
Project Outline



Component software has the potential to revolutionise software development. As systems become larger, more complex there becomes a need to improve the quality, reliability and reusability of software.

Constant innovations in computing hardware and software have brought a multitude of powerful and sophisticated applications to users' desktops and across their networks. Yet with such advances come immense problems for application developers, software vendors and users.

- Today's applications are large and complex – they are time-consuming to develop, difficult and costly to maintain, and risky to extend with additional functionality.
- Applications are monolithic – they come pre-packaged with a wide range of features but most cannot be removed, upgraded independently, or replaced with alternatives.
- Programming models are inconsistent for no good reason. Even when applications have a facility for co-operating, their services are provided to other applications in a different fashion from services provided by different vendors and sometimes even the same vendor.

Component technologies such as ActiveX/COM¹ & CORBA² aim to address this need. They offer the possibilities of a revolution in software development where applications will no longer be complicated monolithic programs; instead they will be made up of a variety of highly portable software objects which can be integrated together with relatively little effort. An example might be a spelling checker sold by one vendor that can be plugged into several different word processing applications from multiple vendors.

Software component objects are much like integrated circuit (IC) components, and component software is the integrated circuit of tomorrow. The software industry today is very much where the hardware industry was about twenty years ago. At that time, vendors learned how to shrink transistors and put them into a package so that no one ever had to figure out how to build a particular discrete function – an NAND gate for example – ever again. The software industry is at a point now where software

¹ COM (Component Object Model) is the programming model and binary standard on which ActiveX and Automation technologies are based; OLE is a group of technologies based on top of COM.

² CORBA (Component Object Request Broker Architecture) is a competing standard for distributed object computing. It defines an abstract object model that describes components and their interfaces. It also provides standard mappings from the abstract object definition to concrete programming languages, but it does not define a binary standard in any way.

developers have been busy building the software equivalent of discrete transistors – standard, similar software routines – for a long time.

The project will examine component technologies with particular emphasis on the PC Desktop Development.

- Why we need component object technologies?
- How does this differ from object orientated programming?
- Who is going to benefit?
- What needs to be considered in the design and implementation process?
- What are the current solutions being used in the real world?
- Where is the future direction of this technology leading?



To put into practise the benefits and learn about the design process of building a reusable ActiveX component; it is envisaged that an object will be written that has a complement of methods and properties to provide data. This will then be used from different applications such as Microsoft Excel, Microsoft Visual Basic, Microsoft Internet Information Server to demonstrate the use of pre-built language independent objects.

Why Project Management?



Three major criticisms of computer system development are

- Projects are delivered late
- Actual costs far exceed estimates
- Systems do not meet expectations / requirements

“More software projects have gone awry for lack of calendar time than for all other causes combined. Why is this cause of disaster so common?

First, our techniques in estimating are poorly developed. More seriously, they reflect an unvoiced assumption which is quite untrue, ie. That all will go well.”

Fred Brookes from the ‘Mythical Man Month’

Why Plan?

- To monitor progress
- To provide motivation
- Input to formal estimating technique
- Contingency
- Basis of lessons learnt
- Mainly to trigger re-planning

A software quality plan is produced to ensure that this project is a “quality” one. It will outline all assurance measures to be applied to the project and will consider why, when and how these will occur. The demands for the plan are concerned with the developer rather than the user ensuring quality through the design and implementation, but should benefit both in the long run.

The quality assurance plan will not follow a specific standard, such as the IEEE standard, but will use points from this as a skeleton framework.

Reference Documents



Reference documents include the original one-page summary of the project outline handed in after agreement with the supervising tutor.

The University guidelines for projects, including the scientific approach to the planning and documentation, as well as the standards for including research sources and literature involvement.

The test plans & reports will be outlined for application in the development.

The anomaly tracking documentation specification will provide information on bug/error correction, which can be used to calculate metrics and be used in a “post-mortem” of the development process, as well as control the defect analysis.

User Documentation will need to be written to explain the available methods and properties of the design business object.

Review Plans for release readiness and checking important criteria.

Checklists provide a clear point for analysis of the completeness status of the project.

Standards for code development and control will be documented in the Software Quality section later.

Management & Responsibilities

The management of this project falls completely to the individual, as it is not a group piece of work. Responsibility for all deliverables ultimately too ends here.

Other people shall be involved in the reviews and feedback of the documentation and system. Namely, the supervising tutor will provide feedback on the work submitted and provide a guiding hand from their experience of previous projects. Friends and family are envisaged to help proof-read and test the system. The developer will in part play the role of the user drawing from observations and experience of using similar objects in development.

Documentation



Software design documentation will be produced in order to state the requirements of the system. This will change throughout the early stages of the project to allow for incorporations not first documented, through a cycle of refinement. A vision statement will attempt to explain the product’s purpose, explaining what the product is as well as what it isn’t.

User documentation will consist of on-line help as well as a guide to each method and property of the developed object. Showing examples of functionality and use.

Design documentation to assist in the quality and maintenance of the system will be produced. This will be of a technical content, from the logical design – detailing inputs/outputs and flow, to the physical design – layout and structure of database tables and files. Cross-checking and reviewing of this against other documents will be essential.

Test documentation shall incorporate the test plans and reviews to be carried out to control development and quality.

All these documents together will form the evidence detailing the project area, research, new development, conclusion and the various methods involved.

Standards, Practices, Conventions and metrics

Standards, practices and conventions for the code development are detailed in the Software Quality section later.

Below details the use of a metric-based checklist for determining readiness of final development & testing.

Testing is done?

