Managing Projects with Intelligence

Paul Gerrard
Gerrard Consulting Limited
PO Box 347
Maidenhead
Berkshire
SL6 2GU
Email: paul at gerrardconsulting dot com
Web: gerrardconsulting.com
Tel: +44 1628 639173

Abstract

This paper sets out some key principles for project management. Managers should be focused at all times on the goal of their project and the risks that threaten them. Projects need to manage goals, not activities to ensure work is done, deliverables are delivered and they are acceptable.

The Project Intelligence concept aims to give project and stakeholder management visibility of project risks, their intermediate and final goals and progress towards them.

Project and stakeholder management are the main customer of Project Intelligence. Early goal and risk analysis enables management to have influence over the way projects are tested and how the gathering of intelligence will be performed. Intelligence is gathered throughout the project lifecycle, from early reviews to final acceptance testing. The intelligence gathered, enables project and stakeholder management to judge how and whether progress is being made. Further, when problems do arise, the intelligence gives an insight into how the current challenges can be overcome. No longer does the project manager have to make a case for slipping a task, accepting or rejecting a deliverable or releasing into production. The intelligence should make those cases self-evident.

Management must learn to appreciate what Project Intelligence can do for them. It dramatically enhances the value of testing to their projects and enables them to manage their projects with more confidence. Project Intelligence gives management better visibility of progress and enables them to make significantly better-informed decisions.

Finally, Project Intelligence is usable in ALL development methodologies. There is an overhead perhaps in the early goal and risk analysis, but this is simply better test practice anyway. The intelligence concept integrates well with risk-oriented project management methods, as these methods pay more attention to the test activities associated with all project deliverables. Project Intelligence simply aims to enhance the understanding of the test process and improve the reporting done by all test activities.

1 PROJECT GOALS AND DELIVERABLES[1]

1.1 Projects Exist to Deliver Business Goals

The usual life cycle of a project begins when senior management in an organisation identify a need to change something. In the case of a user organisation it might be a new way of working to achieve improved performance. This might entail retiring an existing system and replacing it with a new package, or some custom-built software. In the case of a software house, the need might be to launch a product, based on a new idea, or to enhance an existing product to improve its competitiveness.

Change in all organisations is expensive and often traumatic and must be carefully justified. Typically, senior management will want to see a business case made for the change which sets out key business goals and opportunities as well as the organisational impact, costs and risks to success. Sometimes, the decision can be made easily as there is little choice: the regulatory environment might make the change mandatory; pressure on margins or the need to update an aging product might be overwhelming.

In other situations however, the decision is harder to make. The risk of making the change might outweigh the benefits of the change itself or the case for the new project is marginal. Often, the benefits are intangible and the calculations of costs and benefits are based on a vague understanding of the challenge. Many software projects are initiated on the basis of dubious business cases. But let us not delve too deeply into that issue now. This paper is intended to provide a framework for managing projects to successful completion, so let us assume that projects are based on a sound business case.

The mechanism used to implement a change is a Project. Andersen et al. [1] define a project as "a human endeavour that:

- Is unique.
- Is designed to attain a specific result.
- Requires a variety of resources.
- Is limited in time."

Most projects are designed to create change in three areas:

- People – recruiting, reassigning, releasing, training, retraining people to realign an organisation's competences with the new business goal.
- Systems – constructing, acquiring, and retiring business or software systems and processes to support the new business goal.
- Organisation – downsizing, rightsizing and realignment of responsibilities in an organisation to support the new business goal.

There are an infinite number of types of profiles of People, Systems and Organisation (PSO) projects. Although the software-related activities in a project fall into the 'Systems' change category, these projects usually involve both people and organisational change.

Goal-Directed Project Management [1] recognises that changes involves both technical aspects and procedural (or human) aspects. An example of a purely technical change might be change to software to make it more efficient or reliable. A purely procedural project might be a redistribution of responsibilities in an organisation. Most projects involve a mix of both types of change. Experience shows that the technical aspects of projects tend to be easier to manage – they are tangible, more concrete and determination of success criteria is relatively easy. Procedural aspects are harder to manage because changes involve personalities, attitudes

[1] In this paper, we use the term deliverable and product interchangeably to represent the outputs of development processes.

and culture and the criteria for success are harder to define. (A common mistake made by many technically-oriented project managers is to not recognise the importance of the people and organisational aspects of their projects.)

In a goal-directed project, the four aspects of project management, namely: planning, organising, monitoring and control are all driven by the constant need to focus on the end goals of the project.

1.2 Goal and Benefits Analysis

When a project is defined, the business goals are used as the starting point to identify the changes the project must implement and artefacts to be delivered (the "deliverables") to achieve the overall business goals. A typical example of the analysis of business goals into its dependencies is shown in Figure 1.

