Teaching with Dynamic Documents
Web Applications and Local Resources

Jerzy Karczmarczuk
Dept. of Computer Science, University of Caen, Caen, France

Keywords: HTML5, Javascript, SVG, Servers, Python, Tornado, AJAX, LaTeX, Mathematical Tools, Moodle, Nodejs, Websockets.

Abstract: One of the bottlenecks in the application of computer methods in teaching is the limited effort in the development of tools for the *creation* of pedagogical material (documents, presentations, software). These tools exist, but they are often dispersed, and sometimes not well known.
Some methodological work would be very useful. Our talk concentrates on the usage of Web-oriented information processing techniques, from the perspective of an individual: making presentations, scripting them, adding dynamical content from specialized servers, enhancing the interaction between different sources, etc. We give some simple examples belonging to the domain of teaching of sciences. The usage of servers help to modularize the teaching software support, avoiding big, monolithic applications.

1 PEDAGOGICAL CREATION AND WEB TECHNOLOGIES

This talk is addressed mainly, not exclusively, to teachers of mathematically oriented sciences, who want to prepare visually rich, dynamic, and interactive teaching documents, and who feel that the *methodology* of construction of such materials is underdeveloped. Our target reader need not be a Web developer, but she should be acquainted with the basic notions, such as the structure of Web pages, scripting, the concept of server, etc.

We share here our experience with teaching various elements of computer science at the university level, and cooperating with some secondary school teachers. We taught computer-assisted cartography, scientific visualizations, multimedia documents (images and sound), simulation of dynamical systems, etc., which also needed visualization, and interactive experiments during the lectures: animations, executable code snippets, and sound samples, etc. We feel that the progress in this domain is chaotic. The ever growing database of pedagogical software is not always sufficient for making other applications.

We recommend the usage of tools usually associated with the Web programming (HTML5, Javascript, applicative servers), considered as the building bricks of local environments, autonomous, and installable on the teachers' and students' computers, with browsers used as the main interfaces.

The world-wide success of collaborative computer-assisted teaching platforms such as Moodle ((Dougiamas and Taylor, 2003; Moodle, 2015)), Chamilo, etc., is established. Course databases, cohort assembly, management of timed examinations – all this frees the teachers' time. As a course *creation* environment, they seem to be less useful. Davidson and Waddington observe ((Davidson and Waddington, 2010)) that teaching matter they saw on Moodle, are mainly static pages: PDF, Powerpoint slideshows, frozen video-tutorials, etc., manufactured independently of the system. We observed this as well, at our university, and elsewhere. As authoring systems, the LMS, LCMS etc. will evolve, but the progress is slow. Creating under Moodle a course with shared scripts or communicating with some server-side applications, is difficult because of many constraints to fulfil. All this requires some competence in software engineering, while the effort of a teacher should be, and usually is concentrated on the taught matter, and on the teaching itself, they are not Web developers.

1.1 Reasons to Choose "Web-based" Tools

The possibility to deploy the documents on a Web site, enabling thus the distant access is an impor-

Karczmarczuk J..
Teaching with Dynamic Documents - Web Applications and Local Resources.
DOI: 10.5220/0005447803150322
In *Proceedings of the 7th International Conference on Computer Supported Education* (CSEDU-2015), pages 315-322
ISBN: 978-989-758-107-6

tant, but secondary issue, since the teaching operates mostly in direct contact between teachers and students. Our "target context" is the classical teaching with lectures, home assignments, and practical computer work, rather than some MOOC.

The main reason, mentioned already, is the standard possibility to enrich the documents with dynamic elements: animation scripts, linked video sequences, inclusion of other documents (iframes), style modifications, **internationalization**, pop-up windows, etc. We didn't use any other presentation platforms for many years. Difficulties for creators who are not Web specialists persist, but it is easier to control the details in an autonomous environment, than within the LMS such as Moodle, which for rational reasons insist on the coherent usage of built-in tools, and which respect several security constraints, which make experimenting difficult. But there are other reasons for replacing the Powerpoint/ImPress slideshows, or PDFs by HTML, and to use the "Web-based" tools.