- Automated test runs without errors
- Manual test cases have been run
- Each test area defined has been declared “done”
- Ad-hoc testing for each area completed
- All bugs regressed and closed
- Set up and all components frozen for minimum 2 days
- The ever-popular “gut feel” survey shows ready to release

Bug Find/Fix Data

- Bug find rate shows a decreasing trend prior to the zero-bug release (ZBR) and is maintained after ZBR
- No “must fix – severity 1” bugs reported during the last 3 days prior to the release

Reviews and audit



The reviews can provide effective checks, which can be applied during the early stages so that problem areas are discovered before the latter stages of the project. To assist in the reviews, notes will be taken to record discussion during the reviews and a fault list can be therefore be constructed; ensuring procedures and instructions can be later implemented.

The use of different proof-readers and user reviews will be the main methods. The supervising tutor will also be used as a sounding board for discussion and will provide refinement to the overall project. Reviews will also take place for the specification, test plan & user documentation.

Audit of system functionality shall be provided through the check-list of function points cross-referenced will the test reviews. Physical checking of

the development to the specification can also be observed and audited at the end of the project.

Tasks & Deliverables

The project documentation will form the main emphasis of the deliverables. This will include the user documentation as an appendix. The software development lifecycle will be demonstrated in the write-up of the system implementation.

Sources of Information



Information on component design – including COM & CORBA, will come from many different places, including but not limited to,

- Object Orientated Programming books
- Code Reuse discussions
- Microsoft Developer Network both CD-ROM and on-line based
- Microsoft Technical White-papers and press releases
- Object Management Group's library of technical resources
- Microsoft Visual Basic Programmers Guide
- Course notes and lecture material

Feasibility

The feasibility of the project should be first questioned on the hand-in of the project outline to the supervising tutor. Through the use of specification downsizing the feasibility of the project if necessary, be transformed to meet the development deadlines of each milestone.

Estimating Techniques



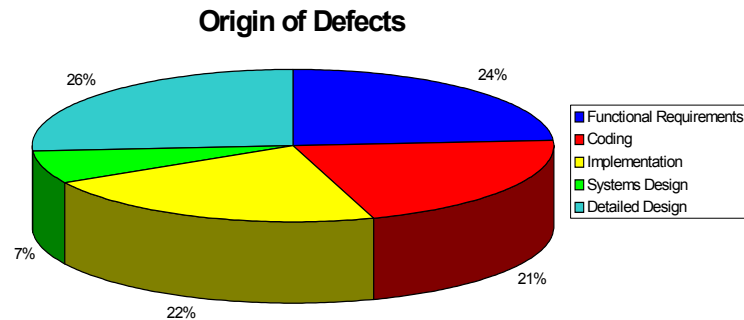
The major problem with project planning is over optimistic estimating. Estimates tend to be 'guess-timates' and therefore are liable to significant variance from reality. It is important that if software engineering is to develop into a serious discipline that the process of formulating reliable estimates is encouraged within the profession.

Metrics

By analysing the results to the following questions, metrics can be devised to better prepare and estimate development time needed.

- Where are the errors found? Location
- Type of error? Classification
- Productivity rate? Effort, modules signed off, lines of code
- Number of test runs needed? Outstanding errors, location
- Success inspections? Effort, Number of defects found
- How to improve development process? Origin of error

The chart shows the origin of defects for an example system.



Testing



The most common quality-assurance practice is undoubtedly execution testing, finding errors by executing a program and seeing what it does. The two basic kinds of execution testing are unit tests, in which the developer check their own code to verify that it works correctly, and system tests, in which an independent tester checks to see whether the system operates as expected.

The effectiveness of testing varies enormously. Unit testing can find anywhere from 10 to 50 percent of defects in a program. System testing can find from 20 to 60 percent of a program's defects. Together, their cumulative defect-detection rate is often less than 60 percent (Jones). The remaining errors are found either by other error-detecting techniques such as reviews or by end-users after the software has been put into production.

The science of software testing encompasses formal verification of safety-critical systems to dropping telephones on keyboards to simulate everyday accidents. The correct mix of code improvement and testing methods depends on the developers and users.

Of course the level of testing needed depends on how mission-critical the application is.

Program Type	Example	Cost of Failure	Tests Required
Vanity Program	Simple game	None	None
Data Display	Charting	None	Check major functions in common circumstances
Data Entry	Hotel reception	Irritated customer, time to re-enter	Check all functions in most circumstances
Business Object	Discount rule	Variable with applications	Check interfaces with correct data, wrong data, high-load testing
Team co-ordination	Restaurant	Temporary loss of revenue	Check all functions in all configurations, typical load testing
Business Critical	Trading floor	Business failure	Check all functions in all configurations, load testing, unit testing
Safety Critical	Fly by wire	Loss of life	Formal methods, Inspections, load testing, check all functions in all configurations

Overall, testing a project requires balancing the difficulty of testing against the cost of failure – how many faults and problems users will tolerate against how much extra work they’ll pay for to achieve the desired level of quality.

The project falls into the category of “Business Object” and such will be tested under the criteria outlined.

The following are seven rules of defensive coding that will be adhered to:

1. Check data types and API calls
2. Always remember to finish part-written subroutines
3. Slow down and do the whole job now
4. Avoid making past mistakes again
5. Get a fellow developer to look over code at check-in
6. Check array bounds and parameters
7. Verify end conditions and loop indices

By making the most of scarce resources and testing only what needs to be tested. Focusing on the areas that are most likely to affect users, and check out those areas thoroughly.

