Modularity for Automated Assessment: A Design-Space Exploration

Steffen Zschaler
Department of Informatics
King's College London
The Strand
London
UK
szschaler@acm.org

Sam White
Department of Informatics
King's College London
The Strand
London
UK

Kyle Hodgetts
Department of Informatics
King's College London
The Strand
London
UK

Martin Chapman
Department of Informatics
King's College London
The Strand
London
UK

***Abstract*—As student numbers continue to increase, automated assessment is an inevitable element of programming education in university contexts. Modularity is a key factor in ensuring these systems are flexible, robust, secure, scalable, extensible, and maintainable. Yet, modularity has not been explicitly and systematically discussed in this field. In this paper, we first present an overview of the modularity design space for automated assessment systems and a discussion of existing systems and their place in this space. This is followed by a brief overview of our novel NEXUS platform, which uses fine-grained modularisation of graders implemented through a micro-service architecture.**

I. INTRODUCTION

Automated grading has been of interest to computer-science educators for a long time. ASSYST [13] was probably the first systems that tried to provide automation support for the task of assessing student submissions to programming assignments. At the time, automation was focused on giving a first assessment of a piece of software, with responsibility for the actual grading and provision of feedback still remaining firmly with the human teaching staff. As student numbers increase, we see a stronger move towards fully automated grading and feedback. Such a system has many benefits, including a reduction in time spent on marking by teaching staff, freeing them up for more productive work, and an opportunity for students to receive more feedback on incremental development stages of their software.

As the functionality and workload of automated grading systems increases, it has become evident that we need to consider sound software-engineering principles in the development of these systems. We have requirements on the reliability, security, extensibility, scalability, and maintainability of these grading systems, which cannot easily be satisfied with a simple set of shell scripts to be invoked by a teacher in response to a set of student submissions. Students are asking for web-based systems well integrated with their preferred tool infrastructure and providing high-quality, near-time feedback. Teachers are interested in using automated grading systems for a range of modules teaching programming in different languages, teaching other aspects of computer science, or even beyond computer science. System operators want automated grading systems to be robust against (intentional or unintentional) attacks by student code under assessment and to be scalable in order to manage the highly bursty workload of large classes of students working against coursework submission deadlines. Errors are inevitable in any software development, so we want to be able to partially update an existing, running automated grading system, as fixes for problems identified become available. Finally, academics want to be able to continuously develop our grading systems as our research into better methods for assessment and feedback evolves.

Modularity is key to satisfying all of these requirements. By breaking grading systems into suitable components and ensuring they can be exchanged and recombined flexibly and robustly, we establish the foundations for extensibility, maintainability and reliability. By ensuring independent execution of components, we achieve scalability through appropriate replication and load-balancing techniques. Modularity also provides opportunities for encapsulation and sandboxing, which can help ensure secure execution of student code.

Some existing systems already reap some of the benefits of modularity. However, to the best of our knowledge, no systematic mapping of the design space for modularity in automated grading systems and the benefits and drawbacks of different spots in this space exists.

In this paper, we provide a first such analysis. We begin by exploring two dimensions of modularity: what to modularise and how to modularise. These dimensions span a design space, and we identify different positions in this space taken by different existing systems. Our preference is for a fine-granular, micro-service–based modularisation and we will briefly sketch the architecture of our NEXUS system, implementing such an architecture.

II. MODULARITY IN AUTOMATED ASSESSMENT SYSTEMS

We discuss two dimensions of the design space for modular automated-assessment systems:

1) *What to modularise.* Which different concerns in a grading system should be separated out into different modules? At what granularity?

2) *How to modularise.* What are the technology choices available for implementing components and re-composing them into a working automated-assessment system?

For each dimension, we discuss a range of choices taken by different existing systems[1] as well as the benefits and drawbacks of these choices.

