Digitization is no longer a choice, but a necessity. In this fast-paced society, keeping up with customer expectations can be difficult. It is no longer enough to build the right software; you must also deliver a superior customer experience quickly and with high quality to survive in the digital world.

How do organizations realize Sustainable Software, in order to face these challenges that come with digitization? This article illustrates that the answer to this question can be found in the combination of using so-called low-code platforms and Automated Software Quality Monitoring.

Low-code and the road to Sustainable Software

Bryan de Vries MSc MBA
is a partner at Omnext B.V.
bryan.de.vries@omnext.com

Dennis Stam MSc CISA EMITA
is a senior manager at KPMG Advisory.
stam.dennis@kpmg.nl

INTRODUCTION

The world around us is changing rapidly and organizations are facing new trends and challenges every single day. Looise, van Riemsdijk and De Lange [Looi98] argue that one of the key drivers behind these changes is the never-ending flow of technological developments.

One of the changes that is being driven by technological developments is the megatrend of digitization ([Bank16]). The world is becoming more digital every day, and this has an impact on nearly every single organization, large or small, selling services or products. Digitization can bring a lot of benefits for these organizations, but it also introduces management questions that, until recently, did not get a lot of attention ([Coll15], [Bank16]). Digitization is no longer a choice, but a necessity. In our modern, fast-paced society, keeping up with customer expectations can be difficult as their demands change.

> It is no longer enough to just build the right software

So, the playing field has changed, and organizations need to adapt to these trends and changing environments in order to survive. In the 1990s, organizations had to become flexible (both in managerial capabilities as well as responsiveness of the organization) to cope with hypercompetitive environments ([Geus88], [Hirs92], [Volb96], [Lin97], [Scho03]). Today, the major changes that organizations face (as a result of digitization) is an increased dependency on Information Technology (IT) and especially on (custom) software.

To stay competitive and differentiate from their competitors, organizations need to invest heavily in their software application landscape. Software must be delivered faster, be easy to maintain, and perhaps above all, flexible enough to 'grow with the organization' and adapt rapidly to any possible changes. It is no longer enough to just build the right software; you must also deliver superior customer experience quickly and with high quality in order to survive in the digital world. To be able to deliver these changes quickly and in a controlled way, organizations must aim to realize what we would like to call **Sustainable Software**.

Sustainable Software definition

Sustainable Software is software that is flexible enough to adapt to any required changes instantly.

That leads to the question: "How do organizations realize Sustainable Software in order to face the challenges that come with digitization?" This article aims to illustrate that the answer to this question can be found in the combination of using **low-code platforms** and **automated software quality monitoring**.

WHAT IS SUSTAINABLE SOFTWARE?

Today's organizations are heavily dependent on the quality of their software. But what characteristics should software have to enable the organizations to gain (and keep) their competitive advantages?

In fact, it may be more obvious than it seems: software should have the exact same characteristics and capabilities as the organization that owns or uses it. From core banking systems, one would probably expect a reliable and well-performing basis, whereas a clothing vendor's web front-end may focus more on usability. The digitization trend, however, expects a common trait for nearly every IT-system: flexibility.

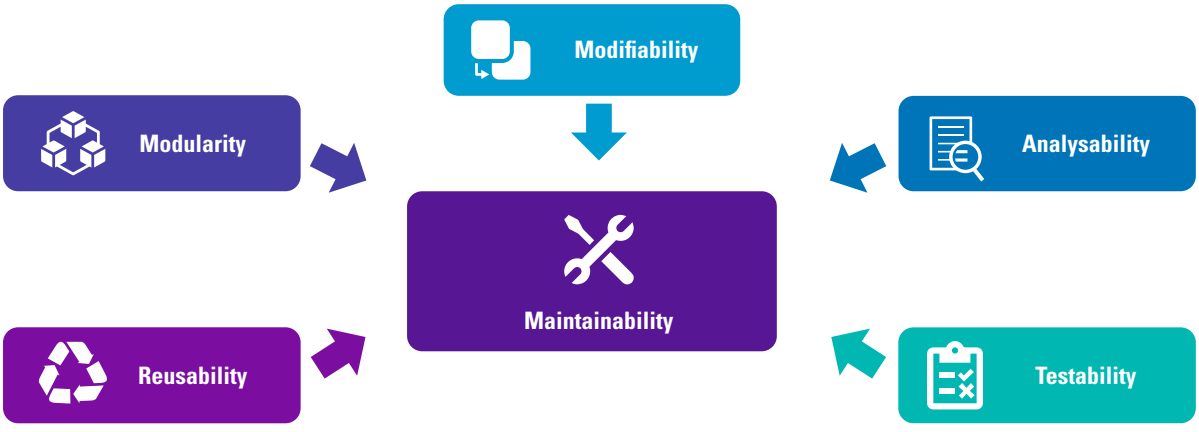
Seacord et al. [Seac03] and Venters et al. [Vent13] define Sustainable Software as software that can evolve and can be modified based on stakeholders' changing requirements in order to make sure it can be used now, as well as in the future.

The Institute of Electrical and Electronics Engineers (IEEE) links software sustainability to software maintainability. According to their definition, software that is easy to maintain can be seen as sustainable. They state that maintainability can be defined as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" ([IEEE90]).

Sunday [Sund89], in line with the IEEE (1990), also links sustainability to maintainability and refers to maintainability as a software characteristic that represents the amount of effort needed to achieve one of four tasks:

1. correction of errors;
2. addition of features;
3. deletion of capabilities;
4. adaption or modification.

