VOCAL: A framework for test identification and deployment

D. Pemberton
I. Sommerville

Indexing terms: Verification, Validation, Software testing, Quality assurance, Viewpoints, Software development life-cycle testing, Review processes, Traceability, Traceability scoping, Viewpoint information mapping, Test process modelling, Multiple test perspectives

Abstract: Viewpoint oriented verification and validation (VOCAL) is a novel technique for the identification and application of a structured life-cycle based software testing process to a broad array of systems. Application of the technique can be specfically tailored to embedded, interactive and multimedia applications, as well as those systems requiring a well defined approach in the quality assurance process. The authors introduce an approach for performance enhancement during the testing process. Problems faced when constructing software systems in time-constrained, market-driven development processes are highlighted. Emphasis is given to test process improvements by stressing some of the 'failures' encountered when using traditional test models. Traditionally, testing follows the stages of specify, develop, test and release in a standard waterfall based approach. The use of viewpoints will be introduced as a technique for structuring, identification, organisation and deployment of test information. Finally, the method is illustrated with a small example using a bank ATM machine. In conclusion, the authors evaluate VOCAL, and outline areas of future work.

1 Introduction

Cuts in development time and the imposition of tight deadlines between project initiation and release have become commonplace during system development. Companies know that they must be first to market a new product, or the first to release an enhanced version mirroring a competitor's product, thus retaining their existing customer base [1]. Furthermore, it is not only tight deadlines that pose problems, but also the notion that the finished product must be *reliable*, *responsive* and *easy to learn and use*, whilst providing customer satisfaction through use of the product. Testing, therefore, is not just the process of looking for defects, but also that of judging quality and validating products with customers' personal requirements.

IEE Proceedings online no. 19971643

Paper first received 3rd June and in revised form 16th September 1997

The authors are with the Co-operative Systems Engineering Group (CSEG), Department of Computing, Lancaster University, Lancaster LA1 4YR, UK

As development practices are changing, so must testing practice. It is no longer feasible to build a product from the ground up. Reuse of existing code, especially through component libraries, is increasing as a standard method of improving productivity [2]. Testing needs to embrace reuse by the capture and creation of distributable, reusable test specifications.

Releasing a product ahead of competition, meeting customer requirements, and ensuring that code is reliable, requires a carefully structured verification and validation (V&V) approach. We propose a method based around life-cycle testing viewpoints, guiding expertise and test identification to achieve these goals.

Viewpoints can be considered as a set of test process perspectives. Typically, we can observe three such perspectives during V&V:

(i) A *development perspective*, that focuses on defect detection and removal, system performance and reliability.

(ii) A *customer perspective*, focusing on robustness, usability and conformance with 'real' requirements.

(iii) An *organisational perspective*, dealing with schedule, cost, standards and conformance with the captured specification.

There is a clear relationship between requirements viewpoints [3–7] and test viewpoints. Both are concerned with understanding a system by looking at its services and constraints whilst identifying and resolving relationships between perspectives. Test, like requirements viewpoints, reflect stakeholder needs by the organisation and separation of concerns. Requirements viewpoints that specifically deal with end-user perspectives are particularly valuable during validation because they are available from the project's outset. Similarly, requirements viewpoints that consider interfacing with external systems are valuable to system testers for the same reason. Non-functional constraints placed on requirements filter through to the production of the quality specification, thus providing a base for assessing product quality.

2 Revising approaches to testing

Traditionally, testing has been seen as a single phase, or step, in the life-cycle; that is, errors, coded into programs, are detected and corrected. For example, to quote Myers [8], '[software testing is] ... the process of executing a program or system with the intent of finding errors', typifies the traditional narrow view of verification. The definition appears to imply that verification is, in itself, limited to executing a program and checking results produced against some form of expected output (program specification). Such a restricted view does not aid location and removal of specification errors, and cannot properly validate a product against customer needs, requirements and expectations of quality. Our preferred definition of software testing has been proposed by Hetzel [9]: '... the process of establishing confidence that a program or system does what it is supposed to do.' This implies that tests take into account user requirements together with the broad based remit of correctness.

There have been some previous attempts to define testing as a life-cycle activity, such as the approach proposed by Perry [10]. Other work in the area of testing maturity assessment [11] has come some way to characterising the goals and objectives required for life-cycle testing, although without defining a framework to implement this. Burnstein *et al.* [11], states that a testing process model should be acceptable to the software development community, and based around agreed software engineering principles and practices, whilst remaining industrially practicable. Our proposed solution addresses these problems through the construction of an integrated test and quality assessment process applicable throughout software development. VOCAL and its use of test viewpoints assists the creation of well planned and clearly defined test life-cycle policies that can be used and reused throughout an organisation.

The verification point of view, popularly described by Boehm [12], states: 'Are we building the product right?' (i.e. assessment of product-to-specification conformance); Boehm also describes validation succinctly as 'Are we building the right product?' (i.e. assessment of user-to-specification conformance). These two views, linked to stakeholder perspectives, add program correctness and stakeholder involvement issues, respectively, to development. Verification is, therefore, the procedure of checking the product against its specified requirements. Validation, on the other hand, is concerned with checking the specified requirements against their originators, being the system stakeholders, and ultimately the users of the product.

Some studies [13, 14] have gone to great lengths to make a point of stressing the costs of poorly formulated, untested requirements. They suggest that errors at the front-end of the life-cycle are the most numerous in software systems, and on average as many as 64% of all faults are attributable to specification and design defects. With only 34% of faults being attributable to coding errors, it can be seen that improving front-end defect detection will reap a large benefit in overall correctness and reliability. Defects undiscovered at their point of injection increase project cost with the time difference between fault insertion and detection. Perry [10] describes a four-fold cost increase through late fault identification. Therefore, it becomes paramount that we do not limit test activities to verification of the executable system. A great deal of cost benefit will be gained from an approach taking into account static front-end review as well as dynamic testing.

Stakeholder views are important sources of quality information. According to Rhenman [15]: '[Stakeholders are] ... the individuals or groups dependent on the company for the realisation of their personal goals and on whom the company is dependent for its existence'. It is because of the stakeholder associated view of quality that we use the terms 'verification' and 'validation' for the process rather than 'testing' to indicate the wide life-cycle view of defect detection. Such a process is more than applying some form of test data selection technique like white or black box testing [16, 17] to achieve a specific level of code (white box), or requirements (black box) coverage. The process actively involves system users.

A final point concerns the importance of structured quality checks throughout development. Technologies, like the Internet, introduce many nonfunctional attributes. Future Internet applications may include home shopping and video-on-demand services. Looking more closely at a video-on-demand service, important nonfunctional attributes include reliability and performance considerations. If these attributes are not verified so that they meet a specified minimum standard, then the final production version of the application software may suffer from dropped and corrupted frames in the transmission, and a poor quality of service. This is clearly unacceptable if a customer is charged for the service. Such problems may only occur under specific network load conditions. Therefore, we must find a congestion limit, and ensure that it will never be exceeded during normal operational use. A similar violation of non-functional requirements (NFRs) could arise in home shopping where customer credit card details are fraudulently captured due to inadequate security measures.

Having now identified the three classes of software test procedures (namely, verification, validation and quality assurance) we can move on to define a more structured testing approach by encapsulating and applying the procedures via interacting viewpoints.

3 VOCAL viewpoints

The idea of using multiple, explicit, viewpoints initially originated in CORE [6], and has been incorporated in later work on requirements engineering by Finkelstein [4], among others. Easterbrook has also encompassed viewpoints in his work at NASA's independent V&V facility during the WHERE project [Note 1]. Here viewpoints are used to manage requirements evolution. They trace how changes affect design objects and test cases.

