GEM IT Review - Reviewer's Report 20 July 2010 Final

[web summary]

Notes area

by Stephen Arnold, ed.

v 0.2 – 20 July 2010

With contributions included from:

Steve Arnold, Gentoo Foundation , Allan Hancock College

Shoaib Burq, Geoscience Australia

Jessy Cowan-Sharp, Sunlight Foundation

Stuart Gill, World Bank

Jocelyn Guilbert, Commissariat à l'Énergie Atomique

Jano van Hemert, Edinburgh University

Andreas Hocevar, OpenGeo

Chris Holmes, OpenGeo

Heiner Igel, Munich University

Linus Kamb, European-Mediterranean Seismological Centre

Phil Maechling, Southern California Earthquake Center

Joshua McKenty, NASA Nebula

Kevin Milner, Southern California Earthquake Center

Timo Multamäki, Eigenor Oy

Alessandro Spinuso, Observatories and Research Facilities for European Seismology

Frank Warmerdam, Open Source Geospatial Foundation

Notes area

Table of contents

Notes area

Notes area

1 - Overview and consensus recommendations

The primary (IT) goal of GEM is to extend the scope, but not necessarily the state-of-the-art, of risk and hazard calculation, as related to earthquakes. In order to achieve this, we expect our IT architecture to be:

- Open (Data, Source, Protocols, Standards, etc.)
- Pluggable (Modular, Loosely-Coupled)
- Dynamic (Fault-tolerant, Distributed, Constantly Updated)

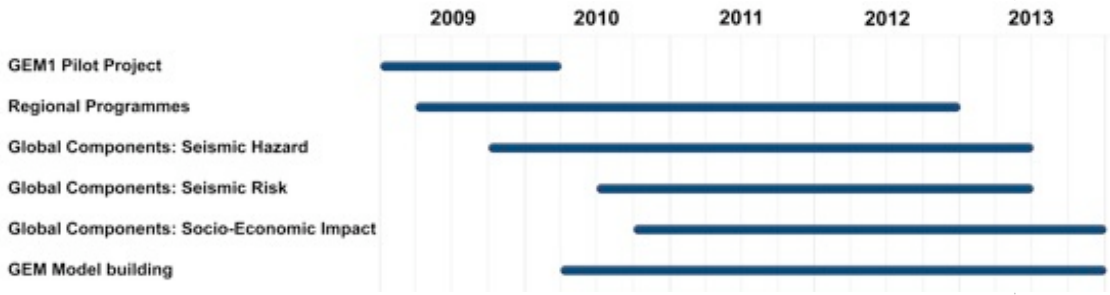

Figure 1. GEM Project Schedule

Note that the organization of this document, including the Section and Subsection naming convention, was more-or-less taken directly from the wiki page structure. The sections were roughly organized by content, with the notes section still in alphabetical order of reviewer's names (same order as the wiki). This document also contains some hyperlinks back to the wiki pages, which may be newer than what was current at the time this report was drafted. That said, any updates made directly to this report may not be reflected in the wiki.

The OpenGEM project has been disected into the following pieces:

- Over-arching architecture
- Database Schema
- Application Layer (including DAC and SOAP-based Web Services)
- Risk Engine (TBD)

Notes area

- Hazard Engine (including SHAML spec)
- Portlets
- Experimental UIs
- MapServer Stack (consumed by Portlets, currently)

Review outcome

What GEM did very well:

There is plenty in the GEM project which is good and commendable. We have listed some specific points, where we jointly agreed on an above average performance.

Looking for help from the outside (us) – in a formal way:

- Putting effort to get expert involvement. (Doesn't happen in most of the other European projects.)
- There is a will to understand the best approaches within GEM.
- The GEM secretariat has managed to find a good cross-section of different backgrounds for the technical reviewers
- Despite having vague requirements, GEM1 has already shown that the existing GEM team is capable of making results and not just idle talk
- Future-looking at an unprecedented scale. (Great potential)

Openess:

- Made a lot of good decisions vis-a-vis being open. Want the code to be FOSS. Thought about licenses. Open to criticism and feedback.
- GEM is a learning organization and it shows. Not afraid to question each other, try to do better the next time. (Vis-a-vis Ben's prototypes)
- WILLINGNESS to throw out the "built one to throw away"
- Hazard side really came together (around openSHA) with very little IT support. (Made good use of OpenSHA external team).
- Trying to do something great, haven't collapsed under their own ambitions.
- Willing to expose themselves for criticism (very atypical).

