The Impact of Cloud Based IDEs on Programming High Level Languages for Educational Purposes on Mobile Devices

[1] Chikwiriro Hilton, [2]Chaka Pharaoh, and[3]Mavhemwa Prudence M
Computer Science Department
Bindura University of Science Education, Zimbabwe
Bindura, Zimbabwe

***Abstract*— An Integrated Development Environment (IDE) is a platform which provides tools which enable software to development on a web browser using mobile devices such as smart phones and tablets which have an internet connection instead of just the usual traditional desktop. The web provides a generic user interface and can also allow for real time collaboration among different users from different locations. This paper discusses the impact that is brought about by the use of cloud base…….d .IDEs in educational setting as a direct result of the improved student participation and collaboration, and also practice which could be due to the cloud IDEs highly availability.**

***Keywords*— Integrated Development Environment (IDE); Cloud server; cloud based IDE; Real time collaboration; language editor**

I. INTRODUCTION

Traditional programming in education relies on programming environments deployed on lab machines, which can be a tedious process at odds with the speed of developments in these tools. While software development environments on the Web may be an appealing vision, it is far from being simple since moving IDEs to the Web is not just a matter of porting desktop IDEs, a fundamental reconsideration of the IDE architecture is necessary in order to realize the full potential that the combination of modern IDEs and the Web can offer [10]. Issues of network latency will also come into play since the code editor and the compiler running on the server will need to be maintaining asynchronous communication.

II. DEVELOPING WEB APPLICATIONS

There are some fundamental differences when designing and implementing an application which will run on the web compared to classical desktop applications. In typical web applications the actual work is done remotely on a web-server or the cloud where the user is presented with a user interface built in HTML. Through the use of GET/POST requests or AJAX communication is handled from the client to the server. This communication layer with the back-end is arguably where most differences between desktop and web applications lie because of its inherent asynchronous nature. At the server side a programmer has virtually unlimited options in which he implements the web application back end. However at the client side the web application has to be presented in a web browser. Currently this means the implementation is bound to only use flavours of (X)HTML, CSS and JavaScript. Even though many Web applications have been created by software developers, there currently are few web applications which provide the necessary tools to actually create applications with. The small amount of tools which do exist, such as CoRED [23] and Cloud9 1, are fundamentally limited in the sense that they only support a select set of languages. Even though Cloud9 supports language plug-ins, these plug-ins still have to be implemented specifically for that platform (in JavaScript) and are mainly implemented using regular expressions which make sophisticated editor feedback impossible.

Over the last decade, the dizzying expansion of the online universe and the growing sophistication of web browsers have turned the Internet into the greatest repository of information in history. At the same time, the increased availability of mobile devices has brought the resources of the Internet to many more people throughout the world. [24] ran a cover story describing mobile devices as the next killer app for the developing world, citing a growing body of evidence that the mobile devices are the technology with the greatest impact on development. Thus to make this transition of devices, from PCs to mobile devices meaningful, it means finding a way to make applications and hence IDEs portable and platform independent, therefore in this report we seek to show that making the IDEs web based is the way to go.

About five decades ago, the first IDE was introduced, targeting the BASIC language [25]. The IDE was purely command-based, and therefore did not look much like the menu-driven, graphical IDEs prevalent today. Still, it integrated source code editing, compilation, debugging, and execution in a manner consistent with a modern IDE. Over the past five decades, desktop IDEs have become mature and are now prevalent in modern software engineering. Even though many IDEs frameworks currently exist such as Eclipse[3], Netbeans and VisualStudio, most IDE implementations are mainly targeted at a small fixed set of programming languages. They provide tools for working with

a wide range of languages, combined with facilities for version management, issue management, and so on.

III. SOFTWARE DEVELOPMENT IN CONTEXT OF DESKTOP IDES

Software development and maintenance is a highly collaborative effort. The crucial role of efficient and precise communication between developers, developers and testers, and developers and end-users is well-known. It is also an accepted truth that developers tend to follow the path of least resistance. If the tools at their disposal make collaboration difficult, collaboration will happen less, or not at all. Despite the many accomplishments and innovations of desktop IDEs, they still operate within the constraints of the desktop paradigm: individual developers work on separate machines, requiring the installation, configuration, and maintenance of separate IDE instances for each developer. None of the major IDEs provide realtime collaborative features to mitigate this problem—even though technology for doing so exists [26].

In the light weight code editing widgets we find code editors without syntactic or semantic services[27], or with just minimal regular expression based syntax highlighting [14]. These tools can be useful for coding small programs, and in the form of widgets they provide ample opportunity for mashups. As an example, WeScheme [18] is an educational programming environment, embedding CodeMirror[14] for syntax highlighting and bracket matching. However, while these widgets can be useful tools for coding small programs, they do not provide a comprehensive environment with all the facilities that are especially important for productivity in larger projects. They also do not offer any support for collaboration.

In the general case, the web browser cannot act as a runtime for the program under development. The web IDE must be connected to some form of runtime provider where the developer can execute the program under development as part of the edit-compile-run cycle. That is, the runtime provider becomes a Web service employed by the web IDE [28].

Can a web IDE support offline mode one would ask. Most web applications simply do not support offline mode, and for some developers, especially Web developers, that is perhaps acceptable also for web IDEs[29]. Even for other types of developers, one must consider how much of their working time is spent offline? What will that figure be in five years? Given developers' reliance on documentation, search engines and collaboration, can one really be productive offline anymore? Even if offline mode is ultimately necessary, one could argue that offline support should be limited to a subset of the full capabilities, starting with only what is necessary for the edit-compile-run cycle [29].

For pragmatical and economical reasons, the web IDE must integrate well with the significant amount of high-quality developer tools already in use[5], such as continuous integration or continuous deployment, issue trackers, version control, static analysis tools. Many of these are on the Web, and already provide a web service API; they are online services designed for integration with other online services. The other tools must become services by acquiring a web service API. Interoperability requires the web IDE to provide a plugin architecture. The plugins must be able to call out to external web services, and to provide the necessary user-interface elements for these services. While the desktop IDE is often a collection of plugins running in the same process, the web IDE is a collection of distributed services connected through web services APIs.

There is a cause for concern when we talk of productivity and web based IDEs. On this issue that is when the desktop counter parts seem to gain headway over the web based IDEs. Literature says web based IDEs might be beneficial where productivity is concerned. Experience from other online services indicates that online services can shield the users, in this case the developer, from the configuration specifics in the runtime environment. The developer can spend less time on installation and configuration of the tools, and more time on development. The daily maintenance which involve, upgrades, backups, redundancy and scaling is handled by dedicated personnel, and the costs are amortized across all users. [28] argue that web based IDEs might be harmful to productivity, by saying any network or cloud provider outages, or takedowns due to legal disputes, will impact the developer severely since there will likely be no backup or offline alternative. The other important factor is that of lack of control of the platform [26] which makes it difficult, or even impossible, to work around bugs and regressions and the developer might not be able to control when and how upgrades to the IDE should happen.

IV. ONLINE COLLABORATION

The Web was conceived as tool for collaboration, and most of the services and techniques developed for the Web are there to facilitate collaboration. Here we consider the potential impact of these services and techniques in the context of IDEs. When all developers are online, how does team collaboration change? A number of Web 2.0 applications, such as Google Docs and Wave, have shown that collaboration changes when the participants interact in real-time, on the same document. These applications emphasize synchronous collaboration combined with versioning. They use the connectiveness of the cloud combined with novel synchronization algorithms such as Fraser's differential synchronization algorithm [30]. Using a realtime connection between clients, every change to a model is reflected from the client to all other active developers working on the same model. By contrast, current desktop IDEs tend to use asynchronous collaboration, where each developer works in their own instance taken from a canonical master copy. Eventually they merge their changes into a new master copy.

Collaboration and version management is an area with a wide range of variability. The connectivity and the centrality of configuration of the cloud makes it an excellent platform to investigate different models. Fully synchronous collaboration is highly effective for editing documents and can facilitate pair programming, but it may not scale to software development projects with more programmers editing and debugging at the same time. One direction for new approaches to online collaboration is to use the

language-specific facilities of the online IDE. With many developers working at the same time, one scenario that should be avoided is synchronous collaboration of invalid or incomplete source code. With a language-aware IDE, source code can be checked for syntactic and semantic correctness, and even tested, before merging [31]. Speculative merging and checking of source code could be the basis of new hybrid models between fully synchronous and asynchronous collaboration. Other online services relevant for online collaboration include any communication channels incorporated into the IDE, in particular issue trackers. Current issue trackers tend to be loosely integrated into the development process. With a fully integrated environment, issue reports could include a versioned snapshot of issues encountered by other developers, or a representation of the runtime state or issues reported by users.