Viewpoints arise from the need to represent chunks of a specification in a particular shareable representation and style. Viewpoints have been described [5] as arising from the need to include many agents' diverse perspectives in software processes. Each agent tends to have a different, incomplete view of the system they are modelling making a viewpoint the combination of the agent and their outlook.

Viewpoint oriented test planning is based on the notion of testing a system from different stakeholder perspectives. These stakeholder perspectives, or groups of stakeholders, are represented by viewpoints. Each viewpoint encapsulates information such as viewpoint scope in relation to the product, test specifications relevant to the viewpoint, and traceable links to other viewpoints. In short, applying viewpoints to testing involves constructing a test plan by combination of test specifications across viewpoints. While this is in progress, any redundant tests discovered across view-

Note 1: Further information on WHERE can be found at http://research.ivv.nasa.gov/projects/WHERE/

points are removed. Detected conflicts of concerns also need to be resolved. Conflicts imply a different understanding of system requirements from separate perspectives. Viewpoints improve the coverage of system tests by reducing the chances that important attributes are not considered.

As an example, testing viewpoints may be concerned with quality assurance, helping to ensure that usability, efficiency and reliability issues are considered throughout development via a program of continuous assessment. Additionally, viewpoints may be introduced for customer evaluation with a specific group of users. Other possible uses for viewpoints include system integration and compatibility viewpoints for use with, say, an external database system or viewpoints focused on defect detection and removal in a software module. Therefore, viewpoints support complexity control, separation of concerns and distribution of work among participants; they are also independent of any particular method or notation.

As with any methodology there are strengths and weaknesses when using viewpoints. The major weakness is the amount of information that can be produced by such an approach, sometimes called viewpoint explosion. Therefore any technique utilising viewpoints requires careful management to ensure that this does not occur. Prototype support tools [18] consisting of a viewpoint test case browser to assist information management are under construction thus helping to avoid such pitfalls.

3.1 VOCAL viewpoint notation

A viewpoint contains partial development process knowledge. This knowledge has a source (the viewpoint specifier), and is associated with a particular agent or role (test perspective). Viewpoints contain a number of information holding slots [3, 4, 19]. Collectively slots are grouped into viewpoint templates. Table 1 describes the generic viewpoint slots that occur in all VOCAL viewpoint templates. When a viewpoint is instantiated, appropriate slots are filled in.

Traceability is an important feature in VOCAL as it allows linking of related information. Hence, users of the method are able to cross-reference test cases to test specifications; also, when necessary, users can link those test cases to other related viewpoints. A traceability link can result from three different relations:

(i) *Interviewpoint traceability* occurs when it is necessary to link information between *different* viewpoints. For example, a link between a quality viewpoint, and a functional viewpoint.

(ii) *Intraviewpoint traceability* occurs when information is linked between slots of the *same* viewpoint. For example, a particular section of the test plan will be intratraceable to a test specification defining the work.

Table 1: Generic VOCAL viewpoint template

Viewpoint slot	Slot contents	Rationale
Viewpoint type	Defines the class of viewpoint under consideration in the template. Types include group (organisational) viewpoints, and those concerning verification, validation and quality (see Fig. 1)	Each viewpoint needs to define itself in terms of its core class. A VOCAL viewpoint must be one of these types.
Style	The test notation used by the viewpoint. This is selected according to the process model of the viewpoint. For example, a quality viewpoint may be represented by a quality inspection template; usability may concern interface design consistency, or usability evaluation; black box testing styles may include equivalence paritioning, boundary value analysis, etc.; developer test styles may include desk checking, peer review, coding standards, walkthroughs, intuitive testing, etc.	Test style is reflected in the information holding content of the test plan slots. Test style defines the criterion to which test information is selected to achieve 'coverage' of the associated intra traceable test specification slot.
Domain	Concerns viewpoint scope. For example, a quality viewpoint such as reliability applies throughout the life-cycle, whereas reusability need only be considered in design, and implementation phases. This helps to focus activity on areas of the product that will gain most benefit from the viewpoint.	In most cases it is necessary to restrict a viewpoint to a subset of the rest of the system, or a particular stage in the life-cycle.
Test specification	Defines test requirements, i.e. *what* must be tested. The test plan defines *how*, according to the selected style of testing.	Test concerns must be clearly and accurately defined so that there is a consistent base to test against.
Test plan	Details test cases associated (intratraceable) to the relevant section of the test specification.	This is the test information that is to be carried out on behalf of the viewpoint author(s) by the viewpoint tester(s).
Test record	The current status of the test plan, and historical information of past work carried out.	This is a record of test progress. Test outcomes, i.e. pass, fail (inc. severity rating), or not yet executed, are also recorded here
Viewpoint owner	Gives details of the owner of the viewpoint (i.e. the test specifier). They are also responsible for change requests to the viewpoint specification and test cases.	Somebody must be responsible for viewpoint management, and resolution of changes to ensure that both inter, and intra viewpoint consistency is maintained.
Viewpoint user	Details the individual(s) assigned under this role to execute the test plan.	People must be assigned to test cases. This slot maps testers onto groups of test cases.
Viewpoint traceability	Inter, intra and temporal traceability is stored in this slot. This makes consistency management easier, and also enables simple application of NFR testing, when traceability relations between viewpoints are showed to hold. Traceability therefore represents how associated viewpoints are affected by actions in this viewpoint.	Traceability is important as it provides a way to ensure that all requirements have a set of defined test cases (intraviewpoint traceability), and that consistency checking is clear (all inter traceability defined).

(iii) *Temporal viewpoint traceability* occurs when a viewpoint is given scope through *time* within the development cycle. Information is therefore traceable to viewpoints whose activities are concerned with different development phases. The temporal traceability link maintains consistency with other life-cycle viewpoints that may affect, or be affected by changes in the source viewpoint. For example, a viewpoint concerned with safety will most likely be applicable and therefore traceable to all development life-cycle stages; whereas, performance is normally only considered during design and implementation stages.

3.2 Applying viewpoints to test activities

The process of applying viewpoints involves three stages [5]: (i) choosing the right viewpoints model, (ii) identifying the viewpoints and (iii) managing information from the viewpoints.

Currently, we have only explored the testing viewpoint model proposed here, although clearly other models are possible. Secondly, viewpoint identification is typically tightly coupled with the attributes of the software under test. For example, a safety critical system's primary concern may be with reliability, testability and correctness; whereas a real-time system's concerns may be placed in the direction of efficiency, reliability and correctness issues. Hence, the idea of viewpoint tailoring, by reuse of previously defined viewpoint templates, comes into play. Collecting and managing information from viewpoints will be dealt with in the following Sections.

4 The VOCAL process

In this Section we aim to outline the VOCAL method, describing the process by introducing a generic viewpoint hierarchy that can be used to guide initial viewpoint identification. This is followed by a description of each viewpoint type necessary to construct a viewpoint oriented test plan. After test plan construction but before we start testing, conflict resolution of information contained in viewpoints should be performed (see Section 4.1.5).

Viewpoints need not be limited to a single stage of development. Non-functional tests and validation activities are often temporally scoped through the life-cycle. For example, a usability viewpoint applies during program development, but also may be traceable to front-end activities. Here a paper-based or dialog-builder assembled mock-up of the interface can be reviewed, possibly leading to cost saving if faults are detected early. This emphasises the need to minimise time between fault injection and removal.