Other issues:

- Funding is appropriately arranged, which is rather rare in scientific related large projects. As project is not only getting funds from a single source, the probability of actually getting commonly accepted results is vastly larger. What they did accomplish (SHAML spec, etc) is very impressive.

Notes area

Productivity:

- Produced the global hazard map.
- Doing int'l distributed software development (which ain't easy to begin with).
- GC and RP organization seems well defined.

History and Context

We did not feel there was enough reference to outcome of Canberra meeting including recommendation that we focus on users that can verify and validate calculations done by system. In future IT reviews, we recommend a concerted effort be made to make explicit reference to previous reviews.

We recommend making public a roadmap with specific milestones, both to guide internal development as well as an aid to coordination and collaboration with external communities. Additionally, the decision tree or review process that's guiding prioritization of development should be explicit and published as well.

Ideally, both of these artifacts would be presented as a comparison against specific, existing projects and programs.

Management recommendations

Team management:

- Hire a full time IT manager with experience in FOSS development
- Manage as a single, distributed team
- Distributed is helpful – it will encourage the use of good FOSS practices
- Force good FOSS practices now (internally), before opening up project
- Frequent project reviews (architecture, code, spec, test plans, continous integration)

Unified and cohesive Team:

- Frequent intercommunication and meetings (face-to-face, virtual)
- Consider FOSS methodologies and tools to promote distributed team cohesion
- One set of tools
- Simplify (use the tools you set up)
- One repo, one document store
- Allow easy creation of branches and merging to allow flexibility
- It could be a good ideas to organize GEM workshops with

Notes area

scientists and ‘users’ to be sure that the DB and software (or portal) are in good correlation with the needs.

Hiring:

- Consider a UI/UX designer early on
- Technical Manager (see above)
- Part-time FOSS community engagement role (also see section on Community – FOSS and User Community). Continue to have IT reviews (small, targeted, frequent).

Testing

We recommend that a strong emphasis be placed on testing during subsequent development. This includes load testing (to identify potential scalability issues), regression testing (to achieve verification of scientific validity), and user testing (which will involve getting rapidly-developed prototypes in the hands of real users and collecting detailed feedback.)

(and don't forget unit testing as part of the personal software process)

Establishing the Continuous Integration environment is a top-priority. Test-driven development without CI, isn’t really test driven development.

Commit to a methodology:

- Might be agile or waterfall, choose and then stick to it.
- Use the same methodology for the entire project.
- Let the selected methodology inform the architecture.
- Follow the full set of practices. E.g:
    - If agile, perform force-ranking of captured user stories. Hold SCRUM meetings, and collect user test data at the end of every sprint.
    - If SDLC, fully decompose identified use cases to a functional specification.

Prioritize Requirements: (Tighter Focus)

Get a clear set of prioritized requirements (user stories, gurkin, etc.) This does not preclude rapid, iterative, or test-driven development, but it requires very clear ranking – even if for short periods of time.

We recommend you consider the GC teams as your initial, high-priority set of system users. As well, from a management perspective, if these teams are developing software, they should be considered and coordinated as if FOSS external contributors. Use FOSS project management, team communication, peer review, etc.

Notes area

The existing prioritization of users is weak. The concept of actors may add confusion – Actors seemed to include both end-users, decision makers, and funding organizations. More specifically, however, these Actors were not referenced in later presentations.
If you are going to advance these as a development tool, consider developing an end-to-end use case example (scenario) for each type of actor.

Again, understanding requirements does not mean spending years writing – short, iterative releases (in weeks or months) with real user testing are a highly effective tool for gathering requirements. Ensure the ICT manager has full responsibility for these use cases; she can present them in the next review, and explain why design decisions match the requirements of these cases.

