Web Servers

Markus Richter (richter@informatik.uni-luebeck.de)

Abstract

Websites play an important part in the Internet, as they are used by millions of people every day world wide, thus this part of the Internet is called *World Wide Web* or, for short, *WWW* or *Web*. In fact many people mean the WWW when they are speaking of the Internet. The WWW is build upon a huge net of Web servers which forms the basic structure of the Web. They provide support for essential technologies that make Web sites comfortable, interactive and attractive. This article focuses on those technologies which are used by servers in order to make appealing and modern Web sites possible. Furthermore, attention will be paid to the research results concerning performance improvements of Web servers by applying the scheduling policy called *Shortest-Remaining-Processing-Time* (*SRPT*), *Least-Attained-Service* (*LAS*) and *Preempt-On-Wait* (*POW*).

1 Introduction

The Internet not only grows larger every day, it also becomes more important and present in nearly every aspect of modern life. Nowadays it is not simply used for information gain between a few exclusive participants as it was initially intended but instead it is a platform for the purpose of communication, business and entertainment for millions of people. Those services, accessed through Web sites, ranging from browser games to e-commerce and online banking, are the key to the success of the Internet.

The astonishing increase in numbers and functionality is made possible not least due to the use of Web servers. Inhomogeneous technologies like *PHP* or *Java Servlet* and *JavaServer Pages* used to implement dynamic behaviour combined with static behaviour and international standards like *HTTP* are being combined and hidden inside the Web servers and being presented as one solution.

So as Web servers are playing such an important role for the internet in general and for Web sites in particular, the questions arise how they can support the development of Web sites towards more attractive and modern Web sites and additionally assure high performance in the Web. To answer this questions we will analyse the functionality behind a Web server which is needed to fulfil those needs. Furthermore we will take a close look at some methods and solutions in order to increase the performance.

1.1 Definitions

First it is essential to define and explain the terminology being used in order to build up an initial understanding of Web servers and the essential technologies associated with them.

1.1.1 Web server

The term Web server is being used to describe hardware as well as software. In the first case the computer is meant on which a Web site is hosted, in the second case the program that runs on the computer. Being part of the *Server Client Architecture* the Web server basically takes *HTTP* request from clients and answers with *HTTP* responses.

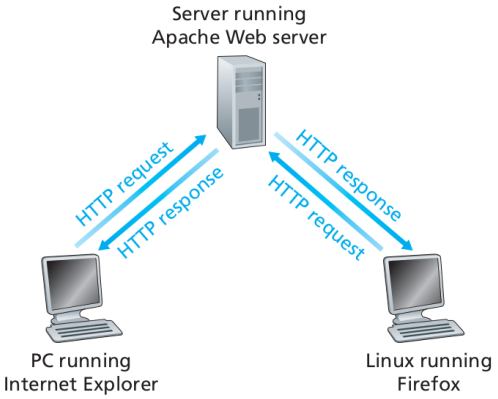


Figure 1: Client Server Architecture.
Taken from [Kurose and Ross, 2012]

1.1.2 HTTP

The *HyperText Transfer Protocol* (*HTTP*) is an *application-layer* protocol that controls the exchange between client and server. It is defined in [Berners-Lee et al., 1996] and [Fielding et al., 1999]. *HTTP* is divided in two parts: a client and a server program. Those two parts communicate with each other by using *HTTP* messages [Kurose and Ross, 2012].

1.1.3 Web site

A Web site consists of one or many Web pages, also called *documents*. A Web page contains objects which are files. Those files can be *HTML files, images, Java applets* etc. Each

object is addressed by an *URL — Uniform Resource Locator*. In most cases the Web site has a base *HTML* file which references many different objects [Kurose and Ross, 2012]. We speak of an *attractive* and *modern* Web site if it is in line with expectations regarding *appearance, security, performance and interactivity*.