4.1 Viewpoint classification

VOCAL implements a hierarchical model that is used to structure viewpoint templates (Fig. 1). The Figure demonstrates how templates fit together providing test activity coverage of the whole development cycle. All templates are derived from the generic template at the root of the hierarchy. Fig. 1 shows how child templates of the same type cluster into parent templates creating a group of 'like' templates (those concerned with similar activities). Viewpoint grouping avoids the explosion of too many viewpoints. Templates are instantiated to provide testing under a selected style. For example, a black box template may be instantiated for equivalence partitioning tests on a software component. Instantiated templates are individual *testing viewpoints*.

There are four fundamental viewpoint classes:

(i) *Group viewpoints* are organisational viewpoints holding no other purpose except viewpoint abstraction, distribution, clustering and integration.

(ii) *Verification viewpoints* are concerned with demonstrating that some domain information complies with a test specification. These viewpoints are particularly useful for representing the 'development perspective' of the testing process (see Section 1).

(iii) *Validation viewpoints* ensure that the viewpoint domain is reviewed with associated stakeholders. This ties in with the 'customer perspective' of the test process. Validation viewpoints can be used to provide a framework for customer evaluation processes.

(iv) *Quality viewpoints* ensure that nonfunctional attributes are met. This viewpoint type ties in with the

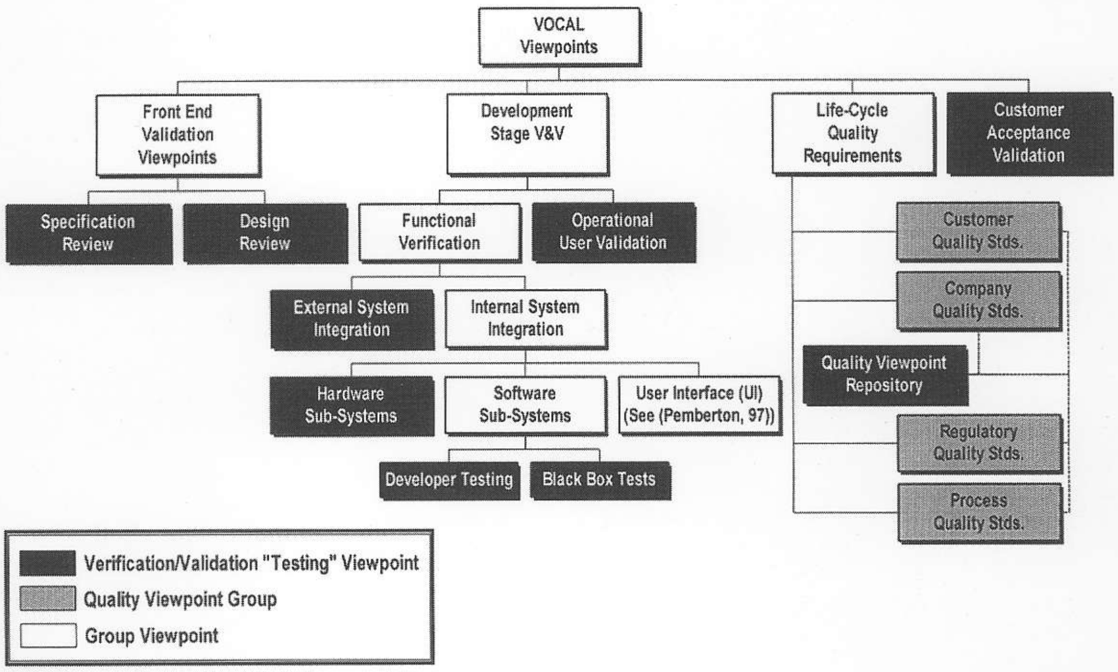


Fig. 1 *VOCAL viewpoint template hierarchy*

'organisational software testing perspective'. A quality template is used to define tests that ensure conformance to standards, schedule and cost.

4.1.1 Group viewpoints: Group viewpoints are used as an owner and tester distribution mechanism. For example, quality viewpoints are the responsibility of the quality assurance department; functional verification viewpoints the responsibility of the test team. Organisation of viewpoints around these groupings aids the distribution of work creating clearly defined responsibilities for each member of the development team. Group viewpoints also form subsystem integration points for use during system testing. Individual functional viewpoints contain logical groups of system modules that are tested to meet correctness issues set out in the viewpoint. If these tests are passed then the viewpoint may be integrated with other functional viewpoints that construct the system. The system is built by collapsing down subviewpoints under a particular viewpoint grouping, hence proceeding with the system build.

Group viewpoints contain no requirements, they only help to confirm that interfaces between their subordinate viewpoint's implementation units match. Group viewpoints therefore do not contain specification, test plan, test record or traceability slots. Integration of viewpoints under a viewpoint grouping is performed through an incremental, regressive strategy. As each viewpoint in the group is added any adverse affects of the integration on previously integrated viewpoints is detected. This is accomplished by progressive reexecution of test information held within formerly integrated viewpoints of the group. A functional viewpoint contains a set of test drivers and stubs necessary to individually test the functionality contained in the viewpoint. As the viewpoint is integrated with others in the grouping we progressively replace test drivers and stubs with the actual components. This is continued until all the subordinate viewpoints in the group have been integrated completing the subsystem. Integration errors, therefore, may be easily traced to the faulty component, and as the process is regressive, adverse affects on previously integrated components can be located.

4.1.2 Verification viewpoints: Verification viewpoints encapsulate domain knowledge of the product. They test from a particular perspective, in a particular style, and in a specified domain. Each verification viewpoint is assigned to a test group consisting of either multiple individuals or a single tester. As the test plan (test cases) are executed, test results are recorded in the test record slot and from this information process control metrics may be calculated.

The developers' role in the software process is also modelled using verification viewpoint instances. Developers are made responsible for producing code according to program specifications, whilst ensuring that code compiles cleanly, is free from obvious bugs, and well commented. Associated with the developer viewpoint may be a coding standards checklist to ensure compliance to company programming standards. Alternatively, the viewpoint may include some form of peer review. The developer test perspective can be seen as the VOCAL counterpart of traditional unit testing.

A specialisation of the verification viewpoint has been designed for use with interactive systems [18]. These systems tend towards unstable specifications and iteration. The model is introduced through two specialised user interface testing viewpoints, and helps fault identification while the interface is built rather than after construction. This helps to support an exploratory, incremental prototyping [7] based approach to user interface development, and V&V.

4.1.3 Validation viewpoints: Validation viewpoints are distinct from verification viewpoints as they are concerned with helping to manage reviews [10]. Reviews may be undertaken using software inspection [20–22] detecting issues in the product for resolution with relevant stakeholder quality groups (see Fig. 1). Validation viewpoints are also closely related to user interface development and evaluation. Other instances of validation viewpoints may arise during acceptance testing, front-end life-cycle, or operational development review. Hetzel [9] gives some key rules for software review:

(i) Check for missing requirements.

(ii) Simplify, and eliminate requirements that are redundant or extraneous.

(iii) Check that the solution is the right choice.

(iv) Check that the solution fulfils requirements.

Validation viewpoints may have scenarios derived from models produced during requirements engineering, and design. Scenarios can provide an invaluable source of test cases for early customer review during operational and acceptance testing (see Fig. 2). A scenario [23] is a sequence of events occurring during a particular execution of a system. For example, changing a TV channel with a remote controller (Fig. 2). Scenarios can be defined using Potts representation [24], where each scenario has a setting, narrative and evaluation slot (see Section 5.3).

