QUALITY AND PRODUCTIVITY - HOLISTIC SOFTWARE PRODUCT ASSURANCE

W Blackburn, R Brown, M Parsons

Logica UK Ltd, Stephenson House, 75 Hampstead Road, London NW1 2PL, UK, Tel + 44 0171 637 9111

ABSTRACT

The perception that quality and productivity are alternatives has held back the software industry for years, but why? Other engineering disciplines have reconciled the apparent conflict.

We can optimise the development process and ensure it matches the organisation's business goals by exploiting a *holistic*[ϒ] approach to software PA. Its main themes are objectivity and reliance on information from modern quality control techniques. This is supplemented with planning and configuration management data.

Its goal is to answer the Project Manager's perennial question "Are we ready to ship the system?" Releasing software into operation early may reduce our direct development costs, but could cost a fortune in warranty, long term maintenance and corporate image.

INTRODUCTION

This paper illustrates one aspect of the Holistic software PA approach using data from a project in the space industry.

It examines the misconception that quality and productivity are alternatives and proposes a six step holistic approach. This allows us to measure the operational readiness of a software system and chart its growth at all stages of development.

QUALITY OR PRODUCTIVITY?

The project team have crafted an innovative, state of the art system and delivered it to the client. They wrote the specifications and reviewed them as stated in the project plan. They painstakingly coded, tested and documented the system and applied all the expected controls. They delivered late and exceeded their budgets. Quality and productivity are alternatives, the Project Manager needs to decide which option to take.

The software sector seems to accept this state of affairs as the norm, but we expect more from other industries. How many times have you heard the following excuse, "Sorry sir, your new car won't be delivered for another three months, integration of the engine with the body is taking longer than we anticipated"?. Any motor manufacturer who can't deliver a quality car and equal their competitor's productivity will not survive. The primary sustainable differentiator in a mature industry then becomes innovation.

WHY (W)HOLISTIC?

We can take a narrow view of quality using it simply to find and fix problems. This is of limited value. Quality will be sidelined when budgets and delivery dates are tight, leading to a "Quality Gap".

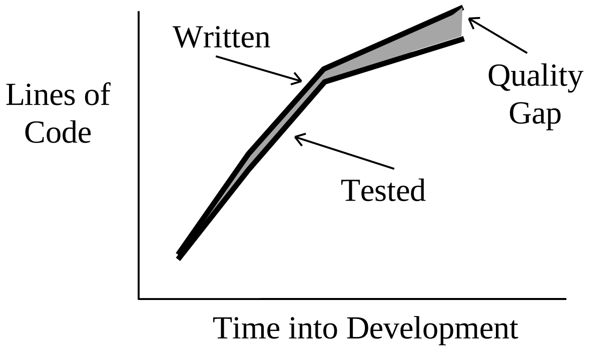

Figure 1 - Short-Term View of Productivity

Instead we must characterise "quality software" simply as the software that maximises the benefits of a system, in terms of the agreed business goals, taking the *whole life cost* into account.

Following a holistic PA approach to software, the conflict between quality and productivity never arises, they are synonymous.

A project manager has to define quality in terms of relevant business and project goals. So quality means tangible objectives like:

- 100% of user requirements can be traced to demonstrable system functions
- evidence collected during test and integration shows that it takes on average 4 hours to fix a defect (maintainability)
- <20% of software components have direct contact with the operating system (portability); etc.

Goals such as these determine exactly what metrics the project should be collecting.

Achieving quality means that the project is ready for roll-out - the two have become synonymous. The quality plan becomes more than a list of standards and procedures. It becomes an "Achievement Source Book".

This is a text book philosophy which many acknowledge, but few put into practice. The reason is equally simple. Quality systems are generally good at making sure the filing is well-organised and authorised signatures appear on the design document. They fail to

[ϒ] Holism - an idea that the whole is greater than the sum of its parts.

Logica

deliver what the project manager and customer need most; a quantitative means to assess when a system is ready to move out of development and into production - a measure called “operational readiness”.

