Just the Facts

VP Engineering Demo Scenario

2/13/13

**Check the Dashboard**

First thing in the morning, I log into Just the Facts to check my dashboard. I inspect the following facts:

1. First, I check my performance numbers. If my **Apdex** score for the last 24 hours is less than .8, that means a significant number of users are not seeing satisfactory response time. I can check my app server **Average Response Time** for details, or simply jump directly into my NewRelic dashboard.

2. Next, I check my **positive and negative sentiment** from my social monitoring tool (socialmention), which gives me an up-to-date analysis of positive and negative tweets about my product. If there is an unexpected uptick in negative comments, this could be an early indicator of a product issue creating customer dissatisfaction.

3. Next, I check the count of currently **open customer support** issues from ZenDesk. If there are more than 25 open issues, this is another leading indicator that there are product quality issues that might require investigation.

4. Now I can rest assured that there are no customer crises requiring my immediate attention, so I turn my attention to tracking my team's progress on the current release. First, I check my **Open Bug Count** from Jira for the current release. More than 10 open bugs means the backlog of issues is building up, and may require attention.

*Another interesting metric would be the ratio of found to fixed bugs. Should we add this or is it too complex for now?*

5. I also check my **Unresolved Story** count from my GreenHopper project tracker. If the count is going up instead of down, that means there are reopened stories that may require attention.

6. I can also choose to show Open Bugs and Unresolved Stories graphed against each other - this effectively gives me a burn down view of work on my current project.

7. Finally, once I am confident that the current release is on track, I check the status of my daily build from yesterday as tracked on my Jenkins server. If there was a **build failure**, that obviously is a red flag. However, if my **code coverage rate** drops below 70%, it is a warning sign that requires investigation. Likewise, if my code coverage goes above 90%, that may be a sign that the team is spending so much time on test coverage that diminishing rates have set in.

If the build succeeded and code coverage has gone up significantly, I can add a comment regarding today's coverage rate with an explanation of why the value changed. I can also view a more detailed graph of how the coverage has changed over time, with my prior comments embedded in the graph.

**Create the Dashboard**

How did the VP of Engineering originally create and populate this dashboard? We demonstrate by logging into a new account and viewing the newly created, empty dashboard.

First, I am prompted to install the Google Chrome plug-in, if it's not already installed.

Next, I have a set of links to the supported and configured dashboards. I can click each link to be launched directly into the dashboard, where I can activate the plug-in and subscribe to any of the supported facts.

The Facts are automatically added to my dashboard. I can also optionally specify target minimums, maximums, or ranges for each fact as described below.

|  |  |  |
| --- | --- | --- |
| **Source** | **Fact** | **Target Range** |
| NewRelic | Apdex Score | Yellow if below .8,  Red if below .5 |
|  | App Server Response Time |  |
| Socialmention | Positive Sentiment Count |  |
|  | Negative Sentiment Count | Red if greater than 10 |
| ZenDesk | Open Ticket Count | Yellow if greater than 25,  Red if greater than 50 |
|  | Solved Ticket Count |  |
| Jira | Open Bug Count | Yellow if greater than 10,  Red if greater than 20 |
|  | Closed Bug Count |  |
| GreenHopper | Unresolved Issue Count | Yellow if current value is greater than prior value |
|  | Resolved Issue Count |  |
| Jenkins | Test Failures |  |
|  | Code Coverage | Red if below 70%  Yellow if above 90% |

*Note that this assumes a previous, per-tenant configuration step that sets up root URL values for each dashboard available to that tenant. Alternately, the first time a tenant user actives the plug-in from a supported dashboard, its URL is saved for use by other users for that tenant.*

After each fact is added, I can view it in the dashboard and see its current value. Historical values will be filled in as the fact subscriptions are updated daily.

**Share the Dashboard**

I want my direct reports to be able to track the same data, so that when I have questions I know that they are looking at the same data as me. I share the dashboard with them by entering their email addresses.

After they receive the email, they complete their account creation by entering their name and setting a password, and then have a read-only view of the dashboard.

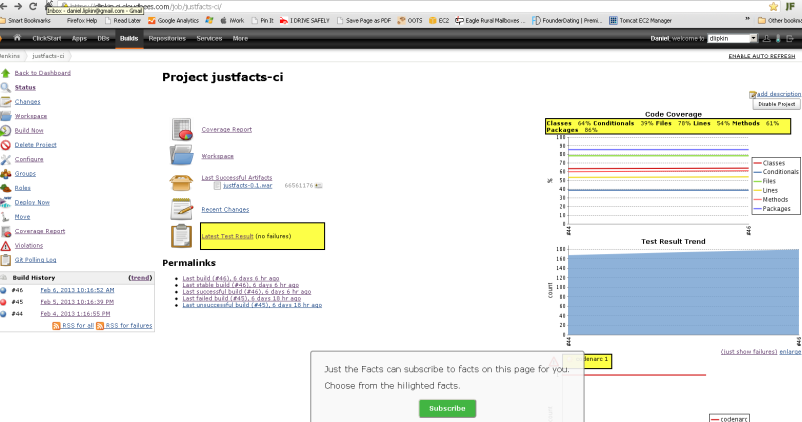
**Maintain the Dashboard**

Periodically, I need to update the facts I am viewing in the dashboard. When I release a product and begin working on the next version, I select the facts associated with the prior version and remove them from the dashboard. I then go to Jira and GreenHopper, bring up the tracking pages for the new version, and subscribe to the new facts for that version.