1.2 Concept

The concept of a Web server is based mainly on *HTTP*. As mentioned before *HTTP* consists of a client and a server program. The client program is in most cases a *Web browser* while the server program is the Web server itself. The communication between those two programs is reduced to the *HTTP* methods *GET, POST, HEAD, PUT,* and *DELETE*. In the majority of cases the GET method is being used, namely when the client requests a file. The request is then answered by the server with a response containing meta data and the payload itself. This concept is simple but completely sufficient and has proved successful over the years. *HTML* is a descriptive language used for visualisation only and is static, hence, contains no logic and therefore is not a programming language.
In order to enable *dynamic* Web sites the Web server can generate — by using *server-side technologies* — those *HTML* files each time an interaction appears, e.g. if the current date needs to appear at the Web site. So the client constantly gets changed static *HTML* files what simulates dynamic behaviour. From the view of the client it seems like the current *HTML* file is changing dynamically but in reality the server keeps responding with new adjusted *HTML* files.
Figure 2 illustrates that almost all responsibility for creating and manipulating content is capsuled inside the Web server, although there are technologies used which allow execution of code on the client side.

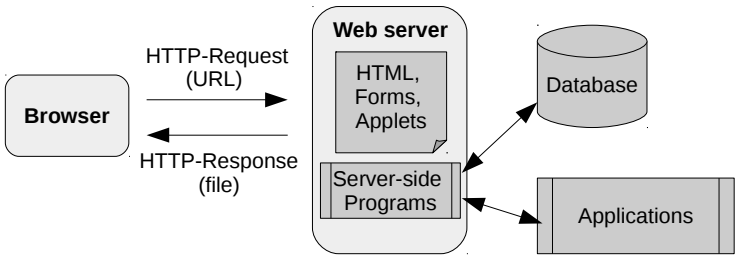


Figure 2: Concept of an HTTP Web server

1.3 History

The history of Web server is closely tied to the history of the Internet. But even though the internet kept growing rapidly since *ARPANET* was established in the nineteen sixties, its size and popularity was not even close to what it is today until the most important development for the Internet to date was accomplished: In 1991 *CERN* released the *World Wide Web* developed by Tim Berners-Lee [Gromov, 1995]. With it came the first *HTTP* Web server: The CERN httpd which worked with *HTTP* and *HTML*[Connolly, 1995]. Until now this principle has not changed significantly. In spite of relying on this principle which only provides static content in the form of *HTML* files there is still dynamic behaviour in the web. This advantage is only possible thanks to server-side technologies like *CGI*, *JSP* or *PHP*.

2 Server-side Technologies

Figure 2 already indicates a functionality not intended in former Web servers (server-side programs, database, applications). As the Web grew, the need for interactivity became more urgent. As a result a number of competing technologies evolved to satisfy those needs for attractive and modern Web sites. All technologies have in common that they are not integrated in the Web server core program but act as an external program or as module located at the same machine the Web server program is running on, hence, on the Web server's hardware. Although [W3Techs, 2013] reveals, that Java based technology *JSP* is used least often than *ASP.NET*, we will concentrate on *PHP* and *JSP* only, because they are not limited to proprietary platforms.

2.1 Common Gateway Interface

In [Coar, 1998] *CGI* is defined as "a simple interface for running external programs, software or gateways under an information server in a platform-independent manner". Figure 3 illustrates the functionality of *CGI*.[Gundavaram, 1996] describes it fairly well. Usually the client requests a *CGI* program the same way it requests any other Web document by using the *HTTP GET Method*. The difference is that the server — provided that it implements the *CGI specifications* — tries to execute the program. Listing 1 shows an example of how such a GET method can look like. All the information, like e.g. the data formats the client accepts, are made available to the *CGI* program via input by using the *standard input*, e.g. STDIN on UNIX systems and from UNIX environment variables. The result of the execution can either be a new document or an *URL* to an existing one. It is then passed on the Web server via *standard output* (STDOUT) as a data stream. Like the definition mentions, *CGI* is platform-independent which also means it is not limited to

certain programming languages, thus, the CGI program or script can be implemented in different ones.
A significant disadvantage of *CGI* is its low scalability and bad performance as it forces the operating system to fork an own process for each *HTTP* request, resulting in high overhead [Brittain and Darwin, 2007]. Because of this, modern Web servers provide their own, more performant, solutions for popular technologies, e.g. *mod_php* or *mod_perl* modules for *Apache* [Welling and Thomson, 2008], so *CGI* is barely used nowadays.

```
1 GET /cgi-bin/welcome.pl HTTP/1.0
2 Accept: www/source
3 Accept: text/html
4 Accept: image/gif
5 User-Agent: Lynx/2.4 libwww/2.14
6 From: shishir@bu.edu
```