Leverage Existing Projects More

- For difficult code, esp. where there are existing well-tested components
- For collaboration, community building – contribute to them, don't just use
- I.e. GeoNode
- Go visit users and ICT experts in person
- Exploit userbases of other projects, e.g., NERIES mentioned the users of their operational services

Community (FOSS and User Community)

- Don't defer this
- Adopt FOSS practices internally (Fogel's 'Producing Open Source Software' is a great handbook for all the best FOSS practices)
- Regarding community management:
    - Each team member should be participating in the community now, even if the source is not yet open:
    - A very rough estimate of time might be 3-4 hours/week for developers to engage in community activities
    - Build in exception notification to promote accountability for the code base
    - GEM Blog: Blogging about the project, interesting technical challenges, etc.
    - External participation – attend meetings and meetups. Consider a training and conference budget for each developer to get out into the community, learn, and simultaneously increase awareness of the project.
    - Organize development sprints -

Notes area

        - within the team, and with the broader community.
        - sprints can seed projects GEM will build on
    - Contribute to other (external) FOSS projects – provide patches, help out their docs, make examples for them, earn commit rights
    - Encourage taking ownership of tasks – by ticketing, nominating domain experts etc.
    - All decisions taken by the project should be done in archivable manner, on mailing lists, irc, or at least documented if decisions have to be done in person. Even if project is not 'open source' for next 6 months then archives should be opened when it gets opened.
- On the question of hiring a full or half time community manager, may consider someone half time, but more important is for each member of the team to have at least 4-8 hours a week where their priority is doing the above activities.
- It could be a good idea to organize GEM workshops with scientists and 'users' to be sure that the DB and software (or portal) are in good correlation with the needs – these can be arranged in concert with the development sprints mentioned above.

FOSS Development Process:

- Use a style guide for writing/contributing code, writing tutorials, using the collaborative space, etc.
- Commit to a consistent naming scheme (current naming schemes have collisions and confusions)
- Plan for ticketing, reviewing, accepting contributions
- Don't call it FOSS until it's FOSS. See http://en.wikipedia.org/wiki/Free_and_open_source_software
- Specify a date at which code will be released, community will become open.

Social Engagement and Media

There was an early discussion of crowd sourcing, and the significance of the GEM work. But the review did not return to this topic of hazard and risk and how GEM can help with this during later discussion.

- If Crowdsourcing is a key requirement, it needs to be addressed earlier in the design
    - Crowd-sourced dataset generation (such as fault or exposure databases) is a role that should be supported in the system – build social tools based on those use

Notes area

cases, not abstract 'social' ideas.
    - Proper metadata and formats might be needed in order to have these user generated datasets interoperable and searchable.. consider the Linked Data and persistent URIs approach for non-sensitive datasets
    - Make the "Haiti Children" aspects of "why we're doing this" a more central part of the messaging (esp. in the FOSS community side)
- Suggested frameworks include django (http://www.djangoproject.com/)
- Exploit existing social networks, how many seismologists are on LinkedIn?
- Regarding "semantic wikis"
    - This was mentioned but not clear what the role is. Why semantic over any other kind of wiki? What does this mean?
    - Efforts around ontology and semantics formalization would be good to start as a public facing, community effort now.
    - With respect to GEM as a system, focusing on developing ontologies and semantics might be out of scope.

Licensing

- CC-zero for content http://creativecommons.org/choose/zero/ (PDDL is a similar option, but CC has nicer awareness)
- A statement on community norms, like http://www.opendatacommons.org/norms/odc-by-sa/
- Add to that a statement that all data contributed to GEM will be open for ever
- Statement on community-contributed content (will be public, open for ever)
- LGPL is good enough, revamp if necessary (be open to relicense if requested)
- Collect Contributor License Agreements (copyright assignment)
- Use PPK to sign authoritative, GEM-certified data assets. (In the future, you could extend this capability to allow any user to sign their data products, both for authenticity and data integrity).
- Design nice logo for GEM-certified data assets, trademark it and control its use