V. DISCOVERY AND RECOMMENDATION

Understanding the source code of a software project is key to efficient software development. Developers navigate the code and documentation to discover its functions, and to learn and follow the architecture and design patterns established for a project. Experience with recommendation engines show that they can be effective tools for helping users navigate many types of content, including source code [20]. As source code is increasingly being placed online under various open licenses, the collective corpus at our disposal for automated mining and indexing is increasing rapidly.

Zeller predicts that discovery and recommendation systems will eventually offer "[...] automated assistance in all development decisions for programmers and managers alike: "For this task, you should collaborate with Joe, because it will likely require risky work on the Mailbox class."[22]. Data extracted from mining software repositories can be used for a number of purposes, including API usage recommendation [31],for example what are the typical protocols that clients of an API use, bug prevention, which are based on historical bugs, which parts of the source code is more likely to have new bugs [32], structural code search, for example show me calls to wait and notify that are not protected by a synchronized block[33] ; automated bug detection, by using static analysis tools such as FindBugs[12].

How exactly will moving to the web change discovery and recommendation. In the Web-based development environment, all the source code is by necessity online. It is collected in centralized repositories, and is increasingly available under open licenses. This simplifies indexing and mining substantially. Measuring the accuracy of recommendation engines is dependent on data from the developers' workspace. User tracking is a basic building block for most modern web applications. Privacy concerns notwithstanding, it is relatively trivial to instrument the web IDEs to track the activity of developers, and thus quickly collect the necessary data needed to tune degrees-of-interest models and thus improve the recommendations. Web-based issue trackers have provided service APIs for some time. This makes it relatively easy to mine and index bug history. Such mining may be used to continuously tune tools for automated bug detection to weed out false positives.

Then looking at another area of concern one would ask the question, then, what happens to the plugin model in a Web-based world? The plugin model is the big enabler of integrated development environments on the desktop. While there are many distributed component models [34], the typical component models used in IDEs are designed to operate only in a single process. In even the simplest of web IDEs, some plugins must execute on the server, and some in the web browser, thus requiring a distributed component model. This change has rippling effects such as every API call might be a remote call, and must be dealt with on a case-by-case basis. Actual remote calls must be handled using asynchronous programming techniques. The synchronous plugin model might still work well for logic that will only execute in the browser, or only inside a single process on the server. For everything else, it is customary to think of it as a Web service – a chunk of functionality provided by a remote machine.

Consequently, the web IDE will require a solid, distributed service model. This model must ensure interoperability across processes, across servers, across cloud providers, across implementation languages, and across geographical locations and timezones. It must support API versioning and system upgrades, so that new versions of components can be provided to a large audience. The design of the service model of the web IDE will set the stage for how open, extensible and centralized a given web IDE is. A restrictive model is likely to promote walled gardens where a flexible model might dissuade the formation of the same gardens.

The open ecosystem of the desktop world has served us very well by fostering innovation and allowing competition on all levels. An important reason why the Internet won out over the thousands of alternative networks of the past, is that openness was architected into the Internet from the beginning [35]. On a technical level, the software development environment should be capable of supporting multiple languages through some form of plugin architecture, and the plugin architecture should be designed so that the end-users, the developers, are in control of the IDE they use. We want the web IDE to be a mashup where users can add in new components, also third party ones, as they see fit. If we only consider the client-side, this is a problem with a number of known solution patterns [36]. Components requiring a server-side component present additional challenges. Examples of these include the semantic components such as type checking and code navigation, which both require access to the entire source code of the program; language-specific execution environments that require a more powerful runtime than what is offered by the JavaScript VM in the browser, and that must be able to run arbitrary code provided by the end-user; and, platform specific deployment systems that might need to run native executables in order to communicate with remote services.