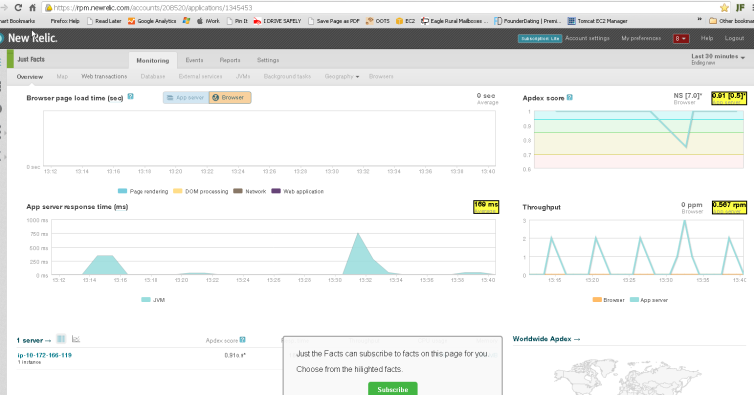
**List of Wireframes/HTML mocks (Proposed Priority Order)**

1. ~~Login Screen~~
2. ~~Subscription Screen~~
3. ~~Dashboard - showing facts~~
4. ~~Dashboard - newly created~~
5. Comment on a fact's current value
6. ~~View fact history~~
7. ~~Graph fact history~~
8. Graph two or more facts together
9. Share a dashboard
10. ~~Remove a fact~~
11. Plugin dialog - subscribe to facts, enter credentials, set optional minimum/maximum/range values