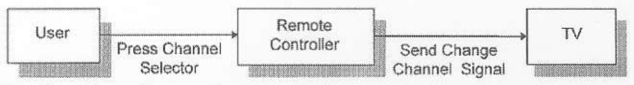


Fig. 2 *Flow of events for an example operational test scenario: selecting a channel on a TV remote controller*

The purpose of having acceptance testing and development validation viewpoints as separate entities arises from their differing aims. Development validation (user operational testing) is similar to the concept of alpha and beta testing where a user is given a set of tasks and performs these with the system. Error observation is therefore carried out under realistic conditions. This form of operational testing helps to uncover faults that only the end-user tends to find, ensuring that we produce a correct system irrespective of the requirements captured during system specification phases. Acceptance testing, on the other hand, is used to prove that a system is ready for operational use, hence ensuring that contractual requirements are adhered to. Such testing may require applying mandatory safety tests for certain systems.

4.1.4 Quality viewpoints: Quality viewpoints are non-functional attributes arising from viewpoint stakeholder sources (see Fig. 1) placing constraints on development [25]. Quality viewpoints, as well as being concerned with *product* quality attributes, may also concern *process* issues. Process quality attributes may arise as part of ISO9000 certification [26], CMM [27], or SPICE [28] accredited software process.

Non-functional requirements can be clustered around quality viewpoint perspectives (marked in light grey in Fig. 1). Clustering aids conflict resolution as a quality viewpoint can mean different things to different groups of people. For example, asking what quality means to users will more likely than not produce a different outcome than when asking a system tester the same question. A typical reply from a user could be 'a high quality system is one that conforms to my expectations and needs as a user', whereas a system tester may reply, 'a quality product is one that conforms to its specifications'. Of course, both these perspectives are valid, and with each quality viewpoint there is likely to be a different perspective to its application. Therefore, the same viewpoint may appear under more than one quality viewpoint group. For example, a customer may require performance throughput to be better than 30 transactions per second; whereas, the developer has specified performance throughput as 10 transactions per second. This conflicting information needs to be resolved during development so that it can be ensured that the product meets all stakeholders' quality constraints. Of course, this may not always be possible as groups will insist that their requirement is more important than others. We therefore require independent arbitration by the quality assurance department to ensure that a trade-off solution can be agreed between parties.

Table 2: Example quality requirements and associated metrics

Quality requirement	Quality requirement description	Example quantifiable metrics*
Architecture flexibilty (Preventative maintenance)	Software should be developed with consideration to future enhancement. It may be necessary to implement changes in a short time period to keep the product ahead of competition. This should be kept in mind during design and development.	(i) Flexibility/adaptability [30]
Interoperability	Can the system communicate with another by requesting operations to be executed, etc.; or, the effort required to link a system with another.	(i) Communication commonality [25] (ii) Data commonality [25]
Maintainability	Design and implementation of maintenance patches should be possible in a limited time period.	(i) Probability that failed system is restored to operable condition in a specified time [30] (ii) Cyclomatic complexity [2, 7, 31]
Performance	The system should be capable of meeting or exceeding a specified throughput. Algorithms taking an unacceptable amount of time to complete should be avoided. Acceptable completion times should be stated within this viewpoint.	(i) Operation throughput per second against increasing system load, e.g. FMV video frames/s; no. of transactions/s [32] (ii) Device utilisation/resource consumption per unit of work [32] (iii) Event-action response time, e.g. screen redraw rate after user operation [32]
Portability	The product should be portable to another target platform within the system domain. Platform dependant code should be avoided, or alternative code included using conditional compilation when this is not possible. Effort required to transfer code to a new platform should be less than the effort needed to totally redevelop it.	(i) % of code which contains platform dependant instructions [7, 30] (ii) No. of target systems [7]
Reliability	The product should have a defined reliability in a specified time period.	(i) Mean time to failure [7] (ii) Availability [7, 30] (iii) Rate of failure occurence [7] (iv) Probability of unavailability [7]
Reusability	Possible reusable components should be identified, and design for reusability taken into consideration.	(i) Domain/range ratio (DRR) [33] (ii) Reuse benefit [2] (iii) Generality [30]
Security	Security risks should be specified, and measures implemented to avoid breaches of security.	(i) Vulnerability [34] (ii) Attack probability [30]
Safety	Code must be failure tolerant. When a fault occurs error handlers must bring the system back to a safe state.	(i) Failure tolerance [34]
Testability	Only testable requirements should be specified. Code should be designed to be testable.	(i) Domain/range ratio (DRR) [33] (ii) Controllability [35] (iii) Observability [35] (iv) Sensitivity analysis [36]
Usability	Requirements should be formulated with user needs and ability in mind. Reviews need to check interface designs against appropriate interface style guidelines.	(i) Total training time [7] (ii) No. of available help frames [7] (iii) Learnability (time required to reach a specified level of performance [37] (iv) Throughput (speed of tasks by experienced users and number of errors made [37]

* NFRs should be testable and quantitative so that metrics can be easily applied and measured. For example, a poor nonfunctional requirement (NFR) is one that cannot be easily tested and is unquantifiable: for example, 'An experienced user of Windows95™ shall be able to operate the product with ease'.

Quality viewpoints have scoping rules applied to them indicated by specifying within the viewpoint the following types of scope:

(a) *Temporal traceability scoping*, the quality viewpoint is traceable and therefore applicable throughout a number of phases of the development life-cycle. The viewpoint may be temporally scoped to apply through specification, design, coding and acceptance testing or to apply to a subset of these depending on the specific viewpoint instance.

(b) *Subsystem specific scoping*, a quality viewpoint may only apply to specified parts of the whole system at a particular point in time. For example, usability is constrained to interaction with the user and is unconcerned with other areas of system functionality. Subsystem specific scoping rules can be summarised by the following scheme,

(i) *Unit specific quality viewpoints* apply only to one particular functional viewpoint (subsystem); for example, usability applies to the user interface subsystem.

(ii) *Group scope quality viewpoints* apply to a defined set of other functional viewpoints; for example, security applies to all components dealing with communication in a bank ATM machine.

(iii) *Global scope quality viewpoints* apply to the entire set of functional viewpoints defining the system; for example, safety applies to all components of an auto pilot system on an aircraft.

Each quality viewpoint therefore has a defined time in the life-cycle when it applies, and at this particular time, specifies the parts of the system affected by it.

Table 2 shows a set of possible metrics [25, 29, 30] for assessing product quality compliance. It should be stressed that the quality requirements are given for demonstration purposes only and are in no way generic for all systems. The Table though could feasibly be used as a starting point to develop specific quality viewpoints, reusing some of the requirements presented in the table.

Associated with the quality viewpoints is a template concerned with VOCAL deployment and process issues (see Fig. 1). This viewpoint contains slots for inter- and intraviewpoint consistency checking. Intraviewpoint checks added here may include test coverage checking and traceability validation. Checks are also required to verify consistency and completeness of external traceability, and also to ensure test result consistency with related viewpoints. It is also important to ensure that quality measures accurately represent the attributes they purport to quantify [38]. The above process checks are amalgamated into the VOCAL deployment viewpoint (see Section 4.2, describing a sample VOCAL deployment checklist).

4.1.5 Viewpoint concern overlap and inconsistency resolution: The test plan for a system developed with VOCAL consists of the set of identified viewpoints. Viewpoints may not be disjoint, and it is often the case that there is some degree of overlap between, say, quality viewpoints and functional system viewpoints. This may be seen as an advantage when testing, as we can determine if a quantified quality requirement is too strict because the actual measured functional performance in the verification viewpoint shows that the constraint cannot be attained. Therefore, overlap can help to ensure and maintain viewpoint consistency. Detected inconsistencies need to be traded off between viewpoints where they occur (i.e. those to which they are traceable). To aid management of trade-offs between faults, VOCAL supports a three point defect classification scheme where less critical faults can be traded against those of a more crucial nature. Severity of faults may be assigned as follows:

(i) *Critical*, a serious corrupting fault having the potential to threaten the whole project.

