Olympiads in Informatics, 2014, Vol. 8, 69–79

Components and Architectural Design of an Autograder System Family

Jordan FERNANDO, M. M. Inggriani LIEM

Data and Software Engineering Research, School of Electrical Engineering and Informatics
Institut Teknologi Bandung
e-mail: fernandojordan.92@gmail.com, inge@informatika.org

Abstract. A new automatic grading (autograder) system has been set up to support the Indonesian selection and preparation process of IOI candidates in Indonesia. As interest in programming competitions is increasing, we need an autograder system that can be prepared for a specific purpose and is scalable to be able to handle the increasing number of users. Therefore, we have redesigned a system with a new concept and presented it in this paper. The new autograder system consists of interchangeable components to fulfil all kinds of operational purposes. Instead of having a big and complex system, the proposed system will be automatically composed and deployed for specific operational needs. Design, implementation and operation of the autograding and contest management systems are supported by IOI alumni.

Keywords: auto grading, training, programming competition, components.

1. Background and Related Work

The process that Indonesian IOI participants (called "TOKI", abbreviation of Indonesian Computer Olympiad Team) go through is a step-by-step one, starting from the school level, then advancing to regency, province and finally to the national level. After national selection, the candidates are prepared and selected by a team of coaches consisting of Indonesian university faculty members and IOI alumni. Faculty members from five national universities – the University of Indonesia (UI); Bandung Institute of Technology (ITB); the Institute of Agriculture, Bogor (IPB); Gajah Mada University in Yogyakarta (UGM); and the Institut Teknologi Sepuluh Nopember Surabaya (ITS) – contribute as coaches. The preparation and selection process for narrowing down the IOI candidates consists of 4 phases, where 30 candidates must take tests to select sixteen, eight and finally four IOI participants. Whether or not they make it through the national process, they form a body of national training participant alumni and IOI alumni that play significant roles in the process in the following years. Some of them become students at national universities or study overseas.

The process of selecting and training IOI candidates is carried out using the autograder system. The first autograder system in Indonesia was developed at the Univer-

sity of Indonesia by Suryana Setiawan, an Indonesian IOI team leader. This first system was the basis of the newer development at ITB.

In 2008, the IOI Selection Committee identified the need for a better autograder system that could be used by the public. Therefore, Petra Barus, an alumnus of the National Programming Competition, developed the *Toki Learning Center (TLC)* as part of his final project in the Informatics Program at the Bandung Institute of Technology (Novandi, 2009). *TLC* was used for the Open National Olympiad in Informatics, for the online training before the national training and selection process and for local competitions. Its user communication language is Bahasa Indonesia.

In 2010, Petra Barus and Karol Danutama, an IOI alumnus, developed a new version of *TLC*, called *LX*. *LX* is open to the public at `http://www.tokilearning.org`, and it offers a new feature called Training Gate that emphasizes self-exercise. Training Gate provides problem sets that are grouped by solution types and ordered by level of difficulty. Until now *LX* has been used for the overall national training and selection process.

Driven by the need to handle tasks submission and to automate the grading process for large programming classes at ITB, and inspired by *Coursera* (`https://www.coursera.org/`) and *Marmoset* (Spacco *et al.*, 2006), Karol Danutama developed an autograder system that combines automatic grading on *LX* with a Learning Management System (Danutama and Liem, 2013) as part of his final project in Informatics Engineering Study at ITB. The autograder system is called *Oddysseus*, and it provides grading services to many clients such as *Moodle* and *Doppel-Ganger* (Chandra and Liem, 2013). *Doppel-Ganger* is an educational programming tool designed for simple PCs and mobile devices that enables students to run simple programs anywhere. *Doppel* is a dedicated source code editor that provides assessment of the coding process. *Ganger* provides for the compilation and execution processes through *Oddysseus* or a local compiler. The architecture of *Oddysseus* consists of three layers and it is scalable. It has been tested in an Object-Oriented Programming course with 164 students and in an Algorithm & Programming course with 173 students. The autograding process is more adapted to teaching than to competition, and until now has been used in courses in Informatics Engineering at ITB. We have found it very useful in reducing man-hours for grading student assignments as well as motivating the students to do more exercises. However, program execution assessment is not enough for teaching. Therefore the grader should be enriched with various types of source code assessments, which we categorize as white box grading.

The evolution of the autograder systems maintained by ITB is shown in Fig. 1. The usage of the autograder has evolved from competition to teaching programming.