- According to the philosophy that learning tools should be freely and widely available, the readability and facility to copy and edit a document is a major point. If the author wants to protect the document, it is easy to export a frozen slideshow, but this should not be frequent.
- A Web page is a *distributed document*, with images and other fragments kept separately. This makes it easier to share them, and the transfer process of fragments may be parallelized, and/or delayed using AJAX.

We recommend the installation, and active maintenance of local resources, including servers. This means more initial work, but less problems afterwards. It should then be easy (for the teachers and for the students) to work with or without Web access, to profit from the learning sites if accessible, but to use more often their own local copies of documents and programs, it is cheaper, safer, and controlled.

In building our teaching environment, we insisted that only free, legally clonable, and easily installable software (apart from the platforms themselves, such as Windows, or Mac OS), should be used; several third parties packages, scripts, and images, could thus be shared. We avoided "experimental" or unknown packages, pedagogical environments should be reasonably stable.

For the dynamically generated documents, the server-based communication is a necessity, even if one works *in situ*. Server installation is an easy task, web servers, locals and remote, begin to be included by default in every operating system and every programming language support, this tendency is manifest. Installing Apache is straightforward, but one of the harmful, persisting myths among the casual users of Internet, is that the users are just clients, and the servers, heavy "monsters", reside in some mythical, "professional" domain.

2 HTML5 FOR PEDAGOGICAL DOCUMENTS

A good course may be prepared and delivered using Powerpoint, or chalk, since good tools are well-mastered tools. Old HTML pages were badly formatted. But the typesetting quality of Web documents becomes very good, current browsers recognize also the hyphenation, it suffices to declare the style for an element class, say:

```
p {hyphens:auto; -moz-hyphens:auto;
   -webkit-hyphens:auto;}
```

and to specify the language, e.g., `<p lang="pt">`, and if it is insufficient, it suffices to load and activate the script Hyphenator of Mathias Nater ((Nater, 2015)); it inserts automatically the conditional hyphen `&shy;` where due, and uses the same multilingual hyphenation patterns as TEX and OpenOffice[1]. There are other reasons to use HTML documents:

- It is considerably easier than with other formats, to provide different styles permitting to show the document on a large screen, on a computer, or printed, to dynamically change the background colour, etc. HTML5 with CSS3, address specifically this problem, with such attributes as `media="print"` (or `"projection"` or `"aural"`, etc.), which can be queried and steer the formatting.
- Pedagogical texts profit from many sources. While the "copy and paste" techniques are easier for HTML, since this is a plain text, without hidden style elements (as in Word), it is *much* easier than elsewhere to assembly compound documents by linking, to construct mashups, and other hybrids.
- The responsive, adaptive design of Web-oriented documents, requires the scale neutral, dynamically rasterized "vector" graphics. The SVG (Scalable Vector Graphics) standard (with animations and scripting ((Eidenberger, 2003)) is there, easy to generate, easy to transform and render, but not directly usable for the TEX or XXX-Office writers.

[1] A local polling has shown that almost nobody uses the hyphenation in the Web documents, people are not aware of these facilities. There are many other formatting stylisations, such as `word-wrap:break-word`, unknown by a majority of Web users...

But, being an ordinary (XML) text, a SVG document is easy to transmit through the network, or to paste into an HTML page.

- The audio and visual (canvas+SVG+CSS3, video and WebGL) facilities in HTML5 offer more than just the possibility to render multimedia documents. New browsers are tightly integrated with the underlying operating system, and became ***universal programming platforms/interfaces***, which may control local and remote cameras, synthesize sounds, and display realistically textured and animated 3D objects through WebGL ((Khronos, 2015)), which involves the control of the GPU (Graphics Processing Unit). New high-level intermediate packages targeted at non-specialists, e.g., Three.js ((Three.js, 2015)), or X3DOM ((X3DOM, 2015)) are born every month. For teaching this is much more important, than adding colourful decorations to course materials.