A. What to modularise

In [20], Sclater and Howie describe requirements for the "ultimate" automated assessment system (AAS). In a similar vein, in Fig. 1 we sketch the logical architecture of the "maximal" AAS. Figure 1 is meant to highlight the units of functionality required for any AAS, but should not be read as a description of how to bundle these functionalities into actual components. Any AAS would need to allow assignments to be created and viewed and submissions to be sent to the system (possibly through a number of different submission pathways). Submissions received need to be graded by grader services; multiple grading steps will likely be involved and require some form of scheduling and weighting of the resulting marks according to a mark scheme. Marks and feedback will need to be presented to students and academics in meaningful ways. To execute grading, a number of execution services are needed, such as sandboxing of execution, monitoring, or possibly load balancing across grader servers. System wide, common services are required to provide solutions for submission storage, auditing of the grading process, plagiarism detection, or setting of unique assignments etc. Often, AAS are used in the context of existing virtual-learning environments (VLE) and some functionality needs to be provided to connect VLE and AAS.

In the following, we discuss different options for "packaging" these logical units of functionality into actual system components. Not all AAS provide all functionalities—for example, many AAS do not provide explicit support for VLE integration.

Any automated assessment system will need to provide support for defining and managing assignments, enable students to make submissions (possibly through a number of different submission pathways), execute grading tools on the submissions, and present grades and feedback to students. In most grading systems—including ASSYST [13], Graja [8], PABS [12], PASS [5], ASAP [6], DUESIE [11], BOSS [14], or Marmoset [21]—these two concerns are separated into at least two components: one component provides functionality for teachers to define assignments and for students to upload submissions, while a second component performs the actual grading.

In modularising graders, we differentiate two levels of granularity:

1) *Coarse-grained grader modularisation* packages a complete grading pipeline (often for one course) into one module. For example, a grader might compile Java code, perform some style checks or static analyses, run a number of unit tests, and provide a combined grade and accumulated feedback to the student. The automated-assessment systems listed in the previous paragraph are examples of coarse-grained grader modularisation.
2) *Fine-grained grader modularisation* considers graders building blocks for constructing a marking scheme for an assignment. This implies graders can be more flexibly re-used across assignments and assignments can choose to use only the graders required. Fine-grained grader modularisation has, for example, been applied in ASB [9], JACK [23], and CourseMarker/Ceilidh [10].

Clearly, fine-grained grader modularisation has many benefits. In particular, some graders are easily reusable across assignments and courses (e.g., a peer feedback grader as described for Praktomat [24]), and with fine-grained modularisation, these can be easily reused for different courses. Similarly, fine-grained grader modularisation makes it easy to select exactly the set of graders that should be applied for a particular assignment and to weight the grades provided specifically for the expected level of teaching. For example, for beginning programmers, we might put more weight on compilation, while for more advanced programmers this would become merely a check at the start of the assessment, with substantial weight placed on the assessment of functionality.

At the same time, there are drawbacks to fine-grained modularisation. First, graders must be developed independently and cannot easily make assumptions about other graders. This may mean some duplication of effort (*e.g.,* for compilation) and requires explicit support for workflow management to ensure redundant feedback is not given to students (*e.g.,* when compilation fails, attempting to run unit tests would only confuse students with redundant feedback). With coarse-grained grader modularisation, these interdependencies can be easily hard-coded. With fine-grained modularisation, responsibility rests with the assignment author. An important question is who "owns" which part of the data: most current techniques seem to focus on giving ownership to the central system, so that graders must fit all their configuration data into a centrally pre-defined schema (*e.g.,* using ProForma [22]). ASB [9] supports hierarchical configuration, where configuration at the course level is automatically reused for all assignments *etc.* Configurations still need to correspond to a centralised data schema. Typically (*e.g.,* as described for JACK [23]) this centralisation means that the data is also managed by the central "assignment manager" so that graders need to request it every time they grade a submission. As we will describe in Sect. III, we prefer a modularised configuration, where each grader "owns" its own configuration data (both the schema and the actual data) as this decouples different parts of the system better and requires less network traffic when marking submissions.