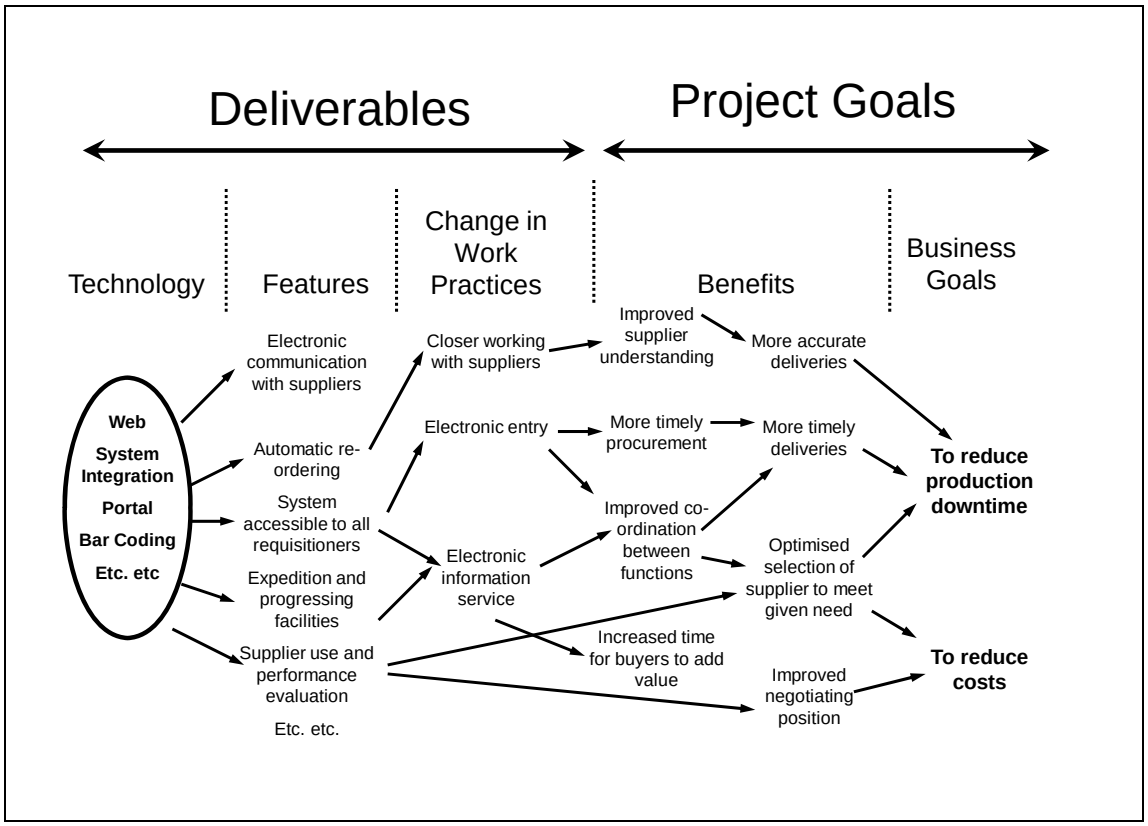

Figure 1 Analysis of goals and deliverables (Based on [2])

Business goals tend to depend on beneficial changes in the way a business operates. These business "benefits" are often improvements in quality, speed, reliability or the cost of delivering services, performing internal processes or interfaces with customer or suppliers.

Business benefits are usually achieved by adding new business processes or work practices or changing them to make them more efficient or effective. Often, these practices require automated support and the features of a new or amended computer system are specified to provide that support. These system features require underlying technical infrastructure (technology) some of which might exist, but other technology might have to be acquired, configured, implemented and tested to support them.

This type of analysis is useful for identifying the deliverables of the project. Although projects vary, the deliverables of most projects involving software include technology, new features and work-practices and processes. These three types of deliverables span the technical thru procedural aspects of PSO projects.

In the PRINCE methodology [3], the deliverables of a project are referred to as Products, each having a Product Description, setting out their composition, purpose and dependencies on other products. PRINCE uses a Product Flow diagram to identify the dependencies between products which determines the sequencing of deliveries and as a consequence, is the basis of the schedule of activities for planning.

All popular project and development management methodologies depend on the notion of products and dependencies.

1.3 Risk in Software Projects[2]

We will use the following definition of risk [3]:

A risk is a threat to one or more of the goals (cardinal objectives) of a project and has an uncertain probability.

Risk only exists where there is uncertainty. That is, if the probability of a risk is zero, it's not a risk. The undesirable event will never happen, so there is no concern. Suppose there is a hole in the road ahead of you, you can see it, you're walking towards it, and if you keep walking you'll fall into it. If the probability of that undesirable event is 100%, then it is inevitable and it is not a risk. To avoid falling in, you would take a detour or stop walking – there is no uncertainty about that. In the same way, a known bug, detected through testing is not a risk – it provides some additional information. The particular 'micro-risk' associated with that bug has been eliminated when corrected. However, your perception of other, related risks may be higher because you might have found a bug in code that previously, you trusted.

Unless there is a potential for loss, then also, there is no risk. Unless I put money on a horse to win, I'm not going to enjoy a horse race, because if I don't put money on a horse I don't care who wins, there's no gain, no loss, there's nothing to worry about. If an undesirable event is trivial, then who cares whether it happens or doesn't happen? It is not a risk.

All our experience of software projects tells us that they are inherently risky. As almost every practitioner has directly experienced, the probability of success in a software project is less than 100%. These projects are inherently risky because there's great uncertainty as to whether we will achieve our desired goal (on time and within budget).

The word risk comes from the early Italian word *risicare*, which means, "to dare". In this sense, risk is a choice that we make, rather than an inevitability. If we call someone daring, they would probably take this as a compliment because we usually admire people who are daring. But daring people take risks; we call them daring because we are more risk-averse than they are. If projects wish to be daring, one of our roles as testers is to inform our managers of the risks they are taking. If managers are ignorant of the risks they take, they are not daring; they are simply uninformed.

Of course, not every risk can be predicted ahead of time. As we proceed through the design, development and testing, new threats to our objectives appear from perhaps unforeseen or unusual directions. As new risks are identified, testers should be prepared to adjust their test plans to account for these new risks where appropriate. Other risks should be reported back to management perhaps because they raise issues that only management could address.

1.4 Three types of software risk[2]

There are three types of software risk.

Project risk: These risks relate to the project in its own context. Projects usually have external dependencies such as the availability of skills, dependency on suppliers, constraints

[2] This section has been extracted from reference [4] Note that the term 'Product' is used in the book, and reproduced here.

such as a fixed-price contract, or fixed deadlines. *External* dependencies are project management responsibilities.

Process risk: These risks relate primarily to the internals of the project and the project's planning, monitoring and control come under scrutiny. Typical risks here are underestimation of project complexity, effort or the required skills. The *internal* management of a project such as good planning, progress monitoring and control are all project management responsibilities.

Product risk: These risks relate to the definition of product, the stability (or lack) of requirements, the complexity of the product, and the fault proneness of the technology which could lead to a failure to meet requirements. Product risk is the main risk area of concern to project staff that test.

Using the descriptions above, we can associate project and process risks with the management of logistics. These risks relate to how the project is managed overall to its goal. Product risk is associated with the quality of the products of the project.

How many risks did your management identify in your last project? Most projects have between, say, ten risks (for a small project) or maybe fifty for a large programme of projects. Typically, project managers need to manage perhaps twenty risks or so and risk reviews can be included in weekly project progress meetings. From the point of view of project management, managing risks using the standard methods and tools works fine if you have ten to thirty risks.

Testers focus on product risk. If you define product risks as the potential for a product to fail in some way, however, in how many different ways could products fail? Is it ten or twenty or is it closer to one hundred thousand? Of course, there is an almost infinite number of ways in which a product could actually fail. If you really had the time and imagination to work them out, maybe you could identify them all, but for practical purposes, there is no limit.

Testers focus on managing product risks at a lower level, basing their strategy for testing, incident management and progress reporting on those risks and a means of identifying both the benefits available for delivery and the risk of releasing early during test execution.

Specification, design and development methods may minimize the likelihood of failure in products but software testing is the main tool for identifying modes of failure and increasing our knowledge of risks in software products that exist. Testing has a key role to play in addressing software product risks.

Before we go much further therefore, we need to define both products and testing in their broadest sense.

1.5 What is a Product?[2]

A product is a final or interim output from a project activity, i.e. any document, procedure, software component etc. generated by a project in the course of its execution. All projects generate many documents and other deliverables. Products are often separated into management and other, more technically oriented products. Alternative terms such as work-products or artefact are used, but we will stick to the term product in the rest of this paper.

Management products include project initiation documents, project plans, quality plans, stage plans, resource plans etc. These products are mainly the concern of project management.

(The following are all technical products and are associated with product risks).

