HOT OSM Task Manager 3 Tech Challenge

Application: Nilenso Software LLP (nilenso), a limited liability partnership firm

1 Introduction

Throughout this application, we hope to address all questions and concerns outlined in the Task Manager 3 (TM3) Tech Challenge (TM3TC). Our team is familiar with HOT, has participated in several mapping events organized by the local OSM community, and is particularly well-equipped to handle both the technical and non-technical challenges of the TM3TC. Our company also has a very specific vision of progressively delivering more of our work as open source, preferably around open data, with each project we take on. When we evaluate, sell, and staff projects at nilenso it is rare to come across one so well-aligned as this. We are very excited to apply to the challenge!

2 About Us

Nilenso is a boutique software consultancy based primarily in Bangalore (India). In our 3-year history, we have built large machine learning, streaming data, experimentation, and mapping/transportation systems. You can read more

about our recent work at http://nilenso.com/recent-tech.html. We continue to run the largest of these systems in production for our clients today.

[I removed "a bit dated" under the assumption I will fix recent-tech.html tomorrow.]

Our designers and developers speak regularly at conferences and the talks we haven't mentioned can be found here: http://nilenso.com/talks.html

2-1 Our Team

Our team consists of software developers and project managers with decades of experience delivering modular code on lean/agile teams, employing practices which have evolved with the industry over the past two decades.

Our industry experience spans a wide range of verticals: from defense and policing to healthcare, transportation, and climate change. The languages we are familiar with (based on experience within, as well as prior to, nilenso) include Java, Ruby, Python, C#, JavaScript, C/C++, Clojure, Haskell. Collectively, we have made substantial contributions to major open source projects like The GIMP and GNOME.

Members of our design team are familiar with the OSM ecosystem, participate in HOT, and are dedicated tree-mappers of Bangalore. They understand and have experience working with open data systems. (Details are provided in later sections of this document.)

Our project managers have led teams of dozens of developers, locally and across the globe. We have both built software teams by managing hiring and staffing decisions and consulted to enhance existing teams or recover failing teams.

[Need to think about the above paragraph a bit more… we need something PM-specific in there, though.]

If we are awarded this project, we intend to staff people from each of these disciplines on TM3.

2-2 Our Experience in Development

We have experience that is relevant to HOT, OSM, and the Task Manager Challenge.

We have built services and infrastructure within the Staples SparX machine learning team for over 3 years. Staples' Experimentation Platform (EP), in particular, is entirely designed, built, and operated by nilenso. EP handles many terrabytes of streaming data at a rate of 500 requests per second under a Service Level Agreement (SLA) of <10ms in the 99.9th percentile. Akshay

Gupta (@kitallis, on Twitter/IRC/Slack) currently leads the EP team and would be our tech lead for Task Manager 3.

We have presented on the Experimentation Platform architecture at Functional Conf 2015:

https://www.youtube.com/watch?v=YjfXhhxw9Bs

EP also involves nontrivial database clustering techniques, which have evolved with the lifespan of the project. Synchronous replication, failover to standby databases, ZFS snapshots, custom PostgreSQL MVCC AUTOVACUUM configurations, WAL streaming, realtime (non-star-schema) reporting replicas, DB and I/O monitoring, partitioning, logical replication, load balancing, and multi-mode failure detection all must be understood by every member of our team working on database clusters.

We have presented on building a PostgreSQL DB cluster at Rootconf 2016:

https://www.youtube.com/watch?v=sGJDg5ba0iI

We have contributed substantially to monitoring infrastructure in many recent software projects. This includes system-level, JVM-level, application-level, and DB-level monitoring on all of our projects. We have been allowed to release some of the DB and DB monitoring work as open source. All of these repositorires are focused on PostgreSQL:

- https://github.com/nilenso/honeysql-postgres (not monitoring specific)
- https://github.com/staples-sparx/wonko
- https://github.com/staples-sparx/wonko-client
- https://github.com/staples-sparx/pg-cluster-setup
- https://github.com/staples-sparx/repmgr-to-zk
- https://github.com/nilenso/postgresql-monitoring

Unfortunately, although we have implemented internationalization (i18n) and built or extended translation tools on a number of projects, very little of this work is open source.

Similarly proprietary is our most recent legacy replacement work on the driver allocation service for an Indonesian Uber competitor. We rewrote this service in Clojure with an extensive generative testing suite (and minimal unit tests for single-value edge cases). Thanks to generative testing and developer discipline, the service was released with zero production issues and completely replaced the legacy software on its first day in production.