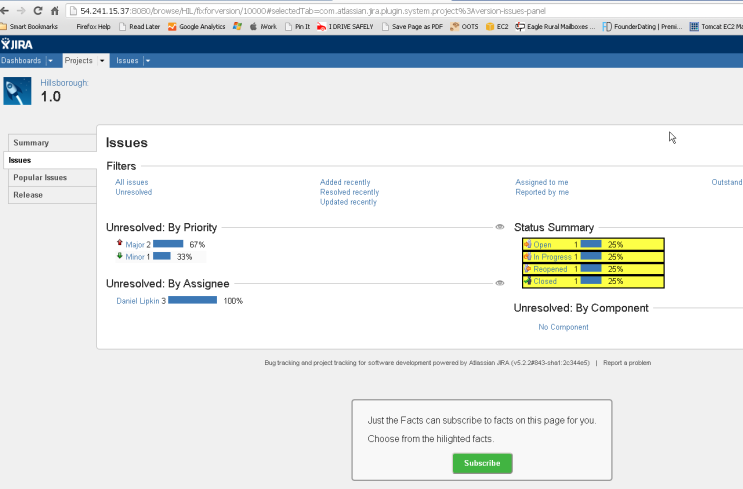
**Appendix: Dashboard Screens**



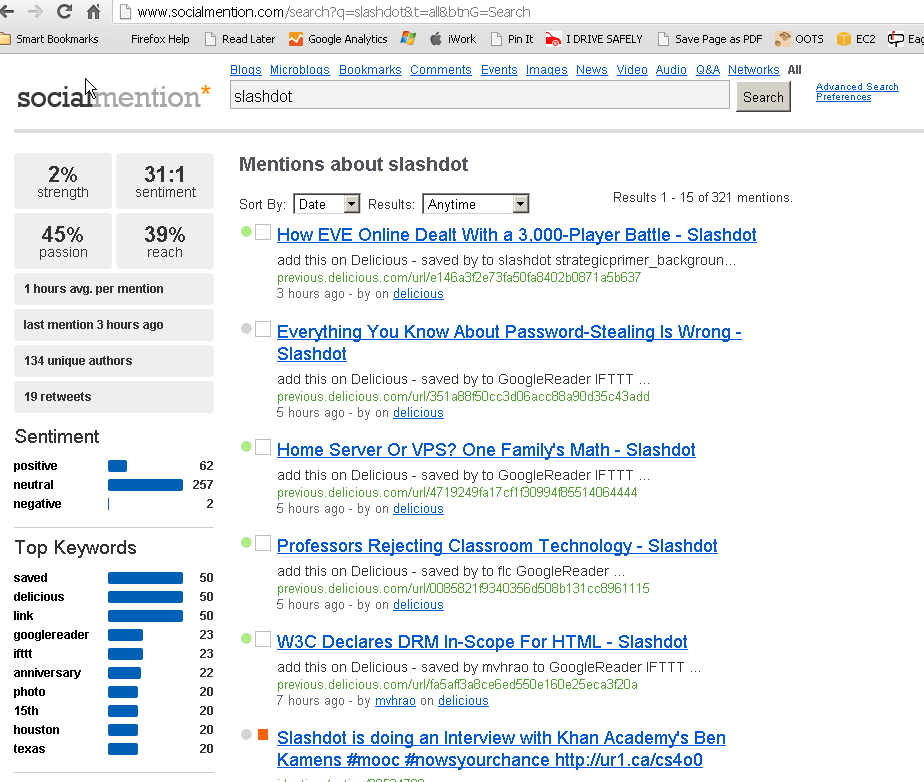
Jenkins



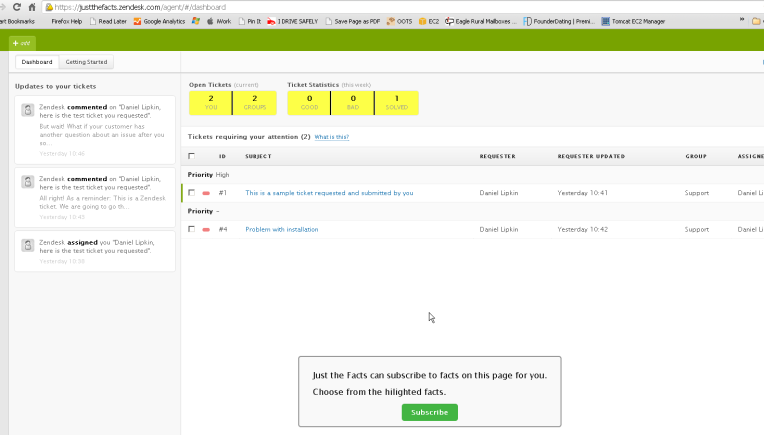
NewRelic



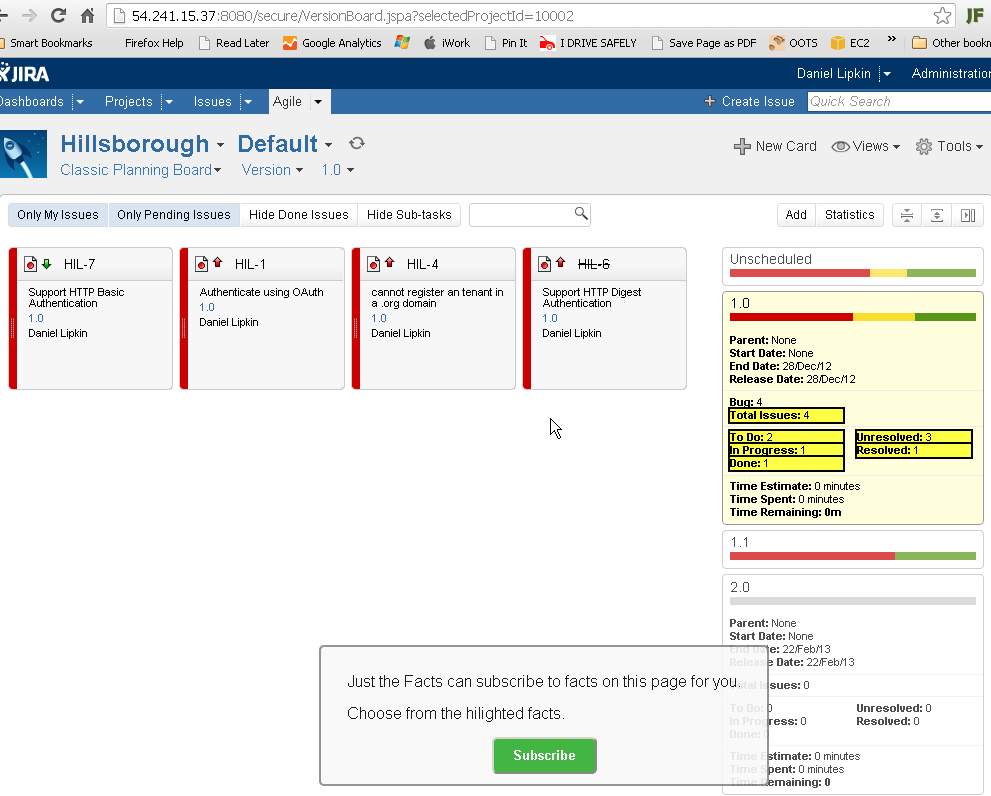
Jira



socialmention



ZenDesk



GreenHopper