Documentation Products are "specification" type documents such as user requirements, designs, specifications, user documentation, procedures etc.

Software Products include custom built or bought-in components, sub-systems, systems, interfaces etc.

Infrastructure products include hardware, computer and network operating systems, database management systems, middleware etc.

Testware products are items such as test strategies, plans, specifications, scripts, test data, incident reports, test summary reports etc.

Other products include conversion plans, training, user guides etc.

The quality of the technical products gives rise for concern as faults in these products can cause failures in the final system. The purpose of testing is to detect these faults (so they can be eliminated) and reduce the risk of failure in production.

1.6 What is testing? [2]

Over the years, quite a few definitions of testing have been promoted and these have often been based on one or other single objective such as "finding faults". Myers in particular, promoted this single-minded objective in his influential book, The Art of Software Testing [6]. A later view from Bill Hetzel in his book, The Complete Guide to Software Testing [7], suggested testing is a broader and continuous activity that takes place throughout the development process.

Testing is a necessary information gathering activity required to evaluate our (or others') work effectively. In our experience, testing has several distinct objectives and as testing progresses through the lifecycle, the emphasis on these objectives changes. There appears to be no definition of testing in terms of a single test objective that can be complete. **A test is a controlled exercise having (potentially) several objectives.** The objectives of a test might include more than one of the following:

- To detect faults in the product under test so they can be removed.
- To demonstrate that the product under test meets its requirements.
- To gain confidence that the product under test is ready for use (or reuse).
- To measure one or more quality attributes (e.g. performance, usability, reliability, accuracy, security etc.) of the product under test.
- To detect faults in the baseline for the product under test.
- To provide data so that the processes used to build products can be improved.
- To provide data on the risk of release (and use) of the product under test.
- To provide data on the (potential) availability of benefits if the product under test is released.

Clearly, there are overlaps in these objectives. For example, if you find and eliminate faults and demonstrate a product meets its requirements, you would probably also have enough data to assess the risk of release of the product and have confidence to do so. We advocate the view that the ultimate objectives of a test strategy are the last two in the list: to provide enough information to judge whether the benefits of release of a system outweigh the risks of release. This is the essential purpose of the risk-based approach to software testing.

What kind of tests should you consider? Testing includes both static and dynamic tests.

Static tests are tests that do not involve executing software. Static tests are mostly used early in the development lifecycle. All human readable deliverables, including project plans, requirements, designs, specifications, code, test plans and procedures, can be statically tested. Static tests find faults, and because they usually find faults early, static test activities provide extremely good value for money. Static tests include reviews, inspections and structured

walkthroughs, but early test preparation can also be regarded as a static test activity, as the process of test design finds faults in baseline documents.

Dynamic tests are tests where you execute dynamic transactions of the software. Dynamic testing is appropriate for all stages where executable software components are available. The traditional component, link, system, large-scale integration and acceptance tests are all dynamic tests. Many non-functional system tests such as performance, volume and stress tests are also dynamic tests.

The test activities in a project are associated with all products and all stages of development. Every activity that involves an evaluation of a product we will call testing. Test strategies focus on the product risks of concern, and you construct your tests to pinpoint the particular product faults that can occur. Tests that focus on fault detection are more effective and more efficient, being less likely to duplicate each other. This is the straightforward approach to test strategy advocated as 'best practice'.

The main deliverable from testing activities, be they reviews, inspections, component, system or acceptance tests, is information. Tests (and the people who test) do not change the products under test. Ultimately, all a test can ever do is give you an insight into whether a product is complete, how it works, whether it meets the goal of the project or has flaws. Although much of the information generated is at a low level, e.g. tests passed, faults found etc. our proposal in this paper is that this low level information can be analysed to provide Project Intelligence. That is, management level information on the status of deliverable and progress towards the project goals.

1.7 Goals, Risk, Testing and Project Status

Let us explore the relationship between project goals, risk and testing summarized in Figure 2.

We have already defined a risk as any threat to a project goal. The deliverables of our project represent intermediate goals and the final deliverable (e.g. a working system, installed, with trained users etc.) is our final goal.

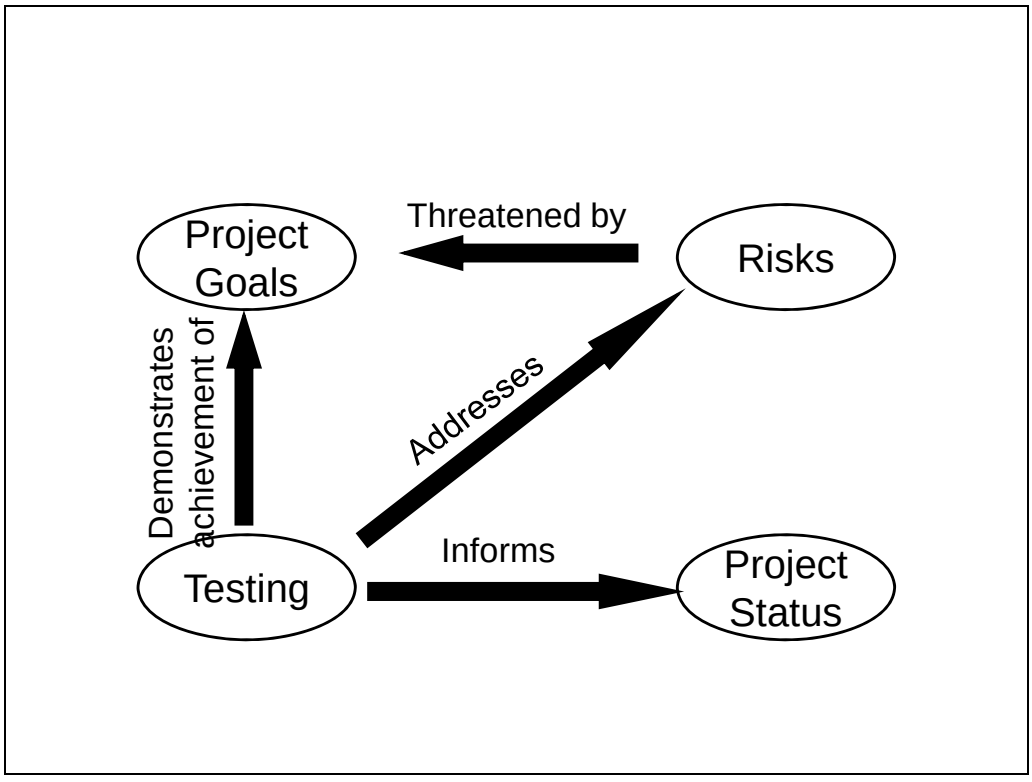

Figure 2 Relationship between goals, risk, testing and project status

The risks to the key deliverables that represent our goals are usually addressed by testing. Testing will either demonstrate our concerns are groundless (if all tests pass), or that our concerns were justified, that the risk has materialised, and we will have evidence that will help us to diagnose the faults to be corrected in the deliverable.