(ii) *Major*, a serious corrupting fault in a localised area.

(iii) *Minor*, a non-corrupting fault that is annoying rather than threatening, and does not adversely affect system operation.

All overlapping and potentially conflicting test plans are defined by including traceability between the viewpoints. This means having the *sink* (target viewpoint) of the overlap defined in the *source* viewpoint's traceability slot, and *vice versa*. Traceability is therefore bidirectional. Traceability can be defined as [39], '... the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use), and through periods of on-going refinement and iteration in any of these phases'. Bidirectional viewpoint traceability makes such cross-referencing possible in VOCAL.

4.2 Viewpoint identification and deployment

Viewpoint identification is primarily determined by the class of system under development (i.e. interactive, real time, safety critical, multimedia, embedded etc.). We have developed a checklist for viewpoint identification and deployment as follows:

(i) Work through the VOCAL viewpoint hierarchy (Fig. 1) in a top down manner, pruning and eliminating viewpoints that have no relevance to the system under examination.

(ii) If any viewpoints are immediately apparent from domain descriptions that are not present after stage 1, add these to the hierarchy under the appropriate viewpoint group in a bottom up manner.

(iii) Identify, and assemble all agents/participants/actors/stakeholders, and associate them with the appropriate viewpoint identified in the first two stages (i.e. associate them with viewpoints in which they have an active concern in terms of authoring and use).

(iv) Map test requirements onto viewpoints, identifying test information according to the viewpoint structure.

(v) Identify relationships (bidirectional traceability) between viewpoints. Remove obvious conflicts of interests along the way. Regularly check viewpoint consistency throughout development.

(vi) Start test activities at the front-end of the life-cycle. Kick off with specification and design reviews. Make sure that temporal traceability is established, especially with the NFRs.

(vii) Continue V&V activities as categorised in the test plan. Make sure that authors who implement change requests follow all traceable links so that each affected test plan is updated and retested as a result of the change. Integrate the system according to the viewpoint structure.

(viii) End with customer acceptance, and hand-over. Identify for reuse any viewpoints or templates that may be useful in the future. Store these in a library so that they can be easily accessed.

Step (iii) associates participants to viewpoints (i.e. maps agents onto roles). It should be noted that this is not performed in an arbitrary manner. Testers exercise their skill based on experience [40]. Therefore, participants should be mapped onto viewpoints with the aim of reducing pretesting costs (e.g. training), thereby choosing the 'right person for the job'.

5 Example: VOCAL application to a bank ATM system

To illustrate the VOCAL process of Section 4.2 we describe a simplified example where testing viewpoints are applied to the V&V of a bank ATM (automated teller machine) [23, 41]. The ATM is a classic example of an embedded system where software within the machine drives cash counting, card reading and receipt printing hardware, etc. ATMs also communicate externally with a central bank database. Typical operations provided by an ATM include, cash withdrawal, statement and chequebook reordering, changing a customer's PIN (personal identification number), etc. The ATM may also have to deal with different classes of customer, each of whom sees a different view of the total functionality. Foreign customers (those without

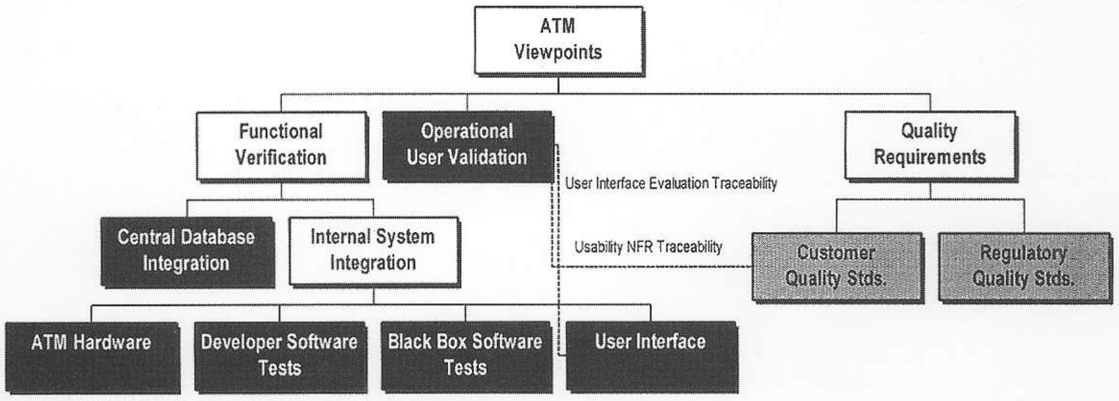
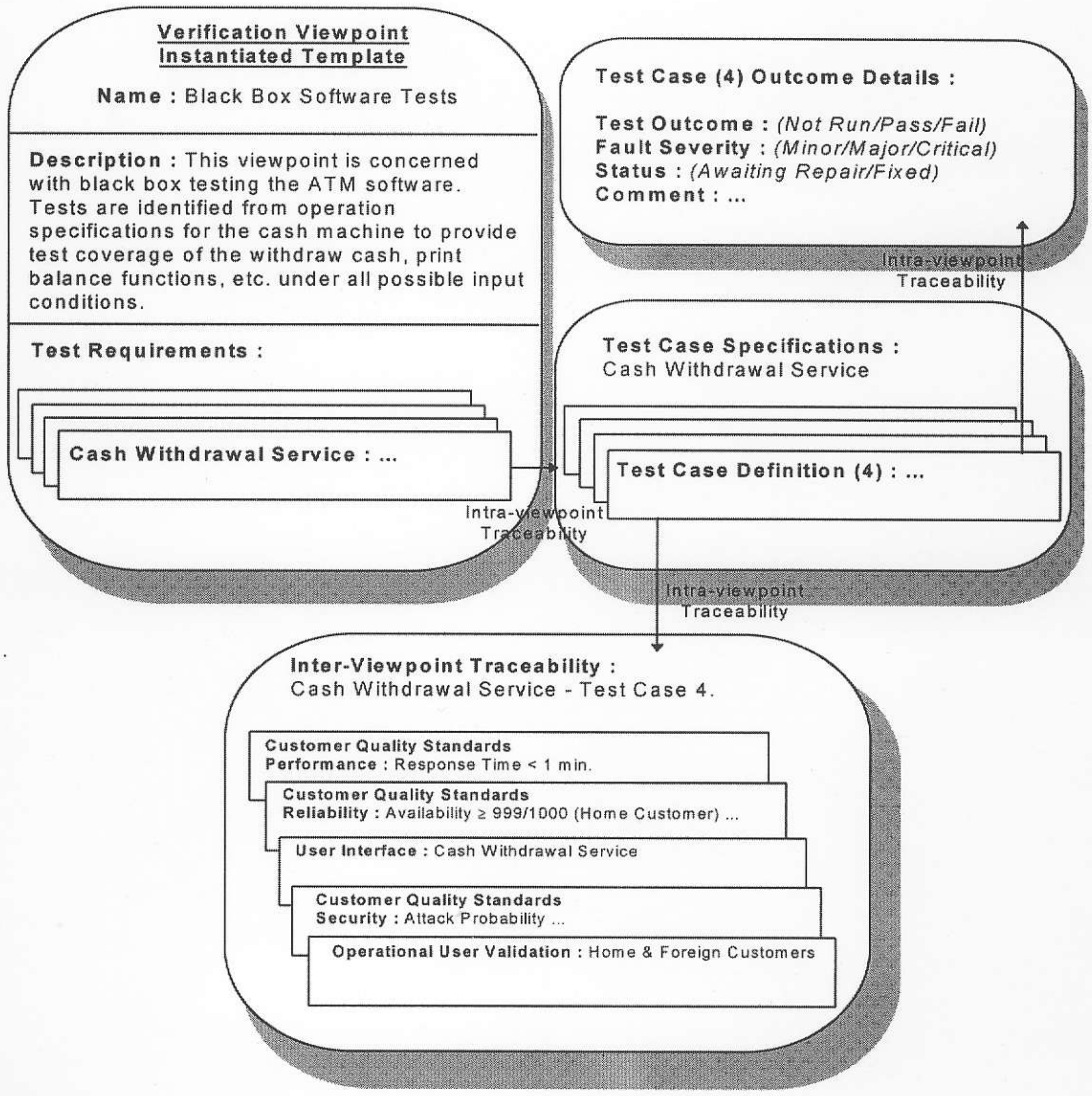