[Are we not allowed to mention Go-Jek anywhere? I guess "Uber competitor" probably works just as well for people who don't know GJ anyway.]

This was not our first Clojure rewrite, however. We also rewrote Staples SparX's configuration management, legacy data feed integration, and reporting tools in Clojure. This work involved a different style of testing. First, these services were built with large and comprehensive unit test suites for all known use cases.

Then the new services were delivered alongside the legacy (Ruby) services in all environments and validated for multiple months. Once the Clojure services produced results consistent with the legacy services (as confirmed by automated validation software provided by us to the QA team), 3rd party services were redirected to new APIs and the legacy services were turned off. This process was slow but not expensive and finished without a single interruption to existing clients.

Developers from nilenso have also run training and courses in ReactJS, Haskell, and Clojure.

2-3 Our Experience in Design

Our small design team lead by Noopur (@9porcupine on Twitter) and Varun (@irrational_pai) has also produced a lot of impactful work in a short span of time. They are currently working with Samanvay Foundation; designing a mobile application that helps ground-forces in rural areas quickly diagnose (with the help of an on-board medical professional) basic health issues and audit all the patient information. All their work has been open source since day one:

- http://samanvayfoundation.org
- https://github.com/OpenCHS
- https://goo.gl/dfSOAs (designs are in-progress)

They have experience conducting research to understand the needs of the user as well as their behaviour in the form of:

- Contextual inquiry: with a focused set of users, by giving them specific tasks to perform, observing their behaviour and talking to them further about their actions during the exercise, in order to better understand their mental model.
- Task Analysis: of major tasks, along with the intent to understand and further reduce complexity in workflows.
- We have built personas, performed focus group discussions and interviews to understand our target audience better.
- There are also certain tools such as http://www.inspectlet.com/ and https://www.hotjar.com/. These record user interactions with online applications in a non-intrusive manner, and can be used for better testing on the web.

They are also familiar with user testing methodologies (either prior to launch of a new product or to test acceptance of new features) by creating mockups and prototypes using prototyping tools.

In addition, Noopur happens to be a proper mapping enthusiast. Apart from participating in the local mapping parties, she's contributed a tourist map of Uttarakhand on Wikipedia and helped build an interactive AQI map on top of OSM, after the recent pollution level spike in Delhi:

- https://en.wikipedia.org/wiki/File:Schematic_Tourist_Map_of_Uttarakhand.jpg
- http://ocsidlab.github.io/earthmap/#4/21.15/79.08

2-4 Process / Collaboration / Communication

Our work has moved us around the country and around the globe, operating in and across many different timezones, with many different project management styles, in a variety of toolchains[1].

Our default delivery style is based on the classic Extreme Programming methodologies with 1-week iterations, daily standups, TDD, pair programming, and heavy communication. Progress on individual features/stories/cards would be updated in real time (daily) so stakeholders can watch work in-flight.

We believe that open lines of communication and planning/projection which employs strict measurement and reduces guesswork are critical to the success of any project, so one of our first objectives would be to document all stakeholders (or as many as possible) to ensure the graph of communication was not only open along the channels themselves, but also transparent from the outside, as a whole.

Because of nilenso's offshore and distributed history, we are very familiar with asynchronous communication channels, discussing issues and ideas over a multitude of channels.

Specifically for TSM3, we would ideally suggest identifying an individual or group of individuals who would assay the role of approver / facilitator / gatekeeper, in other works someone who would take responsility for owning prioritisation and approval.

[Are we comfortable saying the above person is simply Blake, at this point?]

A group approval and prioritization process is possible if it is executed in real time, and we would strongly recommend one such process which we have used to great success on many projects: a project Inception.

[1] We are intimately familiar with most of the project management tools out there: Pivotal Tracker, Mingle, JIRA, Trello, paper-stickies-on-a-wall, and of course, Github.

2-4-1 Inception

An Inception is a high-intensity kickoff to a project, which captures a shared understanding of the high level goals, their priorities, their budgets, and their limitations. The remainder and majority of the Inception then attempts to capture features (or "user stories", preferably, as these frame each feature in terms of how it benefits the end user), estimate them if necessary, and prioritize them. Whether stories are individually estimated or not, they are then packed into a

few sample weeks of the project to rough out the expected pace of development. With this initial projection, 4 months worth of TM3 stories can be laid out in priority order and we will know roughly how much the team feels they can accomplish in that period.