In a programming competition, solutions are graded using black box grading, where the system compares the provided program output with solution output. A pair of input and solution output is called a testcase. Black box grading needs checkers when a problem solution has many possible output. An interactive problem solution needs a special grading process.

There are several ways to grade a source code. Grading types used in competition are subtask, batch, output-only, and interactive. In subtask grading, the testcases are grouped into several subtasks. Each subtask has a score and contains testcases that require certain

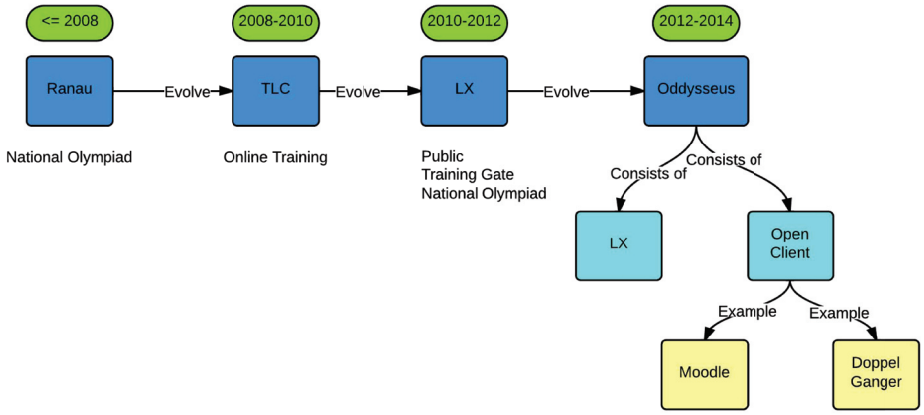

Fig. 1. Evolution of Autograder System maintained by ITB.

algorithm as a solution. In order to gain a score in a subtask, the solution must pass all testcases in the subtask. Testcases of batch grading are not grouped and can be scored partially depending on the correct number of testcases. In output-only grading, a user submits solution in the form of output file(s). In interactive grading, the solution interacts with judge programs to solve the problem.

Currently, *LX* supports subtask, batch, output-only, and Interactive grading types and C, C++, Java, and Pascal programming languages.

Based on experience in using *Oddysseus* for teaching programming, it has been found that program execution assessment is not sufficient. In a formal teaching programming context, the autograder system must be equipped with source code inspection, such as static analysis, bad smell detection, and plagiarism detection. Plagiarism detection is used to compare student source codes with optimal solutions (Kustanto and Liem, 2009).

The autograder system needs problems sets (problem description and testcases) to do the autograding process. The preparation of problems sets takes time to ensure the problems descriptions are clear and the testcases cover all cases. Because *LX* and *Oddysseus* have been used consistently over the years, each system contains many problems-sets. Unfortunately these sets of problems are not well organized and are rarely reused.

With the growing number of users in Indonesia and new problems types in IOI systems, new needs have come up, such as how to set up a competition quickly, how to scout for talent in preparation for the IOI, and how to give support to new IOI grading types as the IOI evolves and is improved. The problem sets need to be pooled in a repository, since existing problem sets are decentralized due to coaching being done at different locations, not only at ITB. Therefore, the IOI Selection Committee requires new autograder systems to meet national IOI preparation and selection objectives. With the benefits of using the autograder system in teaching, the new autograder systems are also designed to support programming courses in the university.

In this research and development, *Oddysseus* serves as the baseline for the development of the new autograder system. The authors have used reverse-engineering tech-

niques to study and improve the design of *Oddysseus* (Pressman, 2010). Other than *Oddysseus* and *LX*, there are other autograder systems such as *Marmoset*, *Open Judge System* (`https://github.com/NikolayIT/OpenJudgeSystem`), and *CMS – Contest Management System* (`https://github.com/cms-dev/cms`). The existing autograder systems only support specific purpose such as competition or learning. The new autograder system is designed to build and deploy many specific purpose autograder systems easily.

2. Problem Statement

The evolution of the autograder system shown in Fig. 1 has produced many versions of the autograder system that exhibit a lack of controls, and the system has been reassembled according to various needs. This situation raises a research question: How can we easily and quickly provide a specific-purpose autograder system and control its versions?

Referring to the needs described in the background, we identify the following specific autograder systems:

a) Competition system.
b) Programming learning system.
c) Programming training system.
d) Problem set repository.

New types of systems are potentially needed in the near future.