By adopting a holistic approach in creating a quality system we can address this issue properly. There are six basic steps that need to be taken:

1. create a “Process Map” and use this to identify the requirements for software PA
2. refine the planning and tracking metrics
3. identify techniques available in the Product Assurance armoury
4. implement the techniques, gathering the agreed metrics
5. quantify the effectiveness of these techniques and identify critical success factors
6. use the metrics to determine operational readiness.

This paper looks at these steps and examines how two techniques, software inspection and testing, have been incorporated into a project’s process map. We study these techniques using data from a project within the space industry. The system is a real-time Ada application with stringent safety requirements.

THE 6 STEPS

Step 1 - Create the Process Map

Our business and project goals require that we do certain tasks - like analyse, design, code etc. The individual nature of each system to be built requires that we tailor existing standard process maps (lifecycles) to produce something relevant for our particular needs - to define a project process map.

The process map is a graphical representation drawn using a number of conventions. IDEF0 is a strong contender, offering a dedicated process capture notation.

Our business and project goals also point to the metrics needed to show that we have achieved our goals and have produced a quality product. We refine our project process map to show when and where metrics will be gathered. The metrics form the basis for process entry and exit criteria.

Once this basic process map is in place, it can be used to walk through the project's life to test the feasibility of the map. The map walkthrough can help test the process timescale estimates and dependencies.

It is at this stage that we select PA techniques to fit individual parts of the process map and so show us *how* to collect metrics, adding to the when and where determined earlier.

In our particular case, part of the definition of quality may have been something along the lines of:

“In a safety-related system, should a fault occur, the system fails to a safe condition, following the fundamental principle of ALARP.”

This implies that one aspect of quality can be defined in terms of an acceptable level of residual errors. As Low as Reasonably Practical - ALARP implies that the removal of all errors is not cost effective.

This suggests that our project process map must include a mechanism for collecting the number of errors detected and fixed against time.

Step 2 - Refine the Metrics

In defining our process map and metric set, we sought guidance from the “ami Handbook”. This indicates the three broad metrics groups relating to resources, processes and products. The primary metrics adopted were:

1. stability of requirements, design and code
2. inspection and test coverage
3. measures of error found that can be used to predict residual errors at exit from project phases or stages
4. effort to fix problems during development and maintenance and consequential damage
5. size of system (eg function points, count of design objects, “lines of code”) factored by complexity.

Step 3 - Identify PA Techniques

On this project, we employed a conventional selection of quality control techniques:

1. Static analysis
   a) manual inspection
   b) automated line counting, McCabe complexity analysis, standards checks
2. Dynamic analysis
   a) Unit testing
   b) Integration testing
   c) System testing
   d) Stress testing.

Of these, we will focus on the results of the formal software inspection process we designed and built into the project process map. The implication of the addition of a process like this to the map was assessed during the map's walkthrough. Asking ourselves how much does the process cost, how long does it take, what are the benefits in the short and long term?

Step 4 - Implement Techniques

People are not born with the ability to inspect or perform tests, they must be trained. This is one of the more time-consuming aspects of implementing the process. Inspections are deceptively simple in concept, but we must adhere to some basic principles to realise the benefits. Gilb provides a comprehensive reference on this topic.

The various techniques are defined in the development and quality plans and personnel trained in their use.

The use of a design method, in this case HOOD and a dependability programme, guided the inspection planning. A hazard analysis formed an intrinsic part of the process and identified key components for inspection.

We gathered data on the effort required in the inspection preparation and walkthrough stages. This allowed us to understand and refine the process. For example, the data showed that inspection effectiveness was strongly related to the inspection package size.

We found the optimum amount of code that can be checked by a trained inspection team is around 400 FIDSIs (format independent source instructions). This easily measured metric is now defined as one of the inspection process entry criteria.

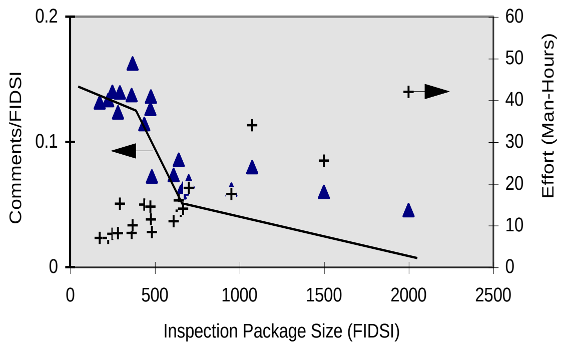

Figure 2 - Optimum Inspection Package

We minimised the effect of complexity by defining a ceiling for McCabe number.

