Presentation
Paper
Bio

Return to Main Menu

P R E S E N T A T I O N

W12

Wednesday, Dec 6, 2000

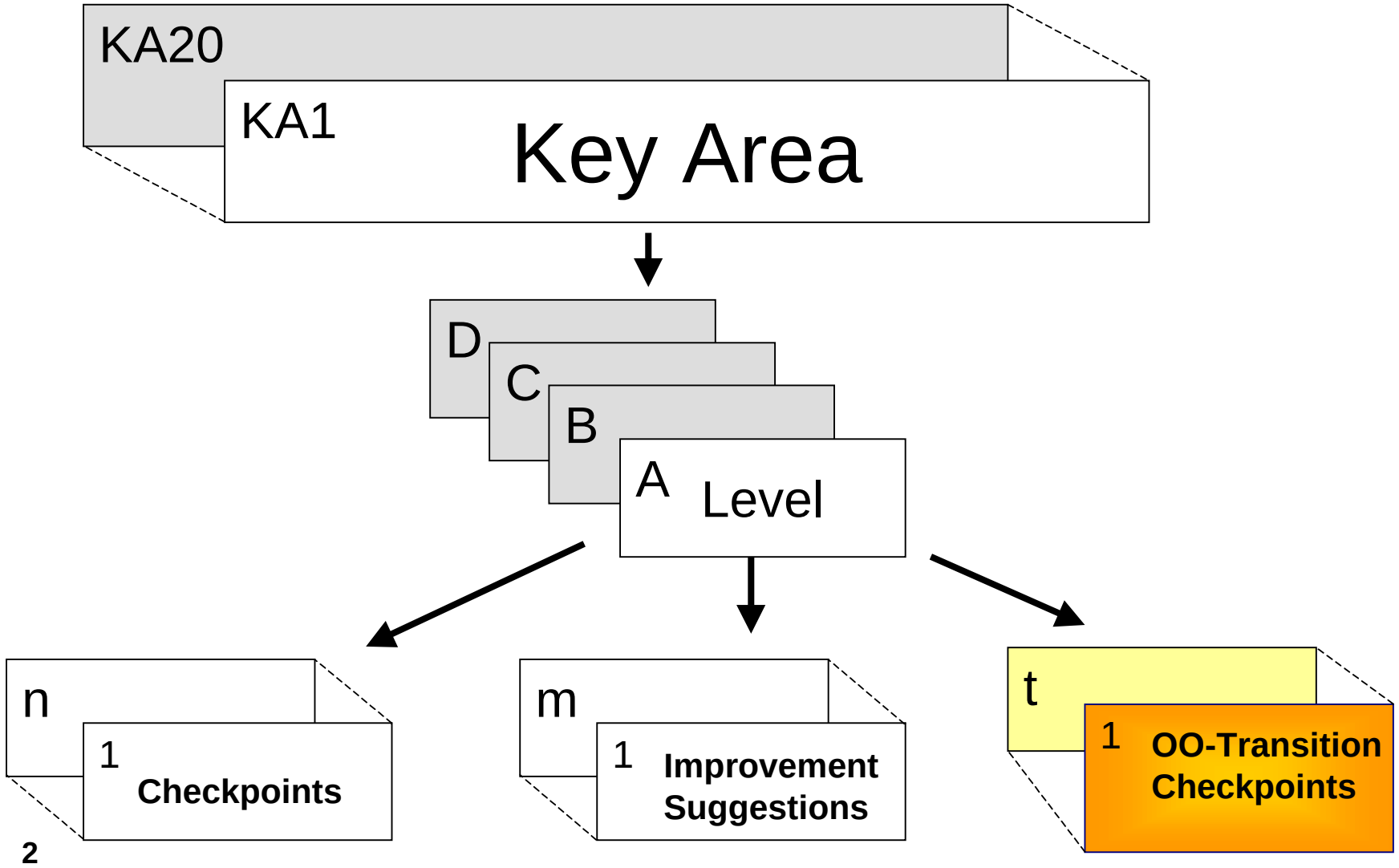
Going “00” : A Model for Test Process Transformation

Graham Bath

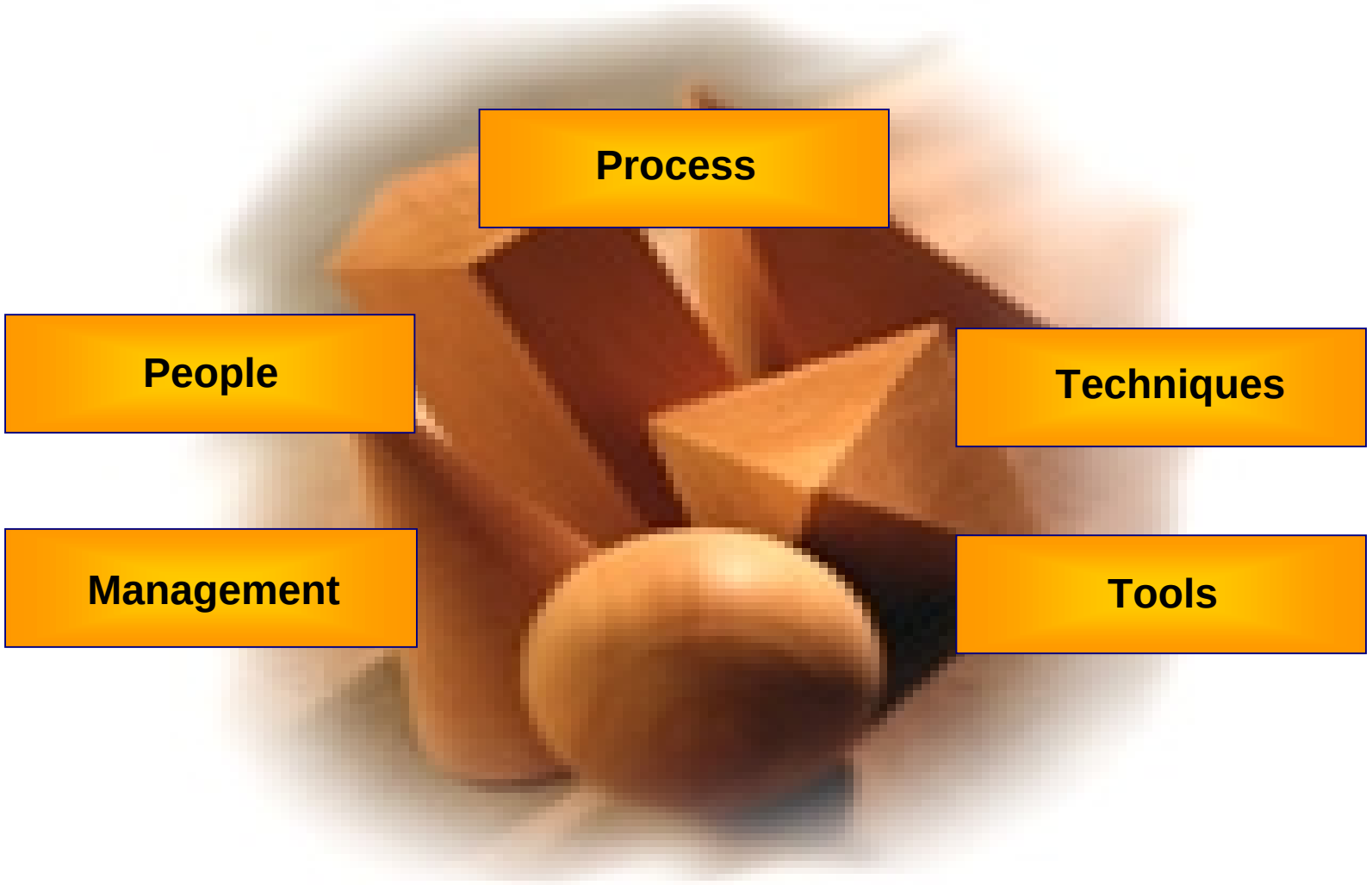
International Conference On
Software Testing, Analysis & Review
DEC 4-8, 2000 • COPENHAGEN, DENMARK