Where should the server-side code run? How does a web IDE provider, deal with the security issues related to running third-party code on their servers? How does he track and bill the users for the resources consumed by third-party components? The web IDE provider might offer a server-side

sandbox, for example in the style of Google App Engine. Third party components must be written to be compatible with this sandbox, which could then be designed more like a traditional desktop plugin-model, albeit with stricter resource control and potentially by separating each plugin into its own process space [12].

Another possibility is to require every component-provider to host the server-side part of their plugin, and expose it using an agreed-upon web service API. This API might then be forwarded to the client so that the server side processing is offloaded to a third party cloud, potentially a different one for every third party component. This presents challenges related to latency, authentication and security, harmonization of service-level agreements across components, and design of the interoperability protocols required between components. A third, hybrid model, is to allow both, and also to allow developers to host third-party plugins on their own hardware, and register these as services with their web IDE provider. A fourth variant is the fully peer-to-peer system with no central authority. This presents significant challenges with regards to trust. Either the computational nodes must perform obfuscated algorithms on obfuscated data, or the users must be able to trust each other to keep each other's data safe. The model that becomes prevalent in the end will have an impact on the openness of the web IDE concept in general, and a number of social questions, such as: Who gets to decide which components and therefore which languages should be allowed? Even if you provide free-of-charge service for your new language, how are you going to get others to use your new language if they're all on a walled up web IDE? Is the service model fundamentally inclined towards censorship, thus easily disallowing languages of the competition?

What are the implications for innovation of programming languages, IDEs, and language workbenches? Given that these technical issues are resolved, the remaining social or commercial aspects are largely a matter of policy, some web IDE providers will be open to integration with third party plugins, other will not. Experience from other walled gardens, such as mobile app stores, suggests that we might end up with a spectrum of openness among web IDE providers. Research and innovation is likely to thrive on the providers that are placed more toward the open end of the spectrum. In a market where users demand services, providing the software behind the service is suddenly not only feasible, but a now established way of gaining credibility and popularity with the user base. Just as was the case with Eclipse, this is likely to benefit the research community, as the basic infrastructure will be freely available to build on[34]. What if something breaks at the web IDE service provider? This is more of a pragmatic issue. Upgrades and system changes often result in regressions.

By hosting the IDE on the Web, the developer gives up a control over when and how potentially devastating upgrades should happen. While it is often possible to track down and come up with workarounds to upgrade problems on your local machine, or to roll back, doing the same for a web service is often impossible. Outages and regressions are usually covered legally by service level agreements, but experience shows that even the biggest and most reliable service providers with the strictest SLAs can go down for days—and may have faulty backups [34]. By hosting the web IDE on multiple, different cloud providers, in different versions, or by designing the IDE around a decentralized peer-to-peer architecture, it might be possible to mitigate this problem somewhat, since an old (presumably working) version is then likely to be around, and available.

VI. RESEARCH METHODOLOGY

A. Research designs, data collection and approaches

The research was implemented at Bindura University of Science Education in the Computer Science Department. The research provided a platform for the learning and practising of programming languages in particular Java, HTML, C++, C and C Sharp, by making use of the web based integrated development environment which we implemented in the quest to achieve our objectives.

B. System design

The Web based IDE is a tool for developing programs on the web, these are IDE which can be accessed through a browser, and there is also need for an internet connection. For the system to be able to support many different languages, we had to integrate the CodeRun studio which supports C#.net, as well as HTML, CSS, Javascript, e.t.c and the eXo cloud IDE that supports Java programming.

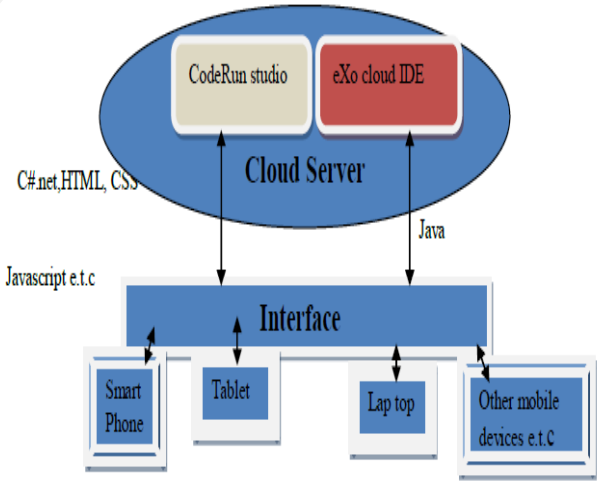

Fig : 1 The Cloud based IDE

The users would write their code and the interface would then direct the code to the right IDE on the cloud server depending on the language that would have been used. The basic features that we have are user registration and login.

