perception that it's often faster to recode than reuse.

Goals

The technology recommended will:

- Increase the amount of source code sharing and reuse in all forms across Software Engineering,
- Increase the exposure to and communications with other software engineers outside Digital,
- Be readily accepted by Digital engineers and incorporated into the culture (rather than change the culture), and
- Meet all of the critical requirements of the Engineering Groups.

Summary of Recommendations

- Deploy the World Wide Web tools to create a Digital Software Engineering infrastructure for reuse discovery
- Providing style guides, concept dictionaries, standard abstract templates, and other tools to ensure consistency across information providers
- Establish a Reuse Library Support Center for ongoing enhancement and consulting.

Summary of Rationale

Finding software components is just one part of software engineering. If we ignore the information services available to us through the network we miss a practical, available, and growing infrastructure to enable Digital software engineers to tap into the attitudes and concerns of the customer base, to interact with their peers in other companies and countries, and to distribute software quickly and efficiently to customers. To use the network and tap into these information sources, Digital must use the tools that the Internet community uses. With a small amount of incremental work, the same tools we use for finding publications, conference announcements, phone numbers, weather maps, software patches, and news articles make a very capable Reusable Resource Discovery System

Windows Development Environment Summary

Problem Statement

Digital needs to provide its software engineers with a contemporary software engineering environment on the Windows desktop. This environment should provide Digital's software developers with contemporary tools and skills and also help attract technical talent to Digital.

Individual developer productivity is heavily dependent upon the tools available. However, good tools do not necessarily work together, and do not present a consistent, easy to use interface to the developer.

In addition to lower productivity, this may also incur opportunity costs by raising barriers to process changes or to teamwork. These opportunity costs are frequently not as visible, but more significant.

Goals

- Provide Digital's software engineers with a state-of-the-industry development desktop, encouraging use of contemporary tools and skills.
- Increase productivity by encouraging the use of IDEs.
- Increase acceptance by being open and avoiding 'religious' issues.
- Provide flexibility for the different technical and business needs of product groups. Address the needs of mainstream developers and projects, while minimizing the impact on individuals who must use one or more different tools (different compiler etc.)
- Allow use of multiple compilers from multiple compiler vendors.
- Integrate the tools recommended by the Internal CASE effort.
- Scale from small to large projects
- Support use of both Intel and AXP for development desktops.
- Although the IDE does not itself support cross-development, allow access to and integration of cross-development tools.

Summary of Recommendations

A Windows-based integrated development environment (IDE) is the current industry approach to address these problems for the Intel and AXP based Windows desktop. (The 'Windows desktop' is defined as MS-Windows v3.1 or later, and Windows/NT.)

- Two-tiered approach to Windows IDE's: allow choice of either Microsoft Visual C/C++ (including the Microsoft compiler), or the Codewright editor as the IDE (with choice of compiler not determined by Codewright).
- Extend and customize VC++ (via extending the Tools menu) and Codewright (via the support extension / customization interfaces) to integrate the tools and technologies otherwise recommended by the Internal CASE effort.
- Customizations of the Codewright IDE should use a generic interface whenever possible, so that a user can swap compilers, s/w configuration managers etc. with as little impact as possible.
- These extensions and customizations should be the responsibility of a central Internal CASE Project, which should continue to develop, maintain, and distribute the extensions to the standard IDEs.

Summary of Rationale

Microsoft Visual C/C++ has the following advantages:

- It is an industry standard visual GUI shell
- Populated with well-integrated tools
- Directly supporting other strategic initiatives, such as COM / CORBA (assumption), and the MFC GUI interfaces (recommended by Internal CASE as the standard GUI API).
- Supporting PC cross-development, including Macintosh.
- Available on AXP (in progress) and Intel, on NT Windows and DOS Windows.

However, Microsoft VC++ has the following disadvantages and limitations that limit its applicability to certain projects and environment:

- Extensibility is very limited; other tools can be invoked as applications by extending the Tools menu, but there is no real integration between the environment and the invoked tool. DLL's cannot be invoked.
- There is no choice of compilation tools (compiler, debugger, etc.); the Microsoft tools must be used.

- VC++ was designed as a tool for individuals and small projects; it does not currently scale well to large projects and code bases.

Codewright, an editor-based interactive development environment, has the following advantages versus Microsoft VC++:

- Codewright is open and extensible: it ships with source code and has a well-designed architecture intended for user extension and customization. This allows good integration of external tools and invocation of DLL's, provides a way to bypass scaling problems (by pushing them down into the integrated tool), and allows well integrated access to cross development tools.
- Codewright is available on both Windows 3.1 and Windows NT.
- Codewright does not dictate choice of language, compiler, debugger, etc.
- Codewright editor emulation includes IBM CUA editor, Brief, UNIX vi, and Emacs. Codewright is a good environment for the "power developer" and developers who frequently utilize non-Windows desktops.

However, there is not yet any plan for a native AXP version of Codewright.

OSF/1 Development Environment Summary

Problem Statement

Digital needs to provide its software engineers with a contemporary software engineering environment on the OSF/1 desktop. This environment should provide Digital's software developers with contemporary tools and skills and also help attract technical talent to Digital.

Individual developer productivity is heavily dependent upon the tools available. However, good tools do not necessarily work together, and do not present a consistent, easy to use interface to the developer.