Going "OO"
A Model for Test
Process Transition
Graham Bath
debis Systemhaus

The TPI-Model with extension for OO-Transition



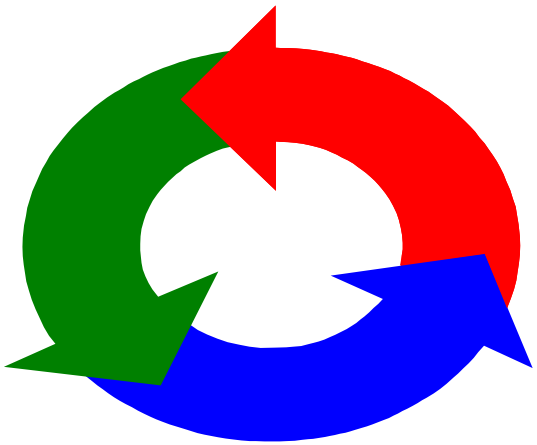
Groups of Key Areas



Process

KA1 Test Strategy

KA2 Life-Cycle Model



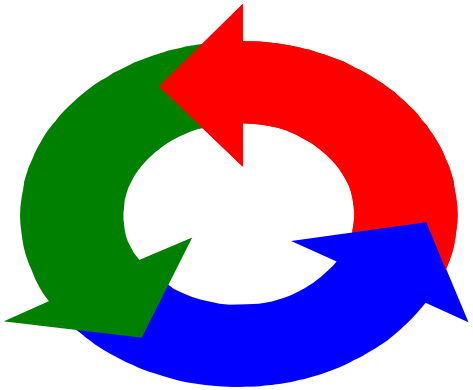
KA15 Reporting

KA3 Moment of Involvement

KA7 Metrics

KA4 Estimating & Planning

Process



KA1
Test Strategy

Checkpoints - Level 1

- Non-functional quality characteristics are adequately covered in the test strategy
- Risk assessment takes OO-Factors into account, particularly when planning a test strategy which includes low-level (class) tests.

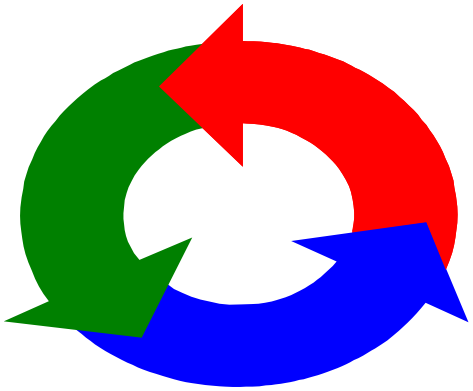
Checkpoints - Level 2

- The test strategy is coordinated across the (larger number of) testing levels such that test effectivity is optimised
- The test strategy calls for the involvement and coordination of skilled personel who are not allocated to QS.

Checkpoint - Level 3

- High-Level test activities include the testing of (potential) business objects

Process



KA2
Life-Cycle Model

Checkpoints - Level 1

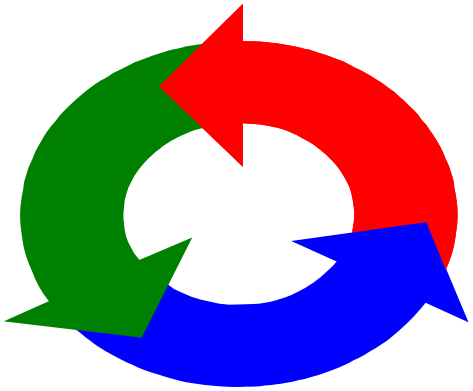
- The Life-Cycle Model is adapted to an iterative development process
- For E-Business applications, the Life-Cycle Model considers in particular the „continuous testing“ strategy required to meet changing user behaviour.

KA3
Moment of Involvement

Checkpoint - Level 1

- Issues of test tooling and test strategy are taken at project initiation

Process



KA4 Estimating & Planning

Checkpoint - Level 1

- The planning of tests is coordinated with other departments and projects to ensure the availability of all necessary skills.

KA7 Metrics

Checkpoint - Level 1

- Specific OO-Metrics are taken and provide input to the risk assessment process.

KA15 Reporting

Checkpoint - Level 1

- Quality characteristics such as scalability, compatability and performance are reported.

Techniques

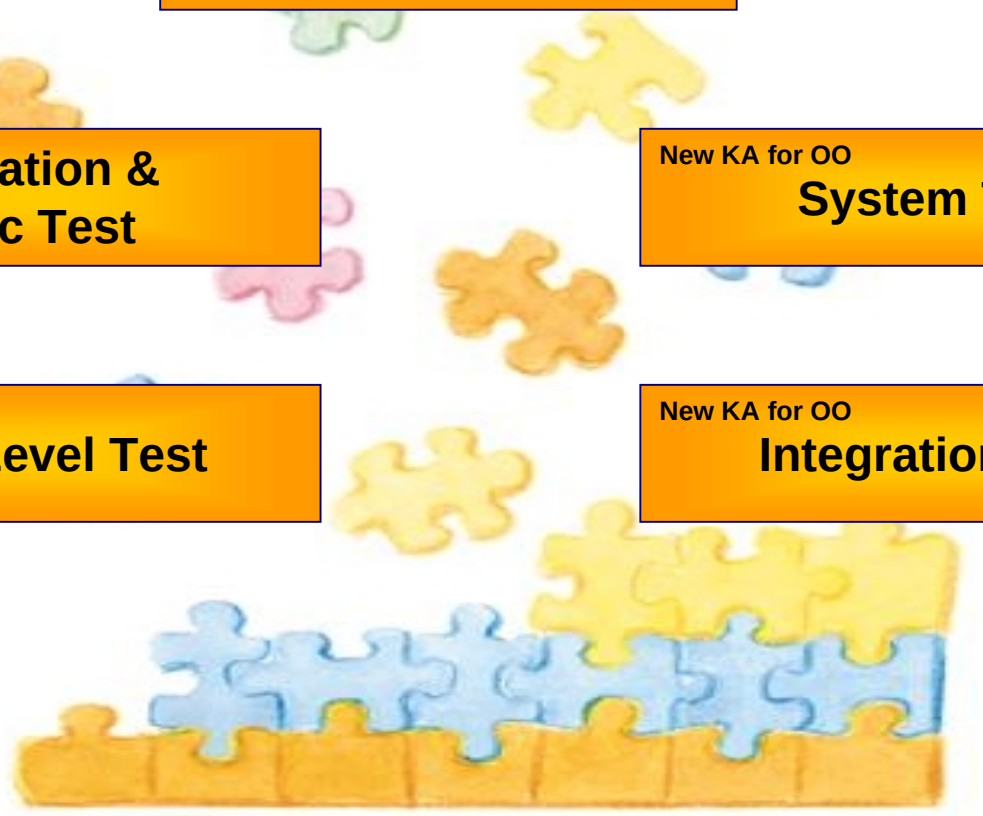
KA5
Testspecification

KA6, KA19
Evaluation & Static Test

New KA for OO
System Test

KA20
Low-Level Test

New KA for OO
Integration Test



Techniques

KA5
Testspecification

Checkpoints - Level 1

- Test cases are derived in a formal manner using techniques which are specific to OO-testing.
- OO-Design documentation includes scenario descriptions which are an adequate basis for functional (system) test specification

Checkpoint - Level 2

- „Load Profiles“ are specified which adequately adress the needs of of the system prior to and after production launch.

Techniques

KA6, KA19 **Evaluation & Static Test**

Checkpoints - Level 1

- Inspection techniques and checklists exist which address the non-sequential nature of OO-Code.
- The inspection of Design models (e.g. Class Diagrams, Sequence Diagrams in UML) are inspected in an interactive manner together with an experienced designer.

Checkpoint - Level 2

- Static Analysis of code is performed to enforce coding standards and detect typical OO-programming errors.

Techniques

KA20
Low-Level Test

Checkpoints - Level 1

- Functional (black-box) test cases take into account the pre-conditions, post-conditions and invariants of a class.
  (i.e. „Design by Contract“)

- Techniques for structural (white-box) testing do not focus on achieving traditional coverage percentages.

Checkpoint - Level 2

- Techniques exist for testing the interaction of methods within a class.

Techniques

New KA for OO

Integration Test

Checkpoints - Level 1

- Techniques exist for generating test cases for class inheritance hierarchies.
- Techniques exist for generating test cases for simple interactions between objects at all levels.

Checkpoint - Level 2

- Techniques exist for generating test cases for complex interactions between objects at all test levels.

Checkpoint - Level 3

- A consistent decision can be made on the need to change/retest sub-classes within a class hierarchy.

