README

Perry Kundert

November 23, 2015

Contents

1 burndown – Charting for agile projects using org-mode and Git

Uses https://github.com/mbostock/d3 to generate burn-down charts from org-mode project Effort estimates and CLOCKSUM time log data. The historical "Effort" (estimate) and "CLOCKSUM" (work) data are collected from the Git repository, and rendered as Burndown charts, as per:

http://www.mountaingoatsoftware.com/scrum/alt-releaseburndown

Normal burndown charts don't differentiate between changes in project scope (adding or removing task), and progress on the project (converting tasks from "todo" to "done"). The "alternative" burndown chart shows project progress by changes along the top of the bar chart, and changes in project scope on the bottom of the bar chart – where the best-fit lines meet is where the project is likely to be finished.

By separating the two concepts, it is absolutely clear where changes in the project finish originate; either due to changes in work progress, or changes in project scope.

1.1 Finish Day/Date Projection

The project is going to finish A) after a certain number of project Days of effort, and B) on a certain calendar Date. These are somewhat independent, since the amount of work applied to the project changes over time (holidays, changes in staffing, etc.)

To compute the Project's finish Day, we use a best-fit curve to project the intersection of the Project's progress and change lines. Based on the burndown style selected, this reflects units of effort, units of elapsed time, or sprints.

Once the finish Day is computed, we also use a best-fit line through each sample's actual calendar date, to predict the finish Date. This gives us a correlation between each burndown unit, and calendar time.

1.2 Burndown Styles

Three styles of burndown chart are available. All of these change the meaning of the X (time) axis. The Y (estimate) axis always represents the amount of task "Effort" (estimate) remaining in the project. By changing the X axis, we can see how the project is progressing as we track:

1.2.1 effort

Tracks project "CLOCKSUM" (work) time. Each graph X axis sample represents a certain number of hours of actual work on the project. The default is 8 hours. This computes the project finsh "Day" based on effort-days, and a finish Date based on a projection of historical work Days per calendar Day.

1.2.2 elapsed

Tracks calendar elapsed time. This produces one X axis sample per calendar day. Therefore, the computed project finish Day and calendar Date are identical.

1.2.3 sprint (not yet implemented)

Tracks project Sprints, one sprint per sample. Works best if your sprints represent roughly equal work effort, but will respond to changes in Sprint effort over time, in a best-fit fashion. Predicts the number of Sprints remaining.

1.3 Maintaining org-mode Project Data

Prepare an org-mode file containing a project, in the form of a tree of TODO entries. Ensure that the top-level TODO entry for the project contains a date entry, and put an org-mode table within it:

```
* TODO Project Test <2012-03-18 Sun>
#+BEGIN: columnview :hlines 1 :id local
| Task                                | Effort | CLOCKSUM |
|-------------------------------------+--------+----------|
| * TODO Project Test <2012-03-18 Sun> |  46:00 |          |
| ** TODO Setup                       |  10:00 |          |
#+END:
** TODO Setup ...
```

Each day, update the CLOCKSUM (time worked) and/or Effort (TODO estimates) data in the org file. Update the top-level Project's date. Finally, refresh the table (hit C-c C-c in the BEGIN: line of the data summary table), save the file and commit it to the Git repository.

The orgserver will parse the project data from the specified project's .org files in the Git repository. By scanning the historical contents in the Git commit history, a burn-down chart will be rendered. We use d3, so make sure your browser supports SVG (eg. Chrome, Firefox or IE9+).

1.4 orgserver – A web.py based HTML and JSON API server

Serves org-mode data from the specified org directory, for the specified project(s). May be configured to update from git, and kill running server(s) if changes to the burndown chart implementation itself are detected.

The running orgserver starts the python orgserver.py and puts its PID in the org directory in an orgserver.pid.##### file ending with the orgserver's own PID. This is used by subsequent orgserver invocations, to locate and terminate the other running orgserver.py instances.

The options are:

```
orgserver [--<option> [...]] [<project> [...]]
    --help            This help
    --verbose         Log activity
    --restart         Restart any running server   (default: no)
    --refresh         Refresh Git repositories     (default: no)
    --address <i[:p]> Bind to interface:port       (default: *:8080)
        Performs a "git pull origin master" on both the org data
        directory, and the orgserver directory.  In either case,
        if the master branch's commit changes, the orgserver will
        forcibly restart any running orgserver.py.
    --org <dir>       org-mode data Git repository (default: ~/org)
    --log <file>      Log appending onto file      (default: -)
        If not relative: (.[.]/*) to current directory, or
        absolute: (/*), we assume it's relative to the org directory
    --server <name>    Specify Web Server platform (default: web.py)
    <project>         .org files to parse          (default: project)
```

The following environment variables may be set, instead of specifying the Org data directory and the Projects list on the command line:

```
ORG_PROJECTS  -- The list of project names (default: project)
ORG_DIRECTORY -- The org directory         (default: ~/org)
```

1.4.1 EXAMPLE

Run a orgserver bound to *:8080 on the host, serving org-mode data from ~/org, for the project 'test' (ie. from org-data in ~/org/test.org):

```
./orgserver --verbose --org=~/org test
```

Start (or restart) the orgserver's underlying webserver, refreshing the Org data (~/org) and orgserver (~/src/burndown) Git repositories. Exits quietly if the orgserver is already running, and restarts it if the orgserver repository 'master' branch has had new commits:

~/src/burndown/orgserver –refresh –org=~/org some project names

This can be set up to occur automatically (say, every 5 minues) using crontab -e:

```
# minute hour mday month wday command
  */5    *    *    *     *    $HOME/src/burndown/orgserver --refresh --log orgserver.lo
```

1.5 orgserver.py – A Python module for process org-mode data in Git repositories

When run as a command (as by orgserver), orgserver.py starts a web.py webserver, by default bound to port 8080 on all interfaces.

