PDF.JS Contribution Plan and Documentation

CSCD01 Deliverable 1

Northern-Horizon

Table of Contents

Overall Architecture

The pdf.js is an open source PDF document reading and analysis plug-in supported by Mozilla that is implemented as the built-in PDF viewer in FireFox version 19 or higher and can be used as a Google Chrome extension. It parses raw arrays of bytes into stream of pdf bytecodes then compiles the bytecode into javascript program which draws on HTML5 canvas.

Most Relevant Files and Folders

The pdf.js system mainly contains two library files (one Library files), pdf.js and pdf.worker.js, which are respectively responsible for API analysis and core analysis.

```
├── build/
│   ├── pdf.js                              - display layer
│   ├── pdf.js.map                          - display layer's source map
│   ├── pdf.worker.js                       - core layer
│   └── pdf.worker.js.map                   - core layer's source map
├── web/
│   ├── cmaps/                              - character maps (required by core)
│   ├── compressed.tracemonkey-pldi-09.pdf - PDF file for testing purposes
│   ├── debugger.js                         - helpful debugging features
│   ├── images/                             - images for the viewer and annotation icons
│   ├── locale/                             - translation files
│   ├── viewer.css                          - viewer style sheet
│   ├── viewer.html                         - viewer layout
│   ├── viewer.js                           - viewer layer
│   └── viewer.js.map                       - viewer layer's source map
└── LICENSE
```

After downloading pdf.js, the decompressed directory structure is mainly composed of build and web packages. Pdf.worker.js in the build package is the core processing package of PDF.js, and viewer.js viewer.html viewer in the web package. CSS is used to display rendered PDFs (borders, toolbars, etc.) on a Web page.
The images folder stores some toolbar icons and other content, and the locale folder stores local language packs.
debugger.js is debug-related. When we use it, we can enable debugging to output certain information for debugging.
compressed.tracemonkey-pldi-09.pdf is a PDF test class. This file is loaded by default when we do not load our own defined PDF.

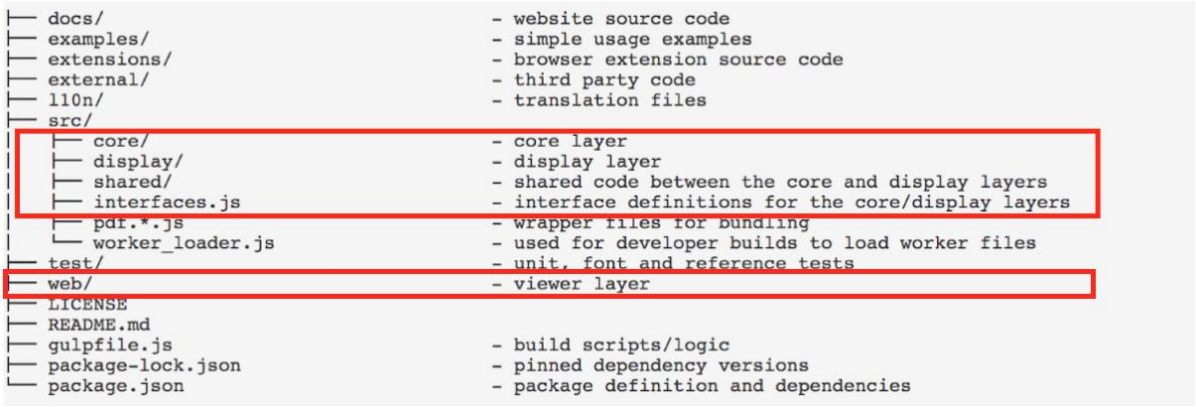
```
├── docs/                                   - website source code
├── examples/                               - simple usage examples
├── extensions/                             - browser extension source code
├── external/                               - third party code
├── l10n/                                   - translation files
├── src/
│   ├── core/                               - core layer
│   ├── display/                            - display layer
│   ├── shared/                             - shared code between the core and display layers
│   ├── interfaces.js                       - interface definitions for the core/display layers
│   ├── pdf.*.js                            - wrapper files for bundling
│   └── worker_loader.js                    - used for developer builds to load worker files
├── test/                                   - unit, font and reference tests
├── web/                                    - viewer layer
├── LICENSE
├── README.md
├── gulpfile.js                             - build scripts/logic
├── package-lock.json                       - pinned dependency versions
└── package.json                            - package definition and dependencies
```

And for the Source part, there are several files included.
docs: mainly static files and reference documents
Examples: Mainly examples that use pdf.js

External and l10n: mainly configuration parameters in js or properties format
Src: is the core PDF parsing component of the entire pdf.js plugin, mainly some js files
Test: some test files
Web: It is a complete online PDF document browsing example that provides good multi-functionality. The JS file is the implementation of these functions.
Gulpfile.js: These are some js configuration parameters of the gulp server, mainly because the project uses gulp server.
Package.json: some dependency parameters configured by webpack packaging

System Design

The pdf.js is designed as a PDF reader. The functionality is broken into 3 different layers of the pdf.js project, as shown below.