Stories can be reprioritized as the project progresses and the initial projection will be updated every week with the team's actual rate of progress. The project's scope is adjusted every week accordingly.

During the project, new feature requests can come from the stakeholders through the various channels into a structured approval and prioritization process. If all cards/stories are ordered by priority from the beginning, it will be easy for the team to collectively prioritize (or for an authority to individually prioritize) individual stories against that total ordering.

3 Implementation

3-1 Design

The nilenso Design team will participate in the development of TM3 with the aim of answering all 4 major user-facing goals laid out in the TM3TC, but Goal 3 is of utmost importance: "a significantly improved UI/UX" means progressively working toward the core of the Task Manager by first eliminating any pain experienced by users.

Our designers will be key in ensuring the development team is building what users need and what the community wants. They will participate in discussions about improving experiences, like those mentioned above for Mappers, for all HOT roles. The proposal and approval process will involve designers early on, from Inception to the fleshing out of each user story. During the Inception, designers will facilitate discussions around a cohesive, overarching design. When defining each user story, designers will describe the specifics of all physical aspects of the software: appearance, interaction, and time.

Designers will own the conducting of user experience testing, whether that happens in the HOT testing environment online or in offline testing such as cafe tests for new users who are completely unfamiliar with HOT, OSM, and their respective software ecosystems.

3-2 Development

3-2-1 Tech Stack

[Explain why Clojure is better: performant, scalable, easy to pick up..]

Regardless of language and stack, our infrastructure, automation, and deployment approach is always built on top of an OSS toolchain. Last year, for cor-

porate political reasons, we had to move all of our services for one client from AWS to a "private cloud". Had we built anything AWS-dependent, this process would have been impossible.

Monitoring is essential for every project. We run own our software in production from our first deployment, and we try to make that first deployment happen as early in our involvement as possible. Operations ("devops", these days) is a group effort but we are often the first or second point of contact for services we build exclusively. Deploying production monitoring, no matter how simple, early in a project's lifecycle is a non-negotiable part of shepherding software in production.

3-2-2 Documentation, Automation, Testing, Deployment

All of our software is thoroughly documented. We not only create user documentation (which is then validated by users as each individual new feature is continuously delivered into production) and developer documentation in the form of in-repository documents, in-line comments (where the domain, abstractions, or algorithms may be confusing), and thorough test suites. We also strive for self-documenting code, verbose and meaningful commit messages on atomic (transactional) commits, short-lived branches, a master branch which is always green and deployable, and a continuous integration (CI) system which not only alerts developers to broken builds but provides proactive insight into deployment and environmental failures. This is done by creating a CI environment which is nearer the production environment than the development environment, where this makes sense.

Major architectural decisions will be captured within the version control system as Architectural Decision Records (ADRs)[1] in an immutable, linear fashion. Every software project should contain a README which serves as the *only* point of entry required for a user, sysadmin, or new developer. Where documentation is not possible (for passwords, keys, and other secrets) the team members responsible are called out explicitly in either documentation – preferably within the README itself – or through automation tools at the point where interjection is necessary. A new user (playing a "sysadmin" role) should be able to install any piece of software with a single package or script for installation. With respect to operations in canonical environments (say, http://hotosm.org) this automation should include provisioning and orchestration.

Unit, integration, functional, and acceptance testing is a part of the development process and a story isn't complete (or even deployable) until it is covered by automated tests. Simulation testing is a bit different and does not necessarily happen in lock step with development. User testing obviously cannot happen until a story is deployed to a testing or staging environment. This is generally also true of cafe testing, to avoid having new users test software that hasn't been proved deployable on production-like infrastructure.

Comprehensive load, integration, acceptance, and unit tests are a good indication of a healthy software project. In recent years, we've started to grow away from these practices to some degree. Load testing can often be completely automated and with small, simple, custom tools and can become a part of CI or even Contiuous Deployment (CD). Integration tests should not be a one-way street and within an integrated service architecture, client-driven service contracts provide integration tests which not only ensure narrowly-defined API correctness but facilitate communication and provide early alerts to both parties when API contracts are accidentally violated. Consumer-Driven Contracts[2] are not as common in the industry as one might expect, and we welcome questions about how we have used them in the past.

