Another Way to Burn

This article describes a burndown chart based upon test cases rather than effort and describes its advantages. It is intended for readers already familiar with the concept of a *burndown chart*.

The rise of the burndown chart

The sprint burndown chart has become a familiar tool for tracking progress during iterations on agile projects. In its most basic form, any burndown chart simply records, at regular time intervals, the amount of work that remains to be done to complete the target scope of functionality. Initially utilized for visualizing progress during sprints, the concept of burndown has been expanded to the larger scope of releases. Thus the *release burndown chart*, whose interval is the sprint and whose scope is the functionality the system should exhibit when it is released to production.

The units, format, and style of any burndown chart are at the discretion of the team using it. Teams may augment their burndown charts with work completed, an ideal burndown trend line, or scope expansion.

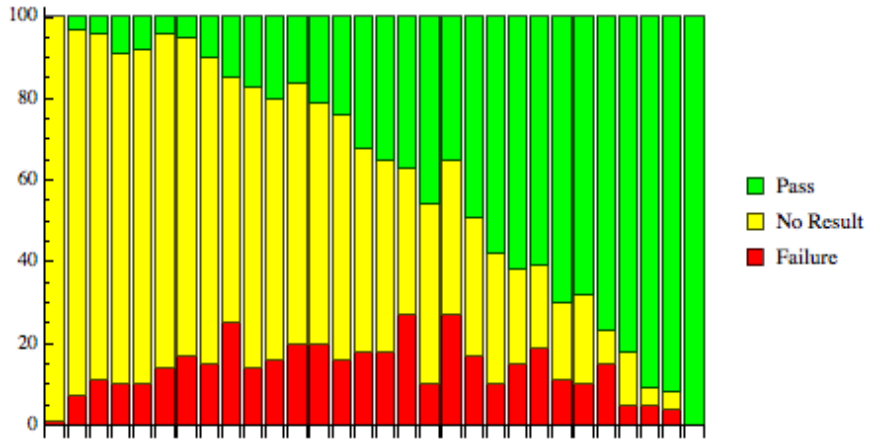
For sprint burndown charts, the typical unit for measuring remaining effort is the ideal hour. Some teams prefer a discrete count of tasks for their sprint burndown chart; others prefer story points. For release burndown charts, the typical unit is the story point.

Burning down test cases

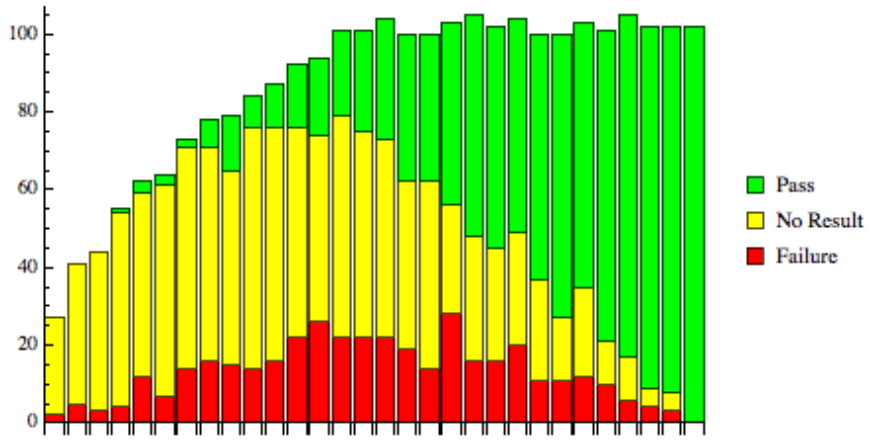
Suppose we created a burndown chart based on test cases. One consideration for such a proposal is whether the data to support it is available; where are the test cases? A variety of test processes are popular among agile teams: test driven development, behavior driven development, acceptance tests embedded in stories, automated unit tests, automated testing as a step in continuous integration. This body of test cases provides teams with immediate feedback as they enhance and maintain their software systems. Many test cases relevant to the current sprint goal already exist when the sprint begins, having been created during story refinement. Over the course of a single sprint, more sprint-specific test cases will be written. The status of these test cases is an indicator of the health of the sprint, just as is the status of tasks. It makes sense to adapt the familiar concept of burning down *tasks* to burning down *tests*.

Test burndown patterns

What patterns do we expect to see with regard to these sprint-specific test cases as the sprint progresses? The following chart illustrates one expected pattern. Early in the sprint, test cases may be failing or not yet executable. By the end of the sprint, no test cases should remain in a failing state.



As the following chart illustrates, it may be the case that the full complement of sprint-specific test cases won't even exist at the start of the sprint. We would expect the full number of test cases to level off well ahead of the end of the sprint.



What is 'burning down' in these charts is the number of non-passing test cases. Teams could choose to display just that quantity over the course of the sprint. Adding passing test cases and splitting the non-passing test cases into failing and unknown provides a more complete and informative view of the sprint's progress.