Data Schema / Database:

- Address the concerns from the scientific community regarding schema

Notes area

- Abandon a monolithic database
- May need to support distributed querying
- Decide if it's important to optimize for the most remote, poor-connectivity regions
- Defer lightweight/standalone clients until primary capability exists.
- Develop tools for DB visualization in terms of quality of the inputs and harmonization of the different tables
- Be sure that the DB embeds all the needs of users in terms of computation or inputs for the computation
- See a first scheme of the organization concerning the DB management (if the DB is distributed this point will became more and more critic)
- Resolve conflicts between SDLC-style data modelling, ORM-based modelling, and the necessary schema evolution (additional mapping layers may not be the ideal solution)
- Consider schema-less, column-oriented, and document-oriented alternatives to RDBMS
    - To deal with Schema extension / churn
    - For scale of certain data types

Data Exchange Formats:

- Various reviewer notes in Section 3 will include specific commentary around the data exchange formats
- SHAML, QuakeML adoption, etc. need more prominent attention as specific IT efforts
- Define the minimum requirement, keep to the minimum
- Plan out extension of SHAML to cover risk and SEI
- Be cautious regarding the level of effort involved in support of these formats
- Several reviewers have concerns about the scalability or suitability of an XML format to the type of data exchange being proposed. Look for various viewpoints in Sections 2 & 3.

Front end

- Central issue: "JSR168/286-compliant frameworks (i.e. portlets)" vs "non-JSR168/286 framworks", see also JanoNotes and Django recommendations
- Emphasize requirements, not implementation spec (eg. JSR168/286)
- Not as important to choose portlet versus non-portlet; instead requirements should dictate appropriate (and help identify inappropriate) tools and technologies
- Doesn't seem like the requirements for front-end are well

Notes area

understood
- Disagreement: majority felt portlets are not the right technology, but there were dissenters. See JanoNotes for more on JSR and portlets.
- Make use of tools to build portlets, do not manually develop everything, this is too expensive and too expensive to maintain in the future
- The rapid prototyping effort (in javascript, python, or other high productivity languages) seemed valuable to explore potential features for implementation. In the context of JSR development, tools such as Rapid will speed up prototyping to days rather than weeks or months.
- Front-end capabilities should be analyzed from two directions
    - bottom-up analysis: portal should expose key functionalities of the system
    - top down: UI mockups need to be done which identify desired user-facing capabilities; those are likely to highlight additional technical functionalities which need to be added to the spec. e.g. user sharing of content, ability to edit/fork models, etc.
- Understanding the front-end

Calculation Engine / distributed computing:

- Don't buy or build a cluster, consider using free solutions first, e.g., clusters and Grids (EGEE, D-Grid, UK-NGS, TeraGrid, NorduGrid, TGCC-CCRT, etc.), then move on to HPC, if you have HPC requirements (PRACE, DEISA2), and cloud vendors (Amazon EC2 and many others).
- CUDA / GPU processing / MPI is all premature optimization, may not fit nature of calculations
- Condor-type, embarrassingly parallel (high-throughput) solution is appropriate, but should not tie into one 'job submission engine'. Keep dependency on these systems as low as possible (See Section 3 for specific recommendations)

Distributed System / Standalone OpenGEM

- Defer addressing construction of lightweight client, standalone system, or federated/distributed system for 2 years
- However, develop calculation engines as standalone applications **now**
- Consider CLI interfaces to these components (as well as WS/REST), many tools exist to take CLI and then wrap these into services. Also many tools exist to web-ify CLIs, including interfacing with cluster/HPC/Grid resources

Notes area

- Keep engine, front-end and datastore loosely-coupled and independent
- Consider these as potentially distributed systems vs. simply distributed data sources, but defer design decisions (specific recommendations in Section 3)
- Modular design (stand alone calculators, decoupled web applications) implicitly distributes computing resources

Deal with Scale

- GEM's scale distinguishes it from similar efforts. Support for scale seems to be a critical requirement
- For RDBMS, generate example data sets, populate tables and evaluate usability. Modify design of database, or design of data storage if system does not support the required global scale.
- Scale consideration include more than just RDBMS. Consider thinking about future data-intensive needs
- Project management courage is required when taking decisions as future scale is difficult to assess

Rapid Prototyping

- Adopt YAGNI: "You ain't gonna need it"…don't add functionality until needed
- Release early and often, get real user feedback
- Look for and use appropriate tools for rapid prototyping (including pencils)
- don't set up tools for the sake of setting up tools. For example, GEM TRAC system is installed but does not seem to be used..if you're not going to use it, abandon it.