Complex systems, such as the Data Science Experimentation Platform (EP) we have built for Staples for over 3 years, can be integration tested with much higher fidelity using simulation-based testing. Simulation tests combine the power of generative testing, explicit system state transitions, and system-wide integration testing. Less expensive than full-blown simulation testing is component-level generative testing using tools like Clojure's `test.check` to replace hand-written unit tests with dynamic and automatically generated tests. Unit tests and TDD still serve to refine component design and test very specific edge cases. You can watch Nivedita and Srihari discuss the EP architecture and simulation testing strategy in their Functional Conf 2015 talk here:

https://www.youtube.com/watch?v=YjfXhhxw9Bs

All of the above of course assumes not just a high level of automation but *complete* automation. To comprehensively test entire architectures with dynamically generated scenarios and interaction simulations, it must be possible to instrument and script the entire set of services. Often, this means that services are built "API first" to accomodate testing infrastructure in the way that object-oriented systems tended to be built "interface first" in the 1990's and early 2000's when TDD and unit testing really took developer mindshare. Though some projects we deliver continue to leave provisioning of hardware to the sysadmin team, absolutely everything else (orchestration, down) must be completely automated and scriptable or high-functioning testing strategies are impossible. The yin and yang in this situation is again reflective of bad modeling corrected by TDD and unit testing: If we try to write a test and get hung up on a manual process, the first thing we'll do is automate it.

[1] ADRs: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions [2] http://martinfowler.com/articles/consumerDrivenContracts.html

3-2-3 APIs

For over two years, the majority of nilenso's software delivery has been on systems with API front-ends. We are very familiar with growing APIs, managing API versions, defining "easy" APIs alongside "performant" APIs, and API-first

development. We all appreciate a line of code is expensive to change, a database schema is much more expensive, and an API schema much, much more expensive than that. Particularly for APIs with unknown clients, "up-front design" is not a dirty term in any sensible agile team.

We see there are five major APIs pending in the existing TM2 software [1]:

1. task details
2. validate/invalidate
3. lock/unlock a task
4. create task by position
5. create task by geography

None of these implementations looks particularly difficult and if we were to extend the TM2 codebase as-is, implementing draft versions of these APIs may be a good way to familiarize ourselves with the TM2 build/deploy process and overall design.

As mentioned above with regard to integration and simulation testing, it is often necessary in modern service architectures to design APIs "outside-in" for the sake of facilitating testing, even before the API has any clients.

In our experience, one major quality of well-designed APIs is a clear and explicit awareness of where and how state transitions occur. Objects and data structure instances are inherently state machines. Entire systems are no different, though it is extremely difficult (and inefficient) to make an entire system immutable, which means state transitions across systems are mutative and it is all the more important to capture state transitions for every API call which writes to a service.

It has only been implied until this point, but to make the point absolutely explicit: We will create APIs early with clear definitions, with the intention of facilitating both testing and integration with other HOT software products. Goal 5 from the TM3TC will be addressed continuously throughout development.

[1] https://github.com/hotosm/osm-tasking-manager2/wiki/API#tasks

3-2-4 Reuse of Code

"Reuse" for us is much more about clean design – well-defined service boundaries, appropriate library usage, and logical abstration – than it is a goal unto itself. Software and team performance are both consequences of a clean design, not the other way around.

4 Timelines

[Add 4-month timeline with standard disclaimers]

In the appendix, we provide a sample set of stories and a general outline of major milestones after that. It should be noted that this list is in no way exhaustive and does not assume build-out of TM2 or a from-scratch server-side rewrite. Therefore, a reader may notice some stories which describe functionality TM2 already provides.

5 Proposed Feature Additions

[we could look at the ToR and pick up more of these]

Between the Github issues and the ToR, an excellent list of suggestions is already available. It appears there are a few areas which would be quite impactful:

1. Exposing all TM3 behaviour by auth-based APIs, first and foremost, will allow all participants to to manage projects & groups and execute tile-level work from outside of TM3 and within their preferred OSM editing environment. Though elided from the cards in Appendix 6-1, new authenticated mutative APIs will require significant work from the development team in tools like JOSM to ensure a full spike through the system, from the user's tool, through TM3's APIs, into project data. The best user interface feels like none at all.

[The above paragraph feels like it has too many words and not enough meat.]

2. User- and Project-level analytics/reporting for Project Managers and Validators will help greatly in informing their workflow and making management of projects more efficient both by improving meaningful (informed) communication with Mappers and helping to smooth the Validation process. The Github issues list contains much low-hanging fruit in this regard and the community's pain points should be our starting position for quick development.

[Above: Kinda vague.]