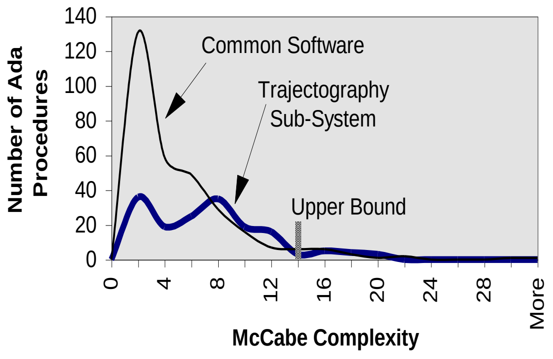

Figure 3 - Typical Complexity Distribution

We have gathered data from over 50 inspections to determine a baseline profile to represent the nature of comments made during an inspection. This profile indicates the effectiveness of each inspection and highlights significant deviations.

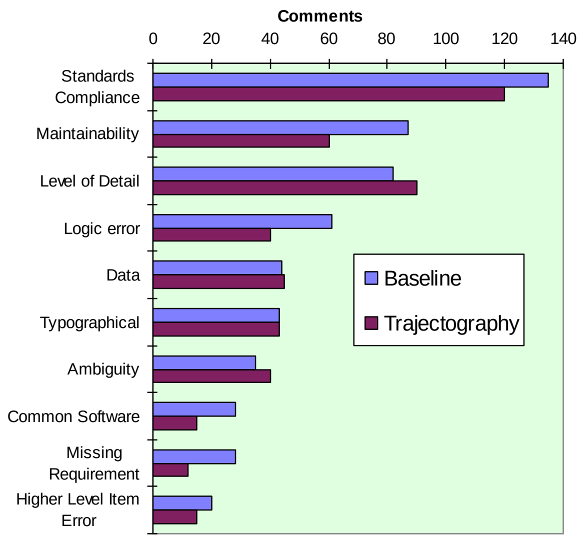

Figure 4 - Baseline Comment Profile

Managers use the model to predict resource requirements during initial planning and to refine their estimates as the development progresses.

Step 5 - Quantify Effectiveness and Identify Critical Success Factors

Our data confirms the results of many case studies that have quantified the payback achieved through finding and curing problems early on in the lifecycle.

Phase	Critical*	Major	Minor
Design	>20	<1	<0.5
Code & Unit Test	>100	<2	<0.5
System Test	>1000	<5	<1
Operations	?	<20	<2

Figure 4 - Man-Days to Fix Problems

(Un)fortunately we don't have data from our project to support the cost of finding a critical problem, such as a fundamental flaw in requirements. This is usually limited only by the total development cost.

Test tools such as AdaTest are invaluable during structural testing, but achieving more than say 98% coverage, can be very expensive. Exception handling code may be difficult to test exhaustively. AdaTest identifies statements that have not been executed and these can be readily inspected. The cost to inspect the last 2%, is orders of magnitude less than using a test tool and effective at finding coding defects.

We have identified the following critical success factors when implementing formal inspections and testing:

- train a core team who will drive the process
- focus on risk areas (hazard analysis, complexity)
- define the right metrics, refine with experience
- measure "drift" to trigger re-inspection and re-test.
- establish the infrastructure to collate metrics

Step 6 - Define Mechanism to Assess Operational Readiness

A development should see operational readiness grow as illustrated in Figure 5 until it reaches the "release threshold". This will indicate that a decision can be made to go into operations. Releasing the system to operation before this threshold is reached might result in expensive maintenance fixes.

The release threshold is an empirical value that takes into account factors like the level of dependability required and anticipated in-service life. A safety critical system will have a much higher release threshold than the latest web browser.

