Scientific Research Publishing

Journal of Computer and Communications, 2018, 6, 118-127
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

A Lightweight MVC Framework Based on Code Decoupling Principle

Guiling Sun, Yingjie Wang, Mengsha Li, Zhenjun Liu

College of Electronic Information and Optical Engineering, Nankai University, Tianjin, China
Email: sungl@nankai.edu.cn, 194991129@mail.nankai.edu.cn,562395057@qq.com, liuzhenjun@mail.nankai.edu.cn

How to cite this paper: Sun, G.L., Wang, Y.J., Li, M.S. and Liu, Z.J. (2018) A Lightweight MVC Framework Based on Code Decoupling Principle. *Journal of Computer and Communications*, **6**, 118-127.
https://doi.org/10.4236/jcc.2018.63009

Received: January 29, 2018
Accepted: March 25, 2018
Published: March 28, 2018

Abstract

This paper analyzed the problems in the existing framework, and achieved a reasonable framework based on reflection and aspect-oriented programming combined with annotations to map the request parameters, and inversion of the control of the various components. The proposed framework reduced the volume of the application context, and achieved the purpose that dependencies between the various components are controlled by the framework. Based on the problems existing in the known framework, the lightweight MVC framework was implemented and the proposed framework was tested by JMeter test tool. The test results showed that the framework proposed in this paper can respond to requests faster, improve access throughput and enhance application performance and user experience. The framework proposed in this paper combined the functions of MVC and IOC to minimize the volume of external dependencies in the development and greatly improved the efficiency of Web applications development.

Keywords

Web, Online Framework, Code Decoupling, MVC Pattern

1. Introduction

In object-oriented programming, the so-called framework consists of a set of interoperable, reusable components that can be customized to create different applications. The MVC (Model/View/Controller) is design pattern most commonly used for today's web application development [1]. The core idea of MVC framework is to separate the view layer from the business logic. When the business is logic changes, we only need to modify the model layer and the controller. In addition, the framework's flexible extensibility makes it possible to introduce IOC (Inversion of Control) for further code decoupling. The

so-called IOC is that the program's components lifecycle management is handed over to the container, and the application is not responsible for the life cycle of the required components [2]. Dependency injection, on the other hand, is an extended explanation of the IOC pattern which defines that the high-level components should not rely on the basic one, and both should rely on abstraction, while abstraction does not depend on the details, but the details depend on the abstraction. Decoupling the program through MVC [3], Dependency Injection, and Inversion of Control can significantly improve program reusability and maintainability. At present, the framework of SSH (Struts 2/Spring/Hibernate) is still widely used in the field of JavaWeb lightweight application development because of its versatility. However, SSH has a problem with version chaos, poor compatibility between versions and low efficiency. Therefore, this paper proposes a lightweight MVC framework based on the idea of code decoupling.

1.1. Existing MVC Framework

Struts 2 is an MVC framework, the core for Struts MVC, is DispatcherFilter which plays the role of controller [4] [5]. It is actually a filter. Struts 2 works as long as it is configured to pass all Http requests to FilterDispatcher. Web container first searches for the web.XML, where the developers must first specify Filter Dispatcher's configurations so that Struts 2 can start working. Struts 2's entire workflow is shown in **Figure 1**.

1.2. Existing Problems

1) The core of the Struts 2 itself is a filter [6], which performs a class-level interception in which one class corresponds to one request context, while a method in Struts 2's Action can correspond to a URL; however, its class properties are shared by all methods, which cannot be annotated or otherwise identified by their own methods. As the request is mapped to an instance of an Action class, it makes a big overhead for you to create a corresponding relationship between the framework components and requests.

2) From the above reason, though the method in Struts 2 is independent, but all Action variables are shared, this will not affect the operation of the program, but give us trouble when coding the program. Each time a request comes, an Action instance is created, Struts 2's class-based development model uses class member variables to receive parameters, which cause that the developers can not apply the Singlton design pattern [7], but only able to use multiple instances.

3) Because Struts 2 must package each request, the request object, session and other servlet life cycle related variables are encapsulated into a Map for the Action to use in the course of processing the incoming request, where you need to ensure synchronization of Map access and isolation, which is more resource-intensive [8].