Listing 1: Client request for CGI program. Taken from [Gundavaram, 1996]

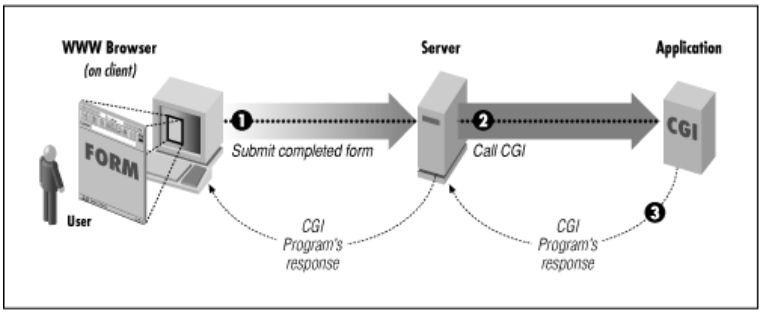


Figure 3: How CGI works. Taken from [Gundavaram, 1996]

2.2 PHP:Hypertext Preprocessor

PHP stands for *PHP:Hypertext Preprocessor*, thus it is a recursive acronym [PHP.net, 2013]. [PHP.net, 2013] defines it as "a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into *HTML*. Its syntax draws upon *C*, *Java*, and *Perl*, and is easy to learn." Because of the fact, that *PHP* is a scripting language it needs to be interpreted. The interpreter can either be a module or a separate *CGI* binary, although the module version is usually being used due to performance reasons [Welling and Thomson, 2008]. *PHP* is relatively easy and straight-forward because, as illustrated in Listing 2, it can be embedded inside *HTML* files and executed every time the *HTML* file is accessed, generating *HTML* or other output relevant for the visitor [Welling and Thomson, 2008]. This way dynamic behaviour is achieved.

According to [W3Techs, 2013], *PHP* is by far the most popular used technology in Web development. The reasons are, amongst others, high performance, scalability, object-oriented support, database integration and low costs. It is also often combined with *MySQL*, an open-source relational database management system. On the other hand, [Bowen et al., 2002] mentions some drawbacks, e.g. the code maintenance, that can be made difficult easily by potential lacking of structure due to mixing up of *HTML* and *PHP* code. Also *PHP* is not fully object-oriented, because some object-oriented capabilities like private variables, multiple inheritance are missing. Furthermore, it has some problems with stability and interdependences, because *PHP* usually relies on external libraries, e.g. for database connectivity, which may be on different stages of development. *PHP* is usually used together with *Linux*, *Apache Web server* and, as mentioned, *MySQL*, forming the well known *LAMP architecture*.

```
1 <?php
2 echo "<p>Order␣processed␣at␣";
3 echo date('H:i,␣jS␣F␣Y');
4 echo "</p>";
5 ?>
```

Listing 2: PHP embedded in HTML. Taken from [Welling and Thomson, 2008]

2.3 JavaServer Pages

JavaServer Pages (*JSP*) is similar to *PHP*, but instead it is easier to achieve more structure and it uses *Java* as programming language. Although *JSP* can be used independently it is meant to be viewed in a *MVC* design fashion [Downey, 2008]. Figure 4 illustrates such an architecture.

This means that *JavaBeans* encapsulates the data and additionally the methods which work on it. The *servlet* is the controller that gets the request and sends back the responses to the Web server. The *JSPs* are responsible for the view. Similarly to *PHP*, *JSPs* are *HTML* files with embedded Java code, but which usually are directives to display data from the model, not logic.

Listing 3 shows a simple case, where the *JSP* uses a *JavaBean* called refData to access the parameters from the *HTTP request*. Before that, the *servlet* (controller) has to add the bean to the session.