1.5.1 HTTP JSON API

The HTTP API respects the Accept: header, and generally responds to "text/html" requests with HTML, and to "text/javascript", "application/json" and "text/plain" requests with JSON. Alternatively, any HTTP request may force JSON by appending ".json" to any URI.

- /api/projects[.json]
  Returns a list of all projects.

```
[
    {
        "project": "burndown",
        "styles": [
            "effort",
            "elapsed",
            "sprint"
        ]
    },
]
```

- /api/data/<project>[/<style>][.json]
  Returns the request project's data in the specified style. The "list" attribute contains an entries with non-empty "estimated" and "work" data for every day actual data, and computed data (inserted in-between existing data points, to ensure that the data points are consistently spaced for the requested X axis style.) All project data is supplied in 2 forms: "xxxx": <textual> and "xxxx#": <seconds>. Relative time values (eg. task "added#") are in seconds from 0 (when the project

started), and absolute time values (eg. the sample "date#") are in seconds since Jan 1, 1970 UTC (the unix Epoch).

If burndown trend lines can be computed, they are included; the "change" and "progress" lines are in seconds since the project started, and "date" in seconds since the Epoch.

Empty data points are appended to "list" between the final day with project data and the computed finish point of the project (or some maximum limit, if the computed finish date is far in the future). These have everything except "lines", "work" and "estimated" are null.


```
{
    "list": [
        {
            "blob": "6c4a8197f1efc91dbb197a0e210ad7b38a03a6b1",
            "date": "2012-03-01",
            "date#": 1330585200.0,
            "estimated": {
                "added": "0:00",
                "added#": 0,
                "addedTotal": "0:00",
                "addedTotal#": 0,
                ...
            },
            "label": "Day 1",
            "lines": {
                "change": {
                    "x1": 0,
                    "x2": 1,
                    "y1": 0,
                    "y2": 0
                },
                "date": {
                    "x1": 0,
                    "x2": 1,
                    "y1": 1330585200,
                    "y2": 1330671600
                },
                "progress": {
                    "x1": 0,
```

```
                    "x2": 1,
                    "y1": 61200,
                    "y2": 61200
                "sprint": 0,
                "work": {
                    "added": "0:00",
                    "added#": 0,
                    "addedTotal": "0:00",
                    "addedTotal#": 0,
                    ...
                }
            },
            ...
            {
                "blob": null,
                "date": "2012-04-02",
                "date#": 1333425728.2857144,
                "estimated": null,
                "finish": "2012-03-25 (Day 15)",
                "label": "Day 19",
                "lines": null,
                "sprint": 0,
                "work": null
            },
        ],
        "project": "burndown",
        "style": "effort"
    }
```

1.5.2 REQUIREMENTS

If you are on a Mac, you might look at https://github.com/pjkundert/setup to see detailed build and installation automation and instructions for these (and other) packages.

The following Python modules are required (all these assume your PYTHONPATH=/usr/local/lib/python2.7/site-packages):

- git-python 0.3; requires nose and mock for tests
  Obtain source from https://github.com/gitpython-developers/GitPython
  Assuming your PYTHONPATH=/usr/local/lib/python2.7/site-packages, this might work:

```
git clone git://github.com/gitpython-developers/GitPython.git git-python
cd git-python
git checkout origin 0.3
git submodule update --init --recursive
python setup.py install --prefix=/usr/local
```

- web.py
  Obtain source from https://github.com/webpy/webpy

```
git clone git://github.com/webpy/webpy.git
cd webpy
git checkout origin master
python setup.py install --prefix=/usr/local
```

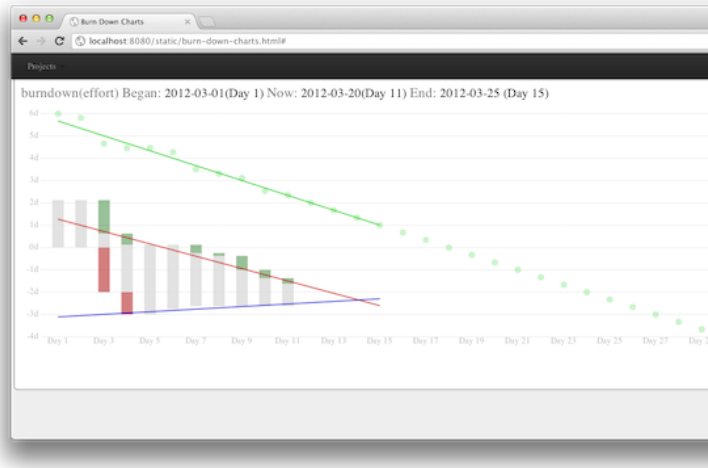
- wsgilog
  Obtain source from https://bitbucket.org/lcrees/wsgilog/src

```
hg clone https://bitbucket.org/lcrees/wsgilog
cd wsgilog
hg pull -u
cd wsgilog # yes, again...
python setup.py install --prefix=/usr/local
```

- argparse
  For pre-2.6 Python, you'll need to install argparse and simplejson:

```
sudo easy_install argparse
sudo easy_install simplejson
```

1.5.3 SCREENSHOT



The burndown project itself, in “effort” style: