EAI Endorsed Transactions
on Serious Games
Research Article EAI.EU

Toward reusable game technologies: assessing the usability of the RAGE component-based architecture framework

Wim van der Vegt[1,*], Kiavash Bahreini[1], Enkhbold Nyamsuren[1] and Wim Westera[1]

[1]Open University of the Netherlands, Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands

{wim.vandervegt;kiavash.bahreini,enkhbold.nyamsuren,wim.westera}@ou.nl

Abstract

This paper investigates the usability of the RAGE component-based software architecture (RCSA). This architecture was designed to support serious game development by enabling cross-platform reuse of game software components. While the architecture has been technically validated elsewhere, this paper studies the perceived usefulness and ease of use of the architecture in practice. An extensive questionnaire based on the Technology Acceptance Model (TAM) was administered to 23 software and game developers that have been creating RCSA-compliant game components or integrating these in actual serious games. The results show that developers are generally positive about the usability of the architecture and that the architecture helps them to do a better job in less time. It turns out that developers effectively use all communication modes that are offered by the architecture, most frequently those based on the component´s APIs and the bridge pattern. Some issues were reported, but could be easily addressed. Most developers reported that they have well understood the effectiveness of the architecture and indicated to keep using the architecture in future projects. The outcomes of this study show that the architecture opens up new opportunities to the cross-platform reuse of advanced game functionalities in serious game projects, to reduce production efforts and to advance the domain of serious games at large.

Keywords: serious games, software components, game development, reuse, cross-platform, portability, game engines.

Received on 05 December 2018, accepted on 28 May 2019, published on 11 July 2019

doi: 10.4108/eai.11-7-2019.159527

[1]Corresponding author. Email:wim.vandervegt@ou.nl

1. Introduction

Although the potential of games for teaching and training has been widely recognised, their uptake in schools and business has been quite limited [1, 2]. The serious game industry displays many features of an emerging, immature branch of business, being scattered over a large number of small independent studios, displaying weak interconnectedness, limited knowledge exchange, and absence of harmonising standards [3]. Notably, progress is hampered by the wide variety of programming languages, game development systems and delivery platforms that are being used, all of which go with specific technical constraints and incompatibilities that pose severe barriers to growth. Moreover, access to emerging media technologies that could be easily incorporated in serious game projects, such as novel adaptation algorithms, artificial intelligence kernels, or natural language processing methods, is limited, while the alternative of in-company development of such technologies is not feasible, either because of required investments or because of lacking know-how.

This paper presents the evaluation results of the RAGE component-based software architecture (RCSA), which was designed to accommodate the development and reuse of advanced software components offering pedagogically relevant functionalities for serious games [4,5]. The RCSA was developed by the RAGE project (rageproject.eu), which is a leading serious gaming research project funded by the Horizon 2020 Programme of the European Commission. RAGE focuses on the development of advanced software

components that can be easily reused and integrated in serious game projects across a wide variety of prevailing technology platforms. To this end, the RCSA provides the technical framework that overcomes many issues of incompatibility and non-portability across different technical environments. Software components based on RCSA would thus greatly amplify the opportunities of serious game developers to efficiently enhance their games with reusable software. Although the RCSA was extensively tested and technically validated with a series of proof cases [4, 5], its usability in professional practice has not yet been studied. This paper presents the evaluation study of the RCSA with respect to technical usability, that was carried out among 18 component developers and 5 game developers, respectively, all involved in RAGE. In addition, detailed data is collected about the usage of specific technical elements of the RCSA.

The research questions investigated are 1) to what extent does the RCSA simplify creation and delivery of components, 2) to what extent does the RCSA simplify reuse of 3rd party components, 3) are there any specific factors preventing acceptance of the RCSA, and 4) to what extent are individual functionalities of the RCSA being used. The first question has the component developers as target group, while the second question targets the component users (e.g. game developers). The third question aims to investigate if game developers experience any trust issues using RCSA based components or other 3rd party code. The final question focuses on the usage of RCSA features.