Other concerns of automated assessment have also been considered for modularisation. However, more detailed analysis of these modularisation choices is clearly still needed:

[1]We tried to include as many existing systems as possible, but do not claim complete coverage of the literature.

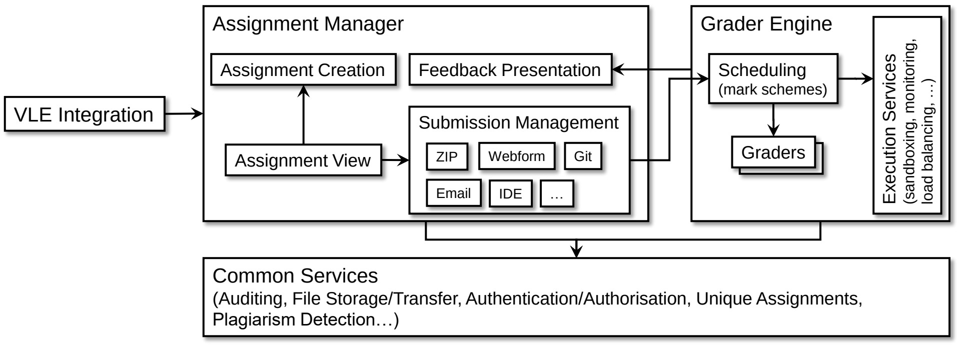

Fig. 1. Logical architecture of the "maximal" automated-assessment system

- Some grading systems have focused on modularising their front-end services. For example, BOSS [14] distinguishes student and teacher servers to support the different roles engaging with the system (in principle, this could be extended to a larger subset of the ideal set of stakeholders [20]). Similarly, FW4EX [19] defines separate servers for creating/editing assignments, accessing assignments, and uploading submissions, so that these can be resourced in accordance with their different performance requirements. FW4EX also has some modularisation of different submission pathways through the introduction of IDE plugins for use by students.
- Some works have focused on modularising general services. For example, Grappa [7] provides generic middleware for connecting graders to a standard VLE. PABS [12] uses a standard SVN server as the shared file-storage system for submissions and assignment data (although, surprisingly, both are kept in the same repository). ASB [9] explicitly modularises different execution environments. A number of systems provide plagiarism detection services as a separate module.

B. How to modularise

The decision of *what* to modularise affects primarily the maintainability, extensibility, and flexibility of the system. In order to modularise for scalability, robustness, and security, we need to consider *how* to implement the modular architecture.

There is a clear trend from monolithic systems with some internal modularisation (typically using object-oriented principles—for example, PASS [5]) to more distributed and loosely coupled multi-process systems (*e.g.,* JACK [23] or [17]). For the latter systems, different communication protocols have been experimented with: [17] uses internal email communication, ASB [9] implements a dedicated event-bus, JACK [23] uses a bespoke protocol with graders pulling new submissions from the assignment manager. Recently—for example in eduComponents [1]—service-based architectures are being discussed. Especially when building on standardised protocols (such as REST-ish web APIs, containerisation through Docker or similar, . . .), these substantially simplify the scaling of automated assessment systems through the use of state-of-the-art virtualisation, load-balancing, and swarm-management techniques. Containerisation also helps with security as containers can be confined in their use of resources. This has been initially explored, for example, for Praktomat [4].

Distribution and using web-based APIs brings its own potential security challenges: if these API endpoints can be accessed by students, extra measures are needed to prevent students from submitting fake marks for their own assignments or from manipulating grader configurations. Service-based distribution potentially also creates concerns about trust between services, especially where they are managed by different organisations.

With fine-grained grader modularisation, an interesting question is how to specify mark schemes (*i.e.,* which graders to use and how to weight the marks provided). For coarse-grained grader modularisation, the mark scheme can be hard-coded into the system or possibly configured by parametrisation of the fixed steps [8]. For fine-grained grader modularisation, different systems provide different answers. Some systems [10] have used Java code to implement mark schemes. This maximises flexibility, but also requires more attention to low-level detail from the assignment developer. Other systems [22], [9], [19] allow configuration through (XML) files at the level of assignments. This is less flexible, as essentially the assignment developer can only choose from a range of pre-defined configuration options, but provides a more standardised interface and higher level of abstraction.