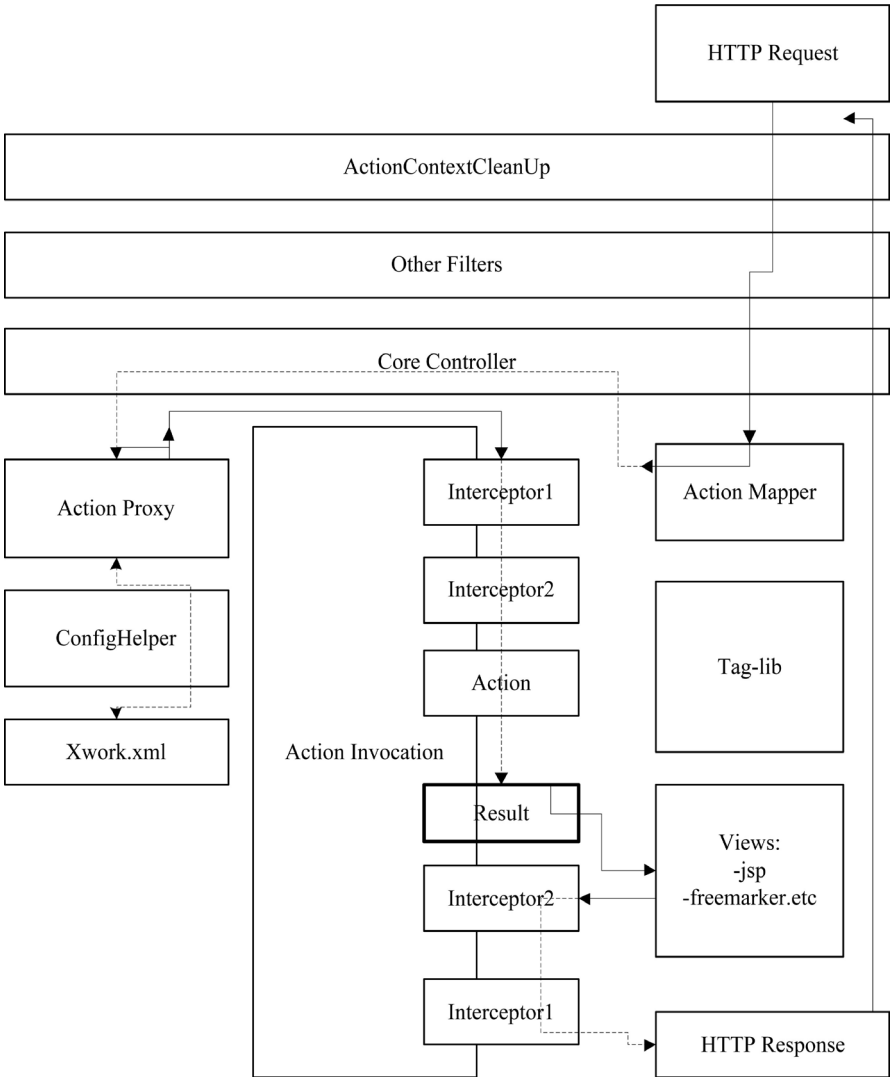

Figure 1. Struts 2 workflow (from Struts 2 official website).

2. Design and Implementation of the Proposed Framework

2.1. Overall Architecture

In view of the above problems, this paper presents an MVC framework based on code decoupling ideas, the overall view of the framework is shown in Figure 2.

Front-End Controller: It is responsible for providing a unified access point for the presentation layer, filtering requests to the rear components of the framework, the requests are gathered here for centralized distribution, so as to avoid duplication of control logic in the old framework, while the front-end controller calls back the function methods provided by the user and provides common logic for multiple requests (such as preparation of the context, encapsulation of the request parameters, etc.), separates the specific view selection and the functional disposal.