Of course, where our tests pass, and we are satisfied that our testing is comprehensive, we can interpret those test passes as evidence that our deliverable is complete and meets our needs. In fact, tests demonstrate that the goal (intermediate or final) has been achieved. Testing (or test results) gives us the evidence either to show our goal is met, or where we need corrective action to achieve that goal.

With this philosophy, a project manager can regard test activities as the key source of information on the status of the project goals. Of course, the project manager also wants to know at intermediate stages how much budget has been spent on a deliverable so he can track it. It is important to know the effort spent on the creation of a deliverable (compared to budget allocated), but effort spent does not equate to value delivered. Timesheets cannot provide information on the status of deliverables, as the status of a deliverable is only known when it has been reviewed or dynamically tested and deemed to have been delivered.

This is a critical distinction: that *deliverable status (and therefore project status) cannot be inferred from the status of the budget and schedule alone*. Project managers need to track progress through the schedule and the status of deliverables throughout the project lifecycle to be in full control.

2 PROJECT MANAGEMENT 1.0.1 MANAGE ACHIEVEMENT, NOT ACTIVITY

Before we look at Project Intelligence in a little more detail, we need to revisit a basic project management principle, which is:

“Manage achievement, not activity”

Let’s take a rather mundane example. Suppose we were fed up with how untidy our teenage son leaves his bedroom. The state of the room is so bad, the floor is covered with clothes, CDs, books and rubbish we cannot see the carpet. The window is closed, there is no fresh air, and we feel sick when we are in the room. The time has come to sort this out, so we ask our son to sort it out.

In a firm voice, we ask our son to clean up and tidy the room. We have been assertive, and absolutely clear as to the task. We want him to clean up and tidy up the room. But, there is a problem. We have assigned an activity to him, but it is unlikely that the room will be clean or tidy enough, when he reports the task is finished. For a boy to live in such a mess, his standards are obviously lower than ours, so it is unlikely he will work hard enough to meet our (higher) standard. It is conceivable, but unlikely, that he would work so hard that he exceeded our standards. Very unlikely!

The key flaw in our approach is that we have assigned a task, but not set any standard for the work. We do not create a clear performance expectation. It is difficult to gain commitment to the task, as it is unbounded. When he is ‘finished’ we can’t recognise success or failure, because we haven’t said how clean ‘clean’ means. Essentially, we cannot decide how to act after the task is complete – do we reward him or punish him? How could either action be seen as fair?

In this case, a better approach would be to say something like, ‘clean up your room, so I don't feel sick when I enter it!’ Here, we have assigned a goal: clean the room enough for me to walk into the room without throwing up. We now have the potential for better performance. The performance expectation is clear and we have an opportunity to develop some commitment to it. If we throw up when entering the room everyone will agree that the

standard has not been met. Rewards (and punishments) have a much better chance of being perceived as fair because the standard was clear.

Now, this is a silly (but realistic) example, but we continually ask our project teams to perform tasks without setting a standard for the work. Suppose we ask a programmer, 'Develop this component to this specification'. The assignment is clear; we want him to write some code in accordance with the specification. The programmer must perform an activity (or two): read the spec and write the code.

Given this brief, however, the work will probably not be up to standard when he reports the task as finished. It is possible, but unlikely, that he will do too much work. The key flaw is we do not create a clear performance expectation. Because of this, we cannot gain commitment to the assignment. When finished, we can't recognise the difference between success and failure. We can't decide how to act after the activity is complete: to accept the code or reject it?

So we must assign a goal, rather than assign a task. A better instruction to the programmer might be 'develop this component and show me (the test results) that it meets the specification.' Here, we assign a goal: provide test evidence that shows that it meets the specification. The evidence of achievement is provided by the test results. If we don't see any evidence, or there isn't enough evidence to convince us, we reject the code. We don't particularly have to understand or even see the code. We don't need to understand what the code is meant to do. All we need is to see the test results. In this case, the acceptance or rejection decisions will be seen to be fair.

2.1 Development and Test Activities are Managed in Pairs

Now, this is a rather trivial example, but it illustrates a common failing in many projects. The task of creating documents or software artefacts that are key project deliverables (and which ones aren't?) is often managed purely in terms of time allocated and effort expended. When time runs out for the task, the task stops, whether the product is complete or acceptable or not. We see this in projects all the time.

It is far better to assign tasks for the creation of a deliverable and the testing of that deliverable at the same time. There are then two products: the deliverable and the test results for that deliverable. These tasks should always be scheduled in sequence (or on occasion with some overlap). But the goal for the pair of tasks is the delivery of test evidence. The first task cannot be deemed to be complete until all tests pass. If the deliverable is incomplete, or tests fail or some tests have not been run, the goal of the two tasks has not been met.

This approach to work breakdown, is implicit in project management methodologies such as PRINCE. Test activities are just as important as the development activities they shadow. The development activity creates the deliverables, but the test activities provide evidence that intermediate and final project goals are being met.

2.2 Testing Provides the "Intelligence" to Support Decision Making

We have argued that the deliverables of our project represent intermediate or final project goals. Towards the end of any development activity for a deliverable, the key questions for the project manager are:

- Is the deliverable available?
- Is the deliverable complete?
- Does the deliverable meet requirements?
- Is the deliverable acceptable?
- Is the deliverable reusable?

The purpose of the test activities is to answer these questions as soon as possible after the deliverable is available for test. Possible status of the tests could be:

- Tests planned, but not run – the deliverable might not be available (or so poor that it cannot be tested) – the development activity is not finished yet.
- Some tests fail – the deliverable has flaws: the deliverable might not be complete, the quality is poor, it does not meet requirements – the goal is not met; we have information to reproduce failures, diagnose and correct faults.
- Some tests pass – the deliverable is partial, some aspects work – the goal is partially met.
- All tests pass – the deliverable is complete, the goal is met.

We are now able to manage project goals using the mechanism we use to address product risks. We make explicit the standard of work for all deliverables by defining the acceptance criteria for all our deliverables and the approach to all testing in our test strategy.

Overall, the value of Project Intelligence depends on how well we achieve and communicate the following four aspects.

- Goal-based testing
- Risk-based testing
- Test Coverage.
- Incidents Management.

We will explore these concepts further in the following pages.

2.3 The W-Model of Testing[2]

Figure 3 presents the V-Model of testing. The V-model promotes the idea that the dynamic test stages (on the right hand side of the model) use the documentation identified on the left hand side as baselines for testing. The V-Model further promotes the notion of early test preparation.

