



# SOFTWARE DEVELOPMENT CYCLE

**Stephen Morris**  
**TESSELLA SUPPORT SERVICES PLC**

Issue V2.R1.M0  
June 2005



## 1. Introduction

At Tessella we believe that a well-structured approach is essential to the success of any software development project. However, each problem is different and the approach used should be matched to it. For this reason, Tessella-managed projects follow one of two approaches, the V lifecycle, and the Rapid Application Development (RAD) method. Both approaches are based on well-known formal methodologies, but include experience from Tessella's 25-year history of developing software for our scientific and engineering customers.

The purpose of this technical supplement is to explain the two development methodologies, and show how they help deliver a quality software solution to a customer problem.

## 2. The V lifecycle

### 2.1 Introduction

The V lifecycle is suitable for projects that have well-defined requirements that are unlikely to change significantly over the life of the project. In this methodology, the requirements are first listed, then the software is written, and finally it is tested and delivered. Since everything is well-defined, this type of approach can be carried out under a fixed-price contract.

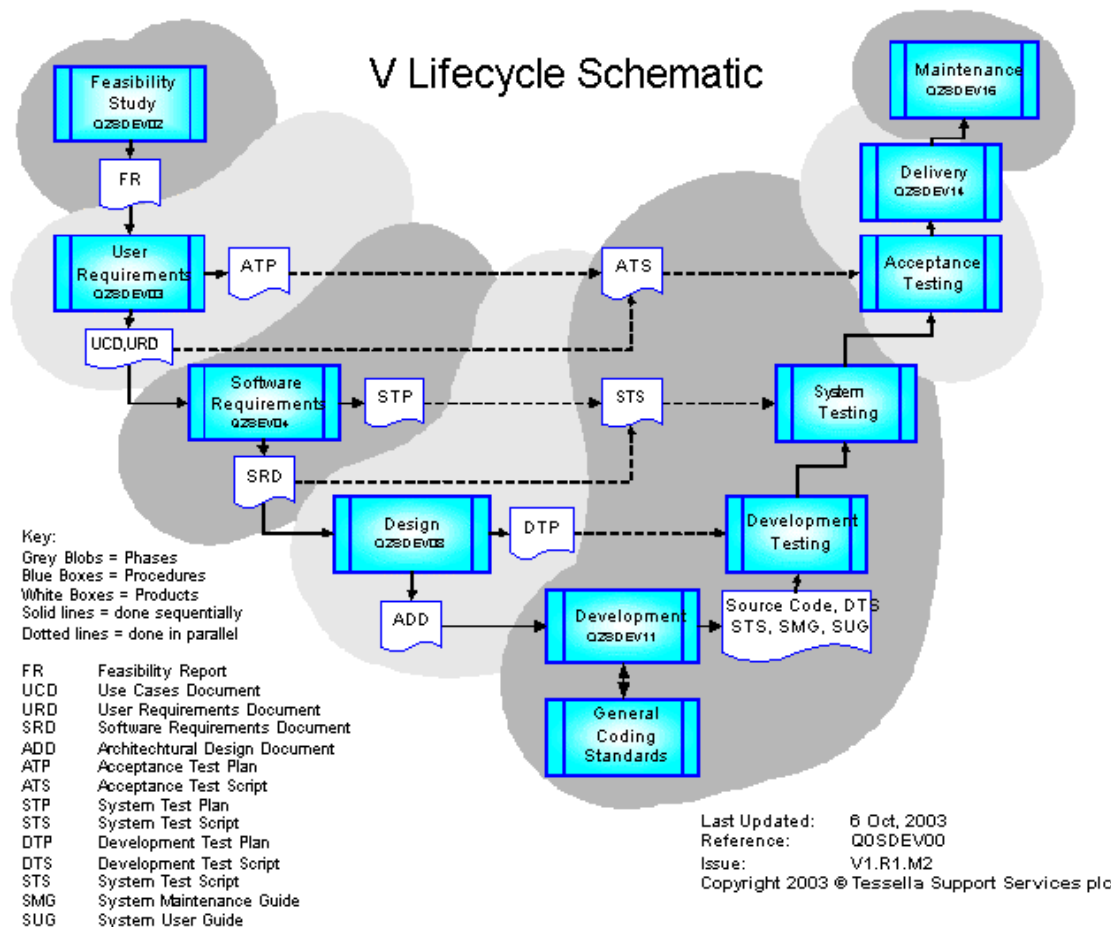
A typical V lifecycle will consist of the following phases executed sequentially:

- ❑ Feasibility Study
- ❑ Requirements
  - User requirements
  - Software requirements
- ❑ Design
- ❑ Development
- ❑ Testing
  - Development testing
  - System testing
  - Acceptance testing
- ❑ Installation
- ❑ Maintenance

Some of these phases may not be relevant to a particular project, for example a project may comprise just a feasibility study to assess the viability of a new idea. Alternatively the customer may already have documented their user requirements, in which case the project could start with the software requirement

stage. Where phases are missed out, the project manager will make sure that all the required inputs to the starting phase are available. For instance, it would be unwise to start design work unless an agreed statement of requirements existed.

The lifecycle diagram shown in Figure 1 illustrates the relationship of these phases with their inputs and outputs. It also illustrates the origin of the name 'V lifecycle', with each of the main requirements and design stages being paired with a testing stage.



**Figure 1. V lifecycle**

## 2.2 Feasibility Study

The feasibility study is intended to determine whether a proposed system can meet the system requirements effectively. It may be initiated either as the result of a new system being proposed or from the analysis of problems perceived in an existing system. A feasibility study should consider both business and technical objectives.

A feasibility study is particularly valuable where the proposed system is contentious, novel or critical to the customer's business. This evaluation will be used to decide whether more resources should be committed to a full system analysis, whether the project should proceed in a different direction, or whether it should proceed at all.

Using documentation for existing systems or procedures and interviewing staff involved with the area in question, the study will assess the effectiveness of a proposed system from a number of different viewpoints:

- ❑ Technical – is the proposed system technically viable?
- ❑ Business – are the business requirements met?
- ❑ Financial – do the benefits of the proposed solution outweigh the cost?
- ❑ Organizational – what would be the effects on present staff members?

The main output from the study will be a feasibility report. The form and content of this will vary but will generally include the following:

- ❑ Context – a description of the current situation, highlighting problem areas requiring improvement
- ❑ Requirements – a description of the situation the customer wishes to achieve. This will form the basis of the user requirements specification
- ❑ Constraints – a description of the constraints – business, technical, financial – on any solution
- ❑ Proposed system – this will be the recommended solution
- ❑ Alternative options – other possible options highlighting their disadvantages over the recommended solution

Of course, it is possible that following review with the customer the project will not go ahead. Alternatively, a hybrid system may be chosen rather than that which is proposed.

## 2.3 Requirements

The purpose of the requirements stages is to define the software requirements completely, consistently and unambiguously. These requirements should be documented and agreed with the customer before passing on to the next phase.

The requirements gathering can be split into two parts; the user requirements and the software requirements. The user requirements stage describes the problem in the customer's language, and is concentrated on what the new system should do, without stating how it should do it. Some requirements may have nothing to do with software, perhaps identifying a required change to a business process. Either way, until the needs of the customer are fully understood, it is not possible to proceed to a solution with confidence.

### 2.3.1 User Requirements

This stage will include interviews with the customer as well as other interested stakeholders, and an examination of existing documentation (including a feasibility report, if there is one). The purpose is to elicit the true requirements without pre-supposing any particular approach or solution. Otherwise there is a risk that the solution is assumed and that the requirements are manipulated to fit it.

The results are documented in a user requirements document (URD), which are agreed with the user and used as input to the next stage. Typically, a URD would include the following sections:

- ❑ Environment – in what physical environment and on what computing platform is the system required to operate?
- ❑ Users – how many and what type of users will the system have to support? What will be their skill level and how much training can be given?
- ❑ Functions – this will be the largest section and will describe each function that the system is to carry out. It is useful to identify functions as Mandatory, Desirable or a possible future Enhancement
- ❑ Performance – some limits should be set for factors such as target and maximum response times, maximum number of concurrent users, and maximum data volumes. Performance figures should always take into account the specification of the computer being used and its operational loading
- ❑ Security – does the data or application code need to be protected from unauthorized users? Does the system need to support multiple levels of access for different users?
- ❑ Constraints – under what time and financial constraints must the system be developed and operated?