Comparing burndown charts

How does a test-based burndown chart compare to an effort-based one?

Test-based	Effort-based
Progress toward working functionality is clear	Without looking into the data, we don't know *why* the line moves up or down
Test cases can be used as proxies for effort, but some analysis and calibration must occur	Since the unit is already effort, no translation is necessary
For automated tests, the chart can be derived automatically	Effort estimates must be updated manually
The team may need more testing knowledge and discipline in order to see the expected trendlines	Very little skill is required, and deviations from the trendline are obvious indicators of trouble
Since test cases have lifetimes beyond single sprints, the data must be structured so that tests can be filtered by sprint	Task lists are created afresh each sprint, so managing the data is relatively trivial
Counteracts the tendency to postpone or minimize testing	Provides no motivation to be diligent about testing
Affords clearer risk analysis when sprint scope adjustments must be made	Provides no insight into the risks and trade-offs if scope were to be adjusted
Well aligned with earned value tracking	Requires analysis to map to earned value
Clarifies dependencies between development and testing	It won't necessarily be clear that a task is on hold for testing

Given the different views that these two versions of a sprint burndown provide, a team may reasonably choose to use both burndown charts, taking advantage of the attributes of each.

Expanding the concept

Could the idea of basing a burndown chart on test cases be extended to release burndown? Absolutely, with certain preparations. It is unlikely that the full complement of test cases verifying a release will be available at the beginning of the release cycle. An effort-based release burndown chart is seeded with the team's velocity and sprint count, and an overall trend line can be plotted. The analogous trend line for a test-based release burndown chart could be plotted if there were a fixed ratio between sprint velocity and test cases per sprint. Can we expect such a correlation?

Story points purport to measure story size and complexity. Surely the number of test cases for a story is also a manifestation of that story's size and complexity. We would expect variation in test case count from one sprint to another. However, over the course of several sprints this *random variation*[1] will average out, allowing us to compute a reasonably accurate[2] *test velocity*.

If there is a strong correlation between the number of story points and the number of test cases required to verify a story, then story sizing, sprint planning, and sprint tracking could all be done with test cases to the same degree of accuracy. Consider some possible advantages of replacing story points with test cases for estimation, planning, and tracking purposes:

Test cases are cross functional. At the same time you are defining test cases, you are also clarifying requirements, establishing the definition of done, *and* providing a size estimate. Accomplishing all of these goals simultaneously with test cases is more efficient, more deterministic, and ultimately easier to calibrate than relying on just story points.

[1] Random variation is unbiased variation. Sample measurements may be above or below an expected value, but are not biased either way. Given enough samples, we can expect the average of these samples to be close to the expected value.

[2] In addition to random variation, there is systematic variation, or biased variation. Measurements that are consistently high (or low) are displaying systematic variation. Systematic variation cannot be corrected just by taking more samples. A measurement process with low systematic error is said to be 'accurate.'

Test cases are precise.[3] The typical process is to first estimate a new story in t-shirt sizes (e.g. small, medium, large, epic). These t-shirt sizes represent increasingly wide ranges of confidence. Refinement into story point equivalents must occur before a story can be accepted onto the sprint backlog. The accuracy of the sprint plan rests on the hope that these wide ranges will contain roughly equal numbers of overestimates and underestimates, thus canceling each other out. While experienced teams are able to achieve a respectable degree of consistency between sprints, it comes through hidden, intuitive adjustments.

Counting test cases can be automated. A sufficiently accurate effort estimate can be computed by simply multiplying the number of test cases by a factor empirically computed and calibrated. The data necessary to derive this equivalence between test cases and effort can be mined from previous sprints or can be computed directly from a chosen seed story. Choose a midsize story, create an initial set of test cases for the story, and then track that story through its development, recording the eventual totals for effort and test cases. A team could, of course, perform this analysis for an entire sprint's worth of stories, creating a sizable amount of data that can be used to seed the subsequent sprint, with ongoing calibration occurring thereafter.

Test cases can come first. Sprint planning requires some amount of story decomposition, and tasks can then be extracted and itemized in the burndown chart. This means that the scope of work for a sprint isn't known until *after* the team has tentatively committed to the sprint goal. Tests can be written as soon as a story is written.

Using test cases for size estimates addresses all of these issues. Instead of arbitrary confidence intervals for story sizes, a confidence interval can be computed statistically with historical sprint data after just a few sprints. The scope of work (again within a computable confidence range) is known *before* the team commits to the

[3] A measurement with low random variation is said to be 'precise.'

goal. The definition of done will have already been established, instead of requiring negotiation during the planning meeting. Finally, sprints can be packed more efficiently, since the count of test cases for a story can be any number instead of a t-shirt size equivalent. If needed, stories can easily be decomposed into sets of test cases to accommodate the sprint packing instead of needing to be re-analyzed and re-written.

♠♢♣♡