Developing curiosity and multimedia skills with programming experiments

J. Henno*, H. Jaakkola** and J. Mäkelä***
* Tallinn University of Technology, Estonia
** Tampere University of Technology, Pori
*** University of Lapland, Rovaniemi, Finland
jaak@cc.ttu.ee

Abstract rowsers have become the most common communication channel. e spend hours using them to get news and communicate with friends, far more time than communicating face-to face. -based communication and content-creation for www will be the most common ob in future wor life for students speciali ing in software engineering.

** e e pect our screens to be colorful and animated, thus students should understand technologies, which are used for e.g. for painting umping ario to screen. ut massive flow of new software engineering ideas, technologies and framewor s which appear in all-increasing temp tend to ma e students passive receivers of descriptions of new menus and commands without giving them any possibility to investigate and understand, what is behind these menus and commands, illing their natural curiosity. There should be time to e periment, compare formats, technologies and investigate their relations. n the presentation are described e periments used for investigating, how different formats for describing animation in T document influence animation rendering speed.**

I. Introduction

Curiosity, active interest, desire to know more about something is a precondition for brain for learning [1]. Curiosity makes us consider, ponder different aspects of new and this is needed to understand it. But constant, all the time accelerating flow of new information, new tools and technologies what we are supposed to introduce to our students is effectively making the whole teaching process a mechanical dumping of new facts, procedures, technologies without leaving students time for investigation, discovering the real nature of new technologies or tools.

Especially rapid is the flow of updates, new technologies, new tools in the field of Software Engineering (SE). This creates constant pressure for SE students, who are in danger to drown in constant flow of new menus and commands without time to investigate, what is behind these new menus. And half of this what they learn now becomes in two-three years already obsolete [2].

More important than just teaching menus and commands of new frameworks is developing analytical, inquiring attitudes, ability to seek to understand how new technologies work, what is in them really new and what – just a marketing noise, how different technologies are intervened, support each other (or not) and why they so quickly replace old ones.

Especially intense is this problem in SE new fields: mobile, web and cloud programming. Here a programmer is working with minimal resources for understanding inner workings, debugging, but at the same the code should not only work, but should be memory-efficient and work quickly. Even small delays in page loading does seriously reduce website's user experience and can mean turning customers away [3].

Current SE students should understand the technical side of this media, this will essential part of their future work.

II. Digital communication

Communication, the base of the advancement of mankind, is rapidly moving online. Online communication is visual, colorful and animated, e.g. Microsoft replaced in Windows 8 and Windows 10 the old start menu with start screen with animated live tiles. Games, game news and game discussion are a permanent section in all social media channels, but also in many old 'solid' news channels.

The major mechanism for presenting common digital content on screen is browser. Only very specialized programs (Photoshop, Excel) or big games still use their own windowing systems but even those have already many browser-based analogues. Browser is the first program what we usually open when we start computer and browsers are already considered as the basis of a whole Operating System (OS) [4].

A browser-based interactive media is essentially an frame-based video. Processor calculates next frame content and accompanying sounds, from these bits is then rendered text/images on screen and played sounds/music. For best results developers want maximal color depth (bpp – Bits Per Pixel, determines color quality) and framerate (fps – Frames Per Second), which determines animation smoothness and game responsiveness. But both these features increase processor load, memory use and decrease speed, thus may make animation jumpy or sound disrupted.

Development of browser content is based on use of the HTML language. With HTML are strongly tied several other languages – JavaScript, CSS (current version – CSS3), SVG (Scalable Vector Graphics, a graphics vector language), Flash, WebGL (for 3D graphics). Due to history and current situation with of development of these technologies their interplay is not transparent, it is often difficult to select one of several possible development strategies.

III. Browsers, HTML5, CSS3

Human society and culture is based on communication. For long time all communication was based on natural languages. But in the second half of the last century appeared programming languages as means to communicate with 'tools for intelligence' – computers and other devices. With development of techno- and info-sphere communication is more and more moving into Internet and the widely used method is browser-based communication using the HTML language, which allows to use all traditional communication modes – text, images, sound, video, but has added a new feature – it is interactive, messages receiver can actively participate in forming received content.