Techniques

New KA for OO

System Test

Checkpoints - Level 1

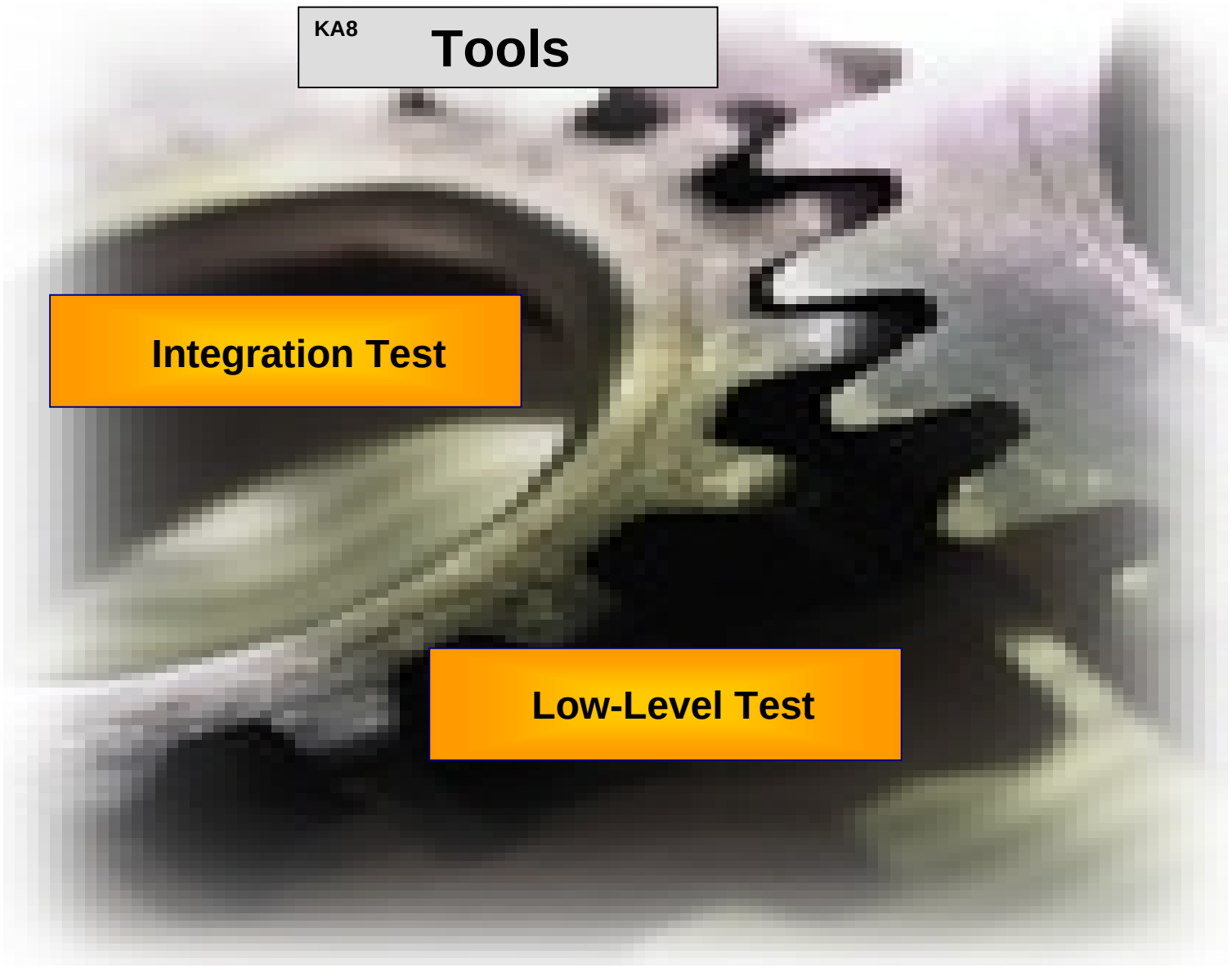
- User Scenarios defined in high-level design documentation (e.g. as „Use-Cases“) are used as the basis for test case design for system test.
- Techniques are available for determining test cases relating to the Graphical User Interface (if defined)

Checkpoint - Level 2

- Techniques exist for defining typical user profiles for use in technical system tests (e.g. for performance, load and stress tests).

KA8

Tools



Tools

Low-Level Test

Checkpoint - Level 1

- Tools are employed to support class testing, where this is part of the test strategy.

Checkpoint - Level 2

- Tools are employed to support the OO-specific dynamic analysis called for in the test strategy, for example:
  - performance „profiling“
  - thread-monitoring
  - exception-handling
  - memory usage

Tools

Integration Test (1)

Checkpoints - Level 1

- Tools are employed to support the testing of class inheritance hierarchies, where this is part of the test strategy.

- Tools for functional client-based testing (GUI-tools) are employed when
  - a large number of Platform/OS/Browser combinations must be tested, or
  - the Life-Cycle Model calls for many test iterations, or
  - the application is expected to be frequently updated or changed after deployment.

- Tools for „Load & Performance“ testing are used when
  - the number of client users is more than 10 - 15, or
  - system scalability is of critical importance, or
  - the system architecture is complex, or
  - self-developed middleware components are used

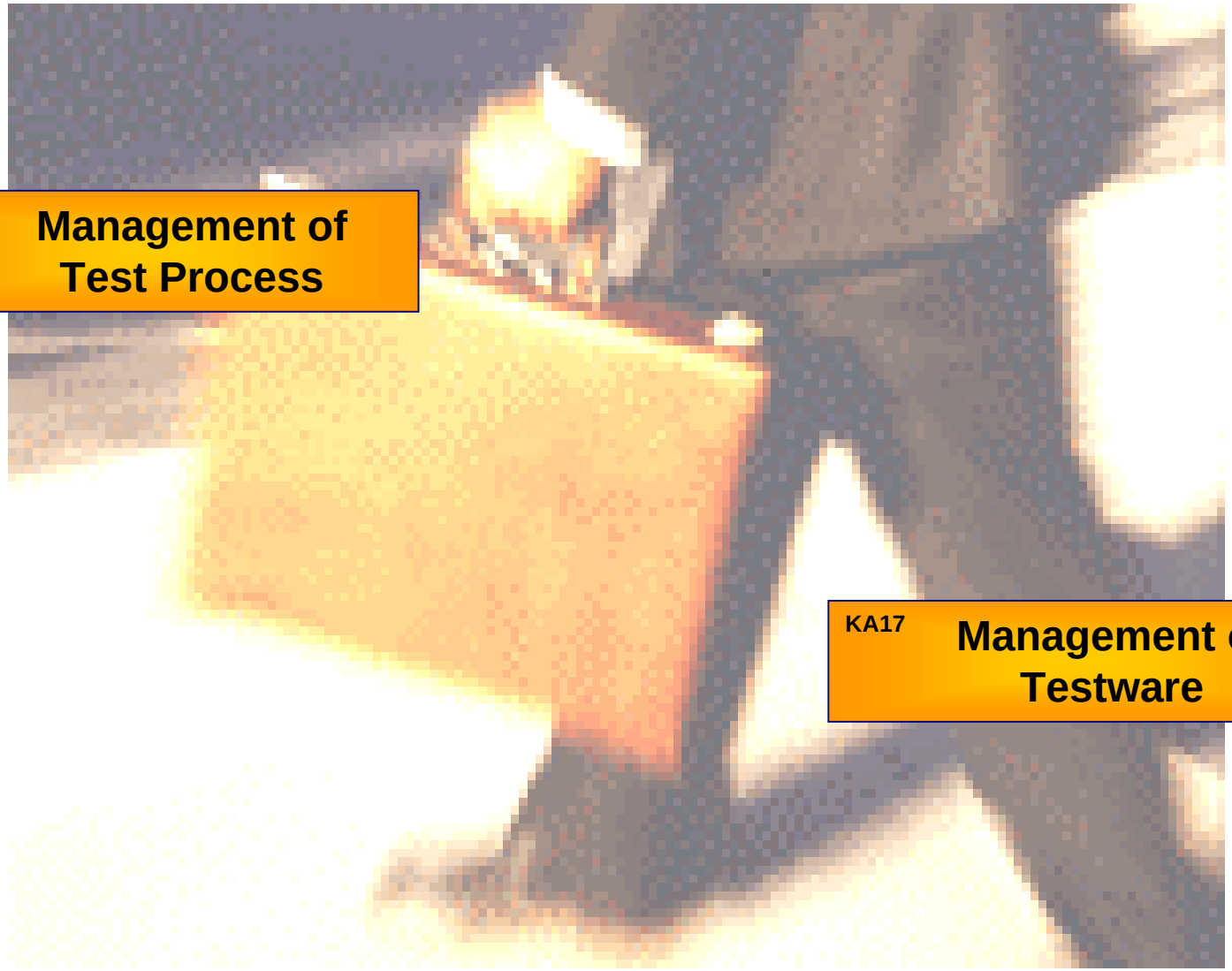
Tools

Integration Test (2)

Checkpoints - Level 2

- Tools are employed to help define and generate web-user „load profiles“ for use by „Load & Performance“ tools
- Tools are used for investigating the distributed aspects of OO-Systems (e.g. business-object invocations via CORBA)

Management



KA18 **Management of Test Process**

KA17 **Management of Testware**

KA18

Management of Test Process

Checkpoint - Level 1

- Additional resources are allocated for managing the test process if any of the following situations apply:

  - The transition to OO also means a change in Life-Cycle Model
  - Processes for testing traditional and OO-systems will run in parallel
  - A number of new tools will support the test process.

