Effectively tracking testing team progress in small/medium companies

Broadly the intent of having matrices defined is to track two things, productivity of the team and client satisfaction.
Most of us at present measure productivity using some of the following computations :

1. (# of test cases created)/Effort spent
2. (# of test cases executed) /Effort spent
3. (# of steps followed to execute the test cases) /Effort spent

or a set of other matrices.And to track whether requirements are met or not we have a traceability matrix in place – which maps requirements to test cases and bug ids.

Current situation

The problem I faced with the above matrices is that it sets the wrong precedence for the team in the sense that people in order to improve productivity tend to granularize the test cases too much in order to increase count plus it does not directly reflect whether all requirements have been covered or not. Then I have to have another metrics for requirements coverage or traceability matrix. Especially if your team is big then it is tough to ensure that everyone writes only meaningful test cases. Then there is the need to have a customer satisfaction metric in place. We can do that indirectly with number of PDDs (Post Delivery defects) and then have a defect seepage metric defined to track that but that is not full proof.On top of all this before delivery we have to ensure that not only all requirements have been covered but also all defects are closed or not.For larger organizations which have dedicated quality team to check and enforce quality as well as analyze data and suggest areas of improvement, this is not a problem. It becomes a pain when you are working in a small or medium sized company where you have to handle everything (as I currently do).

Changes suggested:

Keeping all these in mind I have come up with an approach that combines traceability matrix and data collection for metrics in one place and at the same time ensures that the more your team tries to beat the system to improve their productivity count the better it is for delivery quality. Plus it ensures that all requirements and defects are tracked in the same place so that it is easy to ensure compliance at the time of delivery. Please go through this and let me know your views, if you plan to implement it then do let me know your feedback after usage so that we can improve it further.

The first change is that we will not prepare test cases at all instead we will have only requirements. At the first level we will have customer requirements then let the team think and come up with more granularized sub requirements for each customer requirement – please refer table 1. Let the team have the freedom to come up with as many sub requirements as they can. Then do not focus on how he/she tests to cover that requirement, instead just ask them to mark a pass or fail against each identified requirement. Then we ask them to enter effort spent to test that requirement. Say the total number of requirements identified is "n" and the total effort spent is E, then our productivity is $P = n/E$.
There are three main advantages here. First is if the team member wants to improve his/her productivity then he/she has to either increase the granularity of the requirements or decrease the time spent in executing them – both ways it serves the bigger intent ,i.e, improves quality of deliverable. The second advantage is that since we do not have test cases and instead have only requirements – review of test cases becomes an easy affair (less time consuming and more focused in terms of ensuring every requirement is covered) plus you do not need be have a very deep knowledge of application to review.

The third advantage is that you do not have to have a separate traceability sheet in place – this itself takes care of it. Please see table 1 to understand what I intend to convey.

Mindset Change

Another advantage of this approach is in the mindset change it triggers in the tester. In test case approach the tester tends to focus more on detailing the execution steps of a test case, which not only becomes boring for him (because it is always easy to try out different combinations while doing a testing but tedious affair when you have to write what you did). Plus since his focus is more on test case details, at times the finer points of requirements which should have been covered are ignored. In the new approach since he always has to think about identifying finer categories of the requirement, the focus is always aligned to quality delivery.

If the productivity figure is less then you know where to focus, either the person is not able to analyze properly and come up with finer aspects of requirement or he is taking more time to execute them (both point to a lack of domain or technical competency – which is easy to target for a manager).

Regarding PDDs (Post delivery defects)

In case a post delivery defect comes then the onus of this should be on the developer and the tester. Since I am a testing team manager my focus in on the tester. A small calculation change will ensure that the tester stays focused. Say the number of post delivery defects we got is “m” and the effort spent in re-testing is E1. The we should recompute the productivity of the tester as follows :

$$P1 = (n - m)/(E + E1)$$

where n = number of requirements identified by the tester
and E = the effort he had spent on testing the requirements.

Since this pulls down his productivity – he has to improve his quality.

At the same time the reward mechanism we can have in place for the tester is to add the number of defects he logs (defects which are genuine and accepted) to the number of requirements. This improves his productivity figure. Say number of accepted bugs logged = R ,then his productivity is $P2 = (n + R)/E$. One more area where the tester can improve his productivity figure is by multiplying the number of requirements(n) with the number of browsers he has to test for. Say he has to test the same set of requirements in IE7, Mozilla, Firefox then his total number of requirements for productivity computation should be n*3.

Defect Severity

One area where we tend to have confusion is categorizing the severity of the bugs identified - Fatal or Major or Minor or Trivial. To take care of this my suggestion is to have two requirements as mandatory irrespective of the requirements given by client. They are :
Requirement(1) – Check whether “Look and feel” of the application is good or not.

Here the tester is free to check for any thing that he feels is not correctly done , things like spelling mistakes, misplaced buttons ,color scheme, etc. And any defect found here is a “Minor” or a “Trivial” one.

(Of course this requirement becomes a primary requirement if the client has specifically asked for it. In that case the defects would be categorized as Fatal" or "Major").

Requirement(2) – Once all the requirements identified are tested the tester has the freedom to test randomly – the application as a whole.In this phase he ensures that no existing feature has been broken. Have the tester the freedom to categorize this requirement further if he wants to. Bugs logged here would primarily be "Major" or "Minor".

If any of the client supplied requirement is not met then we can have it as a "Fatal" defect.

Table 1: (The columns in yellow are fixed and the columns in green can be added as and when we come across more requirement levels)

Primary requirements given by Client	Sub-Requirements identified- Level 1	Sub-Requirements identified – Level 2	Browsers tested	Run Date	Pass/Fail	Bug ID	Bug Severity	Effort spent	Version number of code tested
Req 1	Sub-Req1.1	Sub Req 1.11							
		Sub Req 1.12							
		Sub Req 1.13							

Table 2: (All columns are fixed)

Post delivery defect ID	Requirement to which this is related to	Effort spent on retesting