  We taught image processing with live examples, without specialized libraries, and (usually) without having to launch external applications. The Web Audio layer ((WebAudio, 2015)) gave us the opportunity to show within the browser, the implementation of simulated musical instruments. Javascript turned out to be a usable language for dynamical simulations and graphics. The main problem was the lack of good development tools. This is not a critical issue for experienced programmers, but beginners need more user-friendly interfaces.

- Mathematically-oriented elements were, and will remain very important for us. Programs which render TEX mathematics in HTML pages are numerous, actually the most comprehensive, configurable, and well documented is MathJax ((MathJax, 2015)), included in several Moodle sites as a filter. The quality of rendered formulae is almost equivalent to standard LATEX. Since there are several transfers of auxiliary elements (scripts, fonts, etc.), its remote usage involves some overhead, but the entire package is easily storable locally.

2.1 Compound Documents

All text processing software permits to assemble documents from parts, forming a static, bigger whole. Also, everybody knows how to *link* together different pages, loaded separately into the browser. Less is known about assembling a Web page from different, shared fragments during the rendering, but as a one loading process (this is a simplified view of the mash-up concept). The inclusion of images is a known standard, but the aggregation of text fragments remains rare among non-professionals. The inclusion of HTML in HTML is still not standard. However, the existing tools are easy to use.

- Many Javascript libraries offer calls to `XMLHttpRequest()`, a function which accesses a distant document, or a local file by its name, and writes its content. E.g., a user not interested in technical details, who is aware of a popular package `jQuery`, simply provides in his document an identifiable placeholder:

  ```
  <div id="pageHere">
  This part will appear when ready
  </div>
  ```

  and after loading the library, executes: `$("#pageHere").load("otherDoc.html")`. The other document appears where it should, and if it is not available, the document shows the warning. This may be used, e.g., to prepare exercices with solutions, but to postpone the deployment of these solutions. This is an example of AJAX (*Asynchronous Javascript and XML*) call. The loading may be automatic, during the rendering of the document, or triggered by a click on a button, using the same library.

  In some circumstances this will work only if the pages are loaded from a server, *via* `HTTP:`; using a local file might not work. This is one of many reasons to install a local server on a teacher's computer; many dynamic behaviours need an active entity within the document provider.

- Chrome users might use the directive `<link rel="import" href="other.html">`, and it is quite probable that similar contraptions will be implemented in other browsers. (They need some scripting which we will not discuss. The situation is evolving.)

2.2 Another Dynamic Inclusion

This example will use an external *application server*, i.e., a server which takes some data from the client, and *generates* another page or an included fragment, as above. Suppose that a physics teacher wants to plot some programmed, changing formulae, which would demand the installation of a programming language with a good scientific and visual support (say, Anaconda Python ((Continuum, 2015)) with Matplotlib ((Hunter, 2007))).

If during the course the teacher toggles between the descriptive layer (slides) and the Python interface, in order to generate and show the plots, it may show some potentially useful coding details, but normally

will be too disruptive. The idea is to have the program working aside, and communicating directly with the displayed document. Python distributions may include a small universal server, Tornado ((Tornado, 2015)) (if not, it is installable). This is the application server, which can read some parameters from the page, execute any code, and answer by sending back the plot structure formatted as an SVG document or some other format. *The user doesn't need to know much about it, it suffices to load and launch it from his program*, adding a small boiler-plate code chunk.