In addition to lower productivity, this may also incur opportunity costs by raising barriers to process changes or to teamwork. These opportunity costs are frequently not as visible, but more significant.

Goals

The goals for an IDE on OSF/1 is to provide a graphical software development environment that:

- Provides Digital's software engineers with a state-of-the-industry development desktop, encouraging use of contemporary tools and skills.
- Increases productivity by encouraging the use of an IDE.
- Meets the needs for development for an OSF/1 target.
- Increases acceptance by being open and avoiding 'religious' issues.
- Provides flexibility for the different technical and business needs of product groups.
- Allows for easy integration of tools, especially the tools recommended by the Internal CASE effort.
- Scales from small to large projects
- Allows access to and integration of cross-development tools.

Summary of Recommendations

- DEC FUSE as the Integrated Development Environment (IDE) for OSF/1 development.
- Modest enhancement to FUSE in order to meet engineering's needs.

Summary of Rationale

- FUSE meets or exceeds most of the Internal CASE IDE requirements for an OSF/1 desktop and target.
- FUSE is populated with a rich set of highly graphical and tightly integrated tools.
- The FUSE tools are layered on top of native UNIX tools, allowing engineers to "break out" and work in the native UNIX environment as needed.
- The FUSE architecture is open and allows for easy integration of additional tools.
- The architecture is well designed for distribution and FUSE can with some investment be extended to VMS targets.

- Alternatives to FUSE (e.g. Softbench, Centerline, etc.) have essentially the same limitations that FUSE has, and do not offer significant additional advantages in the Digital Engineering environment. These alternatives also do not exist on or support OSF/1 today.

Portable GUI Programming Interface Summary

Problem Statement

Graphical User Interface (GUI) programming on multiple platforms currently requires recoding per windowing system.

- Developers of Digital applications with GUI components are currently spending significant resources developing and maintaining multiple parallel interfaces for Windows, Motif and Macintosh.
- Some groups have investigated various solutions/work-arounds for this, resulting in a hodge-podge of techniques and wasted analysis effort.
- Currently, there are no known solutions with coverage of Digital's strategic platforms, resulting in Alpha AXP being less attractive to ISVs.

Goals

These recommendations are intended to:

- Provide a single GUI coding technique that most groups can use, reducing wasted effort.
- Leverage third party efforts in GUI design and building tools.

- Support Digital engineers keeping their skills current.
- Attract talented engineers to Digital.

Summary of Recommendations

Strategic

Use the Microsoft Windows API everywhere as the windowing program interface. Applications that do not require per-platform GUI development[1] should design and code to the Windows API.

Further, for new application development, use the Microsoft Foundation Class Library (MFC) for GUI access.[2]

Tactical

- Partner with or otherwise influence Bristol Technologies to make Wind/U and related products available on OSF/1 AXP and OpenVMS on VMS and AXP.
- Create a better licensing agreement with Bristol, based on volume. (Leverage Windows effort in OpenVMS group.)
- Use the soon-to-be-available Windows on Macintosh product from Microsoft.

Summary of Rationale

By any measure, Microsoft's Windows API is the overwhelmingly predominant windowing system.

- There are an order of magnitude more desktops installed with Windows than any other windowing system.
- Most new GUI development tools will be written for Windows.
- Most of the world's GUI programmers program to the Windows API.

[1]Some groups interviewed for this effort stated that they are under such great competitive pressure that they must code their GUI on a per-platform basis, and that they would not use any common tool, regardless of type. The competitive pressures that were cited, without metrics, were look and feel, and performance.

[2]***Issue:*** Borland's Object Windows Library is another possible alternative for a GUI base class library. There are corporate level discussions about joint development work between Digital and Borland which may affect this. However, to be a viable candidate, OWL would have to be on Alpha NT, Mac and UNIXes. This will require waiting for Borland's "OWL for AppWare" product, which is probably at least a year away.

By writing to the Windows GUI, Digital's software will have the best possible performance and fit on the primary desktop platform—Windows.

Wind/U is the best technology for this purpose on non-Windows platforms.

Prioritization

Taken by itself, this recommendation only rates a medium priority. Digital internal developers simply don't do enough GUI work for this to be a major area of pain for Digital Software Engineering taken as a whole.

However, the recommendations made here are highly synergistic with efforts planned in OpenVMS development. With a little oversight work to ensure that the negotiations reflect the needs of Digital at large, the goals described here can be accomplished.

Base Class Libraries Summary

Problem Statement

The majority of the programming infrastructure in object oriented environments is provided by objects from a base class library. Both design and coding style in C++ is heavily influenced by the base class library chosen. Since Digital software engineering is expected to heavily migrate to C++, Digital Engineering must decide whether to standardize on one base class library and, if so, what it should provide and how it should be sourced.

Furthermore, C++ has an early, compile-time binding model that inhibits binary code binding. C++ client code must bind at compile time to the detailed implementation of any object it uses; the object implementation virtually cannot be changed without recompiling the client code. This makes it impractical to deploy C++ applications or class libraries as executables. Instead they have to be shipped as source that can be recompiled when supporting classes are upgraded. This is a serious impediment to large-scale code reuse and practical deployment of class library based software.

