Identifying and handling a critical failure

A critical failure is a test failure that has a high impact on the daily work for developers.

This can mean a few different things:

- **Any failure in tier 1 is considered critical.** A single failure, no matter how isolated or trivial, will cause the entire tier to signal failure. As long as there are tier 1 failures, every developer that pushes a change must dig into the results to see what caused the failure. Human nature dictates that after a while one will assume that the failure is the same as it was in the previous run, and at that point tier 1 testing is useless. Unless the results are checked every single time we will not notice when a new failure appears.
- **Any failure that is blocking integration is considered critical.** Integration blockers cause delays in propagation of other changes. People outside of the HotSpot team might be waiting for a change to propagate to master. The performance team needs HotSpot changes to get into promoted builds on a regular basis. In short, integration blockers stop others from doing their work.
- **Any failure that causes multiple unrelated tests to fail is considered critical.** Bugs with a large damage area will hide other bugs and make it harder to fix those other bugs once they are found. Tests that cause massive failures in the nightly will seriously reduce our test coverage.

Please note that there is a significant difference between high priority and high urgency. High priority issues like vulnerabilities for example will always have higher priority than regular bug fixing. Critical issues are urgent, but do not always have high priority. It could be a trivial bug in a test that has no particular significance, a test that nobody cares wether it is fixed or not. But, if it causes failures in tier 1 it is urgent to fix it. Not high priority but still urgent. It is most likely urgent enough to take 20 minutes away from fixing a vulnerability.

Handling a critical failure

It is required that anyone who pushes a change monitors the tier 1 testing and the following nightly, or designates this requirement to someone who can also fix possible regressions. It is considered exceptionally bad engineering to push a change at the end of the day, or on the last day before going on vacation or otherwise becoming unavailable. Critical failures must be handled with high urgency, thus the engineer who caused the failure is best suited to do so.

If a failure in **tier 1** is identified and can be fixed within **two - three hours**, this is the preferred way to go. If a fix is not expected within that time the broken change **must be backed out**.

Integration blockers must be fixed before the next integration per definition. Usually this means that a fix should be available **within a few days**.

Bugs that causes unrelated tests to fail in the nightly should be fixed **before the next nightly**. If a fix can not be delivered within **two days** the causing change **must be backed out**.

Backing a change out is easy. Anyone can do it.

There can never be any ownership involved when it comes to backing out a change. It doesn't matter who made the change or why it fails, if it needs to be backed out it should be backed out. Whoever is on site first should do it. Usually there is some synchronization before doing the actual work to avoid duplicate work.

Use mercurial to create a backout changeset:

```
hg backout -r <revision_id>
```

A backout is considered a trivial change so a single Reviewer is enough and you can push immediately when reviewed. See How to backout a change for details on bug management in JBS related to backing out changes.