The main function of a server is to wait (listen) for a request, and to write an answer. We need only to write the **`Handler`** object, which deals with the "payload". The example below was coded by 1-st year students (knowing some Python, but nothing about servers) in less than 20 minutes.

```
result = ... construct it with "plot(...)"

from tornado.httpserver import *
from tornado.ioloop import *

class MyHandler(
 tornado.web.RequestHandler):
  def get(self):
   self.set_header("Content-Type",
                   "image/svg+xml")
   self.write(result)
application = tornado.web.Application(
   [("/myplot", MyHandler)])
HTTPServer(application).listen(8000)
IOLoop.instance().start()
```

It suffices to open a browser, and to point it to **`http://localhost:8000/myplot`** in order to get the image loaded and rendered by the browser. The command **`set_header`** informs the communication system that the sent string is a drawing, and this must be known, but this is all. The teacher never quits the course pages. We can also insert dynamically generated images. We declare a frame for the plot, say

```
<div id="plot"></div>
```

and the script:

```
$("#plot").load(
    "http://localhost:8000/myplot");
```

loads the image asynchronously, in real time. Such experiences usually need more work, some data might easily be passed to the server, change the plotted function, or colours, etc. ***A full-fledged code example is available upon request.*** Javascript/SVG librairies which cooperate with AJAX protocols are now standard in most modern commercial sites, but several free solutions, such as Snap ((Baranovskiy, 2015)), merit our attention. This library is used among others by PBS Kids site (Public Broadcasting Service) ((PBS, 2015)) for children, but with some educational resources for their parents.

2.3 More on Mathematics on the Web

The inclusion of high-quality formulae within the Web documents through the translation of LaTeX, became standard. But many other tools for the dynamical visualization of mathematics are available. Mathematics teachers usually know GeoGebra ((Geogebra, 2015; Hohenwarter, 2002)), but we found out that several didn't know that it is possible to work with Geogebra structures within a browser. The application includes its own scripting engine. Other Web-oriented mathematical tools are less known that they deserve. Some are online, but for the autonomy and speed, downloadable libraries are better. Our favourite is the JSXGraph project from the University of Bayreuth ((**?**)), a cross-browser, programmable in Javascript, and interactive.

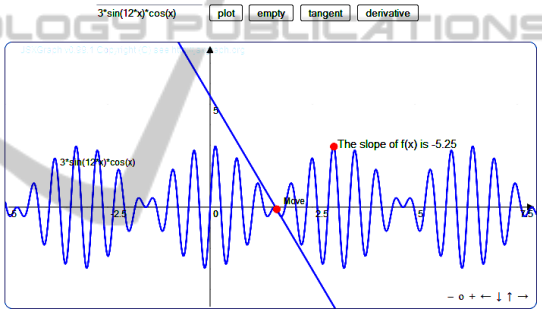

Figure 1: Plot generated by a script.

The essential for such plots (and this applies also for a server-based example) is the facility to compile dynamically a string, say, **`"2*sin(x)+z"`** into an internal expression which can be evaluated. The processors for interactive languages as Javascript, or Python, include the compiler, which translates the program source into its internal structures, and the virtual machine which executes it. The user may call the compiler from inside his program, and generate some code; in such a way a HTML page may transmit not only static data, but also dynamical expressions and statements into a script, or send them to a distant server, to compile them. The Web pages become true, "live" *applications*.

The development tools for Javascript are hardly adapted to beginners (students and teachers). Can we teach it as one of first programming languages, in order to encourage the Web programming afterwards? Marijn Haverbeke, the author of "Eloquent JavaScript" ((Haverbeke, 2014)) thinks that yes, that it became a kind of modern "Basic". People willing

to write and test their programs may use the browser-based development tools, or some stand-alone interpreters, such asRhino ((Mozilla, 2015)), the Mozilla machine, or the Google V8 Javascript processor. Editors, or IDE packages are also available, our preferred is WebStorm ((Jet Brains, 2015)) of JetBrains, who offer it freely to students and educators.

2.4 IPython Notebook Server, and the Jupyter Project

The Tornado example in the section above is an instance of a home-brewed *application server*, a separate application which performs some client-driven actions, and communicates with the user interface. In such a way we can integrate the pedagogical content: descriptions, analyses, static illustrations, etc., with on-line experiments, and *ad hoc* generated visuals, thanks to the Python interpreter behind. But our example is weak, we could parametrize the output style, or the plotted function, but some substantial modifications would require again the direct access to the plotting program.