Potential Problem Areas	Problem Assessment Methods	What the methods check
Requirements, regulations	Functional and visual testing	That the program does what it is supposed to do
Internal interfaces	Unit testing of code modules	That subroutines and functions work correctly
Business rules, external interfaces	Unit testing of object interfaces	That the class in correctly implemented
User feedback	Beta testing	That the program is useful
Bug reports	Regression testing	That old problems have not reappeared
User environment	Combination testing	That the program works with other software
Load analysis	Stress testing	That expected and abnormal loads are handled

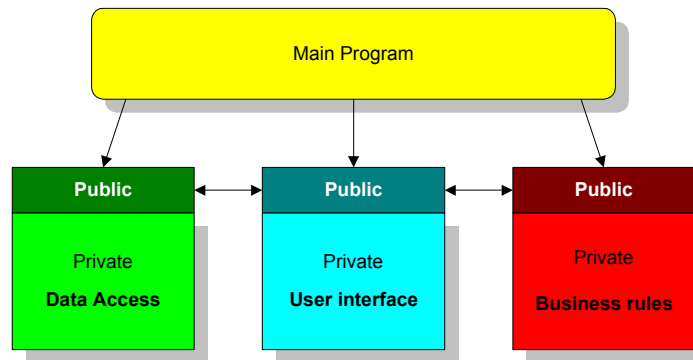
Regression testing will be taken care of by using an automated script developed to test the object with known inputs and examine the results. Much of the testing stages will be finalised as the development cycle continues.

Building Maintainable Code

Planning complex software system means planning for change – for those inevitable modifications, that will be made at some point down the road, whether by the original developer or someone else.

Change rate	Example	Implementation
Never	Days in year	Constant
Rarely	Digits in stock code	Constant
Occasional	Tax rate	Configuration
Frequent	Interest rates	Dialog option
Dynamic	Stock values	Real-time feed

By designing the application in an object-orientated way, encapsulating complexity into a set of public methods and properties, which is an agreement of functionality.



It is also said that “good fences make good neighbours” – so well defined interfaces minimise interdependencies between modules and allows them to be easily re-implemented or extended as necessary.

Problem Reporting & Correction



Problems need to be reported to allow proper tracking and correction. This will happen through the use of an “Anomaly Tracking System”. This will be database that will track issues during the development of the software. Issues can be bugs in the code, inconsistencies between the product and the specification, or future wish-list features. The areas & properties are detailed below. Each issue will have a unique ID number, title, etc.

Area	Properties	Area	Properties
Detail	ID No. Title Steps to reproduce Observed behaviour Expected behaviour Creation Date Revision Date	Status	Pending Active Resolved Closed
Priority	1 2 3 4 5	Severity	1 2 3 4 5
Resolution	By design Duplicate Fixed Not Reproducible Postponed	Assigned to	Unassigned Development Product design Quality assurance User education
Version	1.0, 1.1beta, etc.	Platform	Windows 95, Win NT
Language	English French, etc.	Found by who Area	Jonathan Hodgson Documentation User Interface Maintenance Customisation Performance
Found doing what	Ad-hoc testing Application development Automated testing Documentation review Manual test cases Performance testing		

Sub-area	Setup Queries Data sets Security Properties/codes Reports Printing Toolbars/menus	Issue type	Clarify specification Defect Suggestion Work item
Issue sub-type	Inconsistency Incorrect value Visual display Crash/hang Error message	Source	Beta tester Development Product design Quality assurance User education
History	Issue history History comments Issue notes	Attributes	Keywords Environment Attachments