3. The most direct and TM3-specific feature-set will be interfaces for project/group creation and administration. Though much of the Mapper- and Validator-facing functionality will hopefully move to tools on the edge of HOT/OSM architecture via TM3 APIs, projects and groups of projects are concepts which are rooted in the Task Manager and the unified UI is likely to continue as a feature of the web app. While the primary goal of the user experience for Mappers will be ease and elimination of friction, the goal for Administrators and Project Managers will be power and flexibility. Administrators are not new to OSM and do not require "conversion" to the process; the PM interface of TM3 may forsake time-consuming investment in slick UIs and an easy user experience for richer, heavily-documented features.

6 Appendix

6-1 Sample Stories

The sample story list is as follows. Non-functional requirements are prefixed with "[NFR]".

> *NB: Since one of the Github issues listed is "Consistency in terminology: tasks, tiles, and squares" (https://github.com/hotosm/osm-tasking-manager2/issues/912), we will use "tiles" below to mean all three.*

- Validator can query list of hints in TM3 of tiles where OSMA anticipates an error/mistake
- Mapper will be automatically notified when OSMA anticipates a common mistake on the tile the Mapper is currently editing
- PM/Validator can see # of changesets for a Mapper
- PM/Validator can see # of complete tiles for a Mapper
- PM/Validator can see days since account activation for a Mapper
- PM/Validator can see # of days of active mapping for a Mapper
- PM can view an aerial map of historicaly, current, and new project space by geography
- PM can view a map displaying active and historical projects
- PM can clone a project to create a new project with defaults
- PM/Validator can observe changes made by a Mapper in real time to provide early feedback
- Mapper can self-assign a tile to herself
- PM can assign a Mapper to a tile
- PM can mark a Mapper with a "Validator" role for a project
- PM can create a project group
- PM can add/remove projects to/from a project group
- PM can see current and historical project groups
- PM can mark a project group completed
- PM can mark a project completed
- PM can see nearby/intersecting projects on the project creation screen
- PM can see nearby/intersecting projects on the project summary screen
- Mapper can provide in-app feedback about aspects of mapping they find confusing
- Mapper can make a generic request for help to a Validator
- Validator is notified when a Mapper is identified by OSMA as needing help
- Validator can see a prioritized list of tiles requring validation
- Validator sees likely error-prone tiles highlighted by OSMA in the prioritized list of tiles requiring validation
- Validator can send Mapper quick feedback from a predefined set of common issues

- Validating a tile acknowledges the Validator for that task
- Validator can mark a tile validated in JOSM (implies corresponding API)
- Mapper can optionally add an email id to her profile
- Validator feedback is sent to Mapper's email, if available
- [EPIC] TM3 should provide an API for custom analytics queries (See ToR note "Laura O'Grady 12:38 PM Nov 24")
- PM can set expiration period on a tile; tile auto-expires at the end of the period
- PM can assign a difficulty to a tile
- PM will see auto suggestions from OSMA when assigning difficulty to a tile
- Any User can see aggregate analytics on tiles for all of OSM
- Any User can see aggregate analytics on tiles for a project group
- Any User can see aggregate analytics on tiles for a project
- Any User can see aggregate analytics on tiles for a user
- PM can see a list of Mappers for a project
- TM3 should support 100,000 simultaneous users on XYZ hardware (this card may be defined in terms of N users per M hardware nodes, assuming performance tests are published displaying linear horizontal scalability and the failure conditions for horizontal scaling)
- [NFR] Any API user can read a live data stream of activity from TM3
- PM can view a summary of projects & statuses by geographical area
- PM can adjust tile size to accommodate varying complexity/difficulty
- [NFR] Finish pending TM3 API endpoints for existing features (See TM3TC "Goals" 5.)
- Validator can ask a question of a Mapper for tile she is validating
- Mapper can view a TODO/checklist for her current tile
- Mapper can see a prioritized list of task squares available to work on
- [NFR] TM3 must support i18n
- [NFR] Any user can submit a translation or update to a translation for review
- [NFR] Mutating API calls must be authenticated
- Mapper/Validator can request access to a private project
- Mapper can mark a tile as "too cloudy to complete"
- Mapper can mark a tile as "imagery insufficient to complete"
- Mapper must review & certify TODO/checklist complete to mark tile as "done"
- Mapper can stop working on a tile to unlock it without marking it "done"
- Any User can subscribe to notifications for all comments made on a tile
- Mapper receives a "Welcome to HOT mapping" email upon first login
- Mapper receives a buffer around tile area when beginning work on a new tile