Fig.3 *Simplified VOCAL viewpoint structure for the ATM system*



Verification Viewpoint Instantiated Template
Name : Black Box Software Tests
Description : This viewpoint is concerned with black box testing the ATM software. Tests are identified from operation specifications for the cash machine to provide test coverage of the withdraw cash, print balance functions, etc. under all possible input conditions.
Test Requirements :
Cash Withdrawal Service : ...
Intra-viewpoint Traceability
Test Case (4) Outcome Details :
Test Outcome : (Not Run/Pass/Fail)
Fault Severity : (Minor/Major/Critical)
Status : (Awaiting Repair/Fixed)
Comment : ...
Intra-viewpoint Traceability
Test Case Specifications : Cash Withdrawal Service
Test Case Definition (4) : ...
Intra-viewpoint Traceability
Inter-Viewpoint Traceability : Cash Withdrawal Service - Test Case 4.
Customer Quality Standards Performance : Response Time < 1 min.
Customer Quality Standards Reliability : Availability ≥ 999/1000 (Home Customer) ...
User Interface : Cash Withdrawal Service
Customer Quality Standards Security : Attack Probability ...
Operational User Validation : Home & Foreign Customers

Fig.4 *ATM black box software test viewpoint decomposition*

an account at the bank) only see a subset of the facilities available to home customers.

5.1 VOCAL application to the ATM

Following the rules of Section 4.2, we use the generic viewpoint template of Fig. 1 to derive the structure of the ATM test plan. This produces a testing structure as shown in Fig. 3. Note that we have simplified this by excluding any front-end review, process and company quality standards, although in practice they may apply to such a system. The dotted lines in the Figure indicate usability traceability between nonfunctional quality requirements and operational user validation. Conflicts of concerns between these two viewpoints must be managed and resolved. It should also be noted how traceability indicates the close relationship between user interface verification and operational validation [18].

Assuming a complete test structure (i.e. completed steps (i) and (ii) of Section 4.2), and skipping step (iii) as it is unimportant for demonstration purposes, we can now move onto identification of test requirements for each test viewpoint pictured in black in Fig. 3. This is illustrated by expanding one test requirement (cash withdrawal) associated with the ATM 'black box software tests' viewpoint (Fig. 4). The test requirement is traceable to many other viewpoints in the ATM system and should provide enough diversity to represent the main concepts of the technique. Additionally, an operational test scenario belonging to the user validation viewpoint will be described to provide stakeholder evaluation coverage of the cash withdrawal functionality.

5.2 Test specification: Verification of the cash withdrawal service

Step (iv) of VOCAL involves the identification of test cases from test requirements. In the case of cash withdrawal, this was specified using an event-scenario model based on the notation proposed by Rumbaugh [23]. It is from this suitably formalised specification that the cash withdrawal service's test cases in the black box software test viewpoint can be identified. The model includes all exceptions that may arise from this service. Testing must ensure that the correct event type is triggered under the proper set of conditions, and that exception handlers perform the appropriate action to bring the ATM back to a valid state when an error occurs. Preconditions that must hold are associated with transitions in the event-scenario model, hence if a precondition is false there can be no transition. An event is triggered on each state traversal that performs some processing in the ATM.

Note that stubs are used for yet to be integrated functions. The notation *viewpoint_object::operation* is used to indicate the action to be taken by the stub component. Stub emulation is required until integration with the ATM hardware and external systems. Using this specification, test cases for the cash withdrawal service include:

Viewpoint	Black Box Software Tests
Test requirements	Functional tests for the cash withdrawal service's functionality.

Cash withdrawal service: Test case specification 1

Test requirement	ATM must reject all expired bank cards.
Initial state	Idle ATM machine
Input sequence	Insert an expired bank card
Precondition	[(*Central_Database::get_expiry* (*ATM_Hardware::cardID*()) ≥ *ATM_Hardware::date*())]
Expected action	*ATM_Hardware::eject_card*()
Final state	Idle ATM machine

Cash withdrawal service: Test case specification 2

Test requirement	Vend cash when presented with an unexpired bank card, PIN and valid cash amount, with sufficient funds available in the customer's account and ATM safe.
Initial state	Idle ATM machine
Input sequence	Insert an unexpired bank card **then** Input valid PIN number **then** Select cash withdrawal **then** Select cash amount
Precondition	[(*Central_Database::get_expiry* (*ATM_Hardware::cardID*()) < *ATM_Hardware::date*()) && (*ATM_Hardware::get_PIN*() = *Central_Database::get_PIN*(*ATM_Hardware::cardID*()) && (*ATM_Hardware::get_amount*() ≤ *Central_Database::customer_balance* (*ATM_Hardware::cardID*()) && (*ATM_Hardware::get_amount*() ≤ ATM_FUNDS) && (*ATM_Hardware::get_amount*() **mod** MIN_VEND_AMOUNT = 0)]
Expected action	*ATM_Hardware::eject_card*() *ATM_Hardware::dispense_cash* (*ATM_Hardware::get_amount*()) *Central_Database::update_account* (*ATM_Hardware::get_amount*(), *ATM_Hardware::cardID*())
Final state	Idle ATM machine

For brevity, we have not presented total functional coverage of the withdraw cash service. Each test specification represents an equivalence class of black box tests. Additionally, test cases may be added at boundaries of each equivalence class. For test specification 2, such tests may include:

(i) Entry of a cash amount of £0.

(ii) A cash amount of, *balance* – (*balance* **mod** *MIN_VEND_AMOUNT*), (i.e. a withdrawal of the maximum possible amount of cash from the owner's account).

(iii) A value somewhere between these two, i.e. testing the 'normal' case.

5.3 Test specification: Validation of the cash withdrawal service

This Section outlines an operational test scenario for stakeholder evaluation of the cash withdrawal service. This test case will be contained in the operational user validation viewpoint's test cases, and is specified as,

Viewpoint	Operational user validation
Test requirement	Operational user evaluation of the cash withdrawal service's functionality.

Scenario A home customer requires cash from the ATM

Episode 1

Goal	Reading the customer's card ID number.
Action	ATM displays insert card and welcome message.
Action	Customer responds by inserting card into card reader.
Outcome	Customer's card ID number is read by the ATM. The ATM rejects expired, or unreadable cards.

Episode 2

Goal	Customer identity validation.
Action	ATM displays a message prompting the user for their PIN.
Action	The customer types in their numeric PIN.
Outcome	The ATM checks with the central bank database the PIN for the card. If the PIN has been incorrectly entered three consecutive times that day, the card is retained by the ATM and the transaction terminated.

Episode 3

Goal	Service and withdrawal amount selection.
Action	The ATM displays a menu of available services.
Action	The user selects cash withdrawal.
Action	The ATM displays an option allowing the user to enter an amount on the numeric keypad.
Action	The customer selects an amount.
Outcome	If the customer has sufficient funds, and the amount selected is available in the ATM, then the card is ejected, cash counted, dispensed, and the account updated with the amount withdrawn.