People

KA11
Committment & Motivation

KA14
Communication

KA12
Training

KA12
Test Roles

People

KA11

Committment & Motivation

Checkpoints - Level 1

- All testers are informed of the differences in testing OO-Systems
- A clear strategy for managing the transition to testing OO-Systems is available
- The additional effort required for introducing new techniques and tools is understood by management and is properly budgeted.

People

KA14

Communication

Checkpoints - Level 1

- All testing participants (including those who do not belong to a QA department) are invloved in the test planning process.
- The proposed test strategy is communicated to management (and agreed with them) at the start of the project.

Checkpoint - Level 2

- QA communicate new OO-test techniques and (development) test tools to development staff.

People

KA12

Test Roles

Checkpoints- Level 1

- Roles and responsibilities are defined for the following testing activities:

  - application tester
  - technical tester
  - tools engineer

- Staff are allocated one or more roles according to their abilities and wishes.

People

KA12

Training

Checkpoints - Level 1

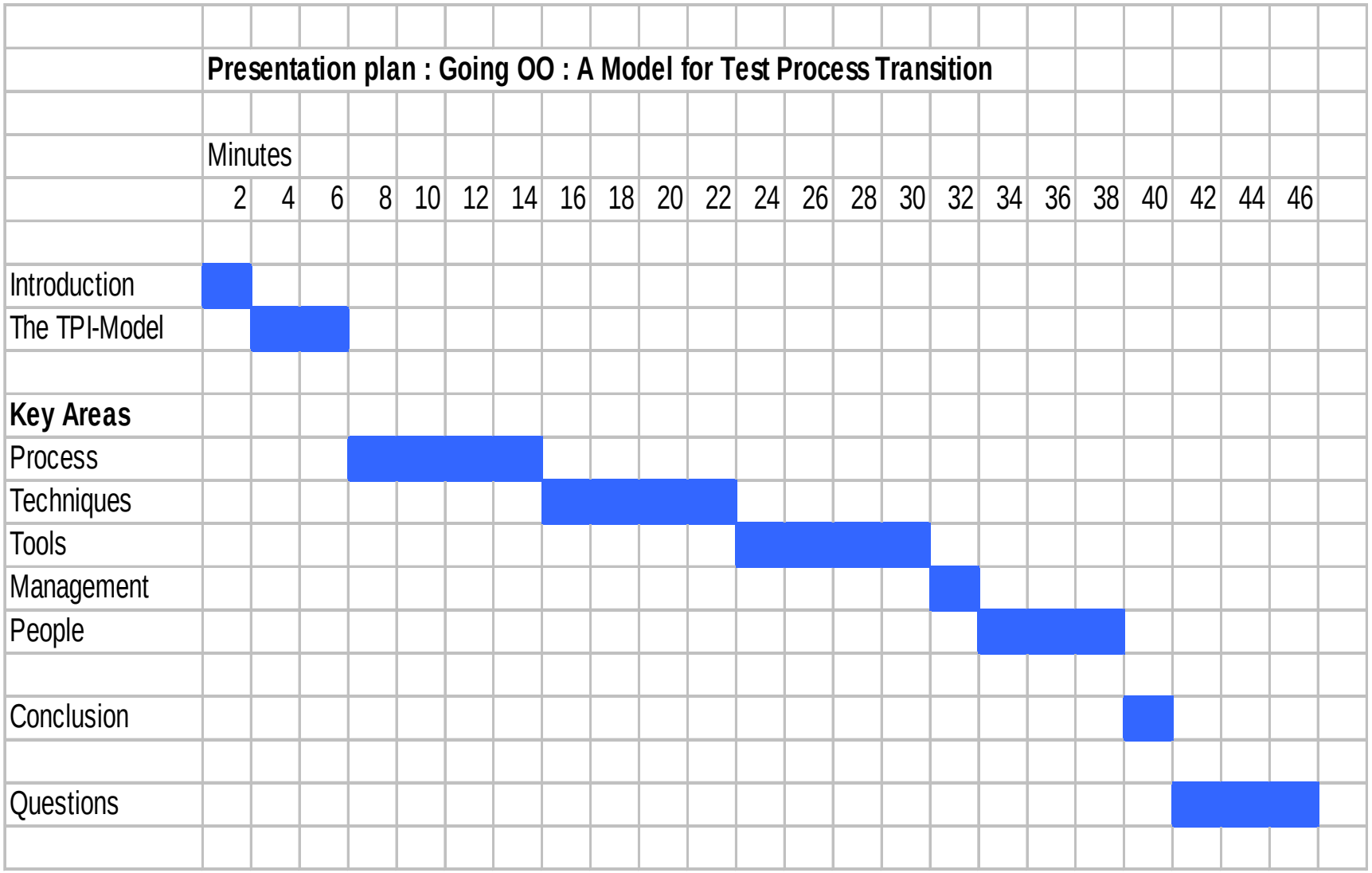
- Training takes place according to specific roles.
- Training takes place on a continuous basis (rather than one-off courses)
- Training is extended beyond the QA-department, such that non-QA staff are able to participate effectively in planned OO-testing activities (e.g. moduletests, performance tests)

Conclusion

What do the „OO“-extensions to the TPI-Model bring ?

- Test process transition can be understood, planned and controlled in a stepwise and priority-based manner.
- By extending an existing and proven model, the OO-Extensions can be viewed within the context of an overall Test Process Improvement model.
- A specific model is available to help solve a specific and frequently occurring problem.

Presentation plan : Going OO : A Model for Test Process Transition



Going “OO”
A Model for Test Process Transition

Graham Bath

debis Systemhaus
Dessauerstrasse 6,
80992, Munich
Germany

Tel: +49 (0)89 14326 367
Fax: +49 (0)89 14326 304
e-Mail gbath@debis.com

0. Abstract

Testers and test organisations are increasingly required to test “object-oriented systems”. Such OO-systems are implemented using the object oriented paradigm both from a programming and from an architectural point of view, and are these days typically web-based.

What does this mean for the testing process? The move from the development of “traditional” to OO systems brings with it a need to consider a wide range of testing issues; it is simply not enough to apply a new testing technique here or a new tool there. The test process itself must undergo a transition.

This paper is about understanding the issues involved in accomplishing such a transition, and outlines a framework model for doing this in a controlled, managed and step-wise manner.The model builds on the well regarded and widely used TPI®-[1]Model developed by IQUIP for general test process improvement. Extensions to the TPI®-Model are proposed which cover those aspects of the test process which are new or acquire greater emphasis when considering OO-Systems.

As a testing consultant, I have been involved in organising and implementing the test process transition for my customers as they move from traditional, often mainframe-based systems to “new technology” OO-Systems. This paper draws on that experience and provides both practical advice and references.

Key words: test process, TPI, object oriented testing, software testing

1. Introduction

For many testers and testing organisations, the move to OO-Systems comes as something of a shock. Managers sing the praises of increased productivity and market exposure, developers are fundamentally keen to take on new technologies and the testers, well, they’ll just test like before, right? Wrong. Testers soon come to realise that their approach to testing has to change “in some way”.

Faced with this challenge, testers can easily fall into one of two categories:

- The “head in the sand” category, for whom there is no need to change anything. Often this is an attitude which has been imposed on testers from above, so that potential productivity gains in development are not offset by increased costs of testing.
- The “lost in the jungle” category, who are aware of the need to change their test process, but are confounded by the multitude of new techniques, tools, articles and issues, and basically find it hard knowing which way to turn.

[1] TPI is a registered trademark of IQUIP Informatica B.V., Diemen, The Netherlands

Let's get one thing right from the start. Good testing practice for traditional systems is still good for OO-Systems. But for OO-Systems we as testers or test managers need a way to bring about the necessary changes to our test process in a managed, focussed and transparent way. For those with their heads in the sand, we need to make clear exactly what the changes are, and for those "lost in the jungle" we need a map which charts a way through.