But since programming languages, such as Python, are adaptable to many user interfaces, why not use a browser as a graphic console for the *complete* computing system? In such a way the teacher of a programming language has a simultaneous access to the static layer of the course and to the full computing power of the language processor. We began to teach programming in Python in such a way. The Tornado server launched from a terminal as `ipython notebook` ((Pérez et al., 2015)) sends pages permitting to write and execute *all* Python programs, and display all results on the browser surface.

The notebooks can be stored and shared, sent by pupils to teachers as transcripts of their exercises, and their potential just begins to be exploited. An offspring of this project, Jupyter ((IPython Project, 2015)), installable as a multi-user server (e.g. in a school computer pool), has the ambitions to include other programming languages, e.g., specialized languages adapted to children.

3 INSTALLING ONE'S OWN WEB SERVER

User-installable servers are more popular than usually thought. It is even possible to make a simple server using the shell (command line), a server is nothing more than a program which "listens" to a *socket*, and which can send some messages, usually textual. All the rest is the protocol, the syntactic construction of those messages, and this can be coded by everybody who spent some hours on the documentation. The details are numerous: message parsing, parallel threads, database access, etc., so a full-fledged server is voluminous, and this gave rise to the myth that servers are beyond the reach of a non-professional computer user. Some reasons to install a server, in the context of this talk, are the following:

- The display of composite, dynamical documents on a browser, usually demands an active actor which sends data to the client. Scripts, applets, etc. usually cannot accede to local file systems because of security restrictions. But they can use the resources of the server. Thus, splitting the communication framework into the client / server domains, gives all the power to the user, and does not violate the established protocols.
- It is then considerably easier to test such documents locally, before deploying them to a distant server, if such is the ultimate goal of the creator. In particular, a local HTTP/PHP/SQL server is necessary in order to install a local Moodle framework in order to test what is, and what is not possible to install on it[2].
- As shown, we can install not just a static file, but a genuine *applicative* server, which performs complicated actions on served documents. The server, disposing usually of more universal resources than the client, can deal with the internationalization of the document: switching of languages, accessing some dictionaries, etc.
- A server can start any program on its site, which is forbidden for a browser. In such a way, the presentation interface acquires the general power of your computer.

One doesn't need the huge LAMP / WAMP / XAMP packages, often recommended for individual platforms. Apache alone, or the LightTPD server of Jan Kneschke ((Kneschke, 2015)) are very light, and their memory footprint is small. There are more solutions which deserve a look. We teach Java Enterprise Edition (since the "market" demands it from our graduates), and here the multi-layer computing, server-client architectures, Web services, etc. constitute the skeleton of the domain. Java-based solutions are usually popular in the enterprise world, since Java protocols enforce the protection of resources better than many other solutions, but such light servers as Tomcat

[2]Some of our colleagues teachers didn't know that they could install Moodle on their laptops, and they structured their documents on a distant server, which was painful.

or Jetty, are trivially easy to install on personal platforms, and they offer directly the possibility to exploit the servlets in order to include some dynamics : simulations, animations, database access, etc., to pedagogical pages, with presentations and/or exercices.

For teaching, this was not our first choice, because Java is not an interactive language, but the Java Virtual Machine is nowadays installed *everywhere* on standard computers, and its variants (Dalvik) are installed on Android devices. Since by default the JVM includes the Javascript processor, users who prefer to program in a dynamically typed language, can do it. The document structure may ressemble our Tornado experience: the HTML page addresses the server, which switches the control to *your* Java program (or to a JSP page, which contains executable fragments inside HTML), this one calls your specific pedagogical library, e.g., a zoological or linguistic database, and returns the information to the browser, perhaps in graphical form.