HTML was not designed as a programming language – the original 18 tags permitted only most-basic text layout options for distributing research reports of scientists working in the Nuclear Research Center in Cern. But it contained one important new tag - the hyperlink; this was the revolutionary concept that created the current-day Internet. And when Netscape invented new programming language JavaScript, which transformed until then passive www-pages into dynamic, interactive media, the web development and use exploded. JavaScript is used by 94.4% of all the websites [5] and is the programming language with highest ranking [6].

Modern web browsers have become a complicated multichannel translators. They accept input from several channels – HTML/XML/SVG text with links to other sources, images, video, sound tags, keyboard and mouse or/and touch input; input may be provided from several files – the HTML-document itself and data from linked external sources, e.g. external CSS (Cascading Style Sheets) and JavaScript files, images, sound/video files (see Figure 1). With these different types of inputs browsers compile complex output for computer/mobile screen, create voice and play sound and video.

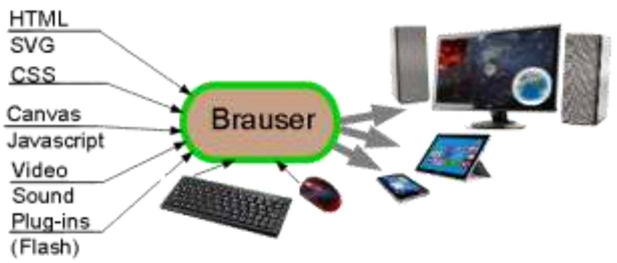

Figure 1. Inputs-outputs of common web browser

Thus in principle browser works like a programming language's translator – it transforms information presented in format/syntax of input channels into format/syntax required by output channels/devices. But its task is essentially more complex – the information presented in inputs is interlinked, depends on each other, e.g. 3D-placement on screen of a paragraph of text depends on size/placement of text/pictures before this paragraph (the CSS box model) and all CSS-formatting rules, which could be introduced in several different places - the 'cascading' feature of CSS.

New features are added to browsers and its input languages in frantic temp. All major browsers – Google Chrome, Microsoft Internet Explorer (IE), Mozilla's Firefox (FF), Opera - publish several major updates per year, minor updates appear in every 6-8 weeks [7] and most of them are already 'evergreen' - they automatically update themselves. In recent years have appeared also several new browsers: Microsoft Edge (for Windows 10 only) [8], Vivaldi [9], Brave [10], Yandex browser [11], Maxton cloud browser [12].

With such a frantic temp of development it is difficult to maintain overall balance and consonance of all parts of HTML-documents - DOM (Domain Object Model [13],all elements of a HTML document) tree, JavaScript, CSS, add-on's) and new features, which are developed by different parties.

Quite illustrative is development of Cascading Style Sheets (CSS) language. This was simple and natural format to describe styling of web pages, when it was introduced 20 years ago. But innovators found, that "CSS is primitive and incomplete" [14] if we could not have borders with rounded corners, thus should be improved and so appeared frameworks - layers of code re-organizing use of CSS features. First was introduced an extension of CSS - Sass (Syntactically Awesome Style Sheets [15]), where are introduced variables, nested rules, inline imports and other features and which has two syntax options. However, to use it you should first know CSS quite well and also have the programming language Ruby compiler installed in your computer, since the extension is a Ruby program. Then Sass was extended with simple scripting language SassScript. But for some people Sass still is not good enough so we get open-source CSS authoring framework Compass [16]. Now writing style rules - something what could be done with any text editor, Notepad or (for highly technical persons) Notepad++ and which few would call programming requires two interdependent frameworks, compiler for the Ruby language and several Ruby Gems to output CSS. And to see the result this should be uploaded to server.

CSS frameworks may be useful for professional developers of big web sites, but even there their use is questionable [17]. For students they are an overkill.

CSS3 introduced several methods for animating elements of HTML document. For instance, CSS property `transform` manipulates the size, shape, and position of a CSS box and its contents through rotating, skewing, scaling and translating. CSS3 animations allows animation of most HTML elements without using JavaScript or Flash [18], but this is a rather restricted type of animation – CSS3 does not allow variables, thus everything should be fixed before, animation cannot be changed on run-time. CSS3 is introducing also several other very advanced features, e.g. media queries, which allow to apply CSS rules depending on properties of device (e.g. screen width) which is used to present the content. Most of these features already work in many browsers; however, currently (Jan 2017) CSS3 is not yet implemented in

Microsoft browsers IE 11 and Edge, these browsers 'know' only CSS2.