Goals

C++ base class library technology should:

- Solve the C++ limitation of early binding and client knowledge of a service's implementation that retard practical software reuse, sharing, and deployment.
- Provide a leadership standard C++ infrastructure inherently supporting internationalization, concurrency, distribution, and persistence.
- Simplify learning C++ by providing focus on what features to use, encapsulation of difficult or complex features, and commonly needed extensions.
- Move C++ into an open OO environment to share objects across other languages and legacy code, broadening the software reuse base and facilitating software engineering's move into C++.
- Lead to consistency in C++ design and coding style and to economy of scale in securing a good C++ infrastructure cheaply.

Summary of Recommendations

- Adopt the CORBA / OLE2 Component Object Model (COM) to provide the binary standard and late binding necessary for binary reuse and deployment of C++ code.
- Procure a single standard base class library through the following steps:
- Negotiate with an existing base class library vendor to get a modifiable source code base for the general C++ base class library.
- Modify that source code base to build inherent COM support, concurrency support, and internationalization into the library.
- Port the base class library to all the target platforms.
- Add distribution and persistence extensions on top of the library later.
- Use the existing Pegasus project (SDT) to negotiate with the class library vendor and port/extend the class library.
- Use the future COM products (NOS) to provide object distribution.
- Extend the COM model by adding a full binary inheritance mechanism and porting an object scripting tool like Visual Basic to Digital's target systems.

Summary of Rationale

- Standardizing on an existing C++ base class library will NOT appreciably help engineering achieve software reuse nor client/server programming because no existing library solves the C++ compile-time binding problem.
- No existing base class library has inherent concurrency support. The best in class libraries are only threadsafe; they don't provide the needed synchronization mechanisms and their objects are not shareable.
- Microsoft's Component Object Model (COM), built into a C++ base class library, will greatly impact C++ reuse and distributivity.
- A COM-based base class library will benefit from the expected host of tools the industry will produce to support and add value to COM.
- Binary binding and concurrency have to be built into the base class library, not added on top of it.
- There are a variety of existing base class libraries that are acceptable starting points for a COM-ized class library. The difficulty will be in negotiating a suitable agreement with a vendor to modify their code.
- If Digital cannot negotiate for a starting code base, we can produce a product quality COM-based base class library internally in a short time (6 months field test, 9 months V1).

C++ Language Subsetting Summary

Problem Statement

C++ will be the language of choice for an increasing number of Digital's projects. The language is powerful, but is easily abused. The problem is to devise a strategy that maximizes the benefits we get from C++, while minimizing the costs.

Goals

The wide-spread successful deployment of C++ at Digital could increase programmer productivity and code reuse; improve communications between different programming groups; and help attract and retain strong programmers to the company.

But projects that have leapt recklessly onto the bandwagon have been bruised. This is clear from the recommendations that Bjarne Stroustrop gives to projects starting to use C++, which can be summarized: Go Slowly.

The internal case effort should provide a "road map" that can be used by projects that are adopting C++, to help them harvest the full potential of the language and avoid the pitfalls.

Summary of Recommendations

Digital's internal case effort should provide

- a well-defined C++ subset to be used for new C++ code written in Digital.
- a conformance checker to test whether code conforms to the subset.
- education in basic object-oriented programming using the subset.

Summary of Rationale

The benefits of C++ are primarily its growing popularity and its support for object-oriented programming. This makes it the language of choice from a long-term point of view.

The costs are primarily its complexity. A team employing the language for the first time must expect a substantial initial decline in productivity as programmers learn to use the language effectively.

The benefit-to-cost rratio of C++ can be increased by subsetting the language. Much of the learning experience consists of falling into traps. Most experienced C++ programmers confine themselves to some subset; many projects and teams using C++ provide written guidelines for using the language; there are a number of published books of such guidelines.

A subset is more valuable if it can be enforced by an automatic conformance checking program. However, this is technically much more difficult than merely formulating vague guidelines. Every effort should be made to define a subset that can be enforced by a checking program. Such a checker must be able to deal with programs that include code from outside the subset. The definition of such a subset and checker should be a part of the next phase of the internal case initiative.

One of the main benefits of C++ is its support for object-oriented programming. Programmer education is required to realize this benefit. An internal education program appears to be the best way to achieve this in a focussed and cost-effective way. Setting up such a program should also be a part of the next phase of the internal case initiative.

Status of Detailed Recommendations

We are making ***no*** detailed recommendations at this time.

This must be an action item at some appropriate near-future time.

At the time of this writing, the Digital C and C++ compiler strategies for internal use were undergoing fundamental examination and revision by an effort outside the Internal CASE effort.

In addition, the COM (CORBA + OLE) initiative is just beginning to be understood within Engineering. The use of COM will influence the C++ object model utilized, which will influence the language features recommended and how those features are used.

All this in turn affects the recommended subset definition, the conformance checking tool, and the choice or development of an education program.

The effort to provide language guidelines to Digital Software Engineering must continue, but detail could not be provided in time for this set of recommendations.

Data and Interface Definition Tools Summary

Problem Statement

"Interface Definition" can have several possible interpretations. For the purposes of this report, it is defined to be a mechanism by which multiple languages share data structures and procedure definitions, either for communication or in an environment where it is necessary for multiple languages to share the same definitions.