All this is wrapped inside a container. A container is needed for communication between the JSP technology and the Web server. Furthermore it handles the lifecycle management of servlets, multithreading support and security [Downey, 2008]. A container, which is very popular today, is for example, *Tomcat* that is based on the *Apache Web server*.

```
1 Hobby:
2 <input type="text" name="hobby" value="${refData.hobby}">
```

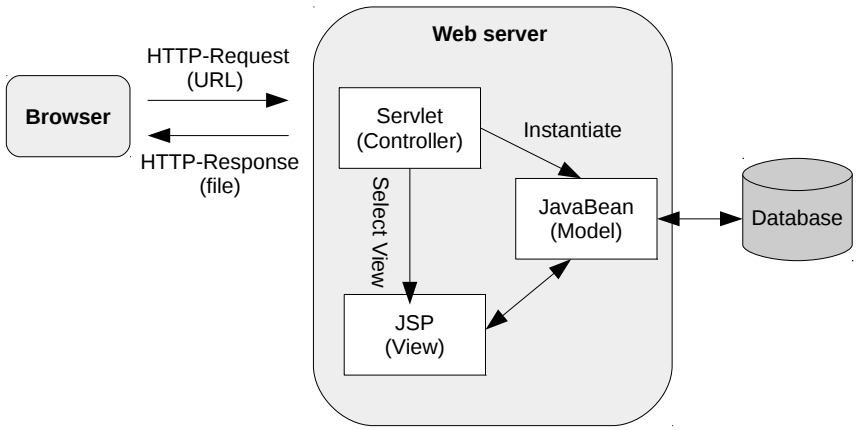


Figure 4: Model of JSP

```
3 <br>
4 Aversion:
5 <input type="text" name="aversion" value="${refData.aversion}">
```

Listing 3: Java embedded in HTML. Taken from [Downey, 2008]

3 Increasing Performance

The number of Web users grows strongly, but as expanding the Web infrastructure is expensive other solutions have to be found in order to maintain, or even increase the performance of the Web. One possible solution, that Web server can participate in, is size-based scheduling proposed by [Harchol-Balter et al., 2003] and [Biersack et al., 2007]. Their work shows that, by simply changing the order in which request being scheduled, the expected response time of every *HTTP request* can be reduced. Most traditional Web server use a *fair* scheduling policy, which means "that the Web server proportions its resources fairly among those requests ready to receive service" [Harchol-Balter et al., 2003]. So the new approach is now to use *unfair* scheduling: Priority is given for short request or those requests with short remaining file size. This approach goes along with the well-known scheduling algorithm *preemptive Shortest-Remaining-Processing-Time-first* (*SRPT*).
Figure 5 shows the results of an implementation using *SRPT* scheduling. An Apache Web server and a Linux operation system was used. In order to adjust the scheduling, implementation had to be done at kernel level. The left figure shows that the mean response time of the Web server using the *SRPT* approach is significantly lower compared to the

Web server working with the default fair scheduling policy. The performance benefits of *SRPT* even increases dramatically as the server load gets higher. The right figure shows for both approaches the response time as a function of the request size. It can be seen that for 99% of file sizes *SRPT* is performing much better and large file sizes belonging to the other 1% are only penalised minimally, which means both response times are either nearly identical or for the very largest files *SRPT* results in 5% larger response time.
Although this results are convincing there are still reservations against *SRPT*. Mostly, it is the fear that big jobs will "starve" [Biersack et al., 2007]. Thus, [Biersack et al., 2007] argues that web file sizes exhibit highly variable statistical distributions with heavy-tails and that for those highly variable distributions *SRPT* does not unfairly penalise long jobs. Additionally, [Biersack et al., 2007] show that *SRPT*-based scheduling is also able to minimise the number of connections at a server "by always working on the connection with the smallest amount of work left" [Biersack et al., 2007] and, hence, can help to face the problem of transient periods of overload.
SRPT therefore is a promising approach for *HTTP requests*, but that means it only can achieve a performance boost with static files, as *HTTP requests*, e.g. using the GET method, concerns physical files like images or *HTML* files. In order to increase performance of dynamic content processing we have to look at the common bottleneck in processing dynamic web requests which is the database backend [McWherter et al., 2004]. This makes it more complicated to apply the *SRPT* approach on dynamic content for two reasons. First, for *SRPT* it is essential to know the length of a transaction before execution. [Biersack et al., 2007] present a preemptive policy called *Least-Attained-Service* (*LAS*) for systems where size estimation is not possible, like for example in scheduling in routers. *LAS* guesses the remaining service time of a job based on the service it has received so far. This way, *LAS* converges towards *SRPT* behaviour.
Given that, the second problem is that existing database management systems do not support effective transaction prioritisation for web-based transacional workloads. This means they are often lock-bound and thus need lock scheduling [McWherter et al., 2004]. Basically, databases lock to protect shared resources like tables, data rows etc.
[Callison, 2009] provides more detailed information about locks. [McWherter et al., 2004] propose a policy called *Preempt-On-Wait* (*POW*). The algorithm preempts low-priority transactions in favour of high-priority transactions, but if and only if the low-priority transaction currently, or in the future, waits for some other lock. By doing this it is guaranteed that already partially completed work will not be lost. Figure 6 shows the response time of *POW* compared to *standard* — transactions are not prioritised — and two other related policies *PAbort* and *NPrioinher*. *PAbort* stands for Preemptive Abort. In that case, a low-priority transaction which blocks a high-priority one is always immediately preempted, what causes overhead caused by rolling back and restarting. *NPrionher* on the other hand grands low-priority transactions that block high-priority ones temporarily high priority to release locks more quickly. This causes worse high priority performance as low priority transactions need to be finished first. *POW*, therefore, is a compromise