A core layer - interpret and parse binary PDF content via HTML5 Web Worker. This is the foundation for all subsequent layers.

A display interface layer - takes the core layer and uses API to render PDFs and get the information out of the document.

A ready-to-use PDF viewer layer - Built on the display layer and is the UI for the PDF Viewer. It supports features like search, rotate, print, page, thumbnails … etc.

By looking at pdf.js main functionality, we can see that the program followed the layered pattern. This pattern is used to structure programs that can be decomposed to group of subtasks where each layer provides information/services to the next layer. Each layer is at a particular level of abstraction as can be seen. Core/ represents the data access layer, Display/ represents the business logic and application layer and web/ represents the presentation layer. These are the most commonly found layers in the pattern.

We think that the architecture style used is appropriate for this project. Since this is an open source project, using a layered pattern makes it easy for contributors to test, develop and maintain due to the well defined components. Layered architecture also allows teams and individuals to work on different components with minimal dependencies.

Software Development Process

Agile Kanban Development

Agile development is a very efficient way of working in teams. The biggest feature is members can complete the set goals in a short cycle, minimize verification, and adjust at any time.

It values the cooperation and interaction between individuals, and we end up delivering problem-free software instead of a pile of heavy documentation. It requires us to understand customer needs and communication with them. And the needs of customers are constantly changing, so we should be more adaptable to change. Instead of simply relying on the product requirements list to execute the project, under the agile guidance, we can continuously update the version to observe customer feedback.

The reason we choose Kanban is that, in order to achieve maximum efficiency, it focuses on balancing the team's capabilities with the work currently underway, thereby it could reduce the risk of any bottlenecks. At the same time, Kanban is also very responsive to changes, as changes can be made at any stage of the project and added to the list of tasks to be performed. And our project will surely have many frequent changes because the problem proposer will find more bugs.

Furthermore, Kanban does not have strict requirements for team roles, and the team is self-organizing. According to our development model on Mozilla, we can't communicate face-to-face with the person who found the problems like scrum, we can only read and reply to their respective information as quickly as possible.

Kanban can sort out the progress and ownership of all projects: to-do, doing, completed. It could let developers understand the relevant information of the project, and make things gradually simple and clear. Each team member takes a task from the to-do list and focuses on completing it. After the task is completed, the member chooses the next one, and so on, until the to-do list is completed.

But here, because the task we need to complete is to fix the vulnerability, we will make some modifications based on it. We will look for six to seven different levels of tasks for team members to choose from. If a team member completes his task, this team member will not choose a new task, but will judge a relatively difficult task based on the progress of other ongoing tasks and join to help solve the task. Or if there is a more complicated task, we will split it into small tasks and let the team members choose.

At the same time, the amount of work in progress during the project will not exceed the capacity of the team. Each task follows the "to-do"-"work in progress"-"done" route, which is very clear. Kanban development will limit the number of tasks, which also matches with our team not many requirements to solve the vulnerability.

Other Processes

The Waterfall Model

The waterfall model better suits large software systems that are developed by multiple teams collaborating together, where all contributors should have a clear understanding of the requirements and have strict consensus regarding each component of the project. This model was not considered since, as a plan driven process, the waterfall model is not flexible to changes of requirement, where the requirements of an open source project might be changing and updating over time along the development process.

Incremental Development

Incremental development might be a good development approach where the customers can actually participate in the development process by providing useful feedback for improvement. However, as a short-term generic project and the fact that the team is contributing to a relatively small part of the whole project, the team does not have control over deployment of the system, at the same time, it is difficult to gain feedback from users.

Spiral Model

The spiral model is a complex risk-driven approach that requires risk assessment for each phase in the development process. With consideration of both the exact and modified version of the model, professional risk assessment is needed for this model, which is not realistic for the team and is not necessary for an existing open source project.

The Rational Unified Process

The Rational Unified Process is an approach of development that is derived from UML design and involves business use cases inceptions. It requires the team to spend more effort on developing the system model with comparison to implementing it with coding. From an existing open source system that requires improvement, it already has well constructed system architecture models that can be used in further development. Due to the purpose of the project and time constraints, this is not considered to be a suitable approach because of its heavy weight required on designing.

Extreme Programming

Extreme programming was also ruled out in the early stage of the discussion, because of its feature that requires a large proportion of customer involvement, including frequent incremental delivery to the customers as well as customer participation in the testing process, where a generic open source project does not have a clearly defined customer or business client. On the other hand, some of its practices can also be applied in the process chosen for improvement such as pair programming, which is to review each other’s code to ensure the component is implemented correctly.