In environments where this is required, it is highly desirable to maintain a single definition. Maintaining multiple language copies of the same definitions is error prone and labor intensive.

The problem is how to write these definitions once and yet have access to them from all languages which need them. The solution must meet the demands of the development environment for performance, stability, and support.

Goals

Digital Software Engineering appears to be moving towards a greater reliance on C and C++, rather than other (multiple) languages.

In addition, the choice of interface definition language for data structures and program interfaces in a distributed heterogeneous is not yet clear, and because of industry activity, cannot be dictated by these recommendations.

In light of these factors, the goal is to meet the requirements for multilanguage interface definition language(s) and tools for those who currently depend on these capabilities, in the most cost-effective manner possible, while providing a growth path towards the future.

Summary of Recommendations

The only firm requirements in this area come from VMS engineering. However, others have expressed interest in this technology or potentially will require it in the future.

The thrust of these recommendations is to solve the immediate problems of VMS, and others currently developing or maintaining VMS-based products in multiple languages, and to provide an evolutionary path towards the future.

- Solidify SDL as an internal development tool.
- Enhance SDL to meet the short-term needs of VMS engineering and others working on multilanguage VMS-based software.
- Implement a C++/C -> SDL conversion utility to provide an evolutionary path away from SDL dependencies. (This will allow new definitions to be written in C++/C, while allowing these definitions to be accessed from legacy code).
- Enhance SDL to better fit into the CASE tool development environment.
- C/C++ development with no multi-language dependencies will define data structures and functional interfaces using language-specific means and, in some cases, using CORBA or DCE IDL.

- Industry activity towards a more abstract interface definition language or tool, perhaps including support of distributed systems and/or multiple languages, needs to be tracked for possible adoption by Digital Engineering when/if appropriate.

Summary of Rationale

Although SDL is hardly strategic for Digital Software Engineering, there are still important dependencies on it. These recommendations are intended to address the immediate needs as well as future requirements in this area.

Prioritization

Taken by itself, this recommendation is rather low priority. Digital Engineering appears to be moving away from dependency on multilanguage software development, and the problems addressed by these recommendations are confined to VMS Engineering and a small number of projects doing similar work.

However, the cost of implementing these recommendations is quite modest, and doing so will have high payoff for VMS Engineering and other similar projects, and will establish a needed path to the future.

Code Checking Tools Summary

Problem Statement

It is easy for developers to make mistakes that are expensive to correct later in the development cycle. This problem is likely to increase as Digital Software

Engineering moves from C to C++, because C++ is a more complicated language, with far greater complexity and "language traps". Many of these mistakes can be caught by lint-like code checkers during development.

Proper use of code checkers should help find software problems earlier and easier than current methods.

Goals

- Use code checkers to find problems in Digital software earlier and easier that current methods (reviews, testing, debugging, ...).
- Get developers to begin using code checkers.

Summary of Recommendations

- Projects using DEC C for development should use the compiler's built-in -check option. This option reports more than 50 questionable practices that could lead to problems.
- All projects should use Gimpel's PC-lint to check their C and C++ sources during development (both individually and during project builds). This is especially important for projects moving to C++.

Summary of Rationale

The DEC C compiler's -check option flags a large number of potential problems for little extra effort if the compiler must be used in the software build process.

Gimpel's PC-lint flags the largest number of potential problems of any of the tools evaluated. It provides an excellent mechanism for filtering unwanted diagnostics and runs on the platforms used by Digital Engineering. It is particularly good at flagging potential C++ problems. This is important because C++ can be more error prone than C, especially for novice C++ users.

Testing Technology Summary

Problem Statement

Software testing in Digital is currently ineffective because it is too late, labor intensive, tedious, and slow.

Goal

Make software testing during development and maintenance comprehensive, efficient, uniform and simple. This includes finding bugs early, convincingly demonstrating function and performance, and minimization of human, computer and software costs.

Summary of Recommendations

- Establish a Software Testing Expertise Center to give the Engineering testing effort direction, and to provide consulting and guidance on applying the variety of available testing technologies to the specific testing goals and proplems of individual projects.
- Provide a Testing Web, giving the latest information on
    - Tool availability and evaluation
    - Technology consultation and courses
- Contract for and support needed tools not otherwise available
    - Component testing capability for C/C++ (based on McKeeman Jig Generator)
    - Port UNIX DejaGNU (Tcl-based) command line driven test scripting to VMS and NT
    - Random testing capability (based on TclREX)
    - Model based testing

Summary of Rationale

Testing has a rich set of categories, a few of which apply to each project. Currently available testing tools are highly competitive and therefore changing.

Therefore we do not recommend a choice, but rather a *means of choice* where tool and method evaluation is centralized and made available to developers. Two necessary testing technologies (component and random) are not available commercially, therefore need to be developed. Using a single scripting language for all platforms simplifies many aspects (especially portability) of build systems (especially test systems).

Debugging Technology Summary

These recommendations address some of the problems inherent in developing software, and the solutions recommended to diagnose and eliminate these problems.

These recommendations address the problems of developing software for VMS, OSF, NT and Windows/DOS given a desktop that is running any of these. Although the ultimate goal is to have a consistent debugging environment for all these scenarios, product availability and cost demand a patchwork solution which is less than ideal.