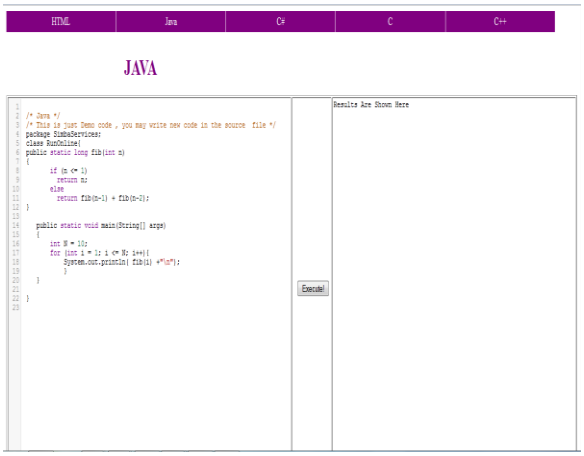

Figure 2: The language editor

To do programming on our website, a user had to log in, and would be taken to the home page. In the home page user had to choose their preferred programming language of choice from a menu bar on top of the web page. The menu bar is in form of links which lead either to the java editor, C# editor, C++ editor or C editor and the HTML editor. Since this editor could run HTML thus by default it runs all the languages that run in a browser that is Cascading Style Sheets and javascript. Our html editor uses event listeners thus it displays results in the results console as the user is editing their code, more like auto compilation and running.

The platform also had other features like saving, deletion, editing, compilation and execution of programs. Once the execute button was hit the code was submitted to the server for processing that is the compiling and running of the code then the result is send back to the results console.

The platform also catered for real time collaboration which is whereby, multiple different users are allowed to edit the same program from different mobile devices if given the permission by the creator of the program. To cater for this facility, there was need to store a backup copies for the user who created the program. Only a single user would be allowed to edit a program at a given time, but his/her editing would be visible to all other users. The platform also allowed users to have private chat and a public discussion forum.

C. Research instruments

The participants of this study were computer science students at the Bindura University of Science Education, Computer Science Department who were doing programming courses Introduction to Programming and also Data Structures and Algorithms. The sample size used was 40 students. Questionnaires, observations and mock tests were the instruments used in this research project. The collected data was analysed using SPSS. The web based IDE was deployed on the Bindura University of Science Education e-learning website where it was accessible by students via the internet. The research provided a platform for students to practice their programming on their smart phones, tablets and other small mobile devices.

ANOVA

Score

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	1221.025	1	1221.025	5.333	.026
Within Groups	8700.950	38	228.972		
Total	9921.975	39			

VII. RESULTS AND ANALYSIS

A. Analysis of data and interpretations of results

The questionnaire contained only closed questions. A total of four closed questions, in the form of positive and negative statements to agree or disagree with, were asked. Each closed question used a five point Likert response scale where each scale point was defined and the results obtained were as follows:

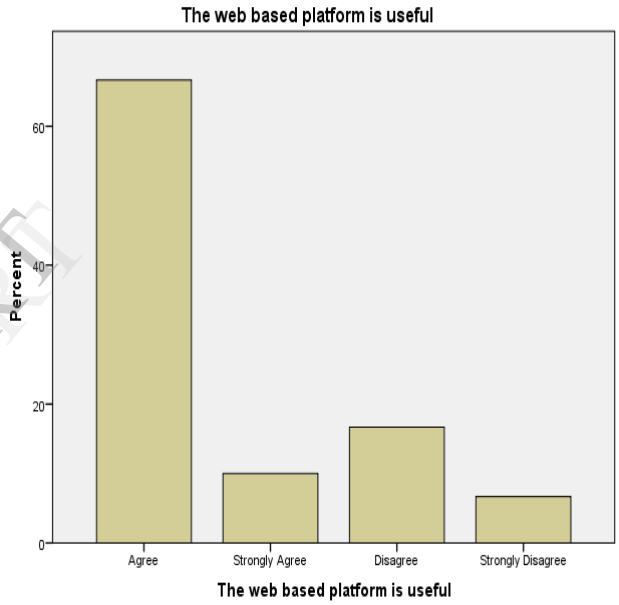

Fig : 3 The Is the cloud based platform useful?