III. NEXUS: A MICRO-SERVICE APPROACH TO FINE-GRANULAR MODULARITY

At King's we are developing an automated assessment platform called NEXUS. This platform was designed specifically to be flexible and extensible, including potentially to non-programming modules. To support extensibility, we aimed to maximise modularity: NEXUS uses fine-granular modularisation of graders and modularises a number of common services: in particular, all submissions are stored in GitHub Enterprise (decoupling graders from submission pathways and providing access to a student's submission history) and we provide generic support for the generation of unique assign-

TABLE I
CURRENTLY AVAILABLE GRADERS

Grader	Purpose
`javac`	Check compilation and code style of `Java` code
`jUnit`	Run unit tests against `Java` code
`io-test`	Run input-output tests against `Java` code
`dyn-trace`	Capture `Java` execution traces & compare against model solution
`matlab`	Grade MatLab-based mathematics submissions
`python`	Unit test python-based submissions
`peer-review`	Enable student peer review of submissions
`manual`	Support manual grading of assignments

ments. Flexibility and security is increased by decoupling all components into their own micro-service [15]; with services interacting through REST-ish web-API endpoints. Figure 2 gives an overview of the architecture. Each service (with the exception of GitHub Enterprise, which is managed separately within King's) is maintained and managed by us in a central monorepo [16] and is deployed in its own Docker containers. As a result, we can easily scale the system by deploying redundant instances of overloaded services and using Docker Swarm to ensure adequate load balancing. Using a monorepo means development can proceed as if the system was a tightly coupled monolith, ensuring continuous compatibility between all micro-services in the overall architecture.

All parts of the system are loosely coupled. For example, the available graders are configured by providing the URLs of their respective API endpoints for configuration and submission-marking. This makes it easy to add in additional graders even at runtime, by adding their API information through the web-based administration interface. We use a fine-granular modularisation of graders (see Table I), including some graders such as `peer-review` or `manual` that can be easily reused across different modules. Graders own their configuration data, allowing for arbitrarily complex data schemata. For example, the `peer-feedback` grader allows the configuration of a web-form for students to fill in when providing a review.

Because graders are independent micro-services, faults are easily contained in the problematic grader. A fault in a grader may mean students having to wait longer to receive a grade or feedback, but will not cause issues for the remaining graders or assignments. This also minimises the opportunities for rogue student code to attack the grading infrastructure. At the same time, it also simplifies incremental improvement of graders. Student submissions can sometimes be highly creative and difficult to predict, occasionally causing graders to fail processing a submission. Because all submissions are managed centrally and the actual submission files are kept in GHE, a faulty grader can easily be repaired and a new version spun up (reusing configuration data from the service's database). NEXUS provides the ability to request a regrading of a particular submission or a set of submissions, whereupon these submissions are simply sent to the relevant graders again for repeat processing.[2] This increases overall reliability and resilience.

We provide explicit support to express grader dependencies and use simple distributed dataflow controls [2]: graders can specify the type of files they produce and require. Given such specifications, NEXUS will automatically determine a maximally concurrent grader execution. When invoking a grader, NEXUS provides information about which kinds of files should be sent to other graders. Graders then send on any files produced as indicated. Subsequent graders can use these additional files without having to recreate them.

Modularisation and the use of web interfaces creates a new attack surface. Students could, in principle, attempt to spoof marks by sending requests directly to NEXUS' `IMark` interface. Similarly, they could attempt to modify the configuration of graders by directly accessing their `IConfig` interfaces. To prevent such attacks, NEXUS uses randomly generated tokens which must be passed along with any HTTP/HTTPS requests. These allow the receiver to check that the invocation did indeed originate from the claimed source. Graders provide feedback as HTML code, which is directly embedded into the NEXUS feedback page. If graders were allowed to be arbitrary web-services, this could easily be a security risk through the potential for XHR attacks. However, in our micro-service architecture, all services are directly under our control so that we can trust their implementations.