Problem Statement

The development of software is plagued by a myriad of problems from logic errors, to coding errors, to problems with the underlying hardware and system components (the operating system, compilers and run-time libraries among other things). The goal of these recommendation is to offer productive approaches to identifying and resolving these problems through various debugging aids. This will be done in the context of the overall programming environment solution that is recommended out of the Internal CASE effort.

Beyond the complications inherent in software development, Digital Engineering is required to develop products which target a wide array of platforms including: VMS/VAX, VMS/AXP, OSF/AXP, NT/AXP, NT/Intel, and Windows/DOS (Intel). In order to avoid requiring multiple targets development systems on each engineer's desk, it is necessary to identify development (and debugging) tools which can be used to cross develop to these various targets from whatever desktop the developer has[3].

Goals

Beyond the overall Internal CASE goals of leveraging high quality third party tools, maintaining the skills of Digital engineers, and making the development environment attractive to talented engineers; the goals of the debugger recommendations include providing:

- Solutions to the broad range of problems which plague developers including: logic errors, coding errors, and memory leaks.
- Debugging aids to the various classes of applications that are developed: application programs (both GUI and character-cell applications), kernel/operating system code, drivers, and run-time libraries.
- Debugging tools which are easy to use without overly restricting the functionality that is available. Where possible, provide consistent solutions across platforms in order to reduce training and maximize programmer portability and productivity.
- Debugging tools which are well integrated into the recommended programming environments (the IDE's).
- Debugging tools to handle the complexities of modern applications such as: very large applications, threads, distributed applications, and optimized code.
- Debugging solutions that are easy and quick to configure and set up (for example not requiring a complete recompilation of the system for simple operations).

[3]Limits in the possible combinations of desktops and targets are outlined elsewhere in this document.

Summary of Recommendations

The problems of the developer are addressed in multiple ways depending on the desktop (host) platform and the target platform, and depending on the type of problem (for example: threads) and type of application (for example: user application versus kernel). A summary of the recommendation (in arbitrary order) is presented here:

Desktop->Target	NT/AXP	NT/Intel	Win/DOS	OSF/AXP	VMS/AXP
NT/AXP	VC++, Windbg	VC++, Windbg	VC++ V1.5 for Prelim Testing	DECladebug & Windows GUI	eXcursion w/VMS Debug & Sys-code Debug w/GUI
NT/Intel	Windbg? DECladebug?	VC++, Windbg Multiscope?	VC++ V1.5 for Prelim Testing	DECladebug & Windows GUI	eXcursion w/VMS Debug & Sys-code Debug w/GUI
Win/DOS	N/A	VC++ for Prelim Testing	VC++, Multiscope, Soft-ICE/W	N/A	N/A
OSF/AXP	N/A	N/A	N/A	DECladebug & Motif GUI	FUSE w/VMS Debug
VMS/AXP	N/A	N/A	N/A	N/A	VMS Debug & Sys-code Debug w/Motif GUI

Key:		
	N/A	This mix of host and target is not a requirement by overall Internal CASE effort
	☐	Denotes a native development environment (i.e. OSF->OSF versus OSF->NT)

As this chart shows, there are a mix of recommended debugging solutions. The reason for this is

- The general lack of availability of third party debug solutions on NT, and particularly on NT/AXP
- The further lack of cross-platform debugging solutions to and from NT
- The fact that unique tools are often required in order to solve the complex and specialized problems which come up in software development.

The shaded squares with "N/A" in them are not addressed by the Internal CASE effort, and squares with "?"-marks in them indicate the lack of a near-term solution. VMS/VAX and Ultrix/RISC are not listed in the matrix; these platforms are not a priority for the Internal CASE effort. The details of the platforms above are discussed in the sections below:

NT Hosted Development

For native application debugging on NT: The Windows development environment recommendation is two-tiered with VC++ as the standard, closed environment and Codewright as the open, extensible environment.

That VC++ product is essentially a closed environment precludes the effective integration of third party debuggers. This means that the default solution for application debugging on NT is restricted to the VC++ debugger. Restricting developers to a single debugger (on NT or any other platform) is unrealistic given the variety of complex problems that developers have. (Symantec is working to port their Multiscope debugger to NT and it is expected to work with the VC++ symbol table on NT but it is not clear when this will be available.)

The second tier IDE solution of Codewright is an appropriate environment to integrate such debuggers. At the moment though, there are no good third party debuggers on NT -- the only one currently available is Windbg from Microsoft which has many deficiencies.[4]

- The recommendation is therefore to rely on VC++ and Windbg for now but to drive internal or external developers to address the debugging needs on NT (particularly on AXP). One other tool which is recommended is BOUNDS-CHECKER (from NuMega) which is being actively ported to NT/Intel.We should encourage the vendor to port to NT/AXP.

For native kernel debugging on NT: The solution for kernel development on NT is restricted because of the fact they will be debugging some of the code that might be needed to run the debugger.

- The recommendation is to use Windbg as the remote debugger (between homogeneous NT systems).

For application cross-debugging targeted toward Windows/DOS: There are no solutions now available for cross-debugging Windows/DOS applications from an NT system, but there is a productive use of a NT/Intel system that can be used to develop Windows/DOS applications:

- Use the upcoming VC++ V1.5 for DOS on an NT/Intel system to generate Windows/DOS code.