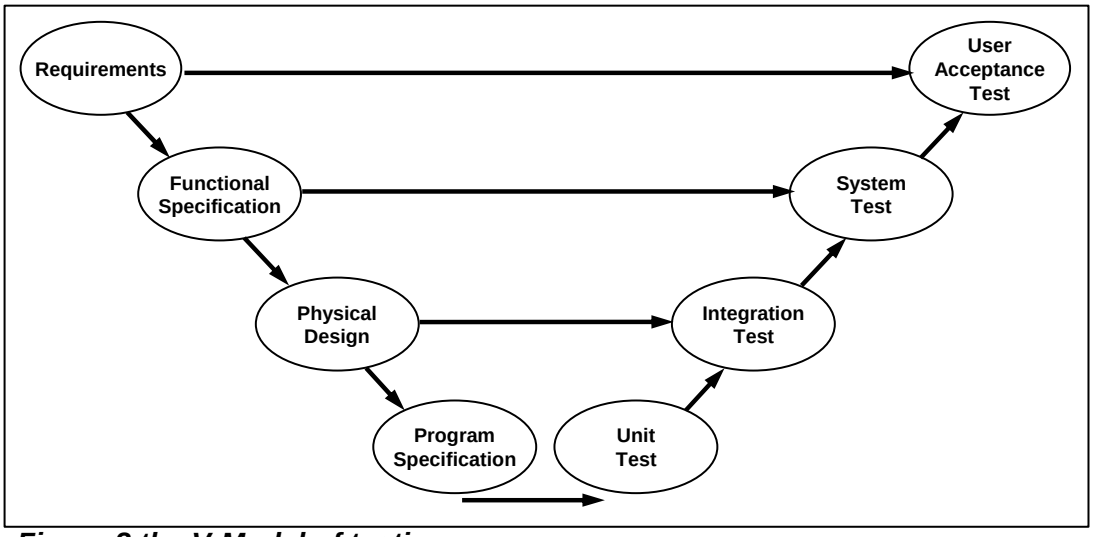

Figure 3 the V-Model of testing

Early test preparation finds faults in baselines and is an effective way of detecting faults early (see Figure 4.2). This approach is fine in principle and the early test preparation approach is always effective. However, there are two problems with the V-Model as normally presented.

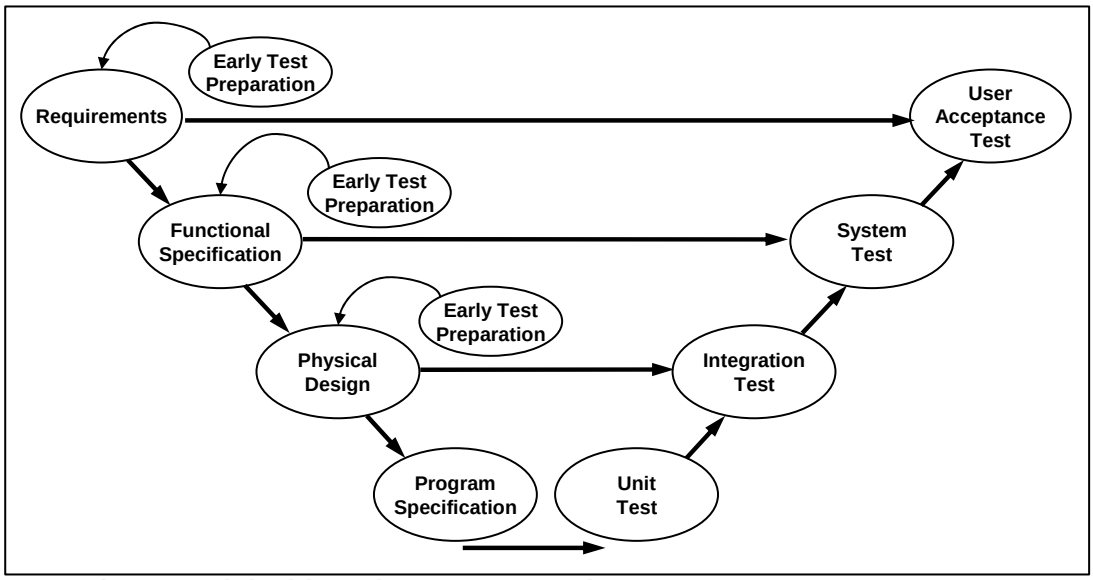

Figure 4 the V-Model with early test preparation

Firstly, in our experience, there is rarely a perfect, one-to-one relationship between the documents on the left hand side and the test activities on the right. For example, functional specifications don't usually provide enough information for a system test. System tests must often take account of some aspects of the business requirements as well as physical design issues for example. System testing usually draws on several sources of requirements information to be thoroughly planned.

Secondly, and more important, the V-Model has little to say about static testing at all. The V-Model treats testing as a "back-door" activity on the right hand side of the model. There is no mention of the potentially greater value and effectiveness of static tests such as reviews, inspections, static code analysis and so on. This is a major omission and the V-Model does not support the broader view of testing as a constantly prominent activity throughout the development lifecycle.

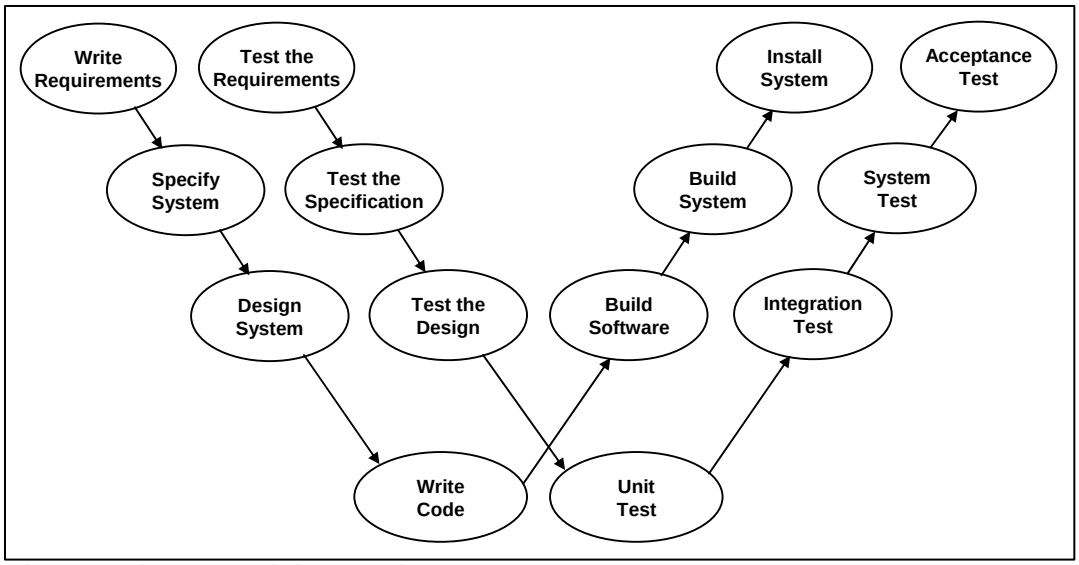

Figure 5 the W-Model of testing

Paul Herzlich introduced the W-Model approach in 1993 [5]. Figure 5 the W-Model of testing attempts to address shortcomings in the V-Model. Rather than focus on specific dynamic test stages, as the V-Model does, the W-Model focuses on the development products themselves. Essentially, every development activity that produces a work product is "shadowed" by a test activity. The purpose of the test activity specifically is to determine whether the objectives of a development activity have been met and the deliverable meets its requirements.