At this time, consideration is also given to the form and content of the acceptance test for the system, although not necessarily in detail. The purpose of the acceptance test is to demonstrate that the completed system meets all the requirements that are specified during this phase. It is likely to be defined in the terms of the customer's needs and will not contain detailed scripts at the key-stroke level.

Once the user requirements have been agreed, the project can proceed to the software requirements stage.

### **2.3.2 Software Requirements**

Having established the user's requirements, the next step is to identify the software required to implement those requirements, and to form a high-level logical model of it. This is then used to describe 'what' the product will do as a series of short demonstrable statements. In this respect it is an initial analysis phase rather than a simple documentation of requirements. During this stage, the user requirements are examined in detail and 'fleshed out', again with the co-operation of the users. So, a simple user requirement of the form:

The user must be able to inspect and manage the data created by the spectrometer

...might be expanded to a whole series of software requirements along the lines of:

- The user must be able to select files to open from a list of current files
- When opening a file, the system must check whether a file is currently opened by another user. If it is, the file must be opened in read only mode
- The user must be able to select files for deletion
- The system must not allow the user to delete files currently opened by another user

The requirements provide the reference against which the design and final system are verified, and are documented in a software requirements document (SRD). Since the software requirements are an expansion of the user requirements, the headings in the SRD are similar to those in the user requirements document. As with the user requirements, the software requirements will be reviewed and agreed with the customer.

Just as thought was given to the acceptance tests when the user requirements were being documented, so thought is given to the system tests as the software requirements are obtained, and a plan for the testing is created. The system tests are of a finer granularity than the acceptance tests, being a test that each software requirement has been correctly implemented. So, for a lab system an acceptance test might be something along the lines of 'run the standard calibration sequence and inspect the results', the system test will be checking that every requirement in the SRD is satisfied, that erroneous input or error conditions are caught, etc. Consideration must be given to the tests at this stage so that the necessary arrangements can be made; for example, if a system is being developed to work with a piece of equipment at the customer site, time will need to be allocated to that equipment for testing.

Again, the requirements document is reviewed with the customer before proceeding. It is far cheaper to make changes at this stage of the development than later on, so it is well worth the customer's effort to get the specification right. Early involvement by the end users can also increase their sense of 'ownership' and commitment to the system.

## 2.4 Design

With the requirements now clearly specified, the design phase involves planning and documenting the global features of the software in preparation for the development phase. Typically, an object-oriented method using the UML design language has proved the most efficient way of doing the design, although other tools and techniques will be used as appropriate.

The Tessella approach is to decompose the system to a level at which an intelligent programmer is able to take the identified components and implement them. It has been found more efficient to leave the programmer(s) to design the low level internal features of the software routines that they develop, such as flow of control, variable types, etc. Sometimes, depending on the complexity of the module, the programmer may elect to refine the design further before starting coding.

The design will typically address issues such as:

- ❑ System breakdown – breakdown of the system into separate modules
- ❑ Module interfaces – how do the various parts of the system fit together? By defining these interfaces, work can proceed on the different modules in parallel

- ❑ Database design – how will the data be stored and structured?
- ❑ Processing – what operations are to be carried out on data and what algorithms are to be used?
- ❑ User interface design – how will the system look to the user? This is at a high-level, outlining overall structure and setting standards (e.g. ‘Windows look and feel’). The actual detailed layout of the various screens is typically left to the programmer.
- ❑ External interfaces – how will the new system connect to existing systems (computer based or manual)? What file formats are to be used to export and import data?
- ❑ Data conversion – if existing data are to be loaded into a new application, will any conversion or re-formatting be required?

The main outputs will be the architectural design document (ADD), which will ultimately become the basis of the system documentation. If the software is large, a decision might be made to perform formal, scripted testing at a module level, testing modules in isolation from the remainder of the code. If so, a plan for such testing is developed at this stage.

## 2.5 Development

Once the design is complete, the development phase can begin. This is where the software is actually written. The main inputs are the software requirements document and architectural design. The software must be written to meet the requirements specification and tested during development. Programming standards and good practice guides for the selected development language or tool are followed during this phase, with the code being reviewed on a regular basis.

Towards the end of this stage, a number of documents are written:

- ❑ The system test script(s). Using the system test plan as the basis, this is a set of one or more scripts that between them check that each requirement specified in the software requirements document has been correctly implemented. It is undertaken now so that the test script accurately reflects the software.
- ❑ A user guide, aimed at end users of the software.
- ❑ A maintenance guide, aimed at future developers or maintainers of the system. Read in conjunction with the architectural design document, it provides a high-level guide to the way that the software is constructed, and a starting point when changes are to be made.