of both. Given that, *POW* improves mean response time of high priority transactions by a factor of 8, while only increasing the mean response time of low priority ones by less than 10% [Biersack et al., 2007]. Combined with *SRPT* scheduling for static content, both, *POW* and *SRPT*, form a appealing solution for increasing performance of Web servers and thus of the Web.

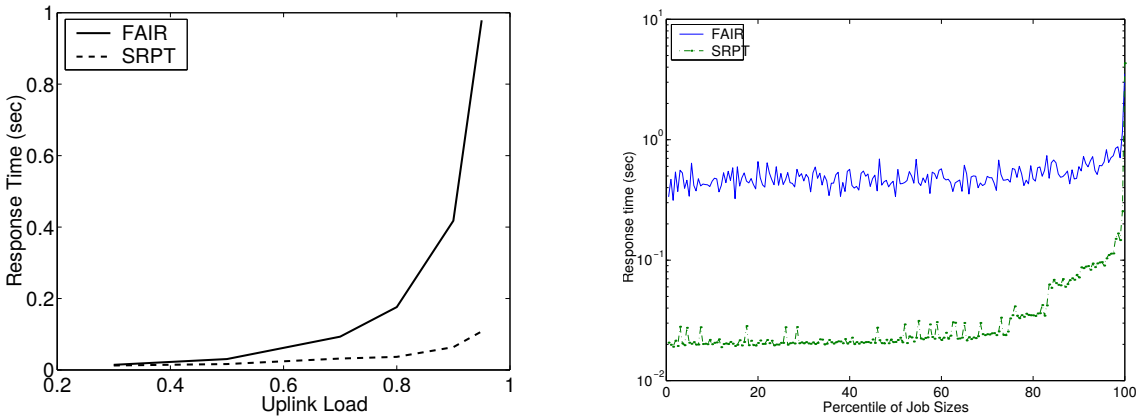
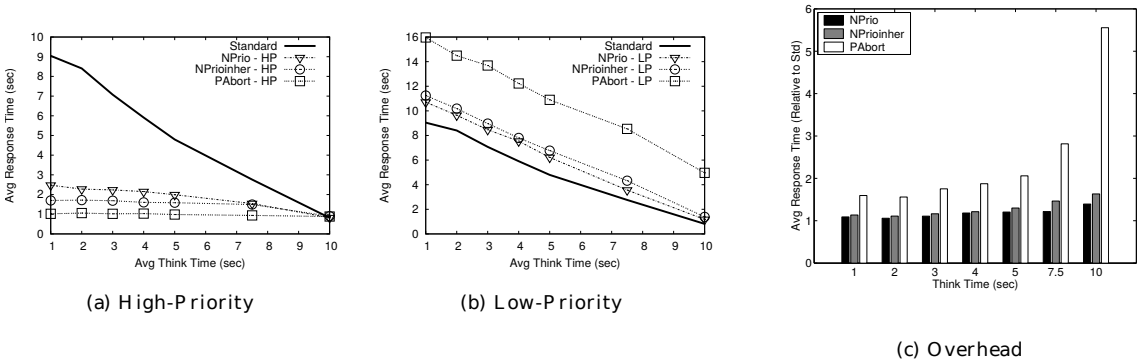