Application Controller (hereinafter referred to as the Handler, described later)**:** After the front-end controller separates the selection of the specific view

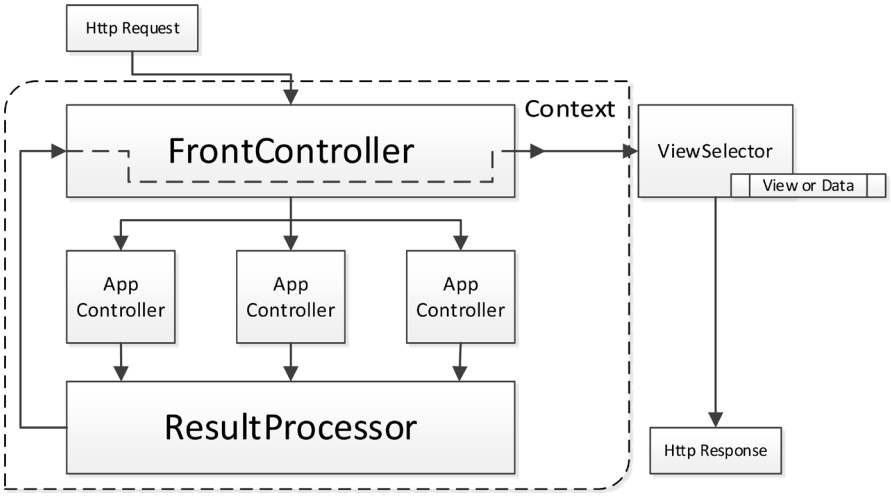

Figure 2. Overall architecture.

and the specific functional action, it needs someone to manage the input parameters to complete the processing logic (such as service layer and data layer), the application controller is used to determine the specific view technology (view management) and the specific function processing strategy, you can easily switch the view/page controller, without having impact on the other part of the program

Result Processor: Handles functional results, collects parameters, encapsulates parameters into the model, transfers them to the business object to handle the model, returns the logical view name to the front-end controller (decoupled from the specific view technology). Delegated to the application controller to select a specific view to show, can be the command design pattern implementation.

View Selector: Processes the result returned to the Front-End Controller for further processing, chooses which specific view to return or the data object to return as a response.

Context: With the context in hand, we prepare the model for the view in the context, thus, we can place the relevant data in the context and access the model data regardless of the protocol (e.g., the Servlet API) through Thread Local mode.

2.2. Components

The framework consists of six major components, the dependencies between the components are shown in Figure 3.

BootStrap Loader:

The framework BootStrap loader is responsible for initializing and starting the whole framework. It first drives the configuration loader to read the relevant configuration files of the framework, and then loads all the classes involved in the framework's base package path into the framework by starting the class loader. Acquire the framework managed bean class (complete the dependency injection) with reference to the relevant dependency injection mapping information

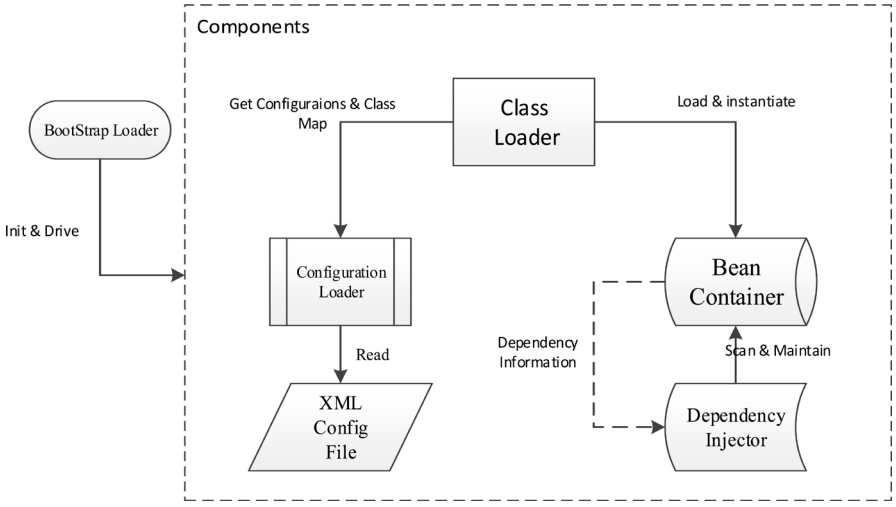

Figure 3. Framework components.

obtained from the dependency injector, add it to the bean container, and finally scan all the bean of the Controller type in the bean container and start the Class Loader.

Class Loader:

Class Loader is the infrastructure of the entire framework, the follow-up work needs to be completed before the class loading, its main responsibility is to load all the classes the framework needs based on the base package path obtained from the configuration loader and to generate the map of classes set for use by other components in the framework.