This paper addresses these various needs by extending a model for test process improvement (the TPI®Model) for the specific factors relating to the testing of OO-Systems. Note that there are some commonalities between what I refer to as an "OO-System" and what could also more generally be called a "Client-Server"-System. This comes as no surprise, since OO-Systems and the type of Client-Server-Systems which we develop today are very closely related. Readers who test non-OO Client-Server systems will therefore already be familiar with a few of the points made, whilst for the many testers transitioning from host-based systems to OO-Systems (or combinations of host/OO) these points may well represent new ground.

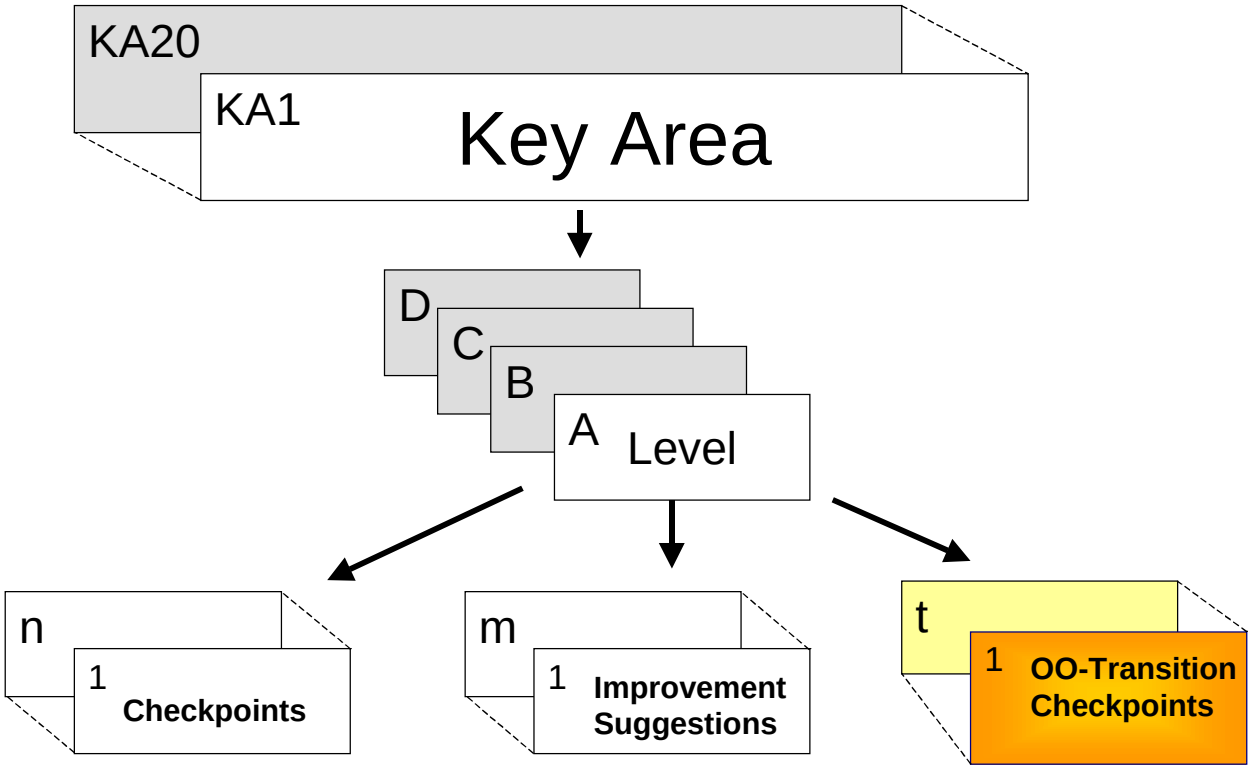
After a brief introduction to the TPI® Model itself, I will go on to cover the OO-specific testing issues within the context of that model.

2. The TPI-Model

Models for process improvement are relatively familiar to many of us. We may have heard of or even use models such as the Capability Maturity Model (CMM) or the Bootstrap Model. These models, whilst differing from one another in detail, all share a basic set of features:

1. "Levels of attainment" are defined and categorised according to predefined criteria.
2. An assessment of the current process status takes place using specific checkpoints for each level.
3. Steps are defined which can be followed in order to acheive a higher level of attainment

Whilst CMM and Bootstrap are examples of models which cover a wide range of IT-related processes, the Test Process Improvement Model (TPI®) focusses, as one might expect, on testing. The fundamental steps outlined above are nevertheless present in the TPI®-Model and are to be found in the following diagram (note that the extensions I proposed for OO-transition are already included).



The TPI®-Model considers 20 Key-Areas (Key Area KA1 for example, covers Test Strategy, KA8 covers Test Tools and so on). Up to four levels of attainment can be assigned to each Key Area, whereby level A represents a lower level of attainment than level B. It is only possible to reach level B when all criteria for level A have been fulfilled.To determine whether a particular Key Area has reached a particular level, Checkpoints are defined. To help the test process attain a higher level, improvement suggestions are proposed.

Whereas the TPI®-Model covers testing in general, this paper proposes an "add-on" to the TPI®-Model for the specific task of OO-related test process transition. An integration into the existing structure of the TPI®-Model is not sought, since this might clutter its currently clear format and structure.

Clearly, dependencies can exist between a level of one Key Area and the level of another. For example, acheivement of level B for the Test Tools Key Area KA8 depends on having attained level A for Key Area KA12 "Test Functions and Training", which states that the required test-tool expertise must be available.

To present the levels for particular Key Areas diagrammatically, the TPI®-Model proposes a "Test Maturity Matrix". Taking account of the dependencies between the levels of particular Key Areas results in a basic matrix which (for the first five Key Areas) looks like the one below. During the course of this paper, the various Key Areas will be introduced and the full Test Maturity Matrix built up piece by piece. At the end of the paper a full matrix is presented.

Key Area													
KA1	A					B				C		D	
KA2	A			B									
KA3		A				B				C		D	
KA4			A							B			
KA5					A			B			C		D

Simplified Test Maturity Matrix Model (first 5 Key Areas)

In this paper, I propose to introduce new levels of attainment for OO-Transition (OO1 to OO3). The basic Test Maturity Matrix shown above therefore needs to be extended by an appropriate number of extra columns. I will also define two extra OO-Specific Key-Areas.The following section of an extended Test Maturity Matrix illustrates these additions:

Key Area														**OO-Transition**		
KA1		A					B				C		D		OO1	OO2	OO3
KA2..KA20															.....	.....	.....
KA21 (new)															OO1		OO2
KA22 (new)															OO1		

Extended Test Maturity Matrix Model

Attainment for a particular test process can can be marked on the Test Maturity Matrix, generally in the form of a shaded area. Here is an example for the first five Key Areas:

Key Area													
KA1		A					B				C		D	
KA2		A			B									
KA3			A				B				C		D	
KA4				A							B			
KA5						A			B			C		D

For a more detailed explanation of the TPI®-Model, refer to the book by the TPI®-Model authors from IQUIP [1].

3. OO-Transition

The remainder of this paper deals with issues of test process transition on a Key Area by Key Area basis. For ease of presentation, the Key Areas are sorted into the following groups:

1. Process
2. Techniques
3. Tools
4. Management
5. People
6. Environment

3.1 Process

As we might expect, the issue of "Process" is at the core of any Test Process Improvement. For OO-Test-Transition this is also the case.

The following Key Areas and OO-Levels are covered within the Process group:

Key Area	**ID**	**OO-Transition**		
Test Strategy	KA1	OO1	OO2	OO3
Life-Cycle Model	KA2	OO1		
Moment of Involvement	KA3	OO1		
Estimating & Planning	KA4	OO1		
Metrics	KA7	OO1		
Scope of Methodology	KA13			
Reporting	KA15	OO1		

3.1.1 Test Strategy

Level OO1

Remarks

- Non-Functional testing:
  Just doing functional system-level testing is not enough when testing multi-tier OO-Systems. It is nevertheless a mistake that is often made when transitioning to the testing of OO-Systems, and is a feature of the „head in the sand" test strategy I mentioned earlier. Non-functional testing issues involve characteristics such as performance, stability under load and stress situations, multi-user issues and scalability. If experience has been already gained with testing Client-Server systems, these point will not be new, but they now take on much greater significance.