3.1 Node.js

Node.js, ((Joyent, 2015)), is a world-wide known Javascript programming framework "on the server side" (independently of any browser, or other client; it may be installed locally, of course). Its kernel is based on the Google V8 engine, and it is just a simple terminal-command based application, which executes a sequence of scripts. We used it to process some pedagogical statistical physics simulations (such as the animation of the Ising model or the percolation), and the speed was sufficient. We could discuss the problem, while the visualization was generated dynamically on-line, without disturbing the presentation.

There exist already hundreds of different, specialized servers based on Node. Several modules are included in the standard distribution, or may be installed in a few seconds. A minimalistic script needs only to load some libraries, to execute, say, `var server = http.createServer(proc_req_resp); server.listen(8080);`, and to declare (before) a function `proc_req_resp(request,response)`, a procedure which gets automatically the request string from the user, if the browser addresses `localhost:8080`. This procedure constructs the server's answer, which is switched back to the user. It may also launch another server, or stop itself, according to the request. The server may act as a go-between the browser (the HTML pages), and all the resources of the platform.

The reader of this paper may have the impression that we speak about the development of Web applications, whose relations to teaching are distant and indirect. This is almost true, only that the dynamical, communicating systems, which begin to **dominate the professional world**, should change our way of teaching as well, and the progress here is not as fast as it should.

4 WEBSOCKETS

This is an advanced subject, which can be barely scratched here, but it really changed the face of *distributed computing*, enabling it for non-professionals, even for children. Most users are acquainted with the classical, HTTP interaction between servers and clients (classical synchronous, or AJAX). The client issues a HTTP request, the server wakes-up, answers, and passivates. This model may be not sufficient, if we want to organize a full duplex, persistent connection, such as chat, which may be quite useful in a computing room, to ask questions by the students, and broadcast the teacher's answers. Also, a fast, persistent communication channel will be useful when our server is supposed to "push" periodically some (unsolicited explicitly)data to the client, for some kind of animation, or teacher ***forced*** messages, warnings, and hints to a group of students working on some assignment.

The administration of client/server channels demand some special work on several computers, which in a pedagogical or not, multi-user environment might be heavy. But then, if the communication channels are already open, it is possible to link together a student's computer and some didactic equipment (data acquisition devices, robots, drones, Arduino assembly, or another computer), and to work as if both software layers were parts of the *same* application. We speak about linking computing sites through ***stable, persistent, full-duplex communication channels***, as if they were parts of the same system.

The professional (industry, scientific, military, etc.) world uses it frequently, and it is the time that the educational community profit from it more intensely. The Websocket constructs are integrated into the HTML5 specification. All serious browsers and server frameworks are compatible with them, and use a specific protocol: `ws://xxxx.zzz`. Moreover, for several programming languages, there are dozens of different Websocket libraries, with slightly different functionalities ((Kaazing, 2015; **?**; **?**)), etc., to cite the most relevant.

Everything is coded in very few lines of code. When a server begins to listen to port 8080, if a script on some remote client executes, say, `sock = new WebSocket("ws://localhost:8080");`,

the server awakes, and a symmetric connection is established. Then, executing `sock.send(someMessage)` once, or many times, one node transmits the information, and maintains a dialogue. The execution of `on_message(funct)` on any side declares that if a partner sends something to this socket (this is this potentially unsolicited information transfer) the appropriate function is executed, as if the other side had called it.

This is a simplified presentation, but with a few lines of code, we can write a simple chat system, and this has been done by adequately instructed high-school pupils. We tested a similar system, and this was quite fast and transparent. The applications of websockets are universal, far beyond such exercises, but they show how the technology interacts with the social behaviour.

5 FINAL REMARKS

Although teaching is a social process, its organization is important, and teaching tools *must* be shared and tested, we think that the basis of this sharing is the *individual creativity*, and we need more effort to enhance the teachers' capacity to use modern computing tools to make and to structure, not only to organize their courses.