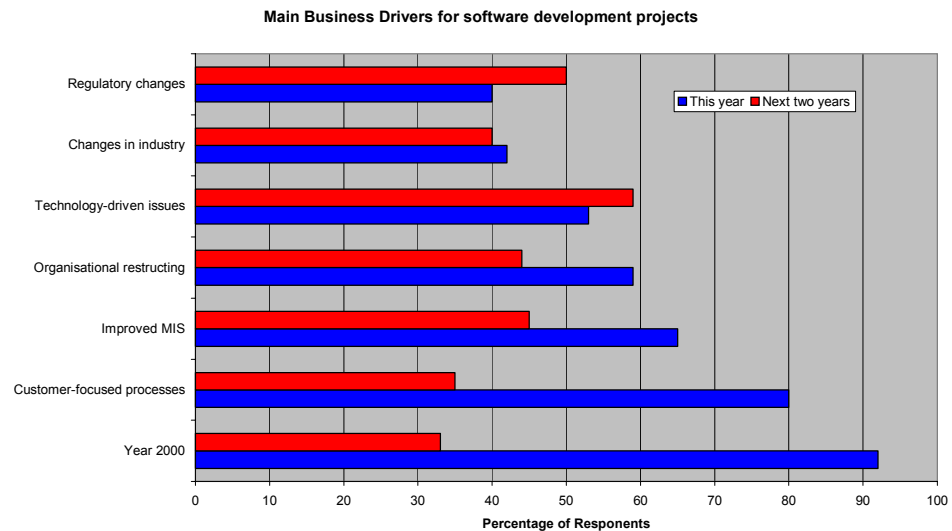
Tools, techniques & methodologies



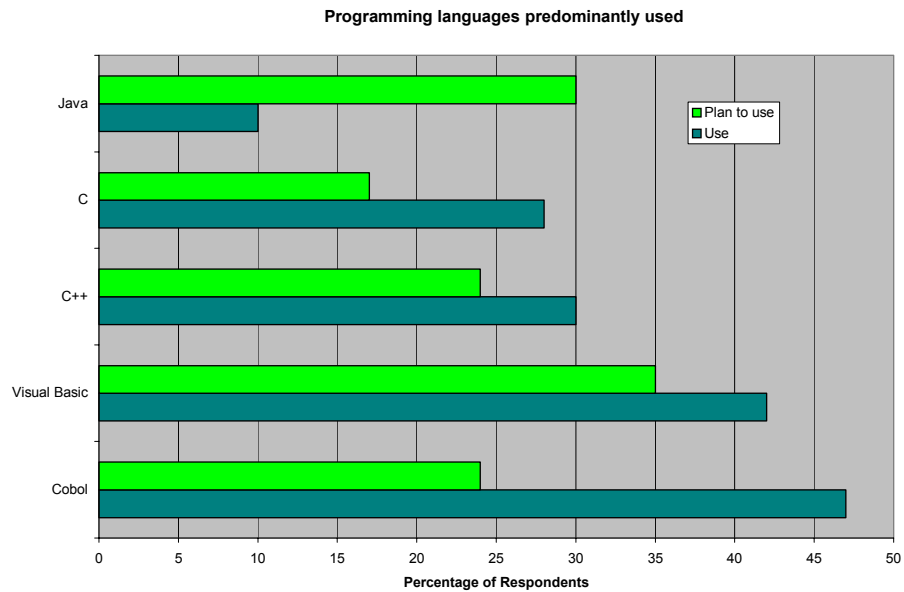
The tools, which are to be used, are:

- Microsoft Visual Basic 5.0 Enterprise Edition
- Microsoft SQL Server 6.5 Workstation Edition
- Microsoft Office 97 Professional
- Microsoft Internet Information Server 2.0

The reasons behind this are discussed now. The following chart shows research into the different projects under development by a variety of companies.



As year 2000 projects dominate the agenda for most companies, Cobol programmers are enjoying a lucrative short-term demand. There also appears to be a greater use of Object Orientated languages; Visual Basic, C++ and Java at the expense of conventional C programming. These figures are drawn from “Conspectus – Software Development Environment – The IT report for decision makers and consultants” issue October 1997.



Having already gained a great deal of knowledge in the use & design of software for Microsoft Windows NT using Visual Basic and its high level of implementation use, this is the decided route.

The methodology with which the system will be designed is Rapid Application Development (RAD). The reason behind this is RAD is becoming more widely used in the software industry, and it is through the use of components such as the one to be developed that RAD is achievable.

Code Control



Code control will come in the form of automated version control through the use of Microsoft SourceSafe, which is directly integrated with Visual Basic. This package provides version control functionality for the development of the software application.

Version control systems track and store changes to files so developers can review a file's history, return to earlier versions of a file, and develop programs concurrently.

Visual SourceSafe can maintain multiple versions of a file, including a record of the changes to the file from version to version. Also version archiving and tracking old versions of source code and other files is useful as these can be retrieved for bug tracking and other purposes.

More on this is discussed in the Software Quality section later.

Media Control



Media control discusses backup and recovery plans.

All application and system software can be re-installed from original media, allowing the development machine to be rebuilt in a matter of hours if necessary.

Backups will be taken daily, using a rotation of different media to provide a history roll-back option if required, as well as the roll-back provision of the source control system.

These include multiple copies of documentation and development both on-site and off-site.

Removable media in the form of diskettes will be used; this will include both 1.44Mb standard floppy disks to 1Gb Iomega Jaz disks for large files and system backups. Postage of completed milestones, to the developer's home-address will also provide an adequate solution.

Attaching the work to an external email and sending it to an address also owned by the developer but not physically part of the computer system, will make the backup and control of a copy of the project the external site's responsibility, adding another level of security.

Supplier Control



If the product's future depends on the future of the vendor who provides the tools, then a number of questions should be discussed. How long has the vendor company been in business? How stable are they? How committed are they to the specific tool of interest? Is it the tool in the vendor's main line of business, or a side-line. Is the tool likely to be supported by another company if the current vendor goes out of business?

With the development using Microsoft technologies, in both developer and productivity tools, this is not seen as a problem. If anything extreme did happen to the Microsoft Corporation it is highly unlikely to cause any problem in the project's time-scale.

Training



Normally training time considerations centre on whether anyone who will use the development tools has direct experience with them. Has anyone on the team attended a training course? How available are freelance programmers who know how to use the tool? How much productivity will be lost to the learning curve?

For this development, little or no training will be required. Proficiency of the development tool, Microsoft Visual Basic, is already at a very high level with over 6 years experience. Any training will be made during the development process and the investigation into new techniques and coding methods.

Risk Management



The job of software risk management is to identify, address, and eliminate sources of risk before they become threats to success completion of a software project.

Levels of risk management include:

1. Crisis management – Fire fighting, address risks only after they have become problems.
2. Fix on failure – Detect and react to risks quickly, but only after they have occurred.
3. Risk mitigation – Plan ahead of time to provide resources to cover risks if they occur, but do nothing to eliminate them in the first place.
4. Prevention – Implement and execute a plan as part of the software project to identify risks and prevent them from becoming problems.
5. Elimination of root causes – Identify and eliminate factors that make it possible for risks to exist at all.

Potential risk areas can be examined and a solution decided upon before the problem manifests itself.