First, we will briefly introduce the RCSA. Then, we will detail the research method and instruments used. Finally, we will present and discuss the outcomes.

2. The RAGE component-based software architecture (RCSA)

The RCSA was devised to accommodate the development of software components that can be easily reused and integrated in serious game projects across a wide variety of prevailing technology platforms. An initial set of state-of-the-art RCSA-based components can be accessed through the RAGE marketplace portal at gamecomponents.eu. The components offer a variety of functionalities ranging from learning analytics, adaptation and personalisation, to language-based sentiment analysis, emotion recognition, social gamification and affective computing, i.e. functionality targeting serious games. The RCSA [4, 5] distinguishes between server-side components and client-side components. While remote communications of server-side components with centralised applications can be easily achieved with web services using the HTTP-protocol (e.g., REST), which offers platform-independence and interoperability among heterogeneous technologies, client-side components need to be integrated into client-machine applications (viz. game engines), which is often problematic. Client-side components should be 1) highly portable, 2) should allow easy integration without interfering with game code, 3) consequently, should not directly access the game´s user interface, and 4) should not access or make assumptions about the underlying operating system. To this end, the RCSA was designed by relying on a limited set of well-established coding practices and software patterns (API, Bridge, Publish/Subscribe and Web Services) aimed at the abstraction of operations. Communications between component code and game code is accommodated by five different communication modes, the usage of which will be investigated in this study [6]. First, games can use the component's API for direct access to the component's core functionality. Second, the bridge software pattern is platform-dependent code implementing one or more interfaces that allow a component to invoke game engine code without having knowledge about the game's implementation details or making an assumption about the underlying operating system. This also makes RCSA components very well suited for performing unit testing. Third, broadcast messaging (Publish/Subscribe) supports a 1-N type of communication, for instance the game engine sending player performance data, which then could be received by multiple components. Also, a component could send broadcast messages to the game engine and other components. Fourth, the Bridge can also be used for web service calls to remote services. Fifth, partly based on the previous modes, component-to-component communication would be an additional mode.

Proofs of concept of the RCSA have been established for C#, C++, Java and JavaScript/TypeScript, which are among the predominant programming languages used game development [6]. Also, RCSA-compliant components have been successfully integrated in multiple game engines, such as Unity3D [23], MonoGame [24], Cocos2D [25] and Xamarin [26], and deployed at the most important desktop and mobile platforms [5]. Although these proofs of concepts have demonstrated the effectiveness of the RCSA, an important question remains: how usable is the RCSA in practice, when used by technology developers creating RCSA-compliant components on the one hand, and game developers wanting to reuse these components in their serious games projects on the other hand.

Although the RCSA and its coding boundaries with regards to game and operating system itself might be seen as a composite game software pattern its main purpose differs from the patterns described in [21] as those are targeting to improve game coding structure or readability where the RCSA is more a nonobtrusive delivery format for 3rd party code and therefor has more in common with software packages like NuGet packages [22]. Preliminary research showed RCSA components can automatically be converted to multi-platform NuGet packages. However unlike the NuGet packages which basically delivers libraries with potentially full access to the game and underlying operating system, the RCSA's boundaries prevent this kind of direct access by design, therefor leaving important decisions about for example where to store data to the game developer.

The same coding boundaries also ensure that RCSA components can be easily tested with agile unit testing techniques, thus improving testability and quality.

3. Method

The study was carried out with two extensive questionnaires that were administered in January 2018 to 18 component developers and five game developers (component users), respectively, involved in the RAGE project. Both groups are users of the RCSA, be it from different perspectives: component developers need to accept the RCSA to build upon, while game developers need to accept RCSA based components and the integration methodology the RCSA provides.

3.1. Target groups

The pool of potential participants familiar with the architecture was necessarily restricted to individuals within the RAGE project. The 18 component developers in the RAGE project were employees at research institutes from different European countries. The five game developers were professionals from the four game studios that were part of the RAGE consortium. In both groups, the age distribution is bimodal, revealing two peaks, one typically under 25 years and one around 40 years, respectively.