In order to exploit efficiently such tools as dynamic, interactive documents, multi-tier client-server architectures, and efficient communication contraptions plugged into personal learning environments, some small competence in "Web programming methodology" is needed, and should be focused upon while forming teachers. This is not easy, but became essential. We should be also aware that the coding standards of HTML5 are already changing, that the next student generation will need new tools.

Such voices as: "I am a teacher of language / biology / music, etc., not a programmer, I cannot afford to learn programming languages", will always be heard, they are rational. However, the teachers should be aware that young people are fascinated by gaming, chats, visual experiments, etc., and they should not be discouraged by a negative atmosphere around programming. Everybody does some programming, even if it is only GPS, kitchen equipment, or telephone configuration.

Our examples strongly suggest that the skeletons of communicating software are short, and their sense is easy to grasp. Writing servers, chat rooms, drone steering software (with adequate firmware libraries) is a question of hours, rather than months.

We couldn't treat many interesting subjects, such as the installation and usage of small, local databases, or local search engines, facilitating the selective rehearsal of the course material. We skipped the "Web based" tools for the creation of video tutorials, which belong to another niche of the construction of pedagogical documents, which ***should*** be taught. It was not possible to discuss the internationalisation techniques (scripted language switching, translations, localised software documentation, etc.)

All these tools exist, and if needed, there may be easily found. Not depreciating the social and organisational aspects of teaching platforms, we believe that more effort should be invested to facilitate the individual work of teachers, to give them the adequate, modern information transfer tools.

REFERENCES

Baranovskiy, D. (2015). URL: http://snapsvg.io/.

Bayreuth, University of (2015). *JSXGraph, Dynamic Mathematics with JavaScript*. URL: http://jsxgraph.uni-bayreuth.de/wp/.

Continuum (2015). http://www.continuum.io/cshop/anaconda/.

Davidson, A. and Waddington, D. (2010). E-learning in the university: When will it really happen?

Dougiamas, M. and Taylor, P. (2003). Moodle: Using learning communities to create an open source course management system. In *Proceedings of the EDMEDIA 2003 Conference*, Honolulu, Hawaii.

Eidenberger, H. (2003). Smil and svg in teaching. In *SPIE Electronic Imaging Symposium*. SPIE. ISBN: 0819448214. URL: http://www.ims.tuwien.ac.at.

Geogebra, I. I. (2015). *GeoGebra 5.0*. http://www.geogebra.org.

Haverbeke, M. (2014). *Eloquent Javascript, a Modern Introduction to Programming*. No Starch Press.

Hohenwarter, M. (2002). GeoGebra: Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene. Master's thesis, Paris Lodron University, Salzburg, Austria.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

IPython Project (2015). *The Jupyter Project*. URL: http://jupyter.org/.

Jet Brains (2015). *Webstorm*. URL: https://www.jetbrains.com/webstorm/.

Joyent (2015). *Node.js on the road*. URL: http://nodejs.org/.

Kaazing (2015). *Websocket.org*. URL: https://www.websocket.org/index.html.

Khronos (2015). URL: https://www.khronos.org/webgl/.

Kneschke, J. (2015). URL: http://www.lighttpd.net/.

MathJax, C. (2015). URL: http://www.mathjax.org/.

Moodle (2015). URL: https://moodle.org/.

Mozilla (2015). *Rhino*. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino.

Nater, M. (2015). URL: https://code.google.com/p/hyphenator/.

PBS (2015). URL: http://pbskids.org/.

Pérez, F. et al. (2015). *The IPython Notebook*. URL: http://ipython.org/notebook.html.

Stangvik, Einar Otto (2015). *ws*. URL: http://einaros.github.io/ws/.

Three.js (2015). URL: http://threejs.org/.

Tornado (2015). URL: http://www.tornadoweb.org/en/stable/.

W3C (2015). The Web Sockets API. URL: http://www.w3.org/TR/2009/WD-websockets-20091222/.

WebAudio (2015). URL: http://webaudio.github.io/web-audio-api/.

X3DOM (2015). URL: http://www.x3dom.org/.