Evaluation

Issue	**Design fault**, the customer requires a record of the transaction. Incorporation of a cash with receipt option is required.
Issue	**Design fault**, the customer would like an option to withdraw a standard amount instead of entering it each time on the numeric keypad via the 'other amount' menu option. This would save key-presses, time at the machine, and mistakes therefore increasing throughput and reducing queues.
Issue	**User interface design fault**, the menu for cash withdrawal amount selection does not display available denominations of note. The user therefore is not warned in advance of the minimum amount of cash that can be withdrawn from the machine, or the amount that their sum must be divisible by.

This scenario then becomes the test plan for customer evaluation of the service.

So far the example has illustrated use of multiple test case representation forms found in VOCAL viewpoints. Scenarios are defined to represent ATM actions and interactions between user and machine. The scenario describes a task that, if successfully completed, must achieve all identified subgoals. Issues arising from this interaction are evaluated as part of customer appraisal. Conversely, the black box test representation, defines functional test cases carried out during system testing. This requires a different representation to reveal faults arising from nonconformance with the specification.

5.4 Traceability identification

Step (v) of Section 4.2 necessitates traceability identification between viewpoints. Test cases dealing with system testing of the ATM cash withdrawal functionality in the black box software test viewpoint are inter traceable to the following viewpoints (see traceability section of Fig. 4):

(i) *Customer quality standards viewpoint (performance requirement)* in terms of transaction response time (specified as ≤ 1 min). This quality requirement would normally be included in the test specification for cash withdrawal after integration with ATM hardware as this is the point where actual end system performance can be realistically measured. Adding temporal scope (through life-cycle) traceability to performance aspects of the product makes all testers, even those working before hardware integration, aware of the intended performance. Any doubts over final projected performance can be rectified as soon as possible. Performance will be held back by the slowest component in the system. Therefore performance traceability is of the type *global* (applies to all subsystems) *temporal* (throughout development) *interviewpoint traceability*.

(ii) *Customer quality standards viewpoint (reliability requirement)* has two definitions one for the home customer, the other for foreign customers. Reliability is specified in terms of availability, which for home customers is $\geq 999/1000$ transactions will succeed, for foreign customers this is $\geq 900/1000$. These two levels apply to the same functionality but under different circumstances, therefore this is not a conflict. The traceable link between cash withdrawal, and reliability requirements is part of the reliability viewpoint's *global temporal interviewpoint traceability*. Traceability between reliability and other functional viewpoints is declared in this manner for the early detection of reliability problems, as is the case with the performance viewpoint (above).

(iii) *Customer quality standards viewpoint (security requirement)* is a global, and whole lifecycle applicable rule concerning the application of security inspections to all systems throughout all development activities. The minimisation of security risks has therefore been given a *global temporal interviewpoint* traceability scope, therefore making security traceable to the functional testing of the cash withdrawal operation.

(iv) *User interface viewpoint* concerns verification and integration with the 'front-end' of the ATM machine, that is code providing the user interface for the cash withdrawal service. Eventually the user interface will be linked with the service provision software. Traceability between these two viewpoints is important as it allows any changes made to the cash withdrawal service's implementation to be reflected in the way the user interface operates, and is tested. The traceability is therefore defined as an *interviewpoint* link between the cash withdrawal service's user interface, and the code implementing the service. The link has no temporal scope as it only applies during the development phase where operational validation activities may cause the service's specification to change (see scenario evaluation of the cash withdrawal service in Section 5.3).

(v) *Operational user validation viewpoint (home and foreign customer evaluation requirements)*. Test case specifications, especially their general descriptions can help formulate operational test scenarios for cash withdrawal. It is particularly important with an ATM machine operated by casual users to validate that the machine is providing the required services. The ATM must provide these services in a manner that users understand and are comfortable with. Therefore, inter-viewpoint traceability between functional test case specifications and operational validation scenarios help to ensure that functional service operation has sufficient user validation.

The scenario test case description for cash withdrawal from the operational user validation viewpoint (see Section 5.3) is traceable to the following:

(i) *Functional verification viewpoint group*, especially in terms of evaluating services provided by the user interface. Traceable item (v) (see above) is reverse traceable to this viewpoint according to the rules of VOCAL. Therefore bidirectionality for item (v) is retained with the functionality concerning cash withdrawal.

(ii) *Quality requirements viewpoints group* assess the results of user evaluation. Evaluation assessment requires measurement [29]. Measurement is provided by quality metrics to evaluate the users' aspect of product quality, quantifying the evaluation. Important quality aspects to consider (to be made inter traceable) from the operational user validation viewpoint are, performance and usability assessment.

Note that traceability is always bidirectional thus the sink viewpoint, in each of the above examples is always traceable back to the source and *vice versa*.

5.5 VOCAL ATM test plan execution

Arriving at step (vi) of the process puts the test plan into action. Ignoring front-end reviews for the sake of brevity we start at the child viewpoint nodes of the test plan and work our way toward the root whilst integrating subviewpoints as we proceed.

As stated in step (vii), change requests require management and update of all traceable viewpoints. For example, changing the structure of the ATM withdraw cash software may require an associated change in the user interface viewpoint. Finally, we hand over for customer acceptance. This is not the end of our process though; it is important to identify viewpoints or templates that have reusable potential. As an example consider interoperability. Interoperability concerns tests to ensure readability of a foreign customer's bank card, and ATM message exchange capability with foreign banks, etc. Such a viewpoint has a high likelihood of being useful in future development, therefore it should be stored as part of the organisation's quality manual.

6 Conclusion

Utilisation of viewpoints in software engineering acknowledges the roles played by the many individuals in the software process. This paper has shown the historical development of viewpoints from a requirements elicitation technique, through to a method for verification, validation, and quality attribute identification and assessment. Information encapsulation of viewpoints in terms of participants, method, and work activity, has also been outlined. It is around this encapsulation that the VOCAL framework is built to apply the benefits of multiperspective viewpoint methodologies to verification, validation, and quality assessment processes.

The deliberate exclusion of prespecified methods for testing helps integrate the methodology with specific company working practices. VOCAL guides and identifies V&V throughout the life-cycle providing an organisational framework for controlled test distribution and activity assignment among participants. This was deemed a more appropriate route to follow rather than trying to convey an unwieldy test strategy that constricts the applicability and real-life value of the method. Using VOCAL can lead to much greater test coverage where it matters in a system. Traditional test methods fail to identify and verify attributes that constitute a system that not only does its job correctly, but in a manner the customer wants. VOCAL addresses this problem directly by including stakeholders in the process wherever possible throughout development. Stakeholders are in regular contact through front-end reviewing, and operational product validation via scenarios.

Emphasis has been given to the importance of the three types of traceability (inter-, intra- and temporal traceability) for conflict management and resolution of product attributes. Conflicts should be detected as early as possible in development to save later correction effort and unnecessary cost.

VOCAL approaches quality assurance and how this integrates with V&V process activities by using viewpoint stakeholder structuring methods. Attribute identification, conflict resolution, and example quality requirements have been described to aid this process. Stakeholder quality structuring assists nonfunctional requirements verification, or if the requirement is not feasible, its resolution to a satisfactory level. The importance of making quality attributes quantifiable is also stressed. Finally, to illustrate the VOCAL method a simplified test plan using a bank ATM system was developed.