[4] NuMega's Soft-ICE/W on Windows/DOS was evaluated, but it neither applies nor is it available on NT. Symantec's Multiscope debugger on Windows/DOS is also appealing but isn't available either. DECladebug is a possibility depending on the level of investment we want to put into DECladebug on NT (AXP and Intel).

Using VC++ will allow preliminary testing on NT and final testing of the same code on Windows/DOS. This will provide a superior development platform for DOS developers at the expense of having a DOS and NT systems available (although rebooting the Intel PC between NT and Windows/DOS is a possibility).

- Development of a traditional cross-debugging environment by Digital should be avoided.

For kernel cross-debugging targeted toward Windows/DOS: There are no tools available to aid in the cross-debugging of Windows/DOS kernel code from an NT platform.

- This is not worth investing in.

For application and kernel cross-debugging targeted toward OSF:

- Remote debugging from NT to OSF could be addressed through having DECladebug cross-debug applications that run on OSF. A Windows GUI is needed to have windowed debugging, otherwise, dbx-style character cell debugging would be the default.
- If no cross-development is needed from NT to OSF, then an eXcursion window can be used to debug the OSF applications using the DECladebug debugger (w/GUI) on OSF[5].
- Cross-kernel debugging depends on the integration of the kernel debugging capabilities into DECladebug.
- eXcursion is also a possibility for Kernel debugging. An eXcursion approach requires network access to source code.

For application and kernel cross-debugging targeted toward VMS:

- The debugging solution from NT and Windows/DOS platforms to VMS is to use eXcursion to run the VMS Debug product. Network access to the source code is required.

Windows/DOS Hosted Development

For native application debugging on Windows/DOS: The solutions available on this platform are the broadest and provide us with something to aspire towards on NT/AXP.

- The debugger for use with the VC++ IDE is the VC++ debugger (because that environment is closed).
- The Multiscope debugger is the debugger of choice for it's general debugging functionality but it's use with VC++ is restricted to the use of

[5] DECladebug currently runs on NT/AXP and could easily be made to remotely debug OSF applications.

C7 compatible symbol tables. Some integration of Multiscope with the Codewright IDE would make sense.

- For special purpose debugging (like kernel debugging or debugging involving system interactions) the Soft-ICE/W debugger from NuMega is the debugger of choice. In addition the use of BOUNDS-CHECKER is also recommended.

OSF Hosted Development

For native application debugging on OSF:

- For developers doing native development on OSF, the recommendation is to use DECladebug as a part of the FUSE environment. This is consistent with the current debug strategy of the corporation.

For kernel debugging targeted toward OSF:

- For native kernel development we recommend that integration of the kernel functionality currently included in the kdbx into DECladebug.

For application and kernel cross-debugging targeted toward VMS:

- For developers who have OSF desktops and want to develop toward VMS, the FUSE environment with the integrated Corporate Debugger GUI[6] will remotely connect to the VMS Debug product on VMS and allow cross-debugging of VMS (AXP or VAX) applications or a VMS/AXP kernel.

VMS Hosted Development

For native application debugging on VMS:

- The recommendation is to use the current VMS Debug product (with its Motif GUI) for all application development.

- Additional Motif workstations are recommended for those situations where GUI debugging is hindered by debugging on the same screen as the application being debugged.

- Worthwhile investments could be made in improving the debugging of GUI applications and making improvements in the support for debugging distributed and optimized applications.

- This recommendation is equally appropriate for AXP and VAX development although there is a question as to whether the investment in new functionality is worthwhile on VAX (unless it comes free with the changes in the AXP debugger[7]).

[6] The FUSE GUI is being replaced with the Motif GUI that is shared between VMS Debug and DECladebug.

[7] The AXP and VAX Debug products share most of their code.

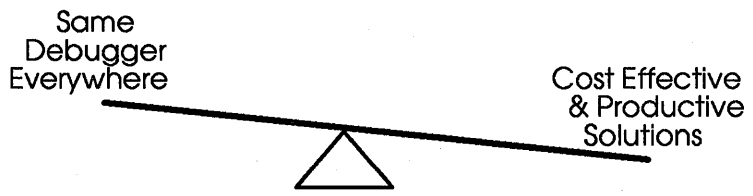
For kernel cross-debugging targeted to VMS:

- For AXP kernel development, use the enhanced VMS Debug product developed by the STD and EVMS Kernel Group (this includes the VMS Debugger with the Motif GUI and enhancements for kernel debugging). We recommend the inclusion of functionality of the existing delta and SDA products into this debugger.
- For VAX Kernel debugging, rely on xdelta rather than port these capabilities to VAX.
- Although cross-development is common in debugging the operating system, it is unlikely that VMS developers will require additional hardware since they already do cross-debugging with more primitive tools now.

For application cross-debugging targeted to NT/Intel: There are no cross-debugging products currently available which will allow users with a Windows/DOS desktop to debug NT applications. There may be something that appears on the market in the future.

- There isn't currently enough demand to invest in developing a solution here.