Potential Risk Area	Likelihood (1-5)	Action / Solution
Feature Creep	2	Use customer-orientated practices Use incremental development Control the feature set Design for change
Disaster – machine crash	2	Use regular backups Minimise use of unnecessary software
Illness	1	Use of buffer-zones on delivery time-scales

Requirements & developer gold-plating	4	Scrub requirements Timebox development Control the feature set Use staged delivery Use throwaway prototyping Design to schedule
Short-changed quality	4	Allow time for QA activities and pay attention to assurance fundamentals
Overly optimistic schedules	4	Use multiple estimation practices Use principled negotiation Use incremental development
Inadequate design	3	Have an explicit design activity and schedule enough time for design Hold design inspections
Research-orientated development	3	Don't try to do research and maximise development speed at the same time Use a risk-orientated lifecycle
Lack of research material	1	Preliminary research Expand scope of project
Weak personnel	1	Staffing with top talent

Application of Quality Methods



The following section explains software-engineering mechanisms to be applied to the design/code to ensure robustness, ease of maintenance, quality of the developed system.

An evaluation of expected usefulness of these techniques will be discussed.

Project Infrastructure

A project infrastructure is something that must be in place to give any development hope of project success. Eisenhower said, “No battle was ever won according to plan, but no battle was ever won without a plan.” Similarly, having a solid foundation for project work won’t guarantee success, but not having a foundation can pretty much guarantee failure for all but the most trivial projects. Most of these issues are not complicated (certainly not in the same sense that performance optimisation techniques or clever coding algorithms are complicated), but they do require a fair amount of up-front investment to properly implement. They also become more valuable the larger your projects and project teams become.

Method	Details
Standards	Naming Coding Documentation
Practices and Tools	Version Control Common Objects/Shared Code Productivity tools Testing
Development time	The key is to drive down the time spent on essentially trivial matters: developers, expensive resources, don’t want to be endlessly debating naming or coding conventions with each other
Maintenance time	Consistently named and formatted code is much easier to maintain than ad-hoc code. The classic example is returning to an application for changes or maintenance six months after it’s been written: if developers have the same conventions now as they did then, this might be easily picked up
Customer acceptance	A consistent design that basically adheres to Windows standards will present a more professional appearance and ease users’ learning and ultimately acceptance of an application.

Standards

Naming Conventions



It is a good idea to establish naming conventions for your code. These outlines are based on the naming conventions used by Microsoft Consulting Services with a strong resemblance to the Hungarian notation.

Naming conventions help developers:

- Standardise the structure, coding style and logic of an application
- Create precise, readable, and unambiguous source code
- Be consistent with other language conventions to help ease different programming language communities
- Be efficient from a string size and labour standpoint, thus allowing a greater opportunity for longer fuller object names
- Define the minimal requirements necessary to do the above

Setting Environment Options

By use of the keywords 'Option Explicit' in Visual Basic, the environment forces the developer to declare all variables before use, this is to save programming time by reducing the number of bugs caused by typing errors (for example, aUserNameTmp vs sUserNameTmp vs sUserNameTemp).

The following tables outline the proposed standard for object and variable naming in the system development.

Prefix	Object Type	Example
cbo	Combo box and drop down list box	cboEnglish
chk	Checkbox	chkReadOnly
cmd	Command button	cmdOk
dlg	Common dialog control	dlgFileOpen
frm	Form	frmEntry
fra	Frame	fraStyle
hsb	Horizontal scroll bar	hsbVolume
img	Image	imgIcon
lbl	Label	lblHelpMessage
lst	List box	lstPolicyCodes
mnu	Menu	mnuFileOpen
opt	Option Button	optRed
txt	Text Box	txtLastName
tmr	Timer	tmrAlarm
vsb	Vertical scroll bar	vsbRate

Prefix	Converged	Variable Use
b	bln	Boolean
d	dbl	Double
dt	dat	Date and Time
e	err	Error
f	sng	Float/Single
l	lng	Long

i	int	Number/Counter
s	str	String
u	udt	User-defined type
vnt	vnt	Variant

Commenting code

All procedures and function should begin with a brief comment describing the function characteristics of the routine (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting unnecessary comment maintenance work, or worse yet, erroneous comments. The code and any necessary in-line or local comments will describe the implementation.

Parameters passed to a routine should be described when their functions are not obvious when the routine expects the parameters to be in a specific range. Function return values, global variables that are changed by the routine (especially through reference parameter must also be described at the beginning of each routine.

Routine header comment blocks should look like this:

Section	Comment Description
Purpose	What the routine does (not how)
Inputs	Each non-obvious parameter on a separate line with in-line comments
Assumes	List of each non-obvious external variable, control, open file, and so on
Returns	Explanation of value returned for functions
Effects	List of each effected external variable, control, file and so on and the effect it has if not obvious

Every non-trivial variable declaration should include an in-line comment describing the use of the variable being declared.

Variables, controls, and routines should be named clearly enough that in-line commenting is only needed for complex or non-intuitive implementation details.

An overview description of the application, enumerating primary data object, routines, algorithms, dialogs, database and file system dependencies, and so on should be included at the start of the system's generic constant declaration file.

Formatting code

Standard, tab-based, block nesting indentations should be four spaces. More than this is unnecessary and can cause statements to be hidden or accidentally truncated.

For example:

```

'*****
'Purpose: Locate first occurrence of a specified item in myList array
'Inputs:  smyList(): the list of items to be searched
'         sTargetItem: the name of user to search for
'Returns: the index of the first occurrence of the rsTargetItem in
'         the array. If target item not found, return -1
'*****

Function FindItem(smyList() As String, sTargetItem As String) As Integer

    Dim i As Integer      'loop counter
    Dim bFound As Integer 'target found flag
    FindItem = -1
    i = 0
    While i <= UBound(smyList) And Not bFound
        If smyList(i) = sTargetItem Then
            bFound = True
            FindUser = i
        End If
    Wend

End Function

```

Scope

Variables should always be defined with the smallest scope possible. Public (Global) variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Public variables also make the reuse and maintenance of code much more difficult.