In its most generic form, the W-Model presents a standard development lifecycle with every development activity mirrored by a test activity. On the left hand side, typically, the deliverables of a development activity (for example, write requirements) is accompanied by a test activity "test the requirements" and so on. If your organization has a different set of development stages, then the W-Model is easily adjusted to your situation. The important thing is this: **the W-Model of testing focuses specifically on the product risks of concern at the point where testing can be most effective**.

3 PROJECT INTELLIGENCE

3.1 Using the Status of Deliverables to Manage Your Projects

There's a familiar saying which goes, "if you don't know where you are, a map won't help". It seems obvious, but having a map is only useful if there is a common point of reference between a visible landmark and the map. Without a landmark, the map is useless, no matter how detailed or accurate. By a similar argument, if you do not know the status of your deliverables, a project plan, no matter how detailed and well thought out, won't help either.

A project plan sets out the staged activities required to produce deliverables (to meet intermediate goals) culminating in a final deliverable which meets the overall business goal. We create staged plans specifically to meet intermediate business goals. This is partly to make management of the project easier, partly to reduce risk (by making the deliverables smaller in scope). The stages also exist because we identified dependencies between those deliverables. There is a natural sequence of delivery of business requirements, technical designs and code, for example. The format and content of deliverables usually depends upon the format and content of previous, predecessor deliverables.

To reduce the risk of rework, we include test activities (reviews, inspections or dynamic tests) to identify faults for correction and the outcome of each test activity is a stable, agreed deliverable. Completion of the test activity for a deliverable is the trigger for the project to "change state". The practitioners start work on the next stage of work on the next deliverable.

You cannot know the status of your deliverables, if the test for that deliverable has not started; you have no evidence to say the deliverable is complete, correct, and consistent and so on. Given that all deliverables are likely to be faulty when first written, the likelihood is that all deliverables are unacceptable before testing has completed. *But the real problem is, we do not know for certain, whether the deliverable is faulty or what those faults are*.

If a deliverable has not been tested, it is impossible to know its status. Without knowing its status, a project manager cannot make a sensible decision:

- To continue working on it or to deem it complete?
- To accept it or reject it?
- To start the next phase of work, or to reassign resources to another task?
- To pay the supplier, or withhold payment, or take legal action?
- To install a system and handover to users or to hesitate?
- To trust the plan, to negotiate for more resources, or a slippage?

In every case, the project manager is faced with a dilemma because they do not have sufficient information to make a reasoned decision. What usually happens is that the project

manager agrees to proceed to the next stage, or refer the decision to the customer or other stakeholders. Without the information from test, it is hard to manage a project at all. Treating testing as an activity essential to deliver management information to drive progress through the plan seems to be an obvious approach but few project managers are that committed to testing.

After all, they might argue, if the deadline for delivery has passed, surely it is better to proceed to the next stage and play catch up later. If we start the next activity on time, surely there is a chance we will finish it on time? If we delay, the deadline will never be met. Apart from that, if we don't start, those developers will be sat around doing nothing anyway. We have to take advantage of every man-hour we can.

The resistance to adopting a more disciplined approach based on the recognition of the role of testing seems unbreakable. The aim of Project Intelligence is to engage stakeholders and management in the management of the risks to the goals of the project and take advantage of test activities to inform decision making. We have already made a case for using testing; we expand the notion of testing a little to define the Project Intelligence concept. Stakeholders and management need intelligence to make decisions, command and control their project, and we will use a military analogy to explain how.

3.2 A Military Analogy

Imagine, for a moment, how military intelligence personnel are utilized in a military campaign. On a battlefield, their role is somewhat different than the traditional infantry. Military intelligence provides background information before the campaign is planned in detail. The intelligence folk infiltrate the enemy perhaps, intercept their messages, take aerial photographs, gather information and analyse it to provide intelligence on the enemy dispositions, their strengths and weaknesses. As the campaign progresses, intelligence is continuously gathered as troop movements, fighting and the capture and loss of territory occur. Intelligence helps commanders to understand where to deploy troops, where the weakest point is, and where the greatest threat to success lies. Intelligence is the nervous system of the command. It helps them understand how progress is being made, and what challenges lie ahead.

The infantry's role is primarily to implement the plans set out by the commanders. They embark on systematic manoeuvres to eliminate opposition, capture materiel and territory. Unlike the intelligence folk, they make things happen. But in the confusion of war, they depend on orders based intelligence reports and feed back no small amount of intelligence themselves. The direction the campaign takes and the understanding of how progress is being made is all based on the intelligence gathered before, during and after the campaign is over.

We can use the military analogy to reflect some of the activities of testers. The overall project plan may be fixed and designed to meet business goals. However, every project faces uncertainty in the nature, number and severity of faults found in deliverables. An early risk assessment indicates where the risks of most concern lie, and this influences where the focus of attention of the testing should be. The test strategy is an intelligence gathering strategy. The test strategy sets out where key intelligence must be gathered and how, by who, where and with what tools.

Intelligence gathering starts with the goal and risk analysis, but continues throughout the project, and involves early reviews, component, system and acceptance testing of the key deliverables. The major output from testing is detail on faults, but also evidence of success and requirements being met. Ultimately, testing provides the evidence of the overall project goal being achieved.

Just like the infantry, developers are more concerned with delivering the products of the project. They work to the plan, but will also take account of the intelligence being provided by the testers. The risk analysis identifies features of the system that are deemed risky.

Developers might take a more conservative approach with those aspects of the system. Developers gather their own intelligence when the component is tested. They act on this intelligence immediately and correct the faults they find themselves. Some of this intelligence is fed back to management with a view to refining the overall test approach now, and possibly the development process for the future.

When testers get the system for testing, they feed back intelligence on the features that do not work. Developers are driven by that knowledge and make changes. As the faults are corrected and more tests pass, a picture emerges of where the final challenges remain. From the point of view of management, they begin to receive clear evidence of the final system and to what degree it meets the overall business goal.

So, Project Intelligence, provided by testers helps the stakeholders, the project manager, developers and users by providing continuous feedback on the status of project deliverables.

3.3 Where is Project Intelligence Obtained?

Just like the project as a whole, the test strategy, being an overall approach to intelligence gathering, depends on the business goals. If the goals are vague, then it will be difficult to understand whether and how the project has met those goals at the end of the project. (If the goals are vague, one must speculate on how easy it is to design and build a system to meet those goals in the first place, but that's another story). Clearly, the business goals should be defined in the business case made for the project as a whole.