IV. CONCLUSIONS

Modularity concerns are important for developing robust, scalable, flexible, and extensible automated assessment systems. Yet, modularity has not been systematically discussed in this field to date. We have presented an exploration of the modularity design space for automated assessment, including how different existing systems are positioned in this space and some of the benefits and drawbacks of different choices. Additionally, we have briefly presented our novel NEXUS platform taking a previously unoccupied position in this space by providing fine-granular grader modularisation realised through a micro-service architecture. We believe that this architecture provides substantial benefits to the robustness and flexible extensibility of our platform as well as to its scalability and reliability. In the future, we plan to extend usage of our NEXUS platform, including to modules outside of computer science and, in particular, will focus on improving the feedback provided by individual graders.

REFERENCES

[1] M. Amelung, P. Forbrig, and D. Rösner, "Towards generic and flexible web services for e-assessment," in *Proc. 13th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE'08)*. ACM, 2008, pp. 219–224. [Online]. Available: http://doi.acm.org/10.1145/1384271.1384330

[2] W. Binder, I. Constantinescu, and B. Faltings, "Decentralized orchestration of composite web services," in *IEEE Int'l Conf Web Services (ICWS'06)*, 2006, pp. 869–876.

[3] O. J. Bott, P. Fricke, U. Priss, and M. Striewe, Eds., ser. Digitale Medien in der Hochschullehre. Waxmann Verlag GmbH, 2017, vol. 6.

[2]NEXUS maintains an audit log tracking any such requests

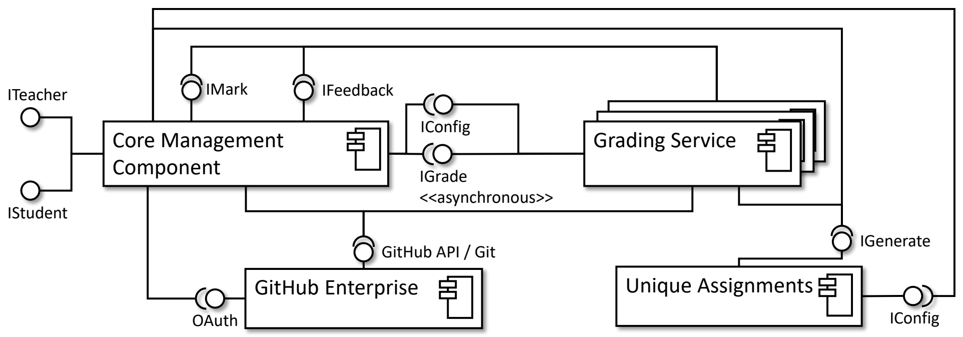

Fig. 2. Architecture of Nexus. Each box is a separate micro-service, all interfaces are REST-ish web-API endpoints.

[4] J. Breitner, M. Hecker, and G. Snelting, "Der grader praktomat," in *Automatisierte Bewertung in der Programmierausbildung*, ser. Digitale Medien in der Hochschullehre, O. J. Bott, P. Fricke, U. Priss, and M. Striewe, Eds. Waxmann Verlag GmbH, 2017, vol. 6.

[5] M. Choy *et al.*, "Design and implementation of an automated system for assessment of computer programming assignments," in *Proc. 6th Int'l Conf. Advances in Web Based Learning (ICWL'07*, H. Leung *et al.*, Eds. Springer, 2008, pp. 584–596. [Online]. Available: https://doi.org/10.1007/978-3-540-78139-4_51

[6] C. Douce *et al.*, "A technical perspective on ASAP – automated system for assessment of programming," in *Proc. 9th Computer-Assisted Assessment (CAA) Conference*, 2005.