## 2.6 Testing

The purpose of testing is to establish whether a system is technically robust, reliable and fit for its purpose. The key features of testing are that it should be identifiable, repeatable and documented.

The Tessella lifecycle includes the following types of testing:

- ❑ Module testing of each software routine
- ❑ Testing of subsystems (development testing)
- ❑ System testing of the whole software application on the same platform as the customer will be using
- ❑ Acceptance testing (part of the installation phase)

Module testing is done by the programmer during development, the development proceeding in a set of code-test cycles. As noted earlier, development testing – by which is meant formal, scripted testing of subsystems – is usually only carried out for large software developments. For smaller pieces of software, the system test is usually sufficient. System testing primarily checks for completeness; that every feature has been implemented, and implemented correctly. Finally, acceptance testing allows the customer to perform a check on the delivered software, and sign off the development if satisfied.

It should be realized that categories listed above cover many types of testing identified in the literature. For example, a system test will check that every software requirement has been correctly implemented. If the project is an enhancement to an existing system, the system test is likely to be a re-run of the system test of the original system extended with tests for the new components. In part then it is a regression test, in that one of the functions of the test will be to ensure that the changes have not caused any part of the existing system to fail.

Testing is a complex area and is often given insufficient attention. It must be understood that no amount of testing can ever guarantee that an application is completely bug-free. The cost of testing must be balanced against the potential cost of failure through software bugs. For non-critical applications it may be acceptable to issue a beta release to users for operational testing. Such an approach would not be appropriate for a safety-critical system.

## 2.7 Installation

Once all the previous phases are complete, the Installation phase of officially handing over the software application and associated documentation to the customer begins. The actual installation process will usually involve:

- ❑ Installation of software on the target computer(s)
- ❑ Running the acceptance test with the customer
- ❑ Signing off of the software by the customer
- ❑ Basic user training

## 2.8 Maintenance

Once the Installation phase of a project is completed, a software project enters the Maintenance phase of the software lifecycle. Maintenance includes the following activities:

- ❑ Problem resolution – detection, analysis and correction of software faults
- ❑ Interface modifications – caused by additions or changes to hardware platform on which the software runs
- ❑ Functional expansion or performance improvement – customer-requested changes after initial delivery and installation

The maintenance period immediately following installation may well be included as part of a warranty agreement supplied with the software. Long-term maintenance should be defined in a maintenance plan that specifies the procedure for dealing with maintenance reports from the customer.

## 3. RAD Lifecycle

### 3.1 Introduction

The 'V lifecycle' forces developers and customers to think through requirements before building a design to meet them. However, it suffers from two drawbacks, which can be severe in certain types of projects. Firstly, it assumes that requirements can be specified correctly at the start of the project. This can be difficult since users may only have vague ideas about their needs and developers only vague knowledge of the domain (so implicit requirements will be missed). Secondly, the V lifecycle approach assumes that these requirements will not change significantly.

In contrast, the Rapid Application Development (RAD) approach aims to retain the benefits of the V lifecycle approach while ensuring that users are kept involved, by providing them with constant feedback on progress and giving them the ability to change the outcome if this does not reflect their needs. The Dynamic Systems Development Methodology (DSDM) is one, well established, means of proceeding on RAD projects.

In RAD, there is a clear change in emphasis from a static requirement-centred approach to a dynamic user-centred approach. Indeed, the users are considered to be part of a wider RAD team that covers all aspects of the project, including training, rollout etc. The aim is to break down the barriers between 'them' and 'us'. Users can feed information back in a number of ways before release e.g. Use Cases, prototyping etc. The best feedback comes from people actually using the software in 'real life' so a RAD project must aim to deliver frequent updates of the software.

In RAD projects, the fundamentals of good practice remain the same as in the V lifecycle method. Indeed, some RAD projects may follow a route closer to traditional V lifecycle than others, or some may even explicitly state that some parts of the project will follow the V lifecycle (e.g. if coding up a scientific algorithm, the algorithm itself is probably not suitable for group discussion although the user interface may well be).

### **3.2 Structure**

There are four phases to a RAD project.