Variables in Visual Basic can have the following scope:

Scope	Variable declared in	Visibility
Procedure-level	Event procedure, subroutine or function	Visible in the procedure in which it was declared
Form-level, module-level	Declarations section of a form or code module	Visible in every procedure in the form or code module
Public	Declarations section of a code module using Public keyword	Always visible

Practices and Tools

Version Control

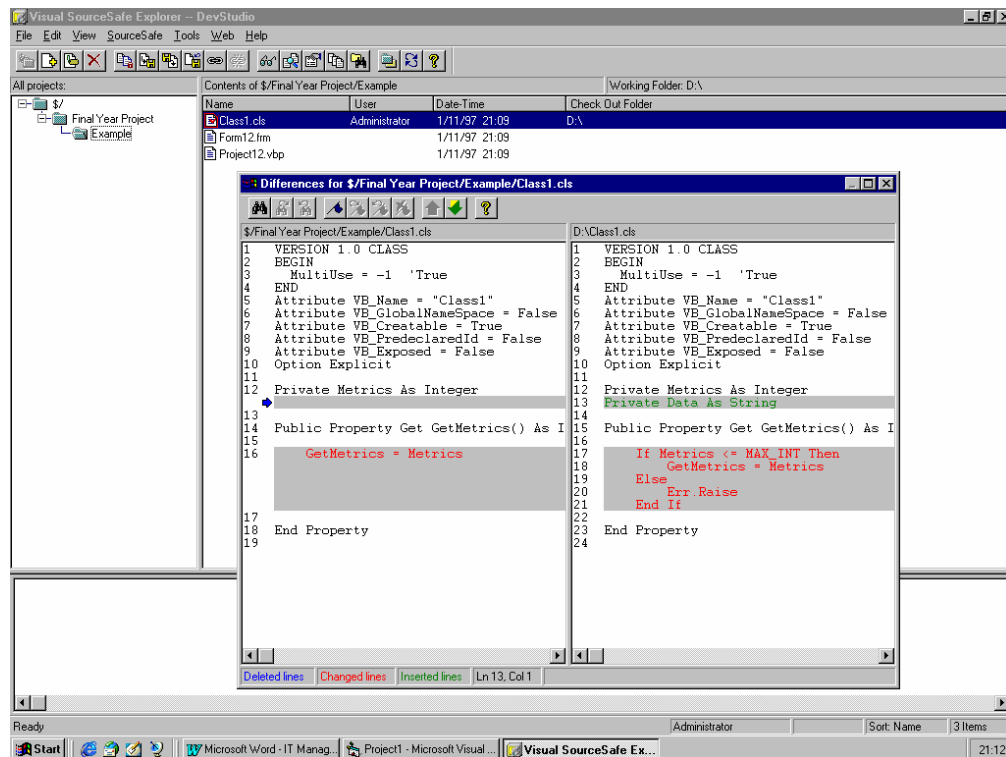


On any but the most trivial development, some form of version control must be in place. Based on experience, version control is one of the most under-invested areas of development: it's far from the most inherently interesting and generally doesn't get paid the attention it deserves.

Thinking of the developer making a mistake. How many times do developers work for a number of hours on a new feature, only to find that it would take more time to fix the old features broken in the process than it would simply to start from scratch and do the new feature right? A good version control package solves this problem easily: simply roll all changes back to the position before work on the new feature.

Efficient testing requires versioned releases. To test and ultimately debug a project, it's helpful to have clearly defined versioned releases (Alpha 1.0, Alpha 1.1, Beta 1.0, and so on) to test.

Below is an example of screen-shot of Visual SourceSafe in action, showing the hierarchical presentation of different projects and applications. The main dialog demonstrates the use of history comparison showing lines deleted, changed and added in different colours.



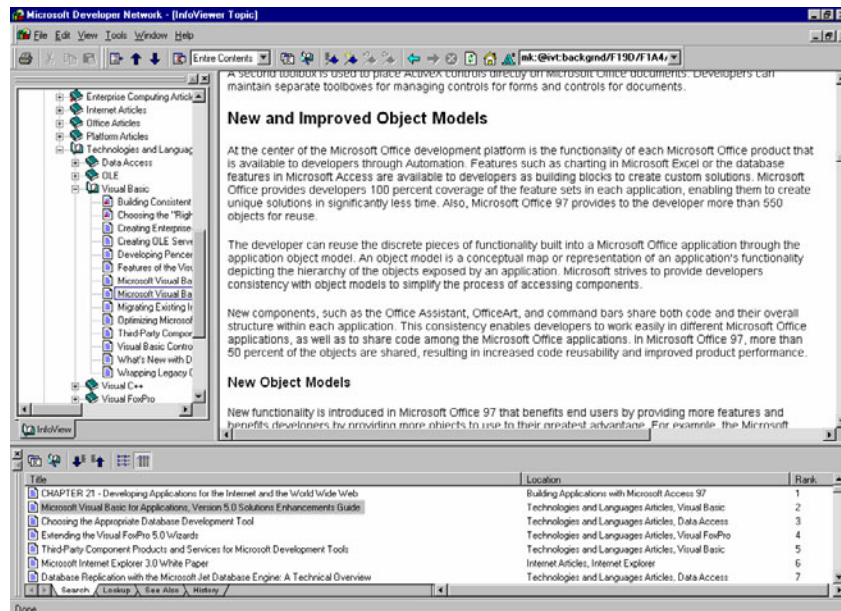
Software Engineering Productivity Tools

Software engineering tools are now becoming vital in the development and maintenance of software systems; three tools that are going to be used are discussed below.