3.2. RCSA Components

The study relies on participants´ operational experiences, either as a developer or as a user, with one or more of up to 30 initial software components developed by RAGE. The quality and nature of the components' pedagogical functionalities are expressly excluded from current evaluation, as these are reported in separate studies. Now, the focus is on the usability of the architecture in the practices of software development and game development. Usability issues might particularly surface for client-side RCSA components, as they are inherently bound to the abstraction layers, e.g. by using the bridge pattern.

Instructions and support to component developers and game developers were provided through manuals, workshops and component code reviews. Component developers were supported with downloadable Visual Studio project templates for both C# and TypeScript (a superset of JavaScript including static typing). Most of the (client-side) components are written in C# and benefit from portable assemblies that are used across Visual Studio, Xamarin as well as the Unity3D game development platform. C# based project templates have been made available, including a regular (.NET 3.5) project and a portable assembly counterpart using the same source code. Both projects preserve portability by using a common subset of the two .NET framework versions in order to compile. Also, code snippets for implementing various bridge interfaces were made available.

3.3 Games

To assess the functioning of components in real games with real end-users, the four game studios in RAGE created seven component-based serious games of which the majority was created using Unity3D. The games focus on various social and entrepreneurial skills and address diverse target groups including school and university students, sports volunteers, policemen and corporate candidates. Overall, over 1500 participants in total were involved in the game pilots. Details about the game pilots and their evaluations can be found in [27, 28].

3.4 Questionnaires

We opted for questionnaires rather than interviews to avoid 1) any influences of interviewers and 2) potential issues resulting from (spoken) language barriers, given the various nationalities involved. Because of the two different target groups, two separate questionnaires were developed, both with a similar setup and structure, but with slightly different questions in some sections. The questionnaires were based on the Technology Acceptance Model (TAM) [7, 8], which was designed to collect information on perceived usefulness and ease of use, both being indicators of technology acceptance and usability. TAM was preferred to USE (Usefulness, Satisfaction, and Ease of use) [9], TTF (Task-Technology Fit) [10] and SUS (System Usability Scale) [11]. The USE and SUS instruments were discarded as they are more focused on the (graphical) user interfaces and associated end-user experiences and are difficult to apply to software coding and architectures. Task-Technology Fit was discarded, because of the lack of a suitable profile and the efforts required to create a new profile and validate it. The TAM-based questionnaire uses six items for each scale; topics are briefly indicated in Table 1.

Table 1. Topics covered by the TAM-based questionnaire for RCSA usability.

	Perceived usefulness	Ease of use
1	Faster task accomplishment	Easy to learn
2	Enhanced job performance	Easy to control
3	Improved productivity	Clear and understandable
4	Enhanced effectiveness	Flexible to use
5	Makes jobs easier	Easy to become skilful
6	Usefulness in job	Easy to use

For the TAM questions we used 'RAGE architecture' as subject, except for the first question on perceived usefulness in the component developers' questionnaire where we expressly used 'the RAGE architecture when creating reusable components' specifying the task more explicitly.

The 7 point Likert scales used the following labels for the perceived usefulness questions: 'extremely unlikely', 2, 3, 4, 5, 6 and 'extremely likely', respectively.

In addition to TAM, sections were included to establish 1) programming experience self-estimation [12] in the most relevant programming languages, 2) the usage of the architectural features, interfaces and communication modes, including required efforts and restrictions encountered, and 3) the architectural elements that were actually implemented or used. For game developers, an additional section was added to determine their attitude towards including third party software in their projects and infrastructure. Although the evaluation is primarily addressing the technical dimensions of the architecture, acceptance could be hindered by trust issues regarding the use of third party code and its origin. All score items used a 7-points Likert scale. Basic demographic data was limited to name, age, company, programming languages and development environments used, and the components or games developed. The questionnaires comprise 53 architecture-related questions, supplemented with 17 open-ended questions allowing for comments. All invited participants completed the questionnaire, possibly as a result of the shared commitment of being part of the RAGE project, be it not without the need for sending reminders.