- Risk Assessment:
  Some of the factors which go into the risk assessment are related to the risk of errors occurring (mostly during integration). For low-level tests we can refer to OO-specific complexity metrics relating to depth of class hierarchy, polymorphism, etc to make informed judgements on which classes to test with what technique. Other OO-relevant risk factors have a more business-related flavour (e.g. if a class or group of classes is intended for re-use as a framework or a business-object there can be greater impact to a business if testing is inadequate). Obviously, non-OO factors such as the criticality of the application also drive the test strategy concerning low-level class testing (embedded code for an airliner flight control system demands low-level class testing, whilst for a less critical on-line information system an alternative test strategy may be more effective).

OO-Checkpoints

? Non-functional quality characteristics are covered in the test strategy

? Risk assessment takes OO-Factors into account, particularly when planning a test strategy which includes low-level (class) tests

Level OO2

Remarks

- New Levels of OO-Testing
  There are many more test types which can be carried out for OO-Testing. For example, instead of the single „integration test“ for traditional systems, Donald Firesmith [5] identifies the following test types for OO-integration testing:
  - Cluster - Scenario -Inheritance -Classification -Aggregation

  The right selection and execution of test type demands a co-ordinated strategy to avoid the inefficiencies of overlaps or the deficiencies of leaving critical testing gaps.

- Strategic decisions may be taken to do thourough inspections and first undertake dynamic testing at the level of cluster integration.

- Coordination of Test Strategy:
  QS-departments often don‘t have all the skills needed for integrating n-tier systems and performing all technical tests. Dependencies on other departments and their skills means they have to be bound more explicitly into the Test-Process than before (the “can you have a quick look” method will not work)

OO-Checkpoints

? The test strategy is coordinated across all recognised OO-testing levels such that test effectivity is optimised

? The test strategy calls for the involvement and coordination of skilled personel who are not allocated to QA.

Level OO3

Remarks

- Testing Business Objects needs extra consideration in a Test Process context. Issues here are, for example:
  - With a wider variety of potential platform/OS/Browser combinations, can all caller possibilities be handled by the business object?
  - Is the public interface absolutely watertight and well documented?
  - Does the user of a business object have a description available of the testing performed (sometimes referred to as a “ Test Contract”)

OO-Checkpoints

? High-Level OO-test activities include the testing of (potential) business objects

? A published „Test-Contact“ exists for business-objects

3.1.2 Life-Cycle Model

Level OO1

Remarks

- The individual steps of the life-cycle model remain effectively the same for OO-Testing. However, the model now has to cope with RAD and similar iterative models.

- „Test“ is typically on the critical path for shorter periods of time, resulting in a higher dependency on tools.

OO-Checkpoints

? The Life-Cycle Model is adapted to an iterative development process

? For E-Business applications, the Life-Cycle Model considers in particular the “continuous testing” strategy required to meet changing user behaviour.

Dependencies
- Level B for the Life-Cycle Key Area should be achieved for testing OO-Systems. This is because the increased dependence on tools makes the additional level B steps “preparation” and “completion” essential.

3.1.3 Moment of Involvement

Level OO1

OO-Checkpoint
? Issues of test tooling and test strategy are explicitly covered at project initiation.

Dependencies
- Level D for the „Moment of Involvement“ Key Area should be achieved for testing OO-Systems since testing issues must be agreed at the very start of the project.

3.1.4 Estimating & Planning

Level OO1

Remarks
- The planning task is more complex for OO-projects; there are typically more individual skills required and these are often spread over different (non-QA) departments or companies. Failure to recognise this can lead to the non-availability of (human) resources at critical times. Although this is true of projects in general, it is a problem which is particularly prevalent when transitioning to an OO testing process.

OO-Checkpoint
? The planning of tests is coordinated with other organisational units and projects to ensure the availability of all necessary skills.

3.1.5 Metrics

Level OO1

Remarks
- Some input from OO-Metrics to risk assessment can be beneficial to estimating risk. Above all, when particular parts of an OO-System exhibit more problems than others, specific OO-metrics can help to identify causal patterns. But beware of excesses in this area! There is little proof of cause and effect available and project time can be lost trying to “generate” such relationships. The best metrics can usually be obtained from a well organised defect management.

OO-Checkpoint
? Specific OO-Metrics are taken to provide input to the risk assessment process.

3.1.6 Scope of Methodology

Remarks
The transition to OO-Testing has no direct influence on whether the test process is applied at the project level (level A), at the organisational level (level B) or is subject to optimising measures (level C).

3.1.7 Reporting

Level OO1

Remarks
- Whilst the role of reporting in the test process does not really change when transitioning to OO-testing, the emphasis on what quality criteria are reported does.

OO-Checkpoint
? Quality characteristics such as scalability, compatability and performance are reported.

Dependency
Key Area „Test Strategy „ Level OO1 : Non-functional quality characteristics are covered in the test strategy.

3.2 Techniques

The following general points all have an influence on the „Techniques“ group of Key Areas:
- Whatever the benefits of OO might be, they are certainly not to be felt in the area of reduced defects (this is born out by reserach carried out by Les Hatton [6])
- Bad test practice in OO tends to show up in more critical failures in production, especially for E-Business systems.
- Integration testing of OO-Systems is the area where the most difficulties and differences to traditional systems exist. This is borne out in a number of test papers (see [2], [3] and [4])

The following Key Areas and OO-Levels are covered within the Techniques group:

Key Area	**ID**	**OO-Transition**		
Testspecification	KA5	OO1	OO2	
Static Test	KA6	OO1	OO2	
Evaluation	KA19	OO1	OO2	
Low-Level Test	KA20	OO1	OO2	
Integration Test	KA21 new	OO1	OO2	OO3
High-Level Test	KA22 new	OO1	OO2	

3.2.1 Testspecification

This Key Area covers the method by which tests are specified.

Level OO1

Remarks
- Formal Methods:
  As previously mentioned, some levels of testing are new to OO-Systems and most of these demand their own technique for test case generation. Regarding the need for formal techniques, (i.e. that there is an unambiguous way of determining testcases from a given source of information), we need to move from the general statement of the existing TPI-Model and be explicit about what specific techniques are needed for particular levels of test. An excellent source of such techniques is presented by Lee Copeland [7].

- OO-Design Improvement:
  The specification „scenarios“ based on descriptions in OO-design documentation (often in the form of Use-Cases) are a valuable test asset. These scenarios can form the basis for formal (and often hierarchical) testcase design. This point serves to underline the fact that Test Process Improvement can also be brought about by improvements to the SW-Development process (in this case, the OO-Design).

OO-Checkpoints

- Test cases are derived in a formal manner using techniques which are appropriate to OO-testing.

- OO-Design documentation includes scenario descriptions which are an adequate basis for functional (system) test specification.

Dependency
Key Area „Testspecification“, Level B : „Formal Testspecification techniques“

Level OO2

Remarks

- The specification of Load Profiles for use in mainly technical tests (load, performance, stress, soak etc.) is general to client-server systems and is highlighted for OO-Systems because of the significance of such tests.
- Web-based OO-Systems such as E-Business applications need to be regularly or even continously tested once launched.

OO-Checkpoint

- A technique exists for generating „Load Profiles“ which adequately adress the needs of of the system prior to and after production launch.

3.2.2 Evaluation and Static Test

The TPI-Model's Key Areas „Evaluation“ and „Static Test“ are combined for the purpose of discussing OO-transition.

Level OO1

Remarks

- Inspection of OO-Code:
  Research from Marc Roper [8] shows that error detection in OO-Code inspections is more difficult, primarily due to the non-sequential nature of the code caused by, for example, inherited methods, polymorphism etc.The code inspection technique for OO-Code should take these issues into account by, for example, providing separate checklists for OO-specifics or even assigning separate inspectors for such issues.

- Inspection of OO-Models:
  John McGregor has provided some useful techniques (see [9], and [10]) for inspecting OO-Models used in the design phases. Not only are these techniques likely to uncover more design problems, they are also effective at increasing the OO knowledge of testers. Typical features of the proposed „Guided Inspection“ technique are questions posed by testers to, for example, identify particular Use-Cases or high-level test scenarios in a given UML Sequence Diagram. The designer responds by tracing the path through the Sequence Diagram and the tester notes the coverage achieved, perhaps by annotating the diagram.

OO-Checkpoints

- ? Inspection techniques and checklists exist which address the non-sequential nature of OO-Code.

- ? The inspection of Design models (e.g. Class Diagrams, Sequence Diagrams in UML) are inspected in an interactive manner together with an experienced designer.

Level OO2

OO-Checkpoint

- ? Static Analysis of code is performed to enforce coding standards and detect typical OO-programming errors.

