QUALITY ASSURANCE

The quality assurance process is different from the quality control. It is a runtime process, that must be planned beforehand and then run in parallel with software development.

What is high quality software?

* [PARADIGM ONE - CLARITY] Functions clearly defined, in terms of: what they do, inputs, outputs/returns, global variables/functions accessed (if any), author(s), date
* [PARADIGM TWO - DELIVERY] They use assertions to guarantee the processing is performed as required.
* [PARADIGM THREE – COUPLING] Their coupling is as tight as possible
* [PARADIGM FOUR - COHESION] They have maximum cohesion
* [PARADIGM FIVE - LOCALITY] They avoid using global objects the most possible, keeping parameters instead of them.
* [PARADIGM SIX - ATOMS] They must be tested individually before [PARADIGM SEVEN - MOLECULES] integration
* [PARADIGM EIGHT - TISSUES] All changes must be carefully tracked.

PROCESSES

1. Pair programming

I propose a pair programming system, where each function is developed for one or more programmers, and then documented [PARADIGM ONE] by another developer who wasn’t in the original development team. He/she must understand it and append: (1.1) the function header, as stated on [PARADIGM ONE]; (1.2) any additional comments among the function lines; (1.3) assertions to guarantee the result(s) of the function will be those expected according to PARADIGM ONE; (1.4) The step for (1.3) might cause errors to be detected and corrected, so corrections will be inserted here.

So, code must be produced by programmers [p1,…pk] (k ≥ 1), steps (1.1) and (1.2) by programmer (pk+1), and steps (1.3) and (1.4) if possible by programmer (pk+2) .

Proposed tool: the programming environment itself or GIT. For Python and other language environments that support notebooks, they would be the best fit as documenting tool – specially if the code requires complex algorithms or mathematics.

1. Code analysis

Before testing, code must be analysed and submitted to a special team, who will apply the other paradigms : [(2.1) PARADIGM THREE] each function must be tightly coupled, what means it uses the minimum of global variables/functions/classes, and can be used as software tool or blackbox; [(2.2) PARADIGM FOUR] each function must have maximum cohesion, what means it returns a very clearly defined result or does some defined processing, without changing globals or running other processes no related to its supposed goal; [(2.3) PARADIGM FIVE - LOCALITY] no global objects should be used, unless they are really required. These 3 paradigms should be applied sequentially.

Code analysis must be scored, and functions under a pre-defined score must be sent back to step (1). See below for score tolerances.

The code score could be proposed as a points system :

COUPLING 🡪 (1 point) loose….. (10 points) tight.

COHESION 🡪 (1 point) minimum ….. (10 points) maximum.

LOCALITY 🡪 (1 point) many globals ….. (10 points) no globals.

For each paradigm, I propose begin with the maximum amount of points (10) and then go decreasing for each infraction (for coupling, e.g., a call to an external/global function; for cohesion, when the function perform some calculation not related to its original goal; for locality, for each global object found). 1 point will be the minimum.

The development team will calibrate the tolerance on scoring: eg, it would accept and forward to testing functions with scores above 7, or be strict and accept only scores 10.

Proposed tool: the programming environment itself or GIT. For Python and other language environments that support notebooks, they would be the best fit as analysis/revision tool – all comments could be recorded for each block of code.

1. Testing

[PARADIGM SIX - ATOMS] must be applied so each function must be tested thoroughly, by a specialized testing team. Testing must be performed if and only if all code analysis and revision teams have already approved it.

The only possible scores here would be : approved or rejected. If rejected, it would be sent back to step 1.

Proposed tool: JIRA, for test automation. The software corporation should decide if automation is really required or it should be tested manually.

1. Integration

[PARADIGM SEVEN - MOLECULES] all software pieces which have been to steps 1 to 3 must be assembled together by another integration team. This team must perform the integrated test, assembling the pieces one by one and performing the test on each assembly.

The only possible scores here would be : approved or rejected. If rejected, it would be sent back to step 1.

Proposed tool: JIRA, for test automation. The software corporation should decide if automation is really required or it should be tested manually.

1. Tracking

[PARADIGM EIGHT – TISSUE] any changes must be tracked, and a version control system used to keep them all on the rails.

It is very important that a new version should be considered if and only if the function has been already delivered (been to all steps above) and evolutive changes are required. Required changes due to errors or low scores on revision (which keeps the function to be accepted) should not generate a new version number, although must be documented carefully.

Versions should be carefully stated on code documentation (step 1), even if there is an automated tool for this.

Proposed tool : GIT.

KPIs

I propose creating one KPI for each paradigm above.

A report, that could be generated anytime, will track, for each function (on a given software project) its current stage (each stage correspond to each paradigm above).

For those which have already finished a stage, its score must be shown. If already delivered, the version control number must be shown.



An additional KPI could be developed, to score the development team individually for step 2 scores, if scores other than 10 are accepted on that stage.

SCRUM

The proposed process could be adjusted to fit within a one-sprint timeframe, distributing each step on a one-week time (first week, development and documentation). Or alternatively Step one could be done in 2 weeks and steps 2 to 4 distributed on the other 2 weeks.

Remember the spring planning session should produce and hand the design to the development team, who will produce the code during the sprint.

The KPIs above could be placed together with the Kanban diagrams.