For each of the two questionnaire versions, we have checked the reliability of the two TAM scales. The perceived usefulness scale shows excellent internal consistency (Cronbach´s alpha: 0.96 and 0.97, respectively), the perceived ease of use scale shows good internal consistency (Cronbach´s alpha: 0.88 and 0.84, respectively) [14, 15].

3.5. Procedure

The questionnaires were administered using Google Forms. The component developer version was pre-tested with one component developer to check for completion time (30-45 minutes) and to test for the clarity of the questions. As the game developer version was similar in length and design no further tests were undertaken. RAGE work package leaders were asked to distribute the questionnaire amongst the software developers that had sufficient hands-on experience or knowledge about the architecture. Reminders where send to increase the response rate. An informed consent was administered as part of the online questionnaire. All collected data were anonymised and handled confidentially, in accordance with RAGE policies to comply with research ethics regulations. Quantitative data from the Likert scales were all normalised to the 0-1 range before further statistical processing.

4. Results

The overall number of participants, in particular the number of game developers was small, because only a small number of individuals within the RAGE project would have sufficient practical experience with the architecture. The data from the component developers is more informative and representative than the data from the game developers, because of the small sample size of the latter group (five respondents). Although the small sample of game developers provided some potentially useful preliminary insights, elaborate statistical processing or direct comparison with the data from component developers was not opportune.

4.1. Self-assessment of software skills

The results of the self-assessed programming skills for both component developers and game developers display relatively high overall scores, typically well above 0.6, except for TypeScript. The skills deficiency in Typescript may be ascribed to the fact that it is the most recently launched programming language, extending JavaScript. Java expertise is rated high among component developers (0.77). This may be attributed to the development of high performance server-based web-services by the component developers, an area where Java is still a popular choice [13]. Overall, the RAGE developers involved can be qualified as (highly) experienced.

4.2. Results from component developers

Responses

From 18 component developers, five only worked on server-side components and skipped the TAM questions, which were mainly referring to the client-side architectural elements. They were then excluded from the TAM analysis but remained included in the remaining functionality usage.

Software communication patterns used

While the RCSA accommodates a variety of software communications modes [4, 5], component developers are quite selective (cf. figure 1). In the RCSA communication from the game to the component is covered by accessing the component's API. The reverse, communication from a component to the game uses, web services, which are used for addressing any remote server, also make use of RCSA's bridge interface. Broadcasting is used to inform any listening service in the system.

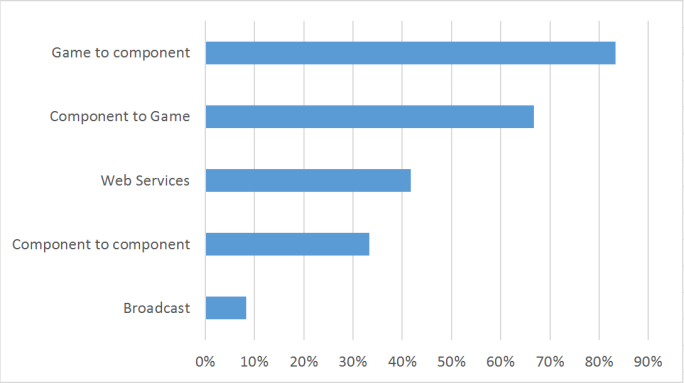

Fig. 1. Usage of software communication modes in client-side components.

Figure 1 shows that game to component communication through component's API is most abundantly used. Communication in the reversed direction, that is, the component using an interface from the Bridge in order to gain access to the game or operating system functionality (such as saving and loading data), is also frequently used. Using this same mechanism to gain access to web-services was less used. Mutual communications between components were not much used as most components work independently from each other. Publish/subscribe broadcasting was the least popular communication mode. In sum, most RCSA communication modes are being used in the components, most frequently the ones using the components' APIs and the bridge interfaces.