IV. Too much is confusing

Animation, movement is the major way to make WWW documents more attractive, thus most of developers want to use this feature. For a long time the only technologies allowing to show movement in WWW were video and Flash. Video is a non-interactive media, thus the main technology for developing interactive, dynamic web content, games and portals was Flash. When Steve Jobs declared in 2010 Flash a 'persona-non-grata' on Apple devices, its popularity decreased also on other platforms. But because of its (very) good performance it is still widely used, so rumors about its death seem to be strongly over-accelerated [19]. Microsoft even included Flash player in its new browser Microsoft Edge - and dropped its own inventions, the ActiveX components and Silverlight. Thus ActiveX and Silverlight become another dying-out stars in quickly changing landscape of web technologies.

The major technology platform for developing interactive web content - games, communication and business portals is currently latest version of HTML – the HTML5. HTML5 introduced a new element – canvas – an area in HTML-document where JavaScript commands can draw and thus create animations.

HTML5 allows to implement animations using several formats and technologies:

- showing animated .gif images either as a part of HTML-document (i.e. with HTML-code) or drawing with JavaScript on canvas; this old image format contains series of frames for storing short animations, is restricted to 8 bpp (bits per pixel) color resolution (i.e. image can have max 256 colors) and animation speed cannot be controlled by browser, it has to be pre-set when creating the .gif animation file, but animation (image) is easy to scale (make smaller, making it bigger destroys quality);

- animation on HTML5 canvas with JavaScript: showing/moving images, possibly clipping them to some figure;

- procedural texture – merging texture images changing their opacity [32]; here this was used to produce Sun's lava texture from only one image;

- CSS3 allows to create frame-based spritesheet animation without using JavaScript [20];

- SVG; SVG elements can be manipulated like HTML elements using transform functions, but many commands and attributes do not work the same way on SVG elements as they do on HTML elements, JavaScript feature detection fails, the local coordinate system of an element works differently for HTML elements and SVG elements, the CSS properties of SVG elements have different names, e.g. instead of *ac ground color* should be used *fill* etc.

When HTML5, CSS and JavaScript arrived, several browsers, especially Microsoft Internet Explorer (IE) browser versions 6..10 did not follow standards, thus web content developers had insert into their code special checks for browser version, e.g. in HTML:

```
<!--[if IE 7 ]>
```

In order to unify development and eliminate browsers incompatibilities was in 2006 introduced a cross-browser JavaScript library `jQuery`, which provided a consistent interface that works across different browsers, i.e. also in Microsoft's browsers. With time `jQuery` has included many properties, e.g. fade ins and fade outs (a 'visual sugar') and animations by manipulating CSS properties. Since JavaScript libraries are a non-transparent layer of code (it is very difficult to check what went wrong is an error occurs in a used library – they act like a compiled module), libraries should be introduced only after students have acquired solid JavaScript skills. But since `jQuery` is currently very popular, we considered also animation created with `jQuery`.

V. What to use ?

Browser input consists of several parts – the HTML-document (HTML code), CSS stylesheet, JavaScript file(s) (there may be several), image, sound, video files. These are handled by different browser's sub-programs – the browser's layout engine with CSS interpreter and the JavaScript engine. All major browsers use their own, independently developed engines:

TABLE I. BROWSERS AND THEIR LAYOUT AND JAVASCRIPT ENGINES

rowser	**ayout ngine**	**ava cript interpreter**
Firefox	Gecko	SpiderMonkey
Chrome	Blink (developed from WebKit)	V8
Internet Explorer	Trident	Chakra
Microsoft Edge	EdgeHTML	Chakra
Opera	WebKit	V8
Safari	WebKit	JavaScriptCore

They all interpret HTML+CSS+JavaJcript code a bit differently and the 'inner working' even of major browser engines are mystery, explanations cover only one particular browser [21], [22], [23] and are presented in rather general terms. There is not yet 'browser theory', which were comparable to IT subject 'Translator theory' [24], developed for classical programming languages. This creates many questions. Performance of some components - JavaScript engines, CSS layout calculation, screen renders etc. may be rather different [25]. How this influences the overall performance, in which order are applied CSS rules, if determining DOM tree element attributes is better from HTML-text or from JavaScript, levelling browser's build-in defaults (e.g. different default margins) – these practical issues are unexplained.