Microsoft Developer Network



The Microsoft Developer Network Library is the most comprehensive source of programming information available from Microsoft today. It is used by developers around the globe – providing a centralised resource of up-to-date information on the latest technologies. Thus assisting in the research and development complexities.



Visual Quantify



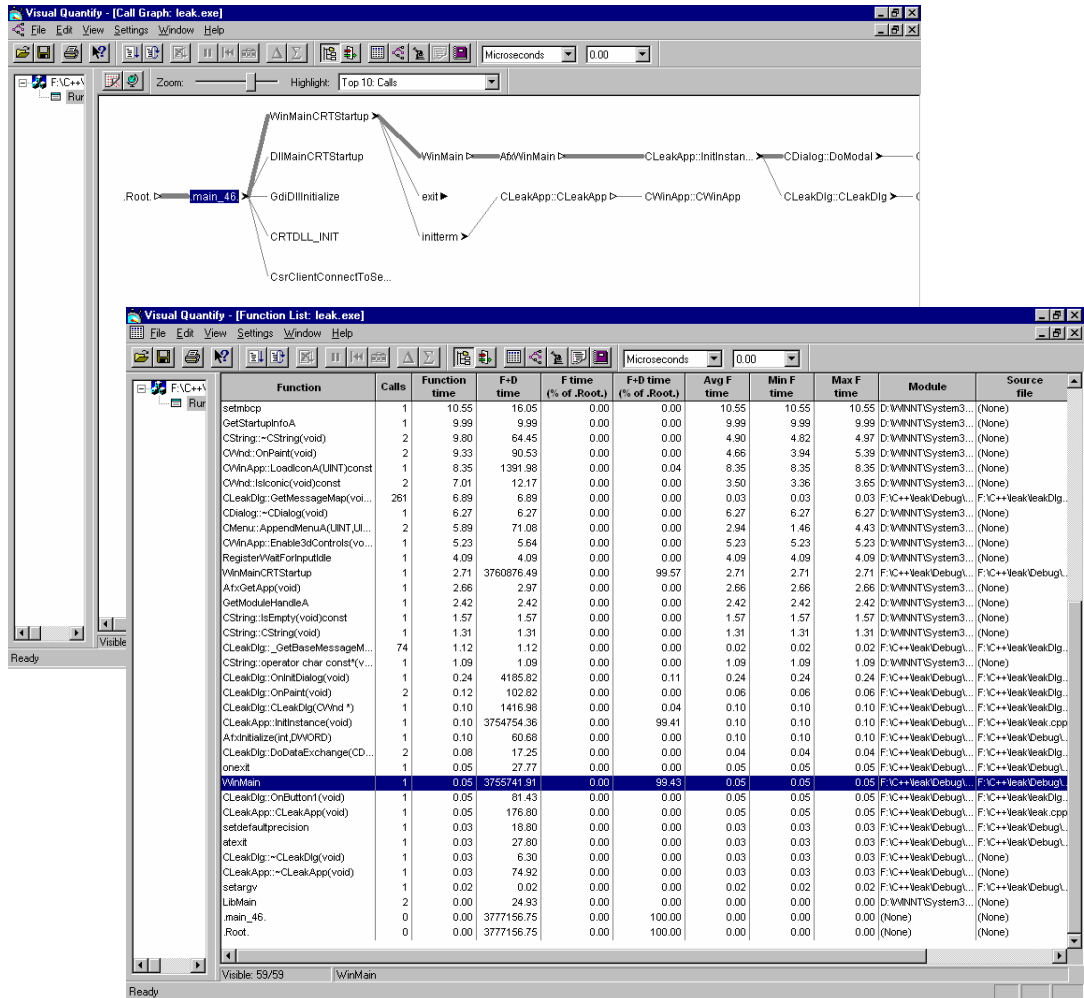
Visual Quantify is an advanced application performance-profiling tool for Visual Basic, Visual C++ Windows NT developers.

It can automatically pinpoint application performance bottlenecks quickly, taking the difficulty and guesswork out of performance tuning. It also delivers repeatable timing data for all parts of an application, including other components including the operating system, not just the parts for which the developer has the source code.

Graphical performance data views allow developers to see exactly how their program executed. Viewing the code in terms of function call architecture as it executed gives developers the insight they need to make the right changes in the program to improve performance. Detailed tabular views and line-by-

line timing data take developers right to the source of bottlenecks. With Visual Quantify, no development time will be wasted tuning code that does not offer significant returns in performance.

Shown below are some examples shots displaying the top 10 function calls throughout the execution of a test program, and then a table will timing metrics.