3. Methodology

The methods used in the research and development of the new autograder system include: the study of related work, reverse engineering, component requirement analysis, design of the proposed system architecture, component implementation, and case implementation.

During the study of related work, the authors scrutinized the evolution of the system and characteristics of specific systems. We learned how to save varying data and added cross-cutting aspects of the system to be used in the new autograder system (Suwandi, Liem, and Akbar, 2014). The authors also adopted the techniques used in continuous integration to be used in the new autograder system (Humble and Farley, 2010).

In the reverse-engineering stage, the author implemented reverse-engineering techniques on *Oddysseus* to uncover autograder system components.

As a result of previous work and reverse engineering, we propose the architecture of the autograder system. Our main focus is on the static aspect (component design) and on dynamic behaviour or runtime systems where components are assembled into one system. The components were implemented and we performed unit tests on each of them. In addition to component testing, integration testing was also done when the components comprised a single system.

4. Component Requirement Analysis

Our analysis is driven from use cases of the new system covering the usage in all specific systems derived from components. The components are arranged in layers adopted from *Oddysseus*. Each layer of the new autograder system contains components derived from the autograder system use cases. The mapping between the autograder system use cases and the components is shown in Table 1.

The results of the analysis and reverse-engineering processes are shown in Fig. 2.

BB consists of black box grading types such as Output Only (BB1), Subtask (BB2), Batch (BB3), and Interactive (BB4). WB consists of white box grading types such as Static Analysis (WB1), Bad Smell Detection (WB2), and Plagiarism Detection (WB3). Other Black Box and White Box grading types are also included in BB or WB.

Components that have been extracted can be used for specific factory systems in a product line; some examples of build script elements using the components are given in Fig. 3.

Table 1

Mapping between autograder system use cases and components

Use cases	Components
Manage contests	Contest Management System (CMS)
Manage problems and test cases	Repository of Problems (RP) Communicator (C)
Manage results	Live Scoreboard (LS), Front-end Result (FR) Communicator (C)
Submit solutions (competition)	Competition Front-end (CF) Communicator (C), Black box grading (BB)
Manage courses and classes	Learning Management System (LMS)
Submit solutions (courses)	Learning Front-end (LF) Communicator (C), Black box grading (BB), White box grading (WB)
Manage training resources	Training Management System (TMS)
Submit solutions (training)	Training Front-end (TF) Communicator (C), Black box grading (BB), White box grading (WB)
Monitor system resources	Monitor (M), Communicator (C)
Manage users	User Management Front-end (UMF)
Autograding request	Any Front-end (F) Communicator (C), Grader (G)

$BB \rightarrow \{BB1, BB2, BB3, BB4, \ldots\}$	[Black Box grading]
$WB \rightarrow \{WB1, WB2, WB3, WB4, \ldots\}$	[White Box grading]
$LX \rightarrow \{TF, CF, CMS, RP, C, BB, LS, FR, M, G\}$	[LX System]
$Oddysseus \rightarrow \{LF, LMS, RP, C, BB, WB, FR, M, G\}$	[Learning System]

Fig. 2. Component extraction from existing autograder systems.

$$\{CF, UMF, CMS, RP, C, BB, LS, FR, M, G\} \rightarrow Competition\ System$$
$$\{TF, UMF, CMS, RP, C, BB, LS, FR, M, G\} \rightarrow Training\ System$$
$$\{LF, UMF, LMS, RP, C, BB, WB, FR, M, G\} \rightarrow Learning\ System$$
$$\{F, UMF, RP, C, BB, WB, FR, M, G\} \rightarrow Problem\ sets\ Repository\ System$$

Fig. 3. Component composition into specific purposes for the autograder system.

Fig. 3 shows how to create an autograder system for competition. Build script will be generated to build and test Competition Front-end, User Management Front-end, Contest Management System, Repository of Problem, Communicator, Black box grading, Live Scoreboard, Front-end Result, Monitor, and Grader component. After building and testing components, all components will be integrated as system and integration test will be conducted. A competition autograder system is ready to be used for a competition.

By having components of autograder systems, we can recreate many types of autograder systems and simplify the deployment and testing process.

5. Proposed Solution

Our proposed solution consists of static system components and dynamic runtime environment.