1. Feasibility Study. The first phase of a RAD project is like any other project. It begins with a feasibility study, the purpose of which is basically to decide whether there is a project and to outline its main purpose. The outputs from these are usually a set of high-level requirements, a budget and a deadline.
2. Functional model. During this phase the emphasis of the project is on clarifying requirements and building up a top-level (architectural) design. Thus, it incorporates elements found in the user requirements, software requirements and architectural design phases of the V lifecycle.
3. Design and build. During this phase the emphasis of the project is on detailed design and 'cutting code'. This phase is roughly equivalent to the development phase of the V lifecycle.
4. Implementation. During this phase, the emphasis is on roll-out and installation (including establishing live data), user training etc. This phase is roughly equivalent to the delivery and acceptance phase of the V lifecycle.

(There is no distinct testing stage – each iteration of each stage involves appropriate testing. Thus testing is spread out across the whole project, instead of being concentrated in one place.)

The main difference between this approach and the 'V lifecycle' approach is thus not what is done (development still follows design which follows requirements), but that change is expected.

### 3.3 Iterations and Involvement

In RAD, change is expected and feedback actively sought from the customers. Thus, the key element of a RAD project is that it is cyclical and that the functional model, design and build, and implementation stages are usually repeated several times. Indeed, developments in one phase may result in the team revising an earlier phase. The structure is diagrammatically similar to that shown in Figure 2.