Weaknesses of the method that need to be addressed arise from the information overhead created by viewpoint encapsulation. This, coupled with defining full traceability, can place a greater focus on information management than with traditional techniques. A prototype test case browser tool supporting VOCAL has been implemented to aid efficient application of the technique [18], and relieve some of the information management overhead. In-process test management is also at an early stage of development in VOCAL. Implementing a well defined test management model helps administer resources to areas of the project requiring most effort to keep the product on schedule. Process management information collection is currently being integrated in the tool to give visual indications of problematic areas in the test plan. Defect tracking is also an integral part of test management. Again prototype support for this has been implemented to further research these issues. Current investigation is being undertaken using tool generated defect tracking information as a test exit criterion to determine the point where test costs outweigh benefits in terms of quality improvement. Currently the approach is undergoing evaluation to assess validity and practicality of the method. This further research is obviously crucial if the approach is to be taken up in favour of more established and mature methods.

7 Acknowledgments

This work is funded by an EPSRC CASE award in collaboration with the Software Engineering and Applications Group, Philips Research Labs., Cross Oak Lane, Redhill, Surrey RH1 5HA, England.

8 References

1 MÖLLER, K.H., and PAULISH, D.J.: 'Software metrics, a practitioner's guide to improved product development' (Chapman & Hall, 1993)
2 DEVANBU, P., KARSTU, S., MELO, W., and THOMAS, W.: 'Analytical and empirical evaluation of software reuse metrics'. Proceedings of the 18th international conference on *Software engineering*, Berlin, Germany, July 1995, pp. 189–199
3 FINKELSTEIN, A., KRAMER, J., and GOEDICKE, M.: 'Viewpoint oriented software development'. Proceedings of third international workshop on *Software engineering and its applications*, Toulouse, France, December 1990, pp. 337–351
4 FINKELSTEIN, A., KRAMER, J., NUSEIBEH, B., FINKELSTEIN, L., and GOEDICKE, M.: 'Viewpoints : a framework for integrating multiple perspectives in system development', *Int. J. Softw. Eng. Knowledge Process.*, 1992, **2**, (1), pp. 31–58
5 FINKELSTEIN, A., and SOMMERVILLE, I.: 'The viewpoints FAQ', *Softw. Eng. J.*, 1996, **11**, (1), pp. 2–4
6 MULLERY, G.P.: 'CORE – A method for controlled requirements specification'. Proceedings of the 4th IEEE Computer Society international conference on *Software engineering*, Munich, Germany, 1979, pp. 126–135
7 SOMMERVILLE, I.: 'Software engineering' (Addison-Wesley/Longman, 1996, 5th edn.)
8 MYERS, G.: 'The art of software testing' (Wiley Interscience, 1979)
9 HETZEL, W.C.: 'The complete guide to software testing' (QED Information Sciences, 1988, 2nd edn.)
10 PERRY, W.E.: 'Effective methods for software testing' (John Wiley & Sons, 1995)
11 BURNSTEIN, I., SUWANASSART, T., and CARLSON, R.: 'Developing a testing maturity model for software test process evaluation and improvement'. International Test conference, ITC 1996, Washington, DC, Oct. 1996, pp. 581–589
12 BOEHM, B.W.: 'Software engineering economics' (Prentice-Hall, 1981)
13 BOEHM, B.: 'Models and metrics for software management and engineering' (IEEE Computer Society Press, 1984)
14 BOEHM, B.: 'Industrial software metrics top 10 list', *IEEE Softw.*, 1987, **4**, (5), pp. 4–9
15 RHENMAN, E.: 'Industrial democracy and industrial management' (Tavistock, London, 1965), p. 25
16 BEIZER, B.: 'Software testing techniques' (Van Nostrand Reinhold, 1990, 2nd edn.)
17 ROPER, M.: 'Software testing' (McGraw-Hill International Quality Assurance Series, 1994)
18 PEMBERTON, D.: 'VOCAL interactive system & tool support'. CSEG Technical Report, Computing Dept., Lancaster University, Lancaster, UK, LA1 4YR
19 NUSEIBEH, B., and FINKELSTEIN, A.: 'Viewpoints : A vehicle for method and tool integration'. IEEE Proceedings of international workshop on *CASE* CASE'92, Montreal, Canada, July 1992, pp. 50–60
20 FAGAN, M.E.: 'Design and code inspections to reduce errors in program development', *IBM Syst. J.*, 1976, **3**, (15), pp. 182–211
21 FAGAN, M.E.: 'Advances in software inspections', *IEEE Trans. Softw. Eng.*, 1986, **7**, (12), pp. 744–751
22 GILB, T., and GRAHAM, D.: 'Software inspection' (Addison-Wesley, 1993)
23 RUMBAUGH, J., BLAHA, M., PREMERLANI, W., and LORENSEN, W.: 'Object-oriented modelling and design' (Prentice-Hall International Editions, 1991)
24 POTTS, C.: 'Using schematic scenarios to understand user needs'. Symposium on *Designing interactive systems: Processes, practices, methods & techniques*, DIS '95, University of Michigan, USA, 1995, pp. 247–256
25 KITCHENHAM, B.: 'Towards a constructive quality model', *Softw. Eng. J.*, 1987, **2**, (4), pp. 105–113
26 SADGROVE, K.: 'ISO9000/BS5750 made easy : A practical guide to quality' (Kogan Page, 1994)
27 PAULK, M.C., CURTIS, B., CHRISSIS, M.B., and WEBER, C.V.: 'The capability maturity model for software', *IEEE Softw.*, 1993, **10**, (4), pp. 18–27
28 KONRAD, M.D., and PAULK, M.C.: 'An overview of SPICE's model for process management'. Proceedings of the 5th international conference on *Software quality*, Austin, TX, October 1995, pp. 291–301
29 FENTON, N.E.: 'Software metrics, a rigorous approach' (Chapman & Hall, 1991)
30 GILB, T.: 'Software metrics' (Chartwell-Bratt, 1976)
31 MCCABE, T.J.: 'A complexity measure', *IEEE Trans. Softw. Eng.*, 1976, **2**, (4), pp. 308–320
32 PERLIS, A.J., SAYWARD, F.G., and SHAW, S. (Eds.): 'Software metrics' (MIT Press, 1981)
33 VOAS, J.M., and MILLER, K.W.: 'Semantic metrics for software testability', *J. Syst. Softw.*, 1993, **20**, (3), pp. 207–216
34 VOAS, J.: 'Testing software for characteristics other than correctness: Safety, failure tolerance, and security'. Proceedings of the international conference on *Testing computer software*, 1996
35 BINDER, R.V.: 'Design for testability in object-oriented systems', *Commun. ACM*, 1994, **37**, (9), pp. 87–101
36 VOAS, J.M., MORELL, L., and MILLER, K.W.: 'Predicting where faults can hide from testing', *IEEE Softw.*, 1991, **8**, (2), pp. 41–48
37 PREECE, J.A. (Ed.): 'A guide to usability, human factors in computing' (Addison-Wesley, 1993)
38 KITCHENHAM, B., and PFLEEGER, S.L.: 'Towards a framework for software measurement validation', *IEEE Trans. Softw. Eng.*, **21**, (12), pp. 929–943
39 GOTEL, O., and FINKELSTEIN, A.: 'An analysis of the requirements traceability problem'. Proceedings of 1st international conference on *Requirements engineering*, April 1994, pp. 94–101
40 CORNELISSEN, W., KLAASEN, A., MATSINGER, A., and VAN WEE, G.: 'How to make intuitive testing more systematic', *IEEE Softw.*, 1995, **5**, (12), pp. 87–89
41 KOTONYA, G., and SOMMERVILLE, I.: 'Requirements engineering with viewpoints', *Softw. Eng. J.*,, **11**, (1), pp. 5–18