5.1. *System Components*

By using reverse-engineering techniques on *Oddysseus*, we found that *Oddysseus* consists of three layers of components: front end, service, and back end. These layers will be adopted with some improvements. The service layer will be changed into a communicator layer and we will add two layers which are cross-cutting and database layers. The communicator layer provides communication methods from the autograder to all external components and provides job distribution. The cross-cutting layer contains components whose function is to take care of other aspects of the system such as security and monitoring. The database layer contains components that take care of the data management system.

Functional components of the system described in the previous section are summarized in Fig. 4.

The general autograder system layers are shown in Fig. 4:

1. Interface Layer.
   User interacts with the autograder system through this layer, using web pages. The functions provided by the interface depend on the type of system autograder. In addition to the functions that depend on the type of autograder system, there are also common functions for all types of user interfaces such as user management.
2. Communicator Layer.

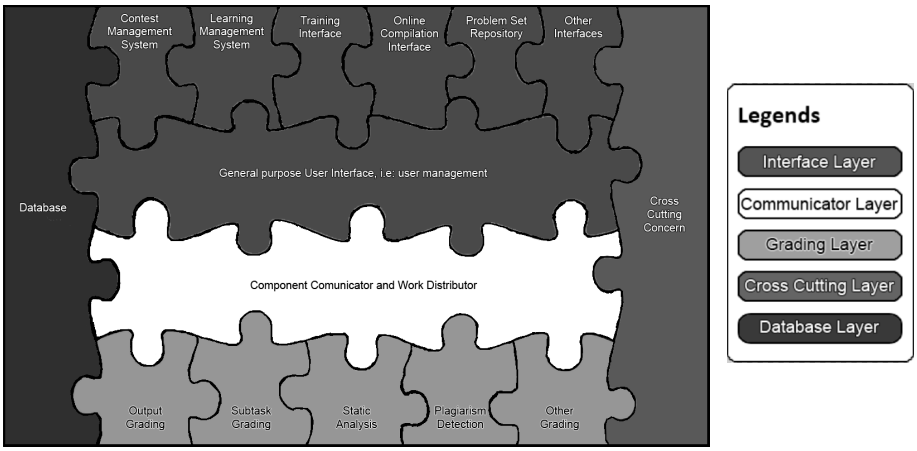

Fig. 4. Layers of multipurpose autograder systems.

In an autograder system, some components need to communicate with each other. Communication can be done through this layer. The main task of the components in this layer is to provide communication between the interface layer and the autograding layer. This layer also provides services to distribute jobs received from the interface layer to the existing components in the grading layer.

3. Grading Layer.
   Automatic grading is the main function of the autograder system. This layer provides grading services to other components. Grading services are invoked by sending a request via the communicator layer. Automatic grading is done by performing a process on the files contained on the grade request such as compilation, execution, and source code analysis. In this layer, each component serves a type of grading such as subtask, outputs only, static source code analysis, and plagiarism detection.
4. Cross-cutting Layer.
   Security and monitoring are cross-cutting concerns that we consider important. All of these concerns can be implemented outside of the main system through aspect-oriented analysis and design.
5. Database Layer.
   The components of the interface layer, communicator layer, and grading layers may need to store specific data to support multiple functions. This function is supported by the database layer components and is implemented as a Database Management System (DBMS). The data model for each use will be designed specifically for that model.

In Fig. 4, components have not been integrated into one specific system. With separation into components, the deployment and testing process will be simplified and automated. Moreover, with the separation of the autograder system into components, component addition or replacement can be done with minimal effort. We have defined components with a higher level of abstraction so that the open-closed principle can be satisfied (Meyer, 1988).

5.2. *Runtime System*

The runtime system consists of two main parts: a front-end subsystem and a grading subsystem. Users of the autograder system interact through the front-end system. The front-end is designed to be implemented as a web-based application, whereas the grading subsystem provides services that are invoked by the front-end subsystem. The web application is being developed based on a data model and a set of available user interface patterns. A new pattern or a specific web page can be integrated as a new link in the web site. Each pattern supports a use case. The patterns are grouped in two categories:

a. User interface patterns that create, read/view, update, and delete (CRUD) data (such as problem set, users, announcements, and events). The data models that are managed through this pattern can be general data models such as users and specific data models such as contests. Data variants are managed by techniques presented in (Suwandi, Liem, and Akbar, 2014). Rules are implemented separately from CRUD.
b. Interface patterns that trigger the autograding process.

The top-level runtime architecture of the autograder system is shown in Fig. 5.