[7] P. Fricke *et al.*, "Grading mit Grappa – Ein Werkstattbericht," in *Proc. 2nd Workshop "Automatische Bewertung von Programmieraufgaben" (ABP'15)*, ser. CEUR-WS, U. Priss and M. Striewe, Eds., vol. 1496, 2015, pp. 9–1–9–8. [Online]. Available: http://ceur-ws.org/Vol-1496/paper9.pdf

[8] R. Garmann, "Graja – Autobewerter für Java-Programme," Fakultät IV – Wirtschaft und Informatik, Hochschule Hannover, Tech. Rep., 2016. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bsz:960-opus4-9418

[9] B. Herres, R. Oechsle, and D. Schuster, "Der grader asb," in *Automatisierte Bewertung in der Programmierausbildung*, ser. Digitale Medien in der Hochschullehre, O. J. Bott, P. Fricke, U. Priss, and M. Striewe, Eds. Waxmann Verlag GmbH, 2017, vol. 6, pp. 255–271.

[10] C. A. Higgins *et al.*, "Automated assessment and experiences of teaching programming," *J. Educ. Resour. Comput.*, vol. 5, no. 3, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1163405.1163410

[11] A. Hoffmann *et al.*, "Online-übungssystem für die Programmierausbildung zur Einführung in die Informatik," in *6te e-Learning Fachtagung Informatik (DeLFI'08)*, ser. LNI, S. Seehusen *et al.*, Eds., vol. 132. GI, 2008, pp. 173–184.

[12] L. Iffländer *et al.*, "PABS – a programming assignment feedback system," in *Proc. 2nd Workshop "Automatische Bewertung von Programmieraufgaben" (ABP'15)*, ser. CEUR-WS, U. Priss and M. Striewe, Eds., vol. 1496, 2015, pp. 5–1–5–8. [Online]. Available: http://ceur-ws.org/Vol-1496/paper5.pdf

[13] D. Jackson and M. Usher, "Grading student programs using ASSYST," in *Proc 28th Technical Symposium on Computer Science Education*. ACM, 1997, pp. 335–339. [Online]. Available: http://doi.acm.org/10.1145/268084.268210

[14] M. Joy, N. Griffiths, and R. Boyatt, "The Boss online submission and assessment system," *J. Educ. Resour. Comput.*, vol. 5, no. 3, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1163405.1163407

[15] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2015.

[16] M. Oberlehner, "Monorepos in the wild," 2017, last accessed 30 January, 2018. [Online]. Available: https://medium.com/@maoberlehner/monorepos-in-the-wild-33c6eb246cb9

[17] A. Pardo, "A multi-agent platform for automatic assignment management," in *Proc. 7th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE'02)*. ACM, 2002, pp. 60–64. [Online]. Available: http://doi.acm.org/10.1145/544414.544434

[18] U. Priss and M. Striewe, Eds., *Proc. 2nd Workshop "Automatische Bewertung von Programmieraufgaben" (ABP'15)*, ser. CEUR-WS, vol. 1496, 2015.

[19] C. Queinnec, "An infrastructure for mechanised grading," in *Proc. 2nd Int'l Conf. Computer Supported Education*, 2010, pp. 37–45.

[20] N. Sclater and K. Howie, "User requirements of the "ultimate" online assessment engine," *Computers & Education*, vol. 40, no. 3, pp. 285 – 306, 2003. [Online]. Available: https://doi.org/10.1016/S0360-1315(02)00132-X

[21] J. Spacco *et al.*, "Experiences with Marmoset," Univ. Maryland, Tech. Rep., 2006.

[22] S. Strickroth *et al.*, "ProFormA: An XML-based exchange format for programming tasks," *e-learning & educaton (eleed)*, vol. 11, no. 1, 2015. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0009-5-41389

[23] M. Striewe, "An architecture for modular grading and feedback generation for complex exercises," *Science of Computer Programming*, vol. 129, pp. 35–47, 2016, special issue on eLearning Software Architectures. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642316300260

[24] A. Zeller, "Making students read and review code," in *Proc. 5th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE'00)*. ACM, 2000, pp. 89–92. [Online]. Available: http://doi.acm.org/10.1145/343048.343090