Purify



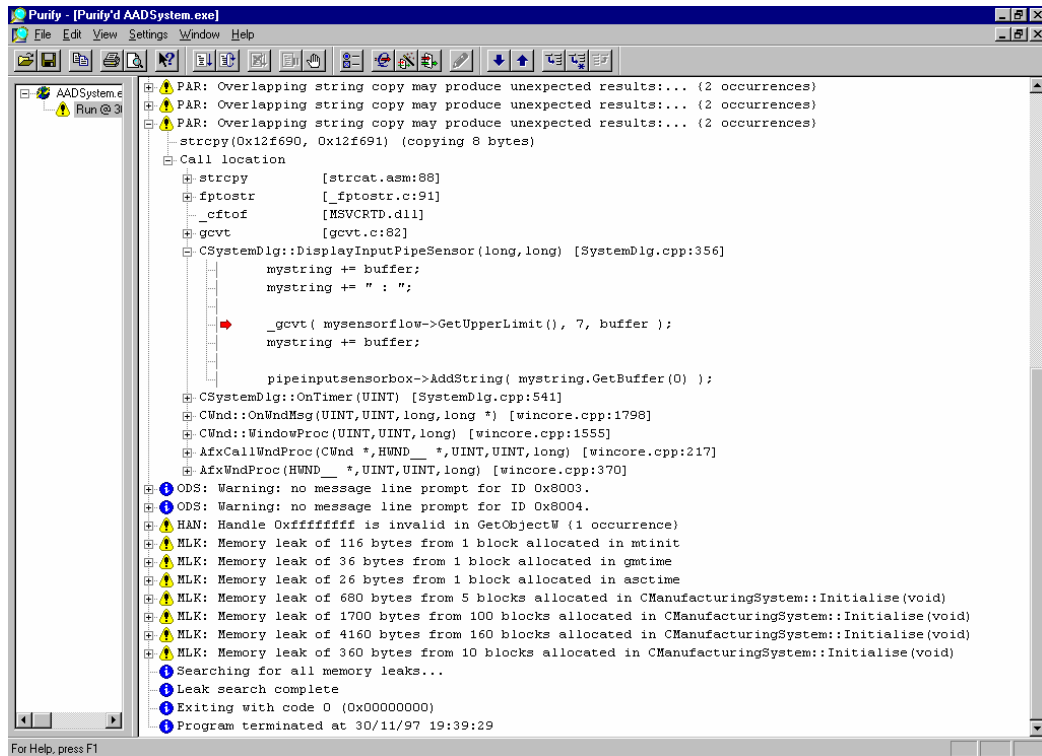
Purify is an advanced run-time error checking for Visual C++ Windows NT developers.

Purify improves software quality through quick and accurate detection of run-time errors and memory leaks in C and C++ Windows NT applications and other development languages.

Purify locates many of the software bugs that are most difficult for developers to find and fix, including reading and writing past the boundary of an array, leaking memory and using uninitialised memory. These errors

can take weeks or even months to identify and resolve manually. Purify unobtrusively monitors executing programs at run-time and informs developers of the problems as they occur. By automatically pinpointing the exact source of the errors, Purify greatly reduces the amount of time you spend debugging code and increases the reliability of your software.

As shown in the demonstration below, when used against the application created for the Advanced Analysis & Design module development. Showing memory leaks and out-of-bounds writes that are otherwise had to debug and find otherwise.



Testing & Maintenance

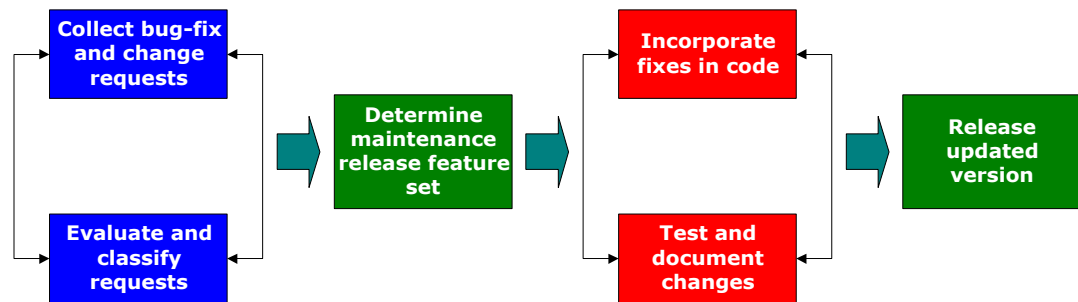
Test plans will not be discussed here, but obviously testing is the main way to assure quality and robustness of any development. Though the use of productivity tools discussed above and automated testing tools a great deal of defects will be found in the early stages.

One point of interest in the different level of perceived development statement variance of languages.

Approximate Language Levels adapted from data in the “Programming Language table” (Jones).

Language	Level	Statements per function point
Assembler	1	320
C	2.5	123
C++	6.5	50
Pascal	3.5	90
Visual Basic 3	10	30

The diagram below details the use of bug-fix & change requests in a cyclic manner through to the release of updated versions.



Miniature Milestones

The Miniature Milestones practice is a fine-grain approach to project tracking and control that provides exceptional visibility into a project’s status. It produces its rapid-development benefit by virtually eliminating the risk of uncontrolled, undetected schedule slippage.

By making milestones binary, they are either done or not. Percentages are not used. As soon as developer are allowed to report that they are “90% done” the milestones lose their ability to contribute to a clear view of project progress.

Development Phase	Feature development in 3 or 4 sequential subprojects that each results in a milestone release
Subproject I	First $\frac{1}{3}$ of features: Most critical features and shared components
Subproject II	Second $\frac{1}{3}$ of features
Subproject III	Final $\frac{1}{3}$ of features: least critical

By breaking the development in subprojects each with their own synchronisation and stability life cycle, the management, development and testing of the program is simplified.

Time scale	Work Area
3-5 days	Code and optimisations Testing and Debugging Feature stabilisation
1-2 days	Integration Testing and debugging
1-2 days	Buffer time zone (built-in allotment for slippage)

Evaluation

The use of all these practices will help ensure that the design & code will be robust and maintainable at the least.

Standards and conventions make coding easier for the initial developer and anyone else who has to maintain the code. Making cost-savings both in the short-term and long-term, for very little cost.

The use of automated productivity tools it is thought will become more and more widespread in the future of all software development. This project will provide a good test-bench to explain their use in real-world development.

The breaking down of tasks into separate binary blocks will allow for use measure of progress. Altogether these techniques should be very beneficial in the development cycle.