For instance, when developing a Christmas game "Santa in Wild Forest" we wanted to introduce a light effect – torch/moonlight moving together with Santa, see Figure 1. It turned out, that the effect can be implemented using different technologies/formats, but they all had some problems (image on mobile screen look a bit dirty) and some solutions worked in different browsers differently.

Figure 2. A sceenshot from mobile game "Santa in Wild Forest", developed as a part of the course, captured from mobile phone screen in original mobile screen resolution – 768x1220px; - Santa Clause has to collect all parcels, which Krampus throw into dark forest; we investigated possibilities to add torch/moonlight effect, i.e. create a half-transparent circle covering Santa and moving together with Santa

Thus instead of following intense flow of marketing shouts: "Use `JQuery`!" [26], "Use `Angular`!" [27], "Use `TypeScript`!" [28], "Use Facebook's `React Native`!" [29], "Use `Intel SDK`!" [30] we decided first to test with a practical application the real value, first of all – speed – of different animation technologies. Implementing test applications and performing tests give students much better understanding of value of different technologies then just implementing something in one (usually rather randomly selected) format.

VI. THE TEST APPLICATIONS

For comparing different animation formats we implemented a scheme of Lunar Eclipse (actually happened during the course on 20.03.2015), which contained several animated and/or half-opaque elements, see Figure 3.

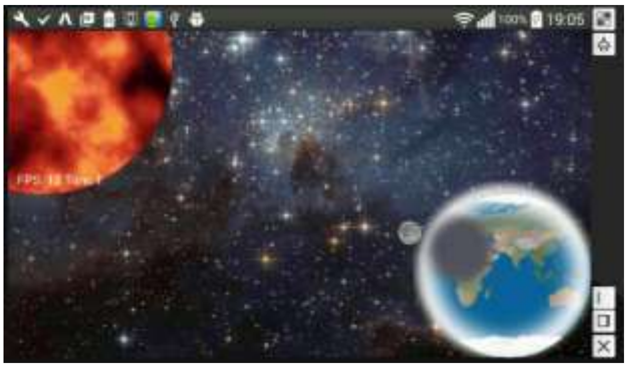

Figure 3. The test application (captured from mobile browser) and its animated objects:Sun (upper left corner with animated lava texture), Earth (lower right corner, rotating), Moon (small grey circle close to earth), Moon shadow (larger half-transparent grey circle on Earth surface), Earth atmosfere (light half-transparent halo surrounding Earth).

To investigate influence of different animation formats and opacity change technologies on animation speed and memory requirements we prepared tests T1..T9 [31], which all are versions of this animation.