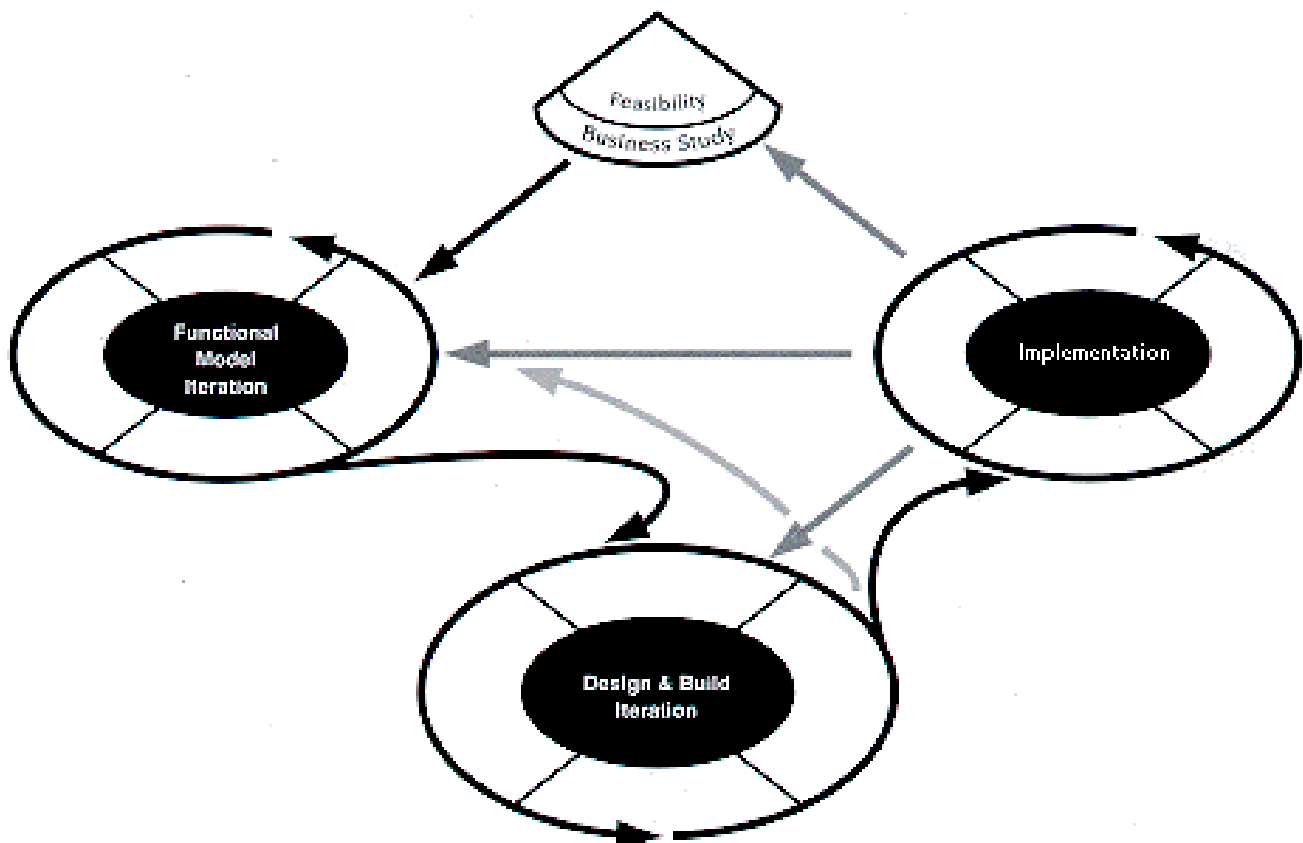


Figure 2. Phases of a RAD Project. Normal workflow is shown by black arrows, exceptional workflow by grey ones.

Each iteration of a stage is kept fairly short. This keeps the focus on interim deliverables, and helps to avoid scope creep during development. Time-boxing is a technique whereby production of a certain set of deliverables is agreed to be completed during a fixed period of time. Time contingency is introduced by prioritizing the requirements, so that those of lower priority are omitted rather than allowing the deadline to slip. Reviews and unit testing are included within the time box, and not left until afterwards.

The end of each stage includes a review comprising the whole team – which as noted above includes both developers and users of the system.

### **3.4 Completion**

With RAD, the question then arises, when is the project complete? Ideally, the project is complete when all the users' requirements have been satisfied. However, in the real world, there will always be some constraints, typically those of cost, and these will have an effect on the end point.

Since by its very nature the deliverables of a RAD project are not defined at the outset, the overall project cannot be carried out on a fixed-price contract; it has to be time and materials. To carry out such a project on a fixed budget means that there has to be careful control by the project managers, both the Tessella manager and the customer manager. The Tessella project manager will ensure that estimates for work are accurate, and that the customer always has an up-to-date picture of the state of the project, both technical and financial. The customer project manager is responsible for utilizing this information to ensure that high-priority features are implemented first, and that the final product – whenever the project ends – adds significant value to the business.

## **4. Conclusions**

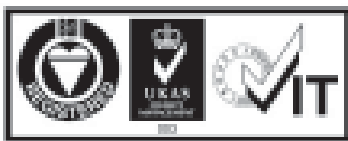
In order to make a success of a software development project a systematic approach is required, although the approach may vary depending on the type of project. The Tessella project lifecycles have been developed to meet the diverse needs of our customers, being adaptable in recognition that no two projects are the same, whilst retaining the structure needed to delivery a high-quality product.

## **Tessella - providing innovative software solutions to scientific, technical and engineering problems**

Tessella specializes in the application of innovative software solutions to scientific, technical and engineering problems. Our services cover software design and development, IT consultancy, infrastructure support and project management.

### **Other Technical Supplements published by Tessella include:**

- Active Server Pages
- Archiving of Electronic Information
- Automated GUI Testing
- Bayesian Statistics
- Beowulf Clusters
- Beyond LIMS
- C++
- Choosing and Using a LIMS
- COM
- Computational Fluid Dynamics
- Computer Image Processing
- Decision Support Systems
- Development for the Mobile Platform
- Digital Preservation Practical Experiences
- e-GIF
- Electronic Data Capture
- Electronic Lab Notebooks
- Evolutionary Computing
- Excel
- Extending the Life of Software
- FDA21 CFR Part 11
- Formulation
- FORTRAN 90
- Grid Computing
- High Throughput Experimentation
- High Throughput Screening
- Instrumentation
- Integrated Lab Systems
- J2EE
- Java
- Linux
- Microsoft .NET
- n-tier Architecture
- Object Oriented Programming
- Open Source and Free Software
- Pocket PC
- Portable GUI Development
- Real Time Systems
- Regression Testing
- Security and the Internet
- Simulation
- Soft Computing
- Software Development Cycle
- Software Documentation
- Software Portability
- Software Re-engineering
- Software Specification
- SQL
- UNIX Inter-Process Comms
- UNIX System Performance
- Web Services
- Windows 2000 Services
- Workflow Systems
- XML
- X Windows



Certificate No. FM 22778



INVESTOR IN PEOPLE

### **Tessella Support Services plc**

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

info@tessella.com

www.tessella.com