Reported issues and comments

A comment was made about the risks of using files with the textual data format. This may cause UTF encoding issues when loading XML files. The .NET framework works internally with UTF-16 encoded strings [16], and as such it defaults to UTF-16 encoded XML files. Forcing UTF-8 output as used by some web-services requires some additional coding [18]. Binary data is currently only supported in C# by base64 encoding it [17].

One component developer highly appreciated using the bridge for platform dependent functionality but expressed concerns about the obligation for game developers to implement interfaces for the bridge, because they are reluctant to implement code that is not strictly related to their games. Their proposed solution was to include a ready to use bridge class with the component. Although the concern is legitimate, the proposed solution of adding a bridge actually undermines platform independence. Pointing towards the available code snippets providing reference was inspired by one of the leading game platforms, Unity3D, not supporting modern async/await type of method invocations during RCSA design. Only recently Unity3D has started supporting a more up-to-date .NET framework [19]. The RCSA easily supports this new framework with its portable assembly counterpart. Preliminary research also indicated that .Net Core 2.0 and newer are easy to add using the same shared sources mechanism as used for creating the portable assemblies. Besides the current interface, which does not enforce async calls, leaves the actual sync/async choice to the game programmer.

During component creation, one-third of the component developers reported having requested (and received) some support for the architecture team. Most component developers indicated that they would use the RCSA in future projects.

Architecture usability

Figure 2 and figure 3 show the normalised mean scores from the component developers on six items of the perceived usefulness scale and ease of use scale, respectively.

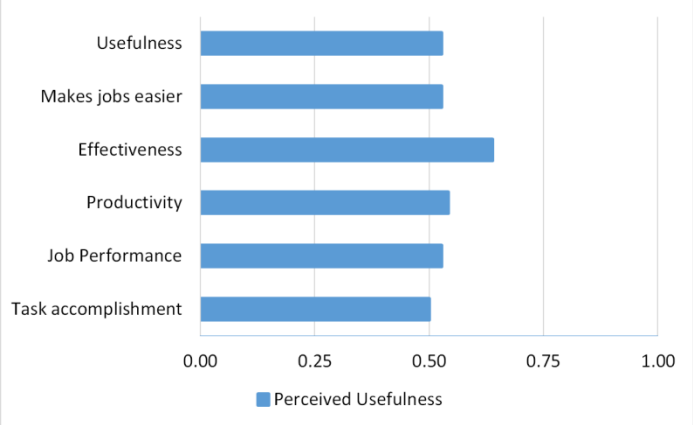

Fig. 2. Perceived usefulness according to component developers (normalized scores).

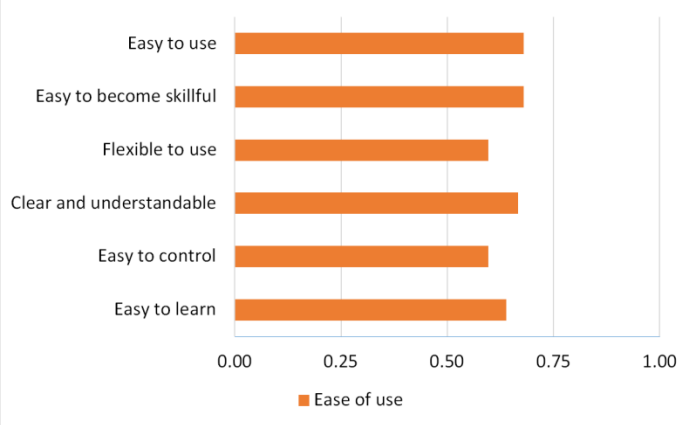

Fig. 3. Ease of use according to component developers (normalized scores).