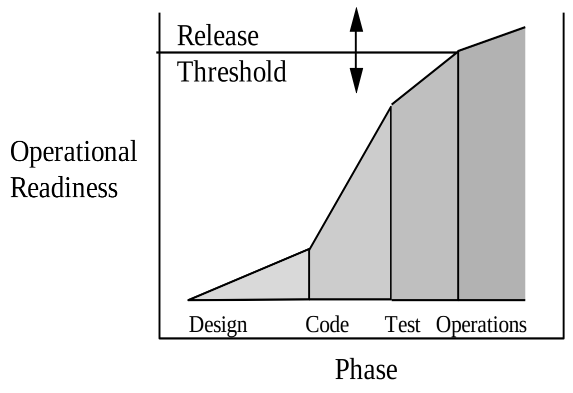

Figure 5 - Baseline Operational Readiness Profile

Operational Readiness is a composite metric, derived using actual and historical data. It takes upstream factors into account with the benefit of hindsight, but also recognises downstream costs.

It captures the key information the project manager needs in a way which can be easily interpreted and compared across projects

The baseline curve is idealistic. In reality, operational readiness grows unevenly. In the actual data below, a late increase in the code modified during the project reduces the stability and test coverage. The operational readiness drops, alerting the manager to a potential problem, but then recovers as corrective action is taken.

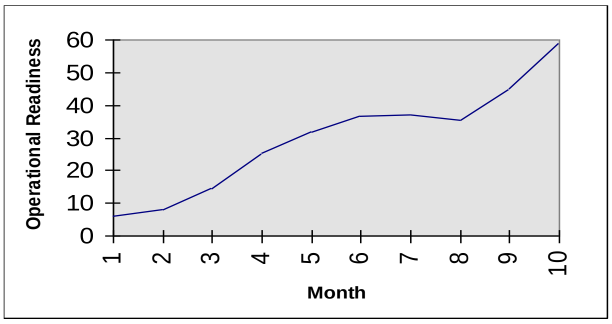

Figure 6 - Actual Operational Readiness Profile

Month	Inspection	Test	Stability	Readiness
1	10%		0.100	6.0
2	30%		0.050	8.0
3	60%	10%	0.010	14.4
4	80%	40%	0.033	25.3
5	95%	60%	0.020	31.8
6	96%	80%	0.033	36.5
7	96%	85%	0.025	37.2
8	96%	80%	0.010	35.6
9	96%	90%	0.200	45.2
10	96%	100%	0.500	59.2

INFRASTRUCTURE

The infrastructure is a prerequisite for the successful application of software PA, especially wih extensive use of metrics. The authors have developed an integrated suite of PC and UNIX utilities supporting the various techniques mentioned here.

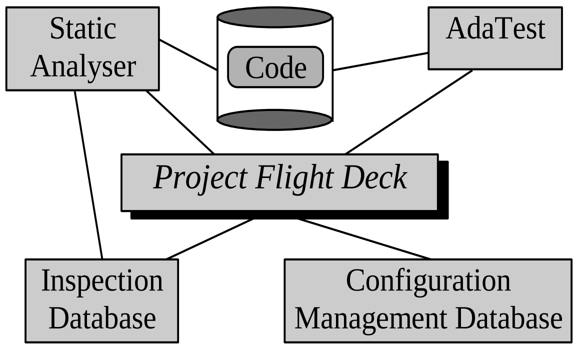

Figure 7 - PA Infrastructure

NEXT STEPS

This simple model described here illustrates the strength of the approach. We are extending the operational readiness concept to encompass all checkpoints on the process map.

The tangible benefit of the holistic approach is that it broadens the project manager's perspective, resulting in achievement of quality and productivity.

REFERENCES

The Readiness Growth Model; Tolochko S, US Army ARDEC. EuroSTAR 1993

A Quantitative Approach to Software Management - The ami Handbook; Pulford K. et al; Addison-Wesley.

Programming Language Table; Software Productivity Research; http://www.spr.com.

Software Inspection; Gilb T, Graham D; Addison-Wesley 1993

ALARP: safety-related systems. Guidance for engineers; Hazards Forum, March 1995.