All More than 70% of the students found the web IDE was useful as a platform for coding, compiling, debugging and executing programs although it had less functionality than the desktop IDE. 30% of the students said it was not useful; this might be attributed to the fact that the cloud based IDE did not show output results for graphical objects. 85% of the students found the system to be more convenient than the usual desktop IDE. The fact that everywhere where smart phones could be used made it more convenient and highly available.77.5% of the students thought that the private chat and the public forums were very helpful in both situation where students may need to feel anonymous and also in situation where they would need collective help from other students. 95% of the student agreed that the real time collaboration feature was a very useful feature whilst 5% did not.

An analysis of the students scores from the mock tests was also performed.

Report

Score

Group	Mean	N	Std. Deviation
1	72.05	20	15.343
2	61.00	20	14.917
Total	66.52	40	15.950

Table : 1 shows comparisons of student scores using ANOVA and mean scores

❖ With the following Hypothesis stated i.e:

H_0: Use of the cloud based IDEs on mobile devices has no effect on the programming skills of students.

H_1: Use of the cloud based IDEs on mobile devices has an effect on the programming skills of students.

Since the P-value from the ANOVA is less than 0,051, I fail to accept H_0 and conclude that the use of cloud based IDEs on mobile devices has an effect on the programming skills of students**.** Also from the observations made by the instructor through checking the system records, students spent quite a lot of time coding on the platform and also the discussions and participation were very encouraging.

VIII. CONCLUSION AND RECOMMENDATIONS AND FUTURE WORK

The results indicated that the web based IDE can be used as a close substitute to the desktop IDE in programming high level languages for educational purposes. The system improved the student's practice and participation time because it could be used anywhere and in particular in places where other devices such as laptops and desktops cannot be used. Therefore this system can improve greatly the student's programming skills since these can be improved mainly through practice. Also through the real time collaboration and the private and public discussion forum, students could get the necessary help they required. This in turn made programming a lot less difficult.

There is need to include as much graphical output objects as much as possible for compatible devices but also being very careful to cater for the non compatible devices as these days smaller devices are getting more powerful by the day.

REFERENCES

[1] *J. Nielsen, "Designing Web Usability: The Practice of Simplicity", New Riders Publishing, 2000.*

[2] T. Berners-Lee, L. Masinter, M. McCahill, et al. Uniform resource locators (url). *CERN*, 1994.

[3] E. Gamma and K. Beck. *Contributing to Eclipse: Principles, Patterns, and Plugins*. Addison Wesley Longman Publishing Co., Inc., 2003.

[4] M. Goldman, G. Little, and R.C. Miller. Collabode: collaborative coding in the browser. In *Proceeding of the 4th international workshop on Cooperative and human aspects of software engineering*, pages 65–68. ACM, 2011.

[5] Z. Hemel and E. Visser. Mobl: the new language of the mobile web. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 23–24. ACM, 2011.

[6] T. OReilly. What is web 2.0. *Design patterns and business models for the next generation*

[7] *of software*, 30:2005, 2005.

[8] A. Russell. Comet: Low latency data for browsers. *alex. dojotoolkit. org*, 2006.

[9] Lucas, H.C., Jr. "Performance and Use of an Information System," Management Science, Volume 21, Number 8, April 1975, pp. 908-919.

[10] Lennart C. L. Kats, Richard G. Vogelij, Karl Trygve Kalleberg, and Eelco Visser. Software development environments on the web: A research agenda. In *Proceedings of the 11th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software (Onward 2012)*. ACM Press, 2012.

[11] Pearson, J., and Pearson, A. "An Exploratory Study into Determining the Relative Importance of Key Criteria in Web Usability: A Multi-criteria Approach," *Journal of Computer Information Systems*, (48:4), 2008, 115-127.