A grading subsystem works when a request of the grading service is invoked. A new grade request is inserted into the job queue through a communicator. A worker pulls the job from the job queue and starts the grading process. After a worker has finished the grading process, the worker sends the grade results back to the communicator. The front-end subsystem then pulls the grade results from the communicator.

The autograding process is the predominant function of the autograder system. In the competition-type autograder system, the grading process only needs to be done using black box with some help from checkers. The autograder system needs to be secure when doing black box grading because the system runs the solution which can be harm-

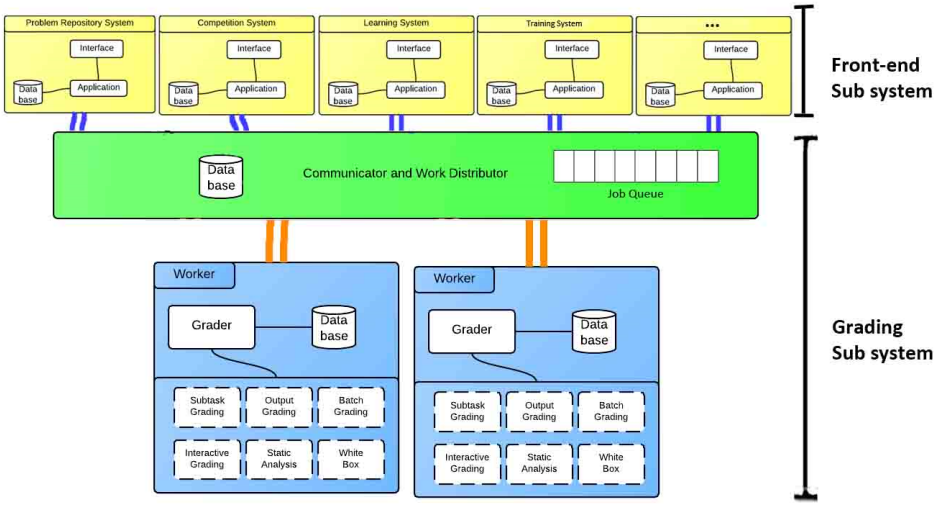

Fig. 5. Runtime architecture of the autograder system.

ful to the system. To ensure security when doing black box grading the solution is run within a sandbox.

A learning-type autograder system requires black box grading and white box grading, such as static analysis, plagiarism detection and bad smell detection. The system can use open source static analyzer tools. Other specific systems that need autograding can have a combination of components and a specific front-end.

The versioning system is set up from the beginning of component development by adopting techniques that exist in the Version Control System (VCS). An overview of the autograder system versioning can be seen in Fig. 6. By implementing these techniques, we have traceable components and also variants of the running system. Each version can be extended into a new branch in which the components can be added or subtracted as needed. Branching is necessary because each system may have a special need that is not the same as for the other systems.

The autograder system must be scalable and robust enough to support many users and a high demand of computing for complex problem solving. The grade requests can be distributed to many worker instances to ensure high performance. Maximum runtime and maximum memory usage of each worker must be set to ensure high performance.

The technology used for the building of the system is *gradle* (`http://www.gradle.org/`), which utilize a Domain-Specific Language to define the building process, and *artifactory* (`http://www.jfrog.com/home/v_artifactory_opensource_overview`) to manage the supply of autograder system components.