3.2.3 Low-Level Test

For OO-systems we are talking about class testing. Generally speaking, the costs of planning and executing low-level tests is relatively high (even when supported by tools). Risk-Assessment must therefore be practiced in order to focus effort and achieve the most effective test strategy.

Level OO1

Remarks
- Functional (black-box) testing:
  For classes we can use essentially the same techniques as as for traditional systems (equivalence classes, boundary conditions etc.). For OO implementations it is helpful to consider „Design by Contract“ (pre-condition, post-condition, invariants) as a suitable technique for specifying tests. Coverage is then measured according to the percentage of post-conditions covered by the test cases.

OO-Checkpoints
- ? Functional (black-box) test cases take into account the pre-conditions, post-conditions and invariants of a class (i.e. „Design by Contract“)

Level OO2

Remarks
- The use of polymorphism and abstract classes makes the traditional test coverage criteria more questionable in value.

The following types of class testing can be identified (see Lee Copeland’s classification in [7]):

- Structural testing (intra-class):
  For OO-Systems structural testing is generally easier, mainly due to the smaller, less complex and cohesive nature of implemented methods. However, there is some evidence that relatively more errors are made in smaller pieces of code that larger ones. Since in OO methods tend to be small, this would tend to point to a need for more Low Level Testing.

- Interaction Testing (between methods of a class): A simple interaction matrix is normally sufficient.

OO-Checkpoints
- ? Techniques exist for testing the interaction of methods within a class.

- ? Techniques for structural (white-box) testing do not focus on achieving traditional coverage percentages.

Dependency
- Key Area „Tools for Low Level Test“ Level OO1: Test Tools support the required test techniques.

3.2.4 Integration Test

For OO-systems we are talking about about integration from the class level up to business object level. Integration is the phase where the most changes are felt compared to traditional systems.

Level OO1

Remarks
- Due to the very nature of the OO-Paradigm, interaction testing (between classes) is especially important for OO-Systems. The level at which interactions take place can be, for example, between objects of particular classes, between object clusters or between distributed (business) objects. Techniques differ according to the complexity of the interaction. For simple interactions (like for low-level intra-class interactions) an interaction matrix can be used which focusses of the areas with the greatest amount of interaction (i.e. where the consequences of failure are likely to be higher). More complex interactions are a feature of Level OO2.

- The testing of inheritance hierarchies requires the use of specific techniques (see, for example, the „HIT Algorithm“ [9])

OO-Checkpoints
- Techniques exist for generating test cases for class inheritance hierarchies.

- Testcases focus on the published (public) interface of an object in order that a consistent behaviour (e.g. of response or state change) can be shown irrespective of the calling object.

- Techniques exist for generating test cases for simple interactions between objects.

Dependency
- Key Area „Tools for Integration Test“ Level OO1: Test Tools support the required test techniques.

Level OO2

Remarks:
For interaction testing with more complex combinations of interactions, the use of techniques such as Orthogonal Arrays (see [9]) are more appropriate.

OO-Checkpoint
- Techniques exist for generating test cases for complex interactions between objects at all test levels.

Level OO3

Remarks
- In the testing of inheritance hierarchies, changes in the hierarchy must take into account the retest issues of changing a class which is high up in the hierarchy. Do the retest consequences warrant the change ?

OO-Checkpoint
- ? A consistent decision can be made on the need to change/retest sub-classes within a class hierarchy.

3.2.5 High-Level Test

Level OO1

Remarks
- Use-Cases as a source of Testcases:
  Use-Cases may already have been documented together with example scenarios by the designer (to be encouraged). If not, the general Use-Case description should be taken as the basis for generating specific scenarios suitable for testing. As a further extension to this technique, the high-level scenarios may be broken down hierarchically to provide finer granularity and improved coverage of the Use-Case requirements.

- GUI-Test:
  The GUI-test issue may already have been adressed for a non-OO application. However, virtually all on-line OO-Systems feature a GUI client. The checkpoint related to this point is therefore particularly relevant for those making the transition to OO-Systems from the host world. In such cases, the transition from character-based to GUI poses a significant testing challenge. Techniques for GUI-Testing are well covered in the available literature (in particular, see Dorothy Graham’s book on Software Test Automation [11]).

OO-Checkpoints
- ? User Scenarios defined in high-level design documentation (e.g. as „Use-Cases“) are used as the basis for test case design for system test.

- ? Techniques are available for determining test cases relating to the Graphical User Interface (if a GUI is defined for the system under test).

Dependency

- Key Area Testspecification“ Level OO1 : OO-Design documentation includes scenario descriptions.

Level OO2

OO-Checkpoint

? Techniques exist for defining typical user profiles for use in technical system tests (e.g. for performance, load and stress tests). For web-based OO-Systems, these techniques are supported by tools.

3.3 Tools

To quote Sigrid Eldh [4], who has provided much input in the area of OO-Testing, „OO-testing is test tool intensive“. The TPI-Model Key Area „Tools“ has been subdivided here into „high-level“ and „low-level“ categories to give some extra structure to this fundamentally important subject for OO-testing.

Note that tools used for test planning and management, which are also covered in the TPI-Model, are basically unaffected by the transition to OO-Systems.

low-level tools

Level OO1

OO-Checkpoints

? Tools are employed to support class testing, where this is part of the test strategy.

Level OO2

Remarks

- The type of dynamic test listed below is typical of those required by (but not entirely exclusive to) OO-Systems. If the current test process does not already include such dynamic analysis, a transition to OO will demand that they be considered.

OO-Checkpoint

? Tools are employed to support the OO-specific dynamic analysis called for in the test strategy, for example:

- performance „profiling“
- thread-monitoring
- exception-handling
- memory usage

high-level tools

Level OO1

Remarks

- Essential Tools for OO
  The TPI-Model Key Area „Tools“ lists 12 possible test tool types. To reach level B for this Key Area, the TPI-Model requires that any 2 these tools are used. For high-level tools, emphasis is placed on the use of specific tools, particularly those for GUI-Tests and „Performance & Load“ tests. These tool types are considered as essential for achieving the Level OO1 because of their significance in the testing of essential OO-System quality characteristics.

- Own Tools
  Don‘t forget - it may be a feasible proposition to develop your own tools for Performance and Load tests, especially if your demands for accuracy and flexibility can be relaxed.

- The use of GUI-Tools for supporting the high-level test of web-based OO-Systems is considered particularly significant since:
  • The cost-benefit ratio of their use is generally acceptable because such systems typically undergo small, frequent changes with an associated demand for (regression) testing.

• Mastering the combinations of platforms/OS/Browser which are demanded of such systems is no longer a feasible proposition for manual testing.

OO-Checkpoints

? Tools are employed to support the testing of class inheritance hierarchies, where this is part of the test strategy.

? GUI-testtools are employed when one or more of the following points is applicable:

- a large number of platform/OS/Browser combinations must be tested
- the Life-Cycle Model calls for many development and test iterations
- the application is expected to be frequently updated or changed after deployment.

? Tools for „Load & Performance“ testing are used when one or more of the following points is applicable:
- the number of client users is expected to exceed 10
- system scalability is of critical importance
- the system architecture is complex or includes previously unused software components or products
- self-developed middleware components are used

Level OO2

Remarks

➢ The analysis of web-site useage requires tools to capture the „hits“ and generate a model of current and expected user scenarios. These are effectively the test cases for use by the appropriate „Load & Performance“ tools.

OO-Checkpoints

? Tools are employed to help define and generate web-user „load profiles“ for use by „Load & Performance“ tools

? Tools are used for investigating the distributed aspects of OO-Systems (e.g. business-object invocations via CORBA)

3.4 Management

Generally speaking, good test management practice for traditional systems is still applicable to OO-Systems.

The following Key Areas and OO-levels are covered within the Management group:

Key Area	**ID**	**OO-Transition**		
Management of Defects	KA16			
Management of Testware	KA17	OO1		
Management of Test Process	KA18	OO1		

3.4.1 Management of Defects

The transition to an OO Test Process has no major influence on this Key Area.

3.4.2 Management of Testware

Level OO1

Remarks

➢ Increased usage of tools for OO-Systems emphasises the important of effective testware management, particularly in achieving benefits at a departmental level. If the transition to OO-Systems means the use of test tools for the first time, this will be a major feature in accomplishing a successful transition (refer to Dorothy Graham‘s book on Software Test Automation [11])

OO-Checkpoint

? If new tools or techniques are introduced for OO-Testing, the management of testware is adapted to cope with the extra volumes and types of testware (scripts, test specifications etc.)

3.4.3 Management of Test Process

Level OO1

Remarks

The task of managing the test process will demand more managerial resources because:

- the greater variety of test techniques demands a more intensive management of the test process (which technique is applicable and gives the best results ?)