Figure 6 shows how Project Intelligence starts with a risk assessment. The Master Test Plan sets out the overall test approach in terms of stages, test objectives, techniques and so on. The test objectives for each stage of derived from the risks in the risk assessment. The test process is designed to systematically inform the reassessment of risk at the end of each test stage. Strictly, the fours activities (test planning, preparation, execution and release) in the diagram are repeated for each test stage.

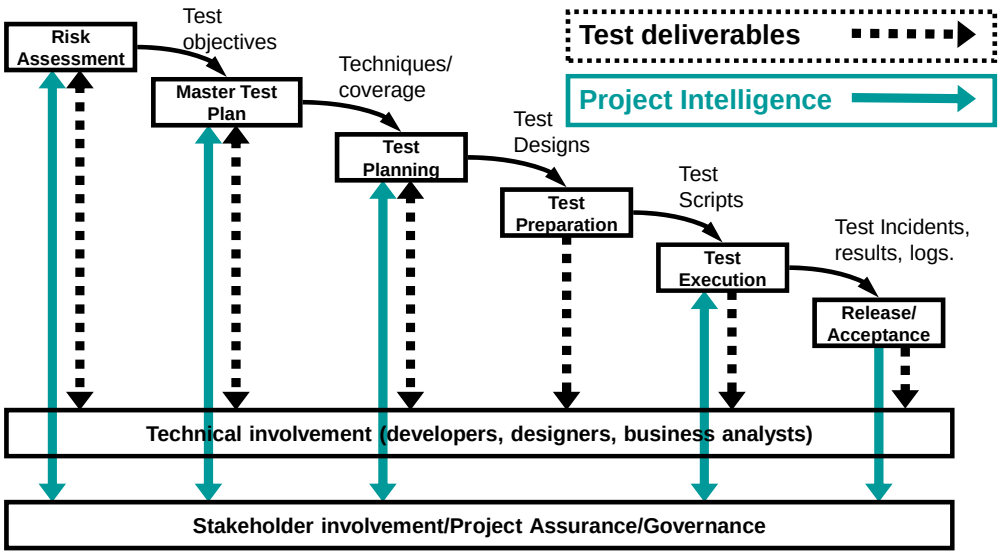

Figure 6 Sources of Project Intelligence

Every activity in Figure 6 involves input from and output to other practitioners in the project. Authors of documents, developers of code, are involved in the planning and definition and execution of reviews and tests, the incident reports generated as well as the decision to accept the deliverable under test. These interfaces involve technically oriented issues.

3.4 Stakeholder Engagement

In parallel with these technical exchanges between the practitioners, there is a higher level interface with the project stakeholders. At this level, the stakeholders are involved in the risk analysis and formulation of the overall Master Test Plan. They are invited to review high level plans of the test stages to ensure they are comfortable with the coverage of the tests. Their priority in this activity is to be satisfied that the risks of most concern to them will be addressed.

Although not directly involved in test preparation, they do have visibility of the outcome of test execution. Test reporting should provide specific evidence of risks being addressed, deliverables being accepted and goals being met. Tests that fail provide insight into where the problems are so that remedial action can be taken. Of course, the release decision is primarily that of the stakeholders and management. At this stage, the decision makers need intelligence presented in a straightforward way: that deliverables are acceptable, goals are met.

Reference [4] sets out a comprehensive consensus-based approach to using risk to drive the creation of a test process. We will not cover that further in this paper.

3.5 Risk, Test Objectives and Coverage

It is straightforward enough to articulate a test objective by examining the description of a risk. Transforming the language used in a risk definition into an objective for testing is fine, but in most test objectives, there is no mention of the quantity of testing to be done. What is missing is a notion of coverage. The question is this, “how much testing (how many tests) are required to provide enough information to stakeholders that a risk has been addressed”?

Let’s look at functional testing first. There are excellent textbooks [6] and [8] and a test standard, BS 7925-2 [9] that describe test design techniques. There are very well documented coverage measures that relate to both functional (black box) and structural (white or glass box) test design techniques. For many techniques, there are relationships between them. For example, the boundary value analysis (BVA) technique will generate more tests than the equivalence partitioning (EP) technique. In fact, BVA subsumes EP because it generates all of the tests that would be generated by EP plus a few more, in most circumstances. BVA is a *stronger* technique, so we might use that technique where concerns about failure are higher. EP might be more appropriate where the concerns are lower. We have sometimes suggested to our clients that BVA is appropriate for functional system testing and that EP is appropriate for user acceptance testing. Many functional test techniques (e.g. decision tables, state-transition testing, syntax testing etc.) do not fit into such an ordered hierarchy however. These (and other) techniques have specific areas of applicability. The tester should explain how these are used, the potential depth and consequent cost of using them to the stakeholders.

Most structural test techniques are based on the path-testing model and there is a more inclusive ordered hierarchy of such techniques. Statement testing, branch testing, modified condition decision testing and branch condition combination testing are increasingly thorough (and expensive) with statement being the weakest and branch condition combination being the strongest techniques, as it subsumes all of the others. Although these techniques are increasingly expensive, there is little data available that compares their cost-effectiveness. British Standard BS 7925-2 [9] presents comprehensive definitions and a discussion of all of these techniques. The “pre-BSI” version of this standards material is also available from the www.testingstandards.co.uk web site, free of charge.

Some test techniques are less formal, or not yet mature enough to have defined coverage levels. Coverage targets for the configuration testing and ethical hacking as a means of detecting security flaws are likely to be subjective.

Where the test budget is limited, the tester should indicate how much coverage (in one form or another) could be achieved in the time available and ask the stakeholders whether that is acceptable. In the case where there is no budget stated, the tester might have to propose coverage targets of increasing thoroughness (and cost). The stakeholders must then take a view on which of these coverage targets will be adequate. There are many techniques discussed in reference [4] that do not have formally defined coverage measures. Where it is possible to conduct such tests with varying thoroughness, the testers and stakeholders need to agree how much "coverage" is enough.

In summary, throughout test planning, the level of coverage should be enough to satisfy the stakeholders that enough testing will be performed, and that the test execution reporting can be trusted: the stakeholders can assume a completed test has addressed risks of concern and when reported at a high level, the stakeholders can be assured that goals are being met.

3.6 Project Intelligence Has a Shelf-Life so Must be Delivered Early

Needless to say, Project Intelligence has a shelf life and this is particularly true during test execution and reporting. In this context, we are including all testing, both reviews and dynamic testing. Any project manager will recognise their project is in trouble if deadlines continue to be slipped, development activities continue while testing proceeds, and late in the project continuous rework and refinement of requirements, designs and other specifications is common.