Figure 1. The ISO/IEC 25010 model for software maintainability.



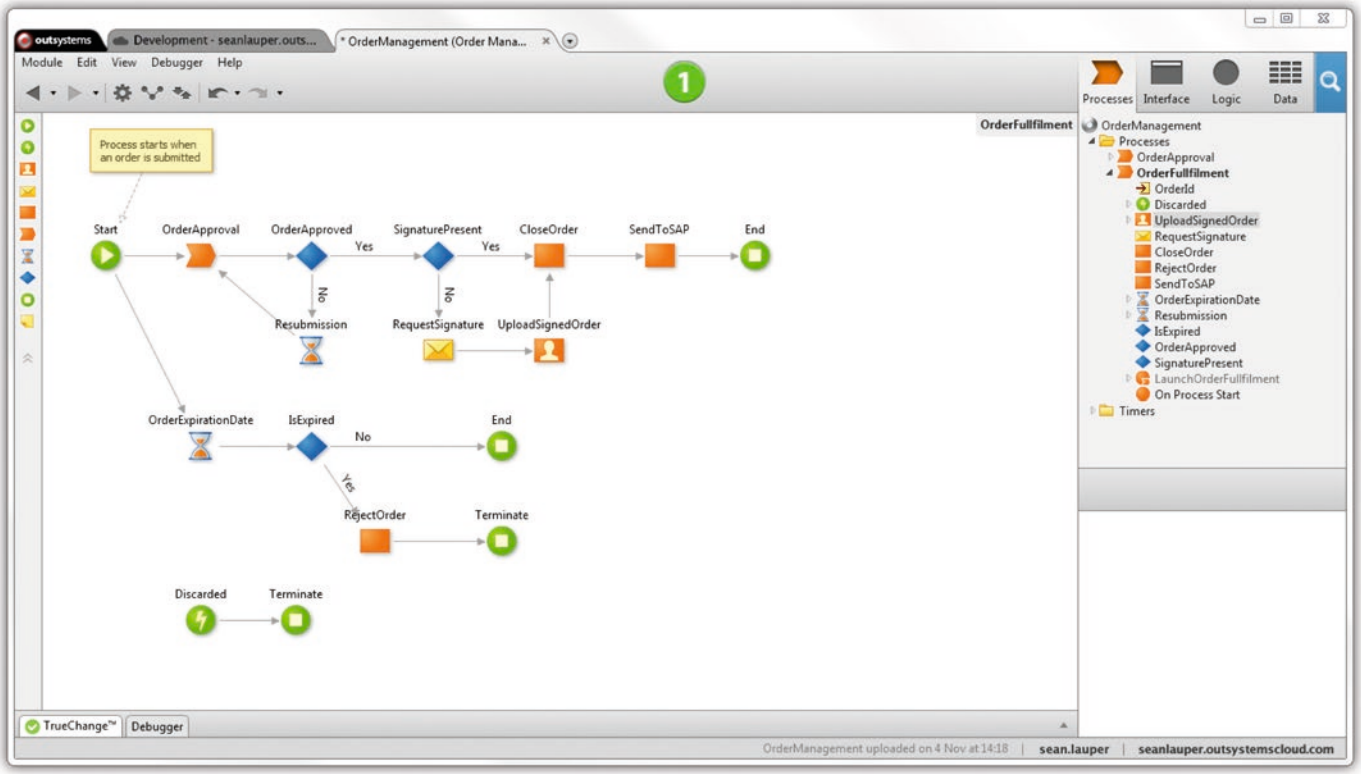


Figure 2. Example screen from OutSystems platform: visual modeling.

The ISO/IEC 25010 [ISO14] standard, an internationally accepted standard for software quality, describes maintainability using five different characteristics:

1. modifiability;
2. analysability;
3. testability;
4. reusability;
5. modularity.

According to ISO/IEC 25010, an application should be taking all these characteristics into account in order to be maintainable (see the box on page 52).

The one thing all definitions of software sustainability have in common is that they all refer to flexibility or adaptability of software when defining it ([Tate06], [Seac03], [Cale13], [Vent03]). We believe this answers the question on what Sustainable Software is: Sustainable Software is software that is flexible enough to adapt to any required changes instantly.

Enter: Low-code

Low-code is a term coined in 2014 by Forrester Research [Rich14], which has quickly gained popularity in the past few years. The low-code platforms, however, first emerged in the early 2010s as a response to the ever-faster pace of developing business applications and the evolution of programming languages ([KPMG19]). These platforms provide system developers with an environment through which software can be created through graphical user interfaces and configuration to "model" the envisioned application instead of traditional computer programming. This reduces the amount of traditional "hand-coding" – hence the term "low-code" – enabling accelerated delivery of applications. The low-code platform in turn generates the required code automatically from these models.

Low-code platforms, such as OutSystems, Mendix, or Appian enable rapid delivery of business applications with a minimum of development, as well as minimal upfront investment in setup, training, and deployment. They are offered as platform as a service (PaaS) and are intended for developing and delivering enterprise web and mobile applications, which run in the cloud, on-premises, or in hybrid environments.

These platforms accelerate the digital transformation by enabling rapid application development. Some platforms, such as OutSystems, even offer full-stack visual modeling capabilities and one-click deployments, which further accelerate development and reduce the need for highly-skilled software developers. By offering full application lifecycle management, from design to deployment and maintenance, low-code platforms empower