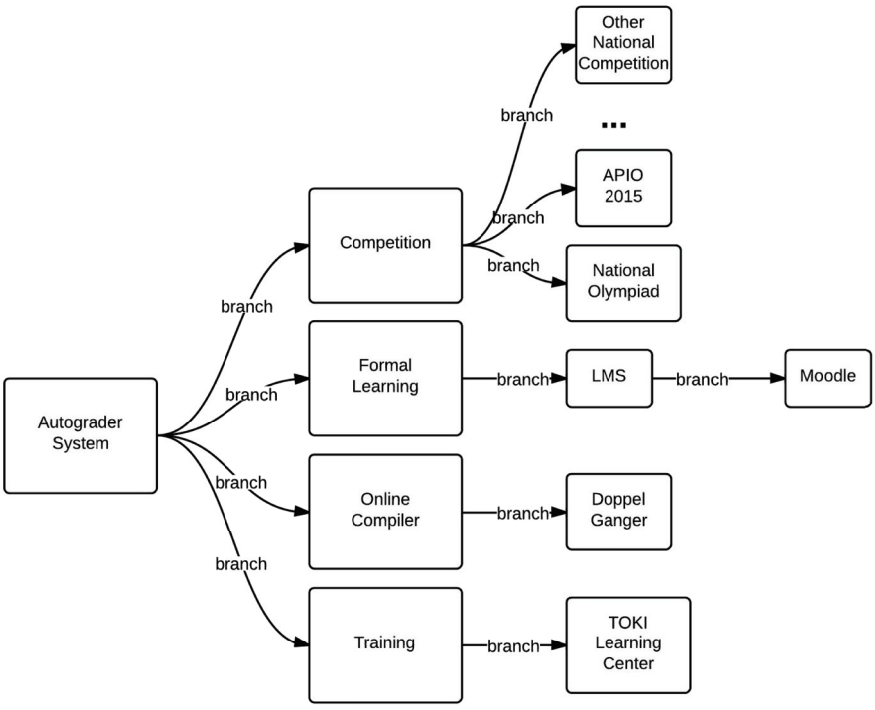

Fig. 6. Autograder system versioning and branches.

6. Case Study

The components will be used to deploy a new competition system, as a part of national preparation in May 2014. In this event, other countries will be invited to participate, since the communication will be in English. The result will be presented in IOI 2014 seminars. The new system is also targeted for use in Asia-Pacific Informatics Olympiad (APIO) 2015, where Indonesia will be the host.

7. Conclusion

The evolution and variation of the autograding system in our national programming competition and formal education is the fruit of IOI alumni and national programming contest alumni contributions. Their experience in competition has led them to be excellent researchers of autograding systems. The results of the work are useful for competition and also for teaching programming and education.

The new component-based system has been proven to make the process of versioning, deployment, testing and synchronization easier. The development of a new grading type or interface could be done with more flexibility to meet particular needs.

The main contribution of this work is a set of components and generic runtime system that can be used to build and deploy a specific-purpose autograder system. The family of specific autograder systems is deployed and tested automatically. The architecture is tested to meet the requirement of scalability, extendibility and adaptabilities.

The new autograder system runs on *Linux* platform. The components are built using Java programming language. The DBMS that is used by the components currently is *MariaDB*. The new autograder system need minimal total RAM of 2GB size, but it is recommended to be run on computer with total RAM of 4GB size. Due to its state as prototype, the source code of the new autograder system has not yet published, but the grading service is opened for public.

Acknowledgements

We would like to thank Mr. Adi Mulyanto and Mr. James Robert Holmboe for their valuable comments.

References

Chandra, T.N., Liem, I. (2013). Source code editing evaluator for learning programming. In: *ICEEI 2013*. 169–175.

Danutama, K., Liem, I. (2013). Scalable autograder and lms integration. In: *ICEEI 2013.* 387–395.

Humble, J., Farley, D. (2010). *Continuous Delivery.* Addison-Wesley, Boston.

Kustanto, C., Liem, I. (2009). Automatics source code plagiarism detector. In: *ACIS/SNPD 2009.* Daigu, Korea, 481–486.

Meyer, B. (1988). *Object-Oriented Software Construction.* Prentice Hall.

Novandi, P.S. (2009). TOKI Learning Center – Sistem Pelatihan Kompetisi Pemrograman Komputer. *ITB Final Project*.

Pressman, R.S. (2010). *Software Engineering a Practitioner's Approach Seventh Edition*. McGraw-Hill, New York.

Spacco, J., Hovemeyer, D., Pugh, W., Fawzi, E., Hollingsworth, J.K., PaduaPerez, N. (2006). Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In: *ITICSE '06*. 13–17.

Suwandi, A., Liem, I., Akbar, S. (2014). Concern-based SaaS application architectural design. In: *ICT-EurAsia 2014*. 228–237.

J. Fernando is a student of Informatics Engineering at Institut Teknologi Bandung. He is the Indonesian technical committee team leader in the national IOI preparation in 2013 and 2014. He is doing his research in the development of autograding components and runtime systems as a part of his final project.

M.M.I. Liem is a member of Data and Software Engineering Research Group in the School of Electrical Engineering and Informatics at Institut Teknologi Bandung (ITB). She has been teaching programming at ITB since 1977. She obtained her doctoral degree in Universite Joseph Fourier in Grenoble, France in 1989, with the teaching of programming as the major topic of her dissertation. Since 2004, she has been involved as a team member in national recruitment, training and IOI preparation for the Indonesian team. She is also the ITB ACM ICPC coach and advisor.