Configuration Loader:

The Configuration Loader reads in the configuration file under the specified path and saves the base package path, log configuration, static resource path, and the location of the view path for further use by other components in the framework.

Bean Container:

The Bean Container saves all the instances used by the framework, maintains the mapping between the classes loaded by the Class Loader and the corresponding instances, takes care of the entire life cycle of the bean, provides services such as acquiring, adding, modifying bean instances externally and interacts with the dependency injector to complete the framework's internal bean relationships maintenance.

Dependencies Injector:

The Dependency Injector sweeps the bean container, establishes the dependency mapping between each bean and the dependent class collection, meanwhile, manages the behavior between the various class instances.

3. Realization of the Framework

The entire framework is realized with Java, which is divided into six parts whose corresponding code structure is shown in Figure 4.

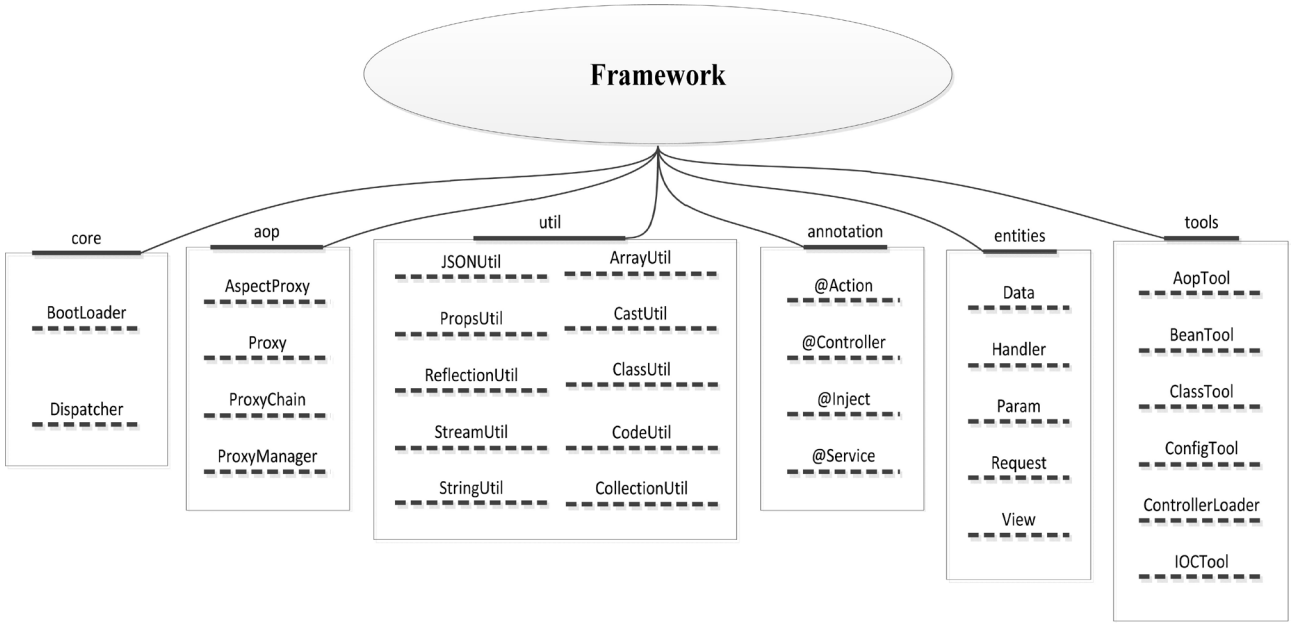

Figure 4. Framework implementation's structure.

The class in the util package is the basic tool that encapsulates the underlying operations and provides basic functionality for other components of the framework. This part of the design follows the toolkit's design methodology and provides all of the functional methods as a static method, but does not allow instantiation and inheritance. Here, identified by class name, functions provided by this package include array operations, data type conversion operations, class-related operations, encoding operations, set-related operations, JSON processing operations, attribute operations, reflection operations, stream operations and string operations.

The class in the tool package uses functions provided in the util (a package name) package, combined with the composition of the framework makes a further functional package encapsulation and abstraction, provides the framework with AOP enhancements, bean management, class loading, configuration file loading, controller loading, dependency injection. It also prepares the startup of the framework by doing the initialization within the static block in each class.