Perceived usefulness has a mean score of 0.55 (standard error 0.05), whereas ease of use received a mean score of 0.64 (standard error 0.05), both representing values well above average. Actually, all separate items received scores above 0.50. Notably, component developers indicate that the RCSA makes tasks easier, helps to accomplish tasks more quickly and efficiently, and thereby improves job performance and productivity (perceived usefulness). Also, the RCSA is easy to understand, provides a flexible way to create components, which can be easily applied.

4.3. Results from game developers

Responses

Five game developers, representing each of the four game studios participating in RAGE, responded to the questionnaire.

Components and game projects

Game developers reported being involved in the development of all seven RAGE games. Six out of seven games were coded using C#. One game developer used C++ as the coding language. Five games used the Unity3D development environment, one game used Cocos-2D, and one of the studios used its own platform. The average number of RAGE components being incorporated in each game is seven, evenly divided among server-side components and client-side components.

Software communication patterns used

In this section, we report how the game developers relied on the RCSA features. We pay little attention to component's API since it is in-dependent of the RCSA. Four game developers used one or more interfaces implementing the bridge pattern that allows a communication from the component to the game. For example, the interface for storing and retrieving local data was used by four developers, and three developers used the logging facility.

A component-to-component communication was used by one developer. There are two options for such communication. One component can directly call the other one if the former implements the latter's API interface. For such cases, the RCSA provides a component manager that offers automatic registry and lookup of available components. Alternatively, if the components are unaware of each other's APIs then the game developer can implement a mediating wrapper code that makes use of the component manager.

Other RCSA features were used to various degrees. Three game developers used RCSA's web-service interface to send a request to remote services. Functionality for handling run-time and default settings to be compiled into the game was used once, which indicates that most game developers prefer to supply the settings by game code. The only communication pattern that was not used so far by the game developers is publish-subscribe for broadcasting messages.

Reported issues and comments.

Scarce issues were reported. An issue was raised about the voice synthesis component, which requires direct access to the underlying operating system. This should be solved by the component developer implementing a simple, generic interface for this. For example, in the facial emotion recognition component, direct access to a webcam was replaced by a simple yet more versatile API that requires the game developer to submit frames from a camera or other sources (e.g. stills or pre-recorded video) to the component, thereby ensuring platform independence.

One game developer needed to port client-side C# components to C++ programming language. This requires some effort, but is doable as such, since the RCSA was proven to be valid for C++ [4].

With respect to adoption barriers, some of the game developers expressed their concerns about the academic origin of the components pedagogical content, while on the other hand being totally confident with using third-party code. We hypothesize that components from academia that are often open-source and do not provide a quality guarantee in the license agreement are perceived to have lower quality than their commercial counterparts. For this reason, architectures such as the RCSA may be highly beneficial for a wider adoption of academic components since a conformance to such architecture guarantees a level of standardization and quality control.

One game developer reported compilation and deployment issues for their game in a highly secured corporate environment, prohibiting for example outwards web-service calls. Although this environmental behaviour is not caused by the RCSA itself, it is a potential issue for those RCSA-based components that expect a web-service to be accessible. Four game developers expressed a preference for 'traditional' direct integration of functionality, which seems to suggest some aversion to the RCSA. Still, four of the game developers reported that they would keep using RCSA-compliant components outside RAGE as well, while the fifth developer said to be using it conditional to the component offering core functionality needed in the game.

Architecture usability

The game developers TAM scores for both scales are slightly above average: 0.53 (standard error 0.12) for perceived usefulness and 0.58 (standard error 0.08) for ease of use. Given the standard errors, the RCSA is to be qualified as moderately usable. However, the scores were negatively biased by one of the game developers assigning systematically much lower scores as compared to the other developers. Removing this outlier (scoring 0.19 and 0.33 on the two scales) would produce perceived usefulness of 0.62 (standard error 0.10) and ease of use of 0.64 (standard error 0.06). This means that most game developers are positive about the RCSA's usability.

