Requirement analysis for autograder system using use case model

R Elsen*, D Kurniadi, S Rahayu and M R Nashrulloh
Department of Informatics, Sekolah Tinggi Teknologi Garut, Jalan Mayor Syamsu 1, Garut 44151, Indonesia

*rickardelsen@sttgarut.ac.id

Abstract. At the beginning of software development, requirement analysis is needed to determine what features and components must be applied in the software to meet its purpose. Autograder have the purpose to grade student submission so the lecturer and student can know the grade immediately. To fulfil it, an analysis of requirements for autograder should be done so we can determine what features and components needed to be applied in autograder. This requirement also can be used to determine the future test plan. In this paper, we analyse the needed requirement for the autograder system and present it in the use case model.

1. Introduction
A distributed model of autograder use multiple backends and all backends managed by a load balancer to manage all requests and responses [1]. The main purpose of this autograder is to grade the student’s code submission by executing the code with some input and compare the result with prepared output. When students submit their work, the load balancer will manage the queue of grading requests and choose an idle backend to execute and grade it. By using multiple backends, student’s submission can be graded fast because it reduces waiting time in the queue compared with a centralized model that using only one backend.

This model can be used by developers who want to build a grader system. This model proposes four components to build: application portal, load balancer, code executor, and grader. We assume that the application portal is stakeholder-based and it can be different between one system and another. But load balancer, code executor, and grader can be reused in another system with few adjustments.

In this paper, we propose a requirement analysis for autograder with distributed architecture. Our scope is focused on load balancer, code executor, and grader. We exclude the application portal since it must be analysed together with stakeholders to meet their goals. We found an example when autograder has been integrated with a learning management system (LMS) [2]. All analysis results will be presented in the use case diagram since it could describe system functionality [3]. A case study we found also showing that the use case diagram can be used to visualize requirements [4].

2. Methodology
In this paper, we use requirement elicitation with goals approach [5-8]. This method focusing on input requirement, but still implement elicitation rule. It has divide requirement analysis process into four steps:
Since we focused only on load balancer, code executor, and grader which can be reused and can be analysed without stakeholder, we exclude the “identify key stakeholders” step and focus on the other three steps.

3. Result and discussion

Requirement divided into functional requirement and non-functional requirement [9]. Functional requirement describes any operation and activities that the system can perform [10]. Non-functional requirement describes how a system should work [11]. It also called quality attributes [12].

3.1. Functional requirement

We determine the functional requirement based on the main purpose of each component in autograder as follow [1]:

- Load balancer's main purpose is managing communication between backends and the application portal by managing requests, responses, and the queue. This component orchestrating requests by sending it to the available backend so it can reduce queue waiting time.
- Code executor's main purpose is to execute code with the input test case. Since there is no information about the student’s submission (non-compiled code or executable file), so we assume code executor should also compile code. This component will be multiplied and managed by the load balancer to reduce queue waiting time.
- Grader’s main purpose is to compare execution results with test case and grade comparison results based on the grades given by the lecturer for every test case passed. Since code executor will be multiplied to reduce queue waiting time, this component will also be multiplied to synchronize grading time with code execution time.

Based on the component’s main purpose, we determine goals, subgoals, and input requirements as follow:

![Figure 1. Requirement elicitation with goals approach [8].](image-url)
Table 1. Goals approach requirements.

| Goals          | Subgoals                        | Input                                      |
|----------------|--------------------------------|--------------------------------------------|
| Manage communication | Manage request | Request body, request header |
|                | Manage response       | Grader result                               |
|                | Manage queue          | Requests, responses, backend status        |
| Code execution | Compile code          | Student’s submission                        |
|                | Execute compiled code | Student’s submission, input test case       |
| Grading        | Compare result        | Execution result, output test case          |
|                | Grade result          | Comparison result, given grade             |

As seen in Table 1, all goals were extracted to subgoals. These subgoals are initial requirements that determine the main requirement of autograder. We describe the use case diagram based on the requirement as seen in Table 1 as follow:

![Use case diagram for functional requirement](image)

Figure 2. Use case diagram for functional requirement [13-15].