The entities package contains the data needed in the framework, the Http request parameters are mapped to the framework's internal definition of the data model, thus, realizes decoupling of code. The data that the framework needs to use are divided into five categories: Request which packs up the incoming Http request, Param which is the controller's parameters, Handler which is the application processor, View is the jsp response and Data is the json format response.

The annotation package defines annotations for the framework controller identification, dependency injection declaration, business layer description, and request handling method.

The aop package contains the necessary proxy component for AOP. Proxy and ProxyChain is for AOP programming, ProxyManager is for management

and Aspect Proxy is the aspect template.

The core package includes the pre-processor Dispatcher and the initial starter Boot Loader.

In view of the problems raised in the first chapter, the framework considers the use of servlet as a front-end controller to carry out the implementation. By setting the path mapped for the servlet, it is set to respond to all requests to the framework and starts with the web container as started, the front-end controller distributes the requests to different application processor for processing, for the first issue, the framework uses @Controller annotated class instance as a controller, the request parameters are encapsulated as Param object, and it is passed into designated controller method, this way the controller development is based on the level of "method", so that each request corresponds to a method, you can use annotation @Action to mark different methods to map different requests

In the Controller, parameters of the method correspond to the request parameters, since parameters of a method are local variables, they are naturally thread-safe, so you can handle different requests in a single Controller instance without causing errors caused by data sharing. In addition, the framework provides only a lightweight encapsulation of the request, together with the context, they provide the parameter basis for each Action mapped to the method to reduce the size of the context in the framework declaration cycle.

The startup process of the framework is mainly done by setting the frame configuration file, making the framework started with the web container, and calling the framework Boot Loader init method within which completes the progress for other components of the framework to start. This is done by initiating the framework's Class Loader to load classes used in the other components of the framework, static initialization block of these classes will be called in the process. Specifically, first, the Class Loader loads the classes needed for the various components of the framework and registers them to the bean container (maintaining the mapping in BEAN_MAP). The dependency injector then scans the entire BEAN_MAP and gets the individual beans and completes the dependency injection. After that, the Controller Loader scans the BEAN_MAP and obtains the Bean corresponding to the application processor and its corresponding request path, and generates the mapping of the request path to the application processor (maintains the mapping in the ACTION_MAP). At this point, the framework is fully started.

After the request arrives at the framework, the request is distributed by the pre-processor Dispatcher according to the requested Http method and the request path, and then the Class Loader is used to obtain the application processor instance—Handler, that previously registered to the ACTION_MAP determined by the path corresponding to the current request. If the acquisition is successful (there is a registered application processor, otherwise does not respond), ask the framework's IOC container to obtain an instance of the application processor, and further request parameters will be encapsulated, the relevant parameters of

the HTTP request is mapped to the processor method parameters, and then through the reflection tool of the framework, the application processor is called and processed. After the result is filtered by the result processor and the View Selector, the method returns either the rendered view or the JSON data to the client.

4. Experiment and Result Analysis

In order to verify the performance of the proposed lightweight MVC framework, two Web projects were established, which respond's to the client's requests by using the Struts 2 framework and the proposed lightweight MVC framework respectively.

The server environment and configuration used in the experiment are as follows:

CPU: Intel(R) Xeon(R) E7-4820 v3 @1.90 Ghz

Mem: 64GB

OS: RedHat Enterprise Linux Server 6.5

Web Container: Tomcat 8.5

On the server, load the two projects into tomcat and start the tomcat. Then, use Jmeter to simulate multiple clients to issue concurrent requests to Web projects on the server. Jmeter configuration details are shown in Table 1.

Above these configurations, the number of threads requested and the number of simulated clients is 100, the number of loops and the number of simulated requests per client is 100, so that a total of 10,000 requests are sent for each Web project. After configuring the test plan, Start the test plan and add listener to the test plan, until the test plan is completed, we can go through the—Aggregate Graph—to view the statistics of the response, the experimental results are shown in Table 2, after averaging the number of tests.

In the test results, the statistical parameters related to the response in Aggregate Graph of Jmeter are intercepted. Among them:

Samples: Indicates how many requests were made in the test, here we call the total number of requests R, number of clients or number of threads CT, number of visits V. Then we have R = CT*V. So, 100 clients were simulated to send 100 requests each.

Table 1. Jmeter's configuration.