Web Services

- Publicly accessible, versioned service API should be included
- Consider using REST – and use it to access services from web applications
- Consider using Web Services Resource Frameworks (WSRF) as the project is exposing resources
- Dogfooding (that is, using the tools that you create) is important (ideally done by different people, and with a different programming language. etc.)
- Make sure an external (or at least distinct) user is also consuming the API, preferably in a second language (javascript is ideal, since it has lots of constraints and is likely in mashups)
- Current system design may overuse web services internally

Notes area

(only make sense internally when accessed from decoupled web applications)
- The current granularity of the web service does not seem quite right. It would likely be a mistake to convert to XML format between seismic application and database, and to use only web service interfaces to access database.

Single Sign On (SSO)

- Most felt the existing SSO model was not quite right – too tightly coupled to portlets, did not adequately protect data sources.
- Adopt an existing standard mechanism, don't reinvent the wheel
- (Specific recommendations in Section 3, many favor SAML/Shibboleth)

Additional resources and links to development tools

- Software for distributed agile project management: pivotal tracker
- Language for user stories: Gherkin
- Collaborative document editing: etherpad
- Test Driven Development TDD
- Behaviour Driven Development BDD a logical extension to TDD
- Grinder – Java Load Testing Framework
- UI Mockup Tools: GoMockingbird and Balsamiq Mockups
- O'Reilly RESTful Web Services
- Fogel's Producing Open Source Sofware
- ReviewBoard – A tool to make code reviews fun…
- Find Bugs FindBugs is one example of a static analysis tool for finding bugs in Java code.
- Schematron to valdate XML
- Rapid to quickly build (without traditional programming) JSP servlets and JSR portlets for running back-end computing jobs
- RelaxNG keeps XML readable
- JING make screencasts to show users how it works and what they can expect and share these

Acronymns

- REST Representational State Transfer Article explaining REST in non-techy speak (slightly sexist)

Notes area

2 - Specific supporting topics

DataSchema - OpenGEM Database Model Ver 1.5

Overview

ORM mapping using Hibernate in JAVA.
ERD Deconstruction
Classic 3rd Normal Form

DB SERVER: gemsun01.ethz.ch

Tools:

- Java Topology Suite
- QuantumGIS
- PgAdmin III
- Hibernate Tools, including Hibernate Spatial

Concerns

- System-level tuning (e.g. complex sharding schemes, postgres-specific DB tuning efforts, expensive hardware, etc) is not necessarily portable for software that should be also run locally.
- The GEM system data may be best expressed in a combination of SQL and NoSQL formats. Innovative approaches to data representation may be necessary, esp. for the (potentially large sets of) point data.

References

- LTree Module

NoSQL Options

“CouchDB”: http://couchdb.apache.org/

MongoDB

Notes area

- Example of (crazy) people using it for GIS: http://www.paolocorti.net/2009/12/06/using-mongodb-to-store-geographic-data/

Cassandra

Riak

MonetDB has OpenGIS included. They are very willing to help. Jano can provide contacts,

Is it feasible to have GEO NoSQL? Some folks say yes…

Quantified Userbase

How many users will GEM initially be supporting, and of what types?

Model Builders: Global Components

- There are currently 5 Global Component consortia on “hazard”
- There are currently 5 Global Component consortia on “risk”
- There will likely be 1-2 Global Component consortia on “SEI”
- Each of these is estimated to represent between 40-100 users.
- (Total of 440-1200 users.)

Model Builders: Regional Programmes

- There are about 10 Regional Programmes
- Each is estimated to represent around 250 **active** users.
- They may require i18n, l8n, training and capacity building.
- (Total of 2500 users.)

Minimum Requirements

Due to opportunities for capacity building, etc., it is reasonable to develop the GEM platform targetting modern browsers (e.g., HTML5 and WebSockets).

GEM use of Web2.0 and Social Media

Much of the data required for a global understanding of risk is currently

Notes area

unavailable.

Crowdsourcing is considered a key approach to collecting this data. This may also include aspects of contesting or other incentive schemes (ala DARPA's red balloon).

Methodology fight: Agile Vs. Waterfall

This section was started by Steve (who wanted others to contribute) and was intended to capture some thoughts about agile methods vs. the more traditional "waterfall" model of software engineering, but focused on the context of the GEM requirements for both "openness" and scientific verification.

Although agile methods hold the promise of both rapid development and a way of handling evolving/unknown requirements, a more traditional approach to requirements engineering, including some form of verification and validation of the software, may lend itself better to the goals of V&V in the context of GEM. The key from my point of view will be integrating a solid V&V effort with an appropriately "agile" project environment.

Areas of emphasis that should always be included:

- Frequent communication among relevant project members (up, down, and sideways).
- Use of the proper tools to understand (and document) source code. Someone else will struggle to figure it out at some point (and it may even be you). See the StaticCodeAnalysis page for details.
- Don't let anyone struggle alone; hold peer reviews, assign mentors and/or partners, and make sure your co-workers aren't stuck on something or waiting for something (and don't be afraid to throw away painful code and re-implement it to get it right).

Static Code Analysis

You should always analyze your own code, including generating metrics & documentation, as well as making full use of static analysis tools. Many great free tools are available, and most provide both GUI front-ends (e.g., an Eclipse plugin) and command-line interfaces (for backend automation). A short list of tools will give some examples:

Notes area

Statistics and metrics

- SLOCCount – SLOCCount is a handy tool for counting source code and basic project estimation. Too useful not to use it.
- CCCC – CCCC generates detailed metrics and points out potential problem areas in your code (with nice html formatting). Another extremely useful tool…

Documentation and design extraction

- Doxygen – Doxygen is a source code documentation, analysis, and reverse-engineering tool for several languages, and can be extremely useful when incorporated into the software development process (ie, by defining your coding style to include doxygen-style comments; can also use JaveDoc style) and incorporating your own internal docs as well.
- JavaDoc – JavaDoc comes for free with the JDK. Use it.

Software engineering and analysis

- Eclipse – Eclipse is an all-in-one software environment for multiple languages, with plugins for pretty much everything from analysis to visualization to test and deployment. Yes, it can get bloated, but it's also the best interface between human and code ever invented.
- Plugins – There are so many plugins for Eclipse they needed eclipseplugincentral.com…
- FindBugs – FindBugs is a tool for finding bugs in Java code. If you write (or work with) more than 10 lines of Java, you should really be using this tool…
- PMD – PMD is another analysis tool for Java (not at all orthogonal to FindBugs). Also highly recommended.

Software engineering tools at tigris.org

- Subversion – Clients, plugins, docs, etc (in the process of moving from tigris.org to apache.org).
- ArgoUML – The best free UML editing system around (supports DoDAF notation).

Other

- Source Navigator – Source Navigator is a cross-platform software development and code analysis tool with multi-language and multiple toolchain support.

A short presentation on this topic is also available on SlideShare

Notes area

Outstanding Questions

Editor's note: Some of these seem to be answered in the individual write-ups.

- What existing open source projects (with attendant developer communities) could be co-opt'd / merged with GEM?
- What primary test case (with attendant full-scale data set, etc) can be used to further develop / validate requirements during the next 6 months of development?
- What sort of "requirements roadmap" can help guide this "agile" process?
- Why Latitude X Longitude instead of Geocode?
- Is it feasible to achieve a unified data format (for exchange as well as internal representations), or will the internal representation always be more complex than exchange format?