Summary of Rationale



The goal of delivering the same debugging solution across all host and target platforms seems noble but the cost of developing such a solution and the effectiveness of such a solution weighed slightly heavier in this recommendation. The balancing of these two issues are addressed in the various rationales below and shouldn't be considered static; decisions around the IDEs used on the platforms and what Digital's priorities are play an important factor on the recommendation. The recommendation at the moment is to postpone recommending a single unified solution and reassess at a later time.[8] In general the strategy around debugging solutions should be monitored and should evolve as circumstances (like NT tool availability) and needs change. The rationale below is broken down in a similar way as the Summary Recommendation Section above -- by desktop and target platform.

[8] The likely candidate for a unified debugger for all native and cross platforms is DECladebug. It is running on OSF/AXP, NT/AXP and Ultrix/RISC, it has been used for remote debugging and is the focus of current debugger language enhancements.

NT Hosted Development

For native NT debugging: For NT debugging: the default debugger is the VC++ debugger since the VC++ environment precludes the integration of any third party debuggers. This could change depending on the source licensing agreement Digital strikes with Microsoft.

The Windbg debugger is the only other solution (besides several kernel debugging tools) that is available. The popular Windows/DOS debuggers are not yet available on NT and although DECladebug is running internally on NT, it is not at a par with PC-based debuggers.

It is premature to select a debugger with such a narrow choice.

We should 1) use the default choices for now, 2) encourage third parties to provide debuggers for NT/AXP, 3) continue to invest in the DECladebug debugger because of it's support of Digital's compilers, and 4) make a decision later when there are real choices.

Several Windows debuggers (such as NuMega's Soft-ICE/W and Symantec's Multiscope debuggers) are available.

Unfortunately Soft-ICE/W addresses the specific debugging needs of the Windows/DOS environment and the functionality isn't applicable to the NT environment. In discussions with NuMega - they think they will be going to NT but don't have much sense about the functionality and positioning yet.

The Multiscope debugger is expected to be ported to NT/Intel but the delivery is unknown.

We must monitor these and other products that appear on NT. Besides the primary debugger that is integrated into VC++, it is likely that in the future another, more specialized debugger can be recommended for common use.

For cross-debugging targeted toward OSF: Debugging OSF applications from NT can most naturally be done with the DECladebug debugger since this debugger already runs on OSF/AXP and the code base has been compiled and run on NT/AXP. The capabilities to do the remote debugging are already there and would cost much less to deliver than any other solution.

Integrating this with the VC++ IDE is unlikely given the closed architecture of the VC++ environment so the most likely environment for this cross-debugging would be the Codewright environment.

A Windows GUI will need to be developed for this debugger. The approach of using an eXcursion window is not an adequate solution given the IDE integration restrictions that would exist.

Windows/DOS Hosted Development

For native application debugging targeted on Windows/DOS: The solutions available on this platform are the broadest and the selection of solutions was based on using the best available solutions that map to the IDE strategies.

OSF Hosted Development

For native application debugging: Of the alternatives for OSF targeted debugging, DECladebug is the most appropriate.[9] DECladebug is being staffed and planned as a replacement for the dbx debugger for all application debugging in the Gold OSF release.

The possible alternatives (such as the HP DDE debugger) had only marginal functional advantage over the DECladebug solution but when the cost of partnering and porting DDE are considered, the best solution is the one which is consistent with the current product strategy.

For kernel debugging: Both dbx and DECladebug were evaluated for kernel debugging, but considering the goal of replacing dbx with DECladebug for application debugging, we should consider doing likewise for kernel debugging in order to completely eliminate the dependency on dbx.

For cross-debugging targeted toward VMS: The approach of developing a true cross development environment from OSF to VMS (by either porting the VMS debugger to OSF or modifying the DECladebug debugger to cross-debug VMS applications) was deemed too costly for the small benefit of being able run the entire debugger on OSF.

The recommendation is to interface the Debug GUI in FUSE running on OSF to the character-cell debugger running on VMS.

VMS Hosted Development

For native VMS debugging: The needs of VMS developers differ from those of OSF and NT in that they are primarily working with legacy Bliss code and the individuals know the environment and tools very well (though the tools are in need of improvement). Using the existing tools (but with some improvements) is appropriate: use the VMS/AXP System-code Debugger with it's Motif GUI[10] and further integrate the kernel debugging components (like xdelta and SDA) on AXP so that those other tools can be retired.

[9] The alternatives are dbx and HP DDE. Both alternatives are discussed further into this paper.

[10]The System-code debugger on AXP is a product that combines the kernel debugging capabilities of xdelta and SDA with the VMS Debug product. This provides a source level Motif debugger for kernel development. This enhancement effort was done by the EVMS Kernel group.

Performance Analysis Technology Summary

These recommendations address the issues of performance measurement and evaluation tools needed by Digital Software Engineers for developing application and system programs. The performance of modern chip CPUs, such as ALPHA AXP and Pentium, are sensitive to the uses of resources within the application. Small changes in the uses of the computational resources can make large variations in the execution speed of the software. We recommend a set of high-level and low-level tools that can be implemented quickly from existing Digital A/D and development projects that can be used to measure the limiting resources for software.