Table 2. Use case explanation.

| Actor           | Use Case       | Explanation                                                                 |
|-----------------|----------------|-----------------------------------------------------------------------------|
| Load Balancer   | Manage requests| Manage request demand from the application portal by add requests to a queue and distribute the request to the available backend. |
|                 | Manage responses| Manage response from backend by sending the grading result to the application portal. |
|                 | Manage queue   | Manage request queue by creating a queue, sort request based on given condition like arrival time or priority, and choose available backend. |
Table 2. Cont.

| Code executor | Compile code | Compile student’s code submission by using a compiler. It uses different compiler based on the programming language used in the code submission. |
|---------------|-------------|----------------------------------------------------------------------------------------------------------------------------------|
| Execute compiled code | Execute code submission that already compiled using the input test cases. This process can be looped based on the number of the input test case. |
| Grader | Compare result | Compare execution result with the output test plan. This process can be looped based on the number of execution results and output test cases. |
| Grade result | Grade result | Grading code submission based on every passed comparison between execution result and output test plan. |

3.2. Non-functional requirement

Autograder with distribution architecture aims to reduce queue waiting time because of the long queue caused by request demand to be executed in the backend [1]. We conclude it as the main reason why autograder must use multiple backends with a load balancer as the orchestrator and determine the non-functional requirements as follow:

Table 3. Non-functional requirement.

| No  | Non-functional Requirement                                                                 | Category         |
|-----|-------------------------------------------------------------------------------------------|------------------|
| NFR1| Code executor should compile or execute code fast. We determine it must not more than 5 seconds. If code execution takes too long, it must be terminated | Performance      |
| NFR2| Load balancer should check backend availability in real-time for reducing waiting time.     | Performance      |
| NFR3| Backend (code executor and grader) should be independent so it easier to multiply and integrate it with the load balancer | Maintainability  |

We believe there will be more non-functional requirements can be described when autograder development begins. Non-functional requirements influenced by various and relative aspects like stakeholder interest, development environment, and available budget.

4. Conclusion

Using distributed architecture on autograder intends to boost performance especially in processing student’s code submission. This architecture focus on optimizing backend by reducing waiting time caused by a long queue. It has three components that have a significant impact on performance: load balancer, code executor, and grader. We describe functional and non-functional requirements based on this proposed architecture. But we know our analysis still need progress since we exclude stakeholder because it will be subjective to analyse it.

Acknowledgment

The Authors acknowledge Sekolah Tinggi Teknologi Garut for supporting and funding this paper.

References

[1] Elsen R, Latifah A and Sutedi A 2019 Distributed architecture for autograding system Journal of Physics: Conference Series 1402 077017.
[2] Danutama K and Liem I 2013 Scalable Autograder and LMS Integration Procedia Technology 11 388-395.
[3] Yang-Turner F and Lau L 2011 Extending use case diagrams to support requirements discovery.
2011 Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems.

[4] Mijanur Rahman M and Ripon S 2013 Elicitation and Modeling Non-Functional Requirements – A POS Case Study *International Journal of Future Computer and Communication* 485-489.

[5] Jaramillo Franco 2015 A Requirements elicitation approaches: A systematic review *9th International Conference on Research Challenges in Information Science (RCIS)*.

[6] Sutcliffe A and Sawyer P 2013 Requirements elicitation: Towards the unknown unknowns *21st IEEE International Requirements Engineering Conference (RE)*.

[7] Niu N, Brinkkemper S, Franch X, Partanen J and Savolainen J 2018 Requirements Engineering and Continuous Deployment *IEEE Software* 35(2) 86-90.

[8] Chand M G, Reddy K N, Rao A A and Kumar J K 2010 An approach to requirements elicitation and analysis using goal *2nd International Conference on Software Technology and Engineering* 1 V1-218.

[9] AltexSoft 2020 *Functional and Nonfunctional Requirements: Specification and Types*.

[10] Ofni Systems 2020 *Functional Requirements (Functional Requirement Specifications, Functional Specs, FRS, FS)*.

[11] Eriksson U 2020 *What are Non-Functional Requirements? (ReQtest)*

[12] Guru99(dot)com 2020 *Functional Requirements vs Non Functional Requirements: Key Differences*.

[13] Heumann J 2020 *Tips for writing good use cases* (New York: Somers IBM).

[14] Visual-paradigm(dot)com 2020 *How to Write Effective Use Cases?*

[15] Savic D, da Silva A R, Vlajic S, Lazarevic S, Stanojevic V, Antovic I and Milic M 2012 Use case specification at different levels of abstraction *In 2012 Eighth International Conference on the Quality of Information and Communications Technology* 187-192.