Early intelligence acted upon promptly can make or break a project. Early reviews of requirements and designs should not be seen as simply a task to be gotten through. These activities provide vital intelligence on the state of the foundations of the project. If all subsequent work on the project depends on requirements or a functional specification, then it is inevitable that faults in these early deliverables, if left alone, may cause catastrophe later on. Issues reported in early reviews should therefore be taken very seriously – project success may depend on an early resolution.

Now, it may be that, when requirements have not yet stabilised and the time to start the design phase has come, the pressure to start design may be irresistible. In this case, the outcome of a requirements review provides a critical insight to the status of that document. A brief analysis of the issues reported may indicate that some aspects of the requirements are stable. Other aspects might be far from complete, or very faulty. Here you have clear information indicating where design may proceed immediately, and where perhaps some design activity may be delayed. Even if it is impossible to design around these 'hotspots' at least it is possible to start setting expectations of where future troubles in development and testing may arise. There may still be time, this early in the project, to reorganise work to address the new risk that has emerged.

Later in projects, when it comes to dynamic testing, test reporting should focus on risks being addressed. Rather than testing being perceived to report only bad news, when problems are encountered, a better approach is to regard the list of outstanding risks (and not yet delivered goals) as a checklist for progress-reporting. If tests are prioritised by risk (or goal) then progress through the test plan can be reported on the basis of risks addressed, and goals delivered.

Partly, this aligns test activity with the goals of the project (to address risk, deliver goals). But it also sets a clear objective for the testers: Get through the test plan and demonstrate working software – as a priority. This objective does not conflict with the lower level goals of testing – to detect faults. Rather, it simply encourages testers to report progress in a way more

consistent with the other activities of the project. No longer will testers appear to be 'pulling in the wrong direction'.

Again, the issue of test reporting is addressed fully in reference [4].

3.7 Organisations Must Learn to Appreciate Intelligence

The discussion of Project Intelligence in the previous sections makes it sound as if the case for creating intelligence is overwhelming, and that intelligence must always be self-evidently valuable to management. But there is still a cultural challenge to be overcome. In general, project managers, while accepting that testing has a valuable role, are less inclined to use the output of testing, particularly in the early stages to influence their decision making.

Slippages in early deliverables mean that review time is cut – after all, it must be more important to deliver than do a review? Problems reported in reviews and testing, particularly early on, can always be overcome by working a little harder, or uncertainty in requirements can be sorted out when the coding starts, or that unreliable components can be identified and fixed when the system is fully assembled and system tested. After all, that's why we have testers, isn't it?

This mentality is still widespread in the project management community. Partly this is because projects are driven by fixed deadlines and budgets. But the majority of project managers also have a development background. They remember well how difficult it was to work in pressurised projects, but when the deadline approached, they could always put in some extra hours to save the day. Many organisations rely on 'hero cultures' to deliver, and if deadlines are fixed, 'our best people will always get us out of trouble'.

Hero cultures are common and they survive and usually deliver, but not because of superheroes. Rather, the cultures tend to over estimate the cost and timescales of projects in the first place. They have learned to live with uncertainty, and compensate by over budgeting. Success may be assured, because the resource allocated to projects is 20-50% higher than it really need be and/or problems can be fixed minutes before release – untested.

If budgets and timescales are truly fixed, Project Intelligence comes into its own. Early intelligence gives warning of problems ahead. Those problems are inevitable, if development activities carry on regardless. Surely it is better to have advanced notice, so expectations can be set: that system testing will be troublesome; that the users may experience failures in some areas; that the scope of delivery may have to be cut. Surely it is better to learn of these problems ahead of time. Perhaps a case really can be made to slip deadlines or obtain more resource or budget in good time? Then, perhaps your project might actually be under your control, rather than drifting into failure as if in a death march.

The willingness of the organisation to accept and act upon Project Intelligence is a key challenge for projects.

4 SUMMARY

This paper sets out some key principles for project management. Managers should be focused at all times on the goal and the risks that threaten them. Projects need to manage goals, not activities to ensure that work is done, deliverables are delivered and they are acceptable.

The Project Intelligence concept aims to give project and stakeholder management visibility of project risks, their intermediate and final goals and progress towards them.

Project and stakeholder management are the main customer of Project Intelligence. Early goal and risk analysis enables management to have influence over the way projects are tested and how the gathering of intelligence will be performed. Intelligence is gathered throughout the project lifecycle, from early reviews to final acceptance testing. The intelligence gathered, enables project and stakeholder management to judge how and whether progress is being

made. Further, when problems do arise, the intelligence gives an insight into how the current challenges can be overcome. No longer does the project manager have to make a case for slipping a task, accepting or rejecting a deliverable or releasing into production. The intelligence should make those cases self-evident.

One way of looking at the test process is as part of the nervous system of your project. The purpose of the test process is specifically designed to gather intelligence. At a low level to aid practitioners to correct faults in deliverables; at a high level to assure management that enough testing has been planned and executed and demonstrate progress toward the overall project goals.

What's new in all this? Well, goal and risk-based testing are relatively new, in terms of their being documented. But this approach is beginning to catch on in testing communities who are finding that their management appreciate being more involved and receiving more informative test reports. Project Intelligence demands that testers engage management and communicate their plans and progress reporting throughout the lifecycle in terms that managers understand – the language of business goals and risk.

Management must learn to appreciate what Project Intelligence can do for them. It dramatically enhances the value of testing to their projects and enables them to manage their projects with more confidence. Project Intelligence gives management better visibility of progress and enables them to make significantly better-informed decisions.

Finally, Project Intelligence is usable in ALL development methodologies. There is an overhead perhaps in the early goal and risk analysis, but this is simply better test practice anyway. The intelligence concept integrates well with risk-oriented project management methods, as these methods pay more attention to the test activities associated with all project deliverables. Project Intelligence simply aims to enhance the understanding of the test process and improve the reporting done by all test activities.

5 REFERENCES

1 Goal Directed Project Management, Erling Andersen et al., Kogan Page, 2004.

2 Maximising the Realization of Benefits Management, Gerald Bradley, Sigma UK, 2002)

3 Managing Successful Projects with PRINCE2, CCTA, the Stationery Office Books, 2002.

4 Risk Based E-Business Testing, Gerrard P. and Thompson N., Artech House, 2002.

5 The Politics of Testing, Herzlich, P., Proc. 1st EuroSTAR conference, London, UK, Oct. 25-28, 1993.

6 The Art of Software Testing, Myers, G. J., John Wiley, 1979.

7 The Complete Guide to Software Testing, Hetzel, W. C., QED, 1983.

8 Software Test Techniques, Beizer, B., Van Nostrand Reinhold, 1990.

9 BS 7925-2:1998 Standard for Software Component Testing, British Standards Institute, BSI, 1998.

This paper can be downloaded from the Gerrard Consulting web site:
http://gerrardconsulting.com