Problem Statement

During software development, the developers must make a significant number of choices which cannot be validated, including process structures, software architecture, and data structures. During the testing process these choices effect software performance. This is particularly true for modern chips, such as the ALPHA AXP and the Pentium. These processors execute available instructions at a high rate, however unusual events may slow these processors to the extent that their high performance is compromised. Such unusual events include:

- Instruction Cache Misses
- Data Cache Misses
- Secondary Cache Misses
- Translation Lookaside Buffer Misses
- Context switching
- Page Faults
- System Call Overhead
- Subroutine Call Overhead

The software developer needs tools to approximate the cost of these events together with the normal performance measurement information needed for tuning the software:

- Real Time for each statement, subroutine, thread, or whole program

- CPU Time for each statement, subroutine, thread, or whole program
- Approximate cycle counts for each statement, subroutine, thread, or whole program
- Correlation of call graph with these other measures

Besides standard measurements, software developers need specialized measurements that are specific to the particular application. They need tools easily automate these measurements.

Goals

- Provide tools for both pc-sampling and cycle counting performance measurement.
- Include facilities in these tools to measure the performance bottlenecks such as cache misses (as listed above).
- Also provide the software developer a flexible performance toolkit to develop special case tools.
- These tools must be available on OSF/1, NT, and VMS, and measure applications for these systems.

Summary of Recommendations

Digital Software Engineering requires two kinds of performance tools.

- For real-time measurements and measurement of production systems, pc-sampling based tools are requires.
- For detailed measurements including information such as cache-misses cycle-counting performance tools are required.

The collector for PCA should be ported to OSF/1 and NT to provide the pc-sampling collector. ATOM, developed at WRL, should be used as the cycle-counter collector and basic performance toolkit. The following operations must be performed:

- Port ATOM/OM to operate on ALPHA/NT. ATOM will be the basis for all cycle-counting collector systems.
- Transliterate the pc-sampling collectors of PCA to operate on OSF/1 and NT. This code is used to provide the pc-sampling collection information.
- Use Tcl/Tk to develop display programs that provide the graphical representation of the information provided by the collectors created above.
- Provide improved documentation and sample systems to demonstrate how to use ATOM for the development of special purpose measurement tools

Summary of Rationale

Digital already has two significant assets in performance measurement tools:

- PCA, which is part of the DECSET tools
- ATOM, which is a research project developed at WRL.

These two systems should be built upon to provide tools for software developers. General performance tools have two components: the collector and the analyzer. ATOM and the collector from PCA can be used as the collector components. The computations needed in the analyzer are simple, but the display of information may be complex. We recommend that the display component be developed using Tcl/Tk to allow quick implementation of the graphic interface.

More specialized collectors are not excluded by these recommendations. However, specialized tools (such as IPROBE) are not normally useful to the majority of Digital software engineers, due to the very fine granularity of the information collected.

Configuration and Build Management Summary

Problem Statement

Configuration management is the ability to identify, manage and control multi-versioned software source code and software-related components, such as plans, specifications, documentation and test suites. All contemporary CM technologies are layer on some form of versioned source code repository (*e.g.* CMS, SCCS, RCS, etc). But true configuration management goes well beyond version control.

In Digital Software Engineering today there is very little configuration management technology in use. The technologies that are being used tend to be home grown and generally bound to idiosyncrasies of the versioned source code repository (frequently CMS), the Digital environment, or a given project.

In UNIX, DOS, NT and the more enlightened VMS-based projects development proceeds with essentially the same technological support as was available nearly two decades ago.

The resulting opportunities for human error, and required labor to manage development artifacts in a sophisticated way, has a negative impact on both the productivity of Digital Software Engineering and the quality and support of the resulting software.

Goals

- Contribute to the succcesful development, maintenance, and use of Digital's software code base:
    - Security
    - Manageability
    - Shareability
    - Resusability
    - Supportability
- Contribute to improved development productivity
    - improve support for parallel work and integration within a given code base
    - enhance confidence that what gets built is what the developer intends
    - automate the detection and tracking of source dependencies ("includes")
    - minimize rebuilding by tracking dependencies and by sharing build results between developers
    - exploit available hardware parallelism to speed up required rebuilds
- Support the required desktop platforms (Alpha/OSF and Windows NT), and integration into the IDE chosen for each.
- Support configuration management and builds on OSF, NT, and VMS servers.

Summary of Recommendations

Standardize on the ClearCase technology from Atria Software.

Atria Software is porting its ClearCase product to Alpha/OSF and to Windows NT. The Alpha/OSF port should enter beta-test in February 1994. Digital has contracted for this port. Multi-Site, a product enhancement to provide explicit support for geographically distributed development groups, should go to beta-test in about the same time frame.

The Windows NT port will follow the OSF version by a few months.

Digital should deploy each product as soon as it becomes available.

Early clients of the Internal CASE technologies may have to use beta version of ClearCase. Atria will be looking to Digital to provide beta sites for the Alpha/OSF and Alpha/Windows NT port. For the Multi-Site technology , however, Atria is looking to organizations that are already actively using ClearCase. Hence, the Multi-Site technology probably will not be available to Digital development groups until it ships.

Summary of Rationale

Multiple processes within Digital Engineering over the last two years have investigated third party CM product from various perspectives. Invariably, these analyses have identified Atria Software's ClearCase product as the first choice.

These Digital analyses are consistent with industry analyses, such as the Ovum Report on configuration management.

Evaluated by the goals of the Internal CASE exercise and the evaluation criteria, ClearCASE is strongly recommended for standard use by Digital Software Engineering.