Items	Values
Threads	100
Cycles	100
Server IP	10.134.3.3
Port Num	8080
Protocols & Methods	Http Get
Struts test access path	/Struts 2Test/test
The proposed framework access path	/Mason Framework/test

Table 2. Jmeter results.

Item	Struts 2	Proposed Framework
Samples	10,000	10,000
Average (ms)	15	2
Median	16	2
90%Line (ms)	26	3
Min (ms)	1	0.4
Max (ms)	58	32
Throughput (KB/sec)	4046.1	8038.6
Received (KB/sec)	1301.25	3769.58
Sent (KB/s)	792.99	1593.59

Average: Average Response Time—By default, the average response time for a single Request, when using the Transaction Controller, you can also display the average response time in units of Transaction. Considering that the chance factor of a response is large, the average response time can better reflect the response performance under the average state of the frame.

Median & 90% Line: The former is the median, which is the response time of 50% of users, the latter is the response time of 90% of users. Both reflect whether the various clients can be responded in a timely manner.

Min: The minimum response time, that is, the response time under the best conditions.

Max: The maximum response time, which is the worst-case response time when the framework may have internal problems.

Throughput: By default, Request per Second reflects the ability of the framework to handle the payload of the request. A higher value indicates that the framework performs better under a large number of requests.

Received KB/sec: The amount of data received from the server per second, reflects the ability of the framework to return data to each client.

Sent KB/sec: The amount of data that can be sent to the server per second reflects the amount of data that the server can accept at one time.

From the experimental results, it can be seen that the framework proposed in this paper has an average response time W faster than that of Struts 2 with the same request type and twice the throughput of Struts 2, with superior performance and ability to handle a large number of client requests. From the response time, throughput, data volume, processing power and so on the proposed framework are better than Struts 2 framework.

5. Conclusion

The MVC framework plays an important role in the current Web applications. It is the key to enhance the development efficiency and simplify the programming complexity. Therefore, the improved architecture proposed in this paper is based on the idea of code decoupling MVC framework, with good performance and a

relatively clear code structure. This framework is used to build the laboratory project (Online Shop for food, http://39.107.115.169:8080/shop), which greatly enhances the development and operational performance. However, there are some limitations of the present framework like lack of documentations and so on. Finally, to further expand the functionality of the framework and enhance the versatility of the framework is the direction for further study.

Acknowledgements

This work was partially supported by Tianjin Science and Technology Major Project (2017ZXHLNC00100) and supported by Tianjin Rural Working Committee and Tianjin Key Laboratory of Optoelectronic Sensor and Sensing Network Technology.

References

[1] Wang, N., Li, L.M., Wang, Y.Z., Wang, Y.-B. and Wang, J. (2008) Research on the Web Information System Development Platform Based on MVC Design Pattern. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, *WI-IAT*'08, Sydney, 9-12 December 2008, 203-206.

[2] Cao, Y., Yang, L.N. and Yang, Y.L. (2008) Machine Tool Distributed Cooperative Design System Based on Extended MVC-Based Web Application Framework and XML Interoperable Information Model. *International Conference on Internet Computing in Science and Engineering*, *ICICSE*'08, Harbin, 28-29 Jane 2008, 423-428. https://doi.org/10.1109/ICICSE.2008.67

[3] Chen, Y.E. (2015) Design and Implementation of Web Software System Development Framework Based on MVC Pattern. *Information Systems Engineering*, **6**, 37-37.

[4] Jiang, P.H. and Xu, J.M. (2017) Web Application Testing Framework Based on MVC Model and Behavior Description. *Modern Electronic Technology*, **40**, 71-74.

[5] Johnson, R., Hoeller, J. and Arendsen, A. (2008) Spring, Java/J2EE Application Framework Documentation Version 1.2.8.

[6] Yu, Y.K. (2017) Construction and Improvement of Development Framework Based on MVC Model. *Strait Science and Technology and Industry*, **5**, 98-100.

[7] Li, Z.F. (2017) Web Software System Development Framework Design in the MVC Model. *Electronic Technology and Software Engineering*, **8**, 61-61.

[8] Cao, J., Li, M., Fu, H.R., *et al.* (2017) A Lightweight J2EE Framework Based on J2EE Application. *Electronic Technology and Software Engineering*, **19**, 153-154.