5. Discussion and conclusion

The outcomes of this usability study can be summarised as follows: developers of software components are generally positive about the usability of the RCSA and indicate that the RCSA helps them to do a better job in less time. All communication patterns offered by the RCSA have been effectively used in the components under consideration, most frequently the communications patterns based on the component´s APIs and the bridge pattern. Some issues were reported, but most of them could be covered without affecting the portability principles of the RCSA. Most component developers reported that they have well understood the effectiveness of the architecture and indicated to use the RCSA in future projects.

Game developers, acting as the users of the software components, are likewise positive about the RCSA. Although the sample was small, unambiguous responses indicated that

most game developers qualify the RCSA-compliant components as useful and easy to integrate. Also, game developers used most of the communication patterns provided by the RCSA. Functionality for handling run-time and default settings to be compiled into the game was scarcely used, however. It seems that most game developers prefer to supply the settings through the game code. The tendency to stay in full control of their game application may pose a barrier to adoption of the RCSA. In [20] it was established that game studios are generally open and positive toward new technologies, but they are critical as such. They look for added value in terms of better games or commercial potential, but at the same time, they are afraid of complex and cumbersome implementation, which is understandable as their games should run smoothly without bugs or crashes. This exploitation requirement inevitably goes with some reluctance toward innovation: game developers first want to see the evidence before adopting something new. Some ambiguity was also shown by game developers raising concerns about software from academic origin, while at the same time they claimed to be confident with using third-party code.

Nevertheless, most game developers in the sample indicated that they would keep using the RCSA in future.

Although the respondents where RAGE project participants and this might have led to a bias in the TAM scores, the absence of high TAM scores and the presence of critical comments indicates that the respondents completed the questionnaire from a professional viewpoint and thus gives confidence the TAM scores are not biased.

Overall, this qualitative study has confirmed the practicability of the RCSA by tapping on the practical experiences of targeted component developers and game developers using the RCSA. The positive outcomes of this study open up new opportunities to flexibly incorporate advanced game functionalities in serious game projects, reduce production efforts and advance the domain of serious games at large. The outlook would be a flourishing market of advanced and affordable serious games that would contribute in purposeful ways to addressing societal problems in the fields of, e.g., media literacy, education and training, cultural heritage and social inclusion.

Future work will include monitoring acceptance by component and game developers outside RAGE and a closer investigation of the not RCSA architecture related questions on acceptance by game developers of foreign code (and especially code with an academic origin) but that might lower acceptation of components created according to the RCSA.

Acknowledgements.
This work has been partially funded by the EC H2020 project RAGE (Realising an Applied Gaming Eco-System); http://www.rageproject.eu/; Grant agreement No 644187.