TABLE II. TEST ANIMATIONS

Test files	Animation description
T1: eclipse1.htm	The whole screen is covered with canvas with cosmos as the CSS-defined background image; Earth is animated with JavaScript (texture is constantly moved behind a clipping circle, drawn on canvas by JavaScript); all other objects are images in HTML document placed using CSS attribute z-order over the canvas: Sun is animated .gif image (32 frames) with transparent background, Moon, Moon shadow, Earth atmosphere – images, transparency is 'built-in' to images with Photoshop (is not adjusted in the HTML-document); placement of images is defined with CSS using position attribute value 'fixed'.
T2: eclipse2.htm	The whole screen is covered with canvas with cosmos as the CSS-defined background image; Earth is animated with JavaScript (texture is constantly moved left-to-right behind a clipping circle, drawn by JavaScript using the 2D-graphics context of canvas); all other objects are images in HTML document placed using CSS attribute z-order over the canvas: Sun is animated .gif image (32 frames) with transparent background, transparency of Moon shadow and Earth atmosphere images is defined with CSS rules
T3: eclipse3.htm	The whole screen is a DIV with cosmos as the CSS-defined background image; Sun is animated .gif image, minimal canvas is used only behind Earth, which is animated with JavaScript (texture is constantly moved behind a clipping circle drawn by JavaScript using the 2D-graphics context of canvas); all other objects are images in HTML document placed using CSS attribute z-order over the canvas, 50% transparency of Moon shadow and Earth atmosphere images is defined in Photoshop
T4: eclipse4.htm	As previous, but 50% transparency of Moon shadow and Earth atmosphere images is defined with CSS rules
T5: eclipse5.htm	The whole screen is a series of DIV-s (no canvas); the main DIV with cosmos as the CSS-defined background image covers the whole screen, smaller DIV-s for Earth, Moon, Moon shadow and Earth atmosphere images are placed over it using the CSS position, width/height and z-order attributes; Sun is animated with CSS3 rules (the 32 frames of the sprite sheet were obtained from the animated .gif image), Earth is also animated with CSS (texture is constantly moved behind a CSS clipping circle); transparency of Moon shadow and Earth atmosphere images is defined in Photoshop
T6: eclipse6.htm	The whole screen is a series of DIV-s (no canvas); the main DIV with cosmos as the CSS-defined background image covers the whole screen, smaller DIV-s for Earth, Moon, Moon shadow and Earth atmosphere images are placed over it using the CSS position, width/height and z-order attributes; Sun is video in WebM format, the video is clipped by CSS clipping circle (works correctly only in Firefox; Chrome has its own proprietary format for alpha transparency in WebM-video); Earth is animated with CSS (texture is constantly moved behind a CSS clipping circle); transparency of Moon shadow and Earth atmosphere images is created in Photoshop
T7: eclipse7.htm	Sun texture is procedurally generated by JavaScript and jQuery on minimal canvas using two additional canvases (the idea from [32]); all other elements are as in previous example
T8: eclipse8.htm	As in previous but jQuery library (253 kb) was removed and replaced with JavaScript
T9: eclipse9.htm	Texture of Sun is procedurally generated (without jQuery), Earth is JavaScript animation on a separate canvas, thus together there are 4 canvases

VII. Results

All test animations *eclipse .htm..eclipse .htm* were HTML5-documents looking similar on screen, but since implemented using different technologies/formats they also produced different results: time needed for constant number of animations (10 rotations of Earth) and RAM memory used by browser. They all had a small JavaScript script, which measured the time was used to run 10 rotations of the Earth; the results were shown on screen in a separate small DIV and stored using the HTML5 local storage feature. For memory performance there is not yet general standards; in Chrome is available a proprietary method `performance.memory` [33], but this can be used only if Chrome is started with specific switch and in our tests Chrome reported always the same values. IE allows to see some memory statistics with the UI Responsiveness tool [34]:

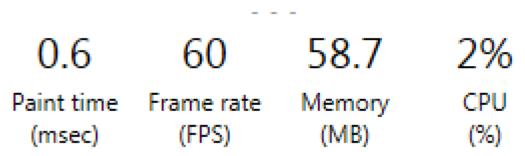

Figure 4. The window of the Microsoft UI-responisveness tool

The values produced by this tool varied 4-10% and therefore not presented here; the only more or less constant change was 3-4% increase in used memory when the `jQuery` library was used (the test *eclipse .htm*).

Thus the main measured parameters were time-based – the framerate FPS (Frames Per Second) and total time needed to execute a constant-size animation (10 rotations of Earth). A JavaScript script measured several parameters of test (FPS, time for one rotation, number of frames rendered) and at the end of animation showed the main result – total time for the animation (10 rotations of Earth) in milliseconds on screen (it was also stored using the HTML5 feature `localstorage`). In order to eliminate computer speed and network latency, all tests were done locally under a local Wamp server [35]. These tests produced lot of numbers; in order to gain better understanding of these were results normalized, using the *eclipse .htm* from Firefox as the control case, i.e. in the following table the first number is *time*(Ti) - total time for this test Ti with this browser and the second – percentage of this time of the 'etalon' time, i.e. calculated with formula *time*(Ti)*100/*time*(T1), where *time*(T1) is the time reported for test T1 by Firefox.