- if the Life-Cycle Model changes to a more interative approach, there will be extra demands placed on managing and introducing that approach (testing will be more often on the critical path but for shorter periods of time).

- if new tools are to be introduced, test management must ensure that:
  • tool evaluations are properly planned and executed (pilot projects etc.)
  • the introduction of new tools does not „swamp“ testers
  • the necessary time and human resources are planned for introducing tools.
  • attention is paid to experiences in the effective use of test tools (in particular GUI-Tools, see case studies in [11])

- if the testing of OO-Systems needs to run in parallel with the testing of traditional systems, there will be increased workload for Test Process management (assuming that effectively two Test Processes are in effect at the same time).

OO-Checkpoints

? Additional resources are allocated for managing the test process if any of the following situations apply:

•The transition to OO also means a change in Life-Cycle Model
•Processes for testing traditional and OO-Systems will run in parallel
•A number of new tools will support the test process.

3.5 People

This is possibly the most critical issue to get right when transitioning to the testing of OO-Systems.

The following Key Areas are covered within the People group:

Key Area	**ID**	**OO-Transition**		
Committment and Motivation	KA11	OO1		
Test Roles & Training	KA12	OO1		
Communication	KA14	OO1	OO2	

3.5.1 Committment and Motivation

Level OO1

Remarks

- What motivates a tester when a transition to OO-testing takes place?
  •they understand the differences between testing OO and traditional systems
  •they understand the overall benefits to be obtained from OO as a concept (i.e. the notion that OO is simply a new „fashion“ must be dispelled)

•they are aware of the strategy to be adopted in making the test process transition (the TPI-Model and the OO-Extensions proposed in this paper support the strategy-building task)

- What demotivates?
  - • hiding the fact that testing OO-Systems is harder and demands a new approach to testing.
  - • lack of a strategy or poor communication of that strategy
  - • simply overburdening testers (new techniques, new tools etc. but timescales and human resources remain fixed)

OO-Checkpoints

? All testers are informed of the differences in testing OO-Systems

? A clear strategy for managing the transition to testing OO-Systems is available

? The additional effort required for introducing new techniques and tools is understood by management and is properly budgeted.

3.5.2 Test Roles & Training

Level OO1

Remarks

- Driving factors for the definition of new roles are :-
  - new testing techniques required for OO-Systems
  - new tools

- It is critical to the sucessful transition of the test process that roles and responsibilities for new activities are identified. It is not efficient, for example, that all testing staff acquire the know-how required to use and program all test tools. The definition of roles can address these issues, for example:

  - **application tester** with specific domain experience and responsible for the specification of functional tests.
  - **technical tester** with responsibility for the specification of end-to-end performance and load testing aspects (amongst others)
  - **tool engineer** with responsibility for the implementation of tests specified by the application and technical testers using the required tools.
- Experience has shown that training in OO-Testing is less efficient when conducted as a one-off exercise. It is far better, for example, to employ a coach who can guide the OO-Transition and provide the necessary inputs on a „drip-feed“ basis.

OO-Checkpoints

? Roles and responsibilities are defined for the following testing activities:
  - application tester
  - technical tester
  - tools engineer

? Staff are allocated one or more roles according to their abilities and wishes.

? Training takes place according to specific roles.

? Training takes place on a continuous basis (rather than one-off courses)

? Training is extended beyond the QA-department, such that non-QA staff are able to participate effectively in planned OO-testing activities (e.g. moduletests, performance tests)

3.5.3 Communication

Level OO1

Remarks
- The Test „Kick-Off“ has been shown to be a valuable communication mechanism between testers, implementers and management. It is recommended to combine a review of the Test Plan with such a meeting, such that the attendees can review strategy and priorities and take decisions regarding staffing and tooling requirements.

OO-Checkpoints
- ? All testing participants (including those who do not belong to a QA department) are involved in the test planning process.

- ? The proposed test strategy is communicated to management (and agreed with them) at the start of the project.

Level OO2

Remarks
- In general, a test mentality should be established at all levels, especially since development staff may be responsible for carrying out low-level tests and may be unaware of specific testing techniques and tool use.

OO-Checkpoint
- ? QA communicate new OO-test techniques and (development) test tools to development staff.

3.6 Environment

The following Key Areas are covered within the Environment group:

Key Area	**ID**	**OO-Transition**		
Test	KA9			
Office	KA10			

The transion to an OO-Test Process has little or no impact upon these Key Areas.

4. The Completed Test Maturity Matrix

All Key Areas have now been covered, and the extended Test Maturity Matrix can be presented as follows:

Group	Key Area	ID														OO-Transition		
Process	**Test Strategy**	KA1		A					B				C		D		OO1	OO2	OO3
	Life-Cycle Model	KA2		A			B										OO1		
	Moment of Involvement	KA3			A				B				C		D		OO1		
	Estimating & Planning	KA4				A							B				OO1		
	Metrics	KA7						A			B			C		D	OO1		
	Scope of Methodology	KA13					A						B			C			
	Reporting	KA15		A			B		C					D			OO1		
Techn.	**Testspecification**	KA5		A		B											OO1	OO2	
	Static Test	KA6					A		B								OO1	OO2	
	Evaluation	KA19							A			B					OO1	OO2	
	Low-Level Test	KA20					A		B		C						OO1	OO2	
	Integration Test	KA21															OO1	OO2	OO3
	High-Level Test	KA22															OO1	OO2	
Tools	**High-Level, Low-Level**	KA8					A			B			C				OO1	OO2	
Mgmt.	**of Defects**	KA16		A				B		C									
	of Testware	KA17			A			B				C				D	OO1		
	of Test Process	KA18		A		B								C			OO1		
People	**Comm. & Motiv.**	KA11		A				B						C			OO1		
	Test Roles & Training	KA12				A			B				C				OO1		
	Communication	KA14			A		B							C			OO1	OO2	
Env.	**Test**	KA9				A				B						C			
	Office	KA10				A													

5. Conclusion

What do the proposed extensions to the TPI®-Model bring for the tester or test manager involved in the act of transitioning to a test process for OO-Systems (in short „going OO“) ?

1. The required changes can be understood, planned and controlled in a stepwise and priority-based manner.

2. By extending an existing and proven model, the OO-Extensions can be viewed within the context of an overall Test Process Improvement model.

3. A specific model is available to help solve a specific and frequently occurring problem.

5. References

[1] Test Process Improvement, Tim Koomen and Martin Pol, Addison-Wesley,1999

[2] Influence of OO and Process Orientation on Software Testing, Dr.Peter Liggesmeyer, EuroSTAR95

[3] Integrating and Testing OO-Software, Peter Jüttner, Sebald Kolb, Peter Zimmerer, EuroSTAR 94

[4] Essentials of Object-Oriented Testing, Sigrid Eldh, EuroSTAR 96

[5] Tutorial on testing object-oriented systems, Donald Firesmith, STAR 95

[6] Testing and Errors: the Avoidable and the Unavoidable, Les Hatton, EuroSTAR 95

[7] Testing Object-Oriented Systems, Lee Copeland, EuroSTAR 98 Tutorial

[8] Problems, Pitfalls and Prospects for OO-Code Review, Marc Roper, EuroSTAR 99

[9] The Fifty-Foot Look at Analysis and Design Models, John D. McGregor, JOOP, July/Aug. 1998

[10] Validating Domain Models, John D. McGregor, JOOP, July/Aug. 1999

[11] Software Test Automation – Effective Use of Test Execution Tools, Dorothy Graham et al, Addison-Wesley, 1998

[12] The Capability Maturity Model, Software Enginneering Institute, Carnegie Mellon University, 1995

[13] Refer to www.bootstrap-institute.com

Wednesday 6 December 2000

W12

Going “00” : A Model for Test process Transformation

Graham Bath

Graham has been involved in the development of mission-critical and commercial systems using both procedural and object-oriented paradigms. As a Project Leader he has been responsible for projects relating to spaceflight, telecommunications, police incident-control and, more recently, e-business. Within real-time, mission critical systems such as the Tornado and Eurofighter military aircraft, he has tested software to the highest levels of rigour and has been involved in implementing several quality standards.

Since joining debis Systemhaus he has mastered the Quality Improvement Programs of several major German companies and is currently involved in implementing Test Process Improvement for a major German tourism company. This entails the modification, definition and coaching of test practices to meet the business demands of the 21st century, which increasingly involves Web-based applications and e-Commerce.

Email address: gbath@debis.com