Figure 5: (Left) Shows the improvement in mean response time of *SRPT* scheduling over *FAIR* scheduling for static requests at a web server in a LAN setting. (Right) Shows mean response time as a function of the size of the requested file, where system load is fixed at $\rho = 0.8$. Taken from [Biersack et al., 2007]

4 Conclusions

In this article, we have defined what is essential for today's Web server and introduced some popular technologies. We started with a brief explanation of the basic principles of a Web server which have not changed much since 1991. We saw, that those principles were usually intended for static content and still, despite of this limitations, modern Web server provide dynamic content. We showed that this is due to innovative server-side technologies like *PHP* or *JSP*. Although today's Web servers are powerful, we also argued, that there is still space for improvements, especially when it comes to performance. Consequently, we looked at some promising solutions being in research and showed that they can help and how they can help.



(a) High-Priority (b) Low-Priority (c) Overhead

Figure 6: Average response time for high- and low-priority transactions for *POW*, *PAbort*, and *NPrioinher* as a function of load (a) and (b). Aggregate high- and low-priority average response time relative to *Standard* (c). Taken from [McWherter et al., 2004]

References

[Berners-Lee et al., 1996] Berners-Lee, T., Fielding, R., and Frystyk, H. (1996). Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational).

[Biersack et al., 2007] Biersack, E. W., Schroeder, B., and Urvoy-Keller, G. (2007). Scheduling in practice. *SIGMETRICS Perform. Eval. Rev.*, 34(4):21–28.

[Bowen et al., 2002] Bowen, R., Ridruejo, D., and Liska, A. (2002). *Apache Administrator's Handbook*. Developer's Library. Sams.

[Brittain and Darwin, 2007] Brittain, J. and Darwin, I. (2007). *Tomcat: The Definitive Guide*. O'Reilly Media.

[Callison, 2009] Callison, J. (2009). Database locking: What it is, why it matters and what to do about it. *Methods & Tools*, Spring 2009:49–64.

[Coar, 1998] Coar, K. A. L. (1998). The WWW Common Gateway Interface Version 1.2. `http://ken.coar.org/cgi/cgi-120-00a.html`. [Online; Date of last access: 21.11.2013].

[Connolly, 1995] Connolly, D. (1995). CERN httpd. `http://www.w3.org/Daemon`. [Online; Date of last access: 21.11.2013].

[Downey, 2008] Downey, T. (2008). *Web Development with Java: Using Hibernate, JSPs and Servlets*. Springer.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616

(Draft Standard). Updated by RFCs 2817, 5785, 6266, 6585.

[Gromov, 1995] Gromov, G. (1995). Roads and Crossroads of the Internet History. http://history-of-internet.com. [Online; Date of last access: 21.11.2013].

[Gundavaram, 1996] Gundavaram, S. (1996). *CGI Programming on the World Wide Web*. Nutshell Handbook. O'Reilly Vlg. GmbH & Company.

[Harchol-Balter et al., 2003] Harchol-Balter, M., Schroeder, B., Bansal, N., and Agrawal, M. (2003). Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233.

[Kurose and Ross, 2012] Kurose, J. and Ross, K. (2012). *Computer Networking: A Top-Down Approach*. Always learning. Pearson Education, Limited.

[McWherter et al., 2004] McWherter, D. T., Schroeder, B., Ailamaki, A., and Harchol-Balter, M. (2004). Priority mechanisms for oltp and transactional web applications. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 535–, Washington, DC, USA. IEEE Computer Society.

[PHP.net, 2013] PHP.net (2013). http://www.php.net/manual/en/preface.php. [Online; Date of last access: 21.11.2013].

[W3Techs, 2013] W3Techs (2013). W3Techs - World Wide Web Technology Surveys. http://w3techs.com. [Online; Date of last access: 21.11.2013].

[Welling and Thomson, 2008] Welling, L. and Thomson, L. (2008). *PHP and MySQL Web Development*. Developer's Library. Pearson Education.