TABLE III. Results of tests

Test	rowser				
	FF 51	***Chrome 55***	***IE 11***	***Edge***	***Opera***
T1	66773ms 100%	68281ms 102%	60008ms 98%	65097ms 98%	65046ms .97%
T2	66744ms 99%	68230ms 102%	59996ms 98%	66007ms 98%	65070ms 97%
T3	66755ms 99%	67874ms 101%	60036ms 99%	66764ms 99%	65065ms 97%
T4	66720ms 99%	68016ms 102%	60032ms 99%	67065ms 100%	65065ms 98%
T5	20011ms 29%	20004ms 29%	CSS clipping with circle does not work	CSS clipping with circle does not work	19974ms 29%
T6	20023ms 29%	19837ms 29%	IE 11 does not play WebM video, CSS clipping does not work	-	20010ms 29% no video clipping
T7	20010ms 29%	19999ms 29%	CSS clipping does not work	-	20006ms 29%
T8	20010ms 29%	19992ms 29%	CSS clipping does not work	-	20004ms 29%
T9	66721ms 99%	67070ms 99%	59626ms 98%	60101ms 99%	64057ms 97%

We performed similar tests also with a mobile phone browsers in an android mobile (LG E975a, Android 4.4.2 'KitKat'); here were used the phone OS built-in browser, Chrome and UC cloud browser (made in China), which currently is a 'rising star' in the landscape of mobile browsers [36]. In the following table are presented the raw results (test times in milliseconds) and results after normalizing using Chrome as the etalon.

TABLE IV. Results of tests in mobile browsers

Test	**rowser**		
	Chrome	***LG OS-browser***	***UC browser***
T1	66972ms 100%	80060ms 119%	78215ms 117%
T2	66882ms 100%	83453ms 125%	98413ms 145%
T3	66973ms 100%	85029ms 124%	69619ms 127%
T4	67308ms 100%	82938ms 124%	85169ms 127%
T5	19894ms 30%	20060ms 30% – no clipping	19109ms 29% no clipping
T6	19835ms 30%	No WebM	19930ms 30% no clipping
T7	19838ms 30%	-	19129ms 29% no clipping
T8	19993ms 30%	-	19835ms 30% no clipping
T9	68086ms 100%	-	71027ms 106%

Although mobile browsers show more differences both between browsers and between tests, the general tendencies were rather similar.

VIII. Conclusions from Tests

These results allowed students to draw several conclusions:

- there are no essential differences in speed between major browsers, but Microsoft browsers do not (yet) implement CSS3 (only CSS2)

– HTML5 canvas+JavaScript allows to create complicated animations, but reduces animation speed ca three times (tests T5,T6,T7 did not use canvas). This result was for students surprising, since canvas is commonly considered the main element in all graphics-intense web applications (games, portals etc.) but it becomes understandable if one thinks what actually is loaded as the canvas 2D context – this is an interpreter 2D graphics commands, which builds its own name table etc.

- using several canvases does not make application slower (test T9);

- changing opacity of bitmaps with JavaScript does not make application slower and allows better to control result (tests T2, T4)

- CSS3 animations and CSS3 clipping (with circle) are quick, but quite difficult to scale (changing size of frame-based CSS animation is very error prone) and did not work in Microsoft browsers

- video in WebM format with transparent background (test T6) can be achieved (using CSS3 clipping) only in Firefox (Chrome has for this a proprietary extension [37]);

- results of tests T7, T8, T9 indicate, that `jQuery` was officious – big, especially for mobile applications (current version 3.1.1 – 261 kB) and did not have any advantages. The `jQuery` library was introduced to make Microsoft browsers IE6..IE10 to understand standards, but currently Microsoft has also started to follow them, so `jQuery` is (mostly) not needed. But `jQuery` introduces rather difficult to understand cryptic syntax (what has to be learned) and is and changing custom semantics, e.g. cryptic *Query* command to get canvas object:-

```
var $canvas = $('#canvas');
```

returnes array (`#canvas` suggests, that there are several canvas objects having the same id?!); equivalent to this, but more understandable plain JavaScript command

```
var $canvas =
document.getElementById('canvas');
```

returnes 'flat' variable, thus all uses of these variables also require different syntax. Use of `jQuery` also increased memory requirements (as measured in IE 11). It was relatively easy to remove all dependencies of `jQuery` - we actually had to change only 8 lines to convert the script into 'clean' JavaScript, where `jQuery` was not used.