References

[1] Carl Abt: Serious games. Viking Press, New York (1970).
[2] T.M. Connolly, E.A. Boyle, E. MacArthur, T. Hainey and J.M. Boyle: A systematic literature review of empirical evidence on computer games and serious games. In: Computers & Education 59 (2), 661–686. DOI: 10.1016/j.compedu.2012.03.004 (2013).
[3] Stewart, J., Bleumers, L., Van Looy, J., Mariën, I., All, A., Schurmans, D., Willaert, K., De Grove, F., Jacobs, A., and Misuraca, G.: The Potential of Digital Games for Empowerment and Social Inclusion of Groups at Risk of Social and Economic Exclusion: Evidence and Opportunity for Policy. Centeno, C. (Ed.), Joint Research Centre, European Commission. (2013).
[4] G.W. van der Vegt, W. Westera, E. Nyamsuren, A. Georgiev and I. Martinez Ortiz: RAGE architecture for reusable serious gaming technology components. In: International Journal of Computer Games Technology. Article ID 5680526. DOI: 10.1155/2016/5680526. (2016).
[5] W. van der Vegt, E. Nyamsuren and W. Westera: RAGE Reusable Game Software Components and Their Integration into Serious Game Engines. In: Proceedings of the 15th International Conference on Software Reuse (ICSR 2016). Springer International Publishing, Basel, 165-180 (2016).
[6] RedMonk: The RedMonk programming languages rankings: January 2015, http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/, last accessed 2018/05/15.
[7] Davis, F. D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. In: MIS Quarterly 13(3): 319-340 (1989).
[8] Marangunić, N. and Granić A.: Technology acceptance model: a literature review from 1986 to 2013. Universal Access in the Information Society 14(1): 81-95 (2015).
[9] Lund, A. M.: Measuring Usability with the USE Questionnaire. STC Usability SIG Newsletter (2001).
[10] Furneaux, B.: Task-Technology Fit Theory: A Survey and Synopsis of the Literature. In: Information Systems Theory: Explaining and Predicting Our Digital Society, Vol. 1. Y. K. Dwivedi, M. R. Wade and S. L. Schneberger. New York, NY, Springer New York: 87-106. (2012).
[11] Bangor, A., et al.: Determining what individual SUS scores mean: adding an adjective rating scale. J. Usability Studies 4(3): 114-123 (2009).
[12] Siegmund, J., et al.: Measuring and modeling programming experience. In: Empirical Software Engineering 19(5): 1299-1334 (2014).
[13] W^{3}Techs: Usage statistics and market share of Java for websites, https://w3techs.com/technologies/details/pl-java/all/all/, last accessed 2018/05/15.
[14] Cronbach, L. J.: Coefficient alpha and the internal structure of tests. In: Psychometrika, 16, 297-334 (28,307 citations in Google Scholar as of 4/1/2016). (1951).
[15] Tavakol, M. and Dennick R.: Making sense of Cronbach's alpha. Int J Med Educ 2: 53-55. (2011).
[16] Microsoft: Character Encoding in .NET, https://docs.microsoft.com/en-us/dotnet/standard/base-types/character-encoding, (2017).
[17] Microsoft: Convert Methods, https://msdn.microsoft.com/en-us/library/system.convert_methods(v=vs.110).aspx, last accessed 2018/05/15.
[18] Lacovara, R.: How To Create XML in C# with UTF-8 Encoding, (2011),

http://rlacovara.blogspot.nl/2011/02/how-to-create-xml-in-c-with-utf-8.html, last accessed 2018/05/15.
[19] Unity: Unity Blog: Unity 2018.1, https://blogs.unity3d.com/2018/05/02/2018-1-is-now-available/, last accessed 2018/05/15.
[20] Saveski, G., Westera, W., Yuan, L., Hollins, P., Fernández Manjón, B., Moreno Ger, P. and Stefanov, K.: What serious game studios want from ICT research: identifying developers' needs. In: Games and Learning Alliance Conference 2015, Rome (2015).
[21] Nystrom, B.: Game Programming Patterns. Genever Benning (2014).
[22] NuGet Gallery, https://www.nuget.org/, last accessed 2018/09/18.
[23] Unity3D, https://unity3d.com/, last accessed 2018/10/22.
[24] MonoGame, http://www.monogame.net/, last accessed 2018/10/22.
[25] Cocos2D, http://www.cocos2d.org/, last accessed 2018/10/22.
[26] Xamarin, https://visualstudio.microsoft.com/xamarin/, last accessed 2018/10/22
[27] Bazzanella, B., Casagranda, M., Molinari, A., Humphreys, S., Sleightholme, G., Lepoivre, O., ... Kommeren, R. (2018). D5.4 – Pilots quality report round 2. RAGE project. https://research.ou.nl/en/publications/d54-pilots-quality-report-round-2, last accessed June 24, 2019.
[28] Steiner, C., Gaisbachgrabner, K., Nussbaumer, A., Mertens, J., Hemmje, M., Nadolski, R. J., ... Santos, P. A. (2018). D8.4 – Second RAGE Evaluation Report. RAGE project. https://research.ou.nl/en/publications/d84-second-rage-evaluation-report, last accessed June 24, 2019.