[12] N. Ayewah and W. Pugh. The google findbugs fixit. In P. Tonella and A. Orso, editors, Nineteenth Int. Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010, pages 241–252. ACM, 2010.

[13] Cloud9 IDE. http://www.cloud9ide.com/ .

[14] CodeMirror. http://codemirror.net/ , Apr. 2012.

[15] CodeStore Inc. Coderun. http://coderun.com, 2010

[16] Z. Hemel and E. Visser. Declaratively programming the mobile web with mobl. In K. Fisher and C. V. Lopes, editors, 2011 Int. conference on Object oriented programming systems languages and applications, OOPSLA 2011, pages 695–712. ACM, 2011.

[17] M. Labs. Mozilla labs: Skywriter, 2010.

[18] D. Yoo, E. Schanzer, S. Krishnamurthi, and K. Fisler. WeScheme: The browser is your programming environment. In Conference on Innovation and Technology in Computer Science Education, 2011

[19] Narcissus. http://mxr.mozilla.org/mozilla/source/js/narcissus/ , Apr. 2012.

[20] [19] M. Robillard, R.Walker, and T. Zimmermann. Recommendation systems for software engineering. IEEE Softw., 27(4):80–86, July 2010.

[21] The Eclipse Foundation. Voidspace – python in your browser with silverlight. http://www.voidspace.org.uk/ironpython/silverlight/index.shtml.

[22] A. Zeller. The future of programming environments: Integration,synergy, and assistance. In L. C. Briand and A. L. Wolf,editors, Int. Conference on Software Engineering, ISCE 2007,Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA, pages 316–325, 2007.

[23] J. Lautam\"aki , A. Nieminen , J. Koskinen , T. Aho , T. Mikkonen , M. Englund. CoRED: browser-based Collaborative Real-time Editor for Java web applications . Conference on Computer Supported Cooperative work pages 1307-1316 ACM 2012.

[24] [Economist05] *The Economist.* The real digital divide: Encouraging the spread of mobile phones is the most sensible and effective response to the digital divide. 10 March 2005. http://www.economist.com/node/3742817?story_id=3742817

[25] Kemeny, John G. & Kurtz, Thomas E. (1985). *Back to BASIC: The History, Corruption and Future of the Language*. Addison-Wesley Publishing Company, Inc. ISBN 0-201-13433-0.

[26] L.-T. Cheng, C. R. B. de Souza, S. Hupfer, J. Patterson, and S. Ross. Building collaboration into IDEs. Queue, 1(9):40–50, Dec. 2003

[27] http://www. processingjs.org

[28] Lennart C. L. Kats, Eelco Visser: The spoofax language workbench: rules for declarative specification of languages and IDEs. OOPSLA 2010: 444-463

[29] L.C.L. Kats (2011): Building Blocks for Language Workbenches , Delft December 13, 2011

[30] N. Fraser. Differential synchronization. In U. M. Borghoff and B. Chidlovskii, editors, 2009 Symposium on Document Engineering, Munich, Germany, September 16-18, 2009, pages 13–20. ACM, 2009

[31] Jungloid mining: helping to navigate the API jungle D Mandelin, L Xu, R Bodík, D Kimelman - ACM SIGPLAN Notices, 2005

[32] Graves, T. L., A. F. Karr, J. S. Marron, and H. Siy (2000). Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, vol. 26, number 7, pp. 653-661.

[33] E. Linstead*, S. Bajracharya*, T. Ngo*, P. Rigor, C. Lopes, P. Baldi. *Sourcerer: Mining and Searching Internet-Scale Software Repositories.* Data Mining and Knowledge Discovery. Volume 2, Number 18. April 2009.

[34] K.-K. Lau and Z. Wang. Software Component Models. IEEE Transactions on Software Engineering. 2007 October; 33/10: 709-724. eScholarID:1a5875 | DOI:10.1109/TSE.2007.70726

[35] L. Lessig. Code and Other Laws of Cyberspace. Basic Books, Inc., New York, NY, USA, 2000.

[36] L. Grammel and M.-A. Storey. The smart internet. chapterA survey of mashup development environments, pages 137– 151. Springer-Verlag, Berlin, Heidelberg, 2010.