Students discovered even more, e.g. the speed of canvas animation depends essentially on size of animated objects – scaling page down decreased rendering time.

IX. Conclusions from the Project

The main benefit of the project was not discovery of dubious properties of use of canvas or other technical results – this may change with the next updates of browsers. The main benefit of the project for students was in making them think and explore, showing exploratory attitude for use of software technologies instead of following blindly the next marketing hype and learning dumbly commands of some commercial framework (e.g. Intel XDK - 643 MB, 35800 files, development for Android phones requires also Android SDK – 27.2 GB, 154999 files). Browsers are the most important communication channel of future and students should understand well the browser technology and for this they should experiment and investigate.

REFERENCES

[1] Scientific American. Curiosity Prepares the Brain for Better Learning. https://www.scientificamerican.com/article/curiosity-prepares-the-brain-for-better-learning/

[2] How fast is our world becoming obsolete? https://www.ericsson.com/thinkingahead/the-networked-society-blog/2014/01/30/how-fast-is-our-world-becoming-obsolete/

[3] A.B. King. Website Optimization: Speed, Search Engine & Conversion Rate Secrets. O'Reilly Media; 2008, 398 pp

[4] Browser Based Operating Systems Reviewed. http://www.tech-tweak.com/browser-based-operating-systems/

[5] Usage of JavaScript for websites. https://w3techs.com/technologies/details/cp-javascript/all/all

[6] The RedMonk Programming Language Rankings: June 2015. http://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/

[7] https://en.wikipedia.org/wiki/Timeline_of_web_browsers

[8] Microsoft Edge. https://www.microsoft.com/en-us/windows/microsoft-edge

[9] Vivaldi. https://vivaldi.com/

[10] Brave. https://brave.com/

[11] Yandex browser. https://browser.yandex.com/desktop/main/

[12] Maxton. http://www.maxthon.com/

[13] JavaScript HTML DOM. https://www.w3schools.com/js/js_htmldom.asp

[14] An Introduction to CSS Pre-Processors: SASS, LESS and Stylus. https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus

[15] Sass – CSS with superpowers. http://sass-lang.com/

[16] Compass. http://compass-style.org/

[17] You might not need a CSS framework. https://hacks.mozilla.org/2016/04/you-might-not-need-a-css-framework/

[18] CSS3 animations. http://www.w3schools.com/css/css3_animations.asp

[19] Usage of Flash for websites. https://w3techs.com/technologies/details/cp-flash/all/all

[20] Using CSS animations. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

[21] WebCore Rendering I – The Basics. https://webkit.org/blog/114/webcore-rendering-i-the-basics/

[22] How Browsers Work: Behind the scenes of modern web browsers. https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

[23] High Performance Animations. https://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

[24] Alfred V. Aho, Jeffrey D. Ullman Theory of Parsing, Translation and Compiling. Prentice-Hall 1972, 542 pp, ISBN-13: 978-0139145568

[25] A Guide to JavaScript Engines for Idiots. http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/

[26] jQuery. https://jquery.com/

[27] Angular. https://angularjs.org/

[28] TypeScript. https://www.typescriptlang.org/

[29] ReactNative. https://facebook.github.io/react-native/

[30] Intel XDK. https://facebook.github.io/react-native/

[31] Eclipse. http://deepthought.ttu.ee/users/jaak/slideshow/

[32] Experiment - HTML5 Canvas Nebula. http://www.professorcloud.com/mainsite/canvas-nebula.htm

[33] Static Memory Javascript with Object Pools. https://www.html5rocks.com/en/tutorials/speed/static-mem-pools/

[34] Improving UI responsiveness. https://msdn.microsoft.com/en-us/library/dn255009(v=vs.85).aspx

[35] Wampserver. http://www.wampserver.com/en/

[36] Browser Market Share Worldwide. January 2016 to Jan 2017. http://gs.statcounter.com/

[37] Alpha transparency in Chrome video. https://developers.google.com/web/updates/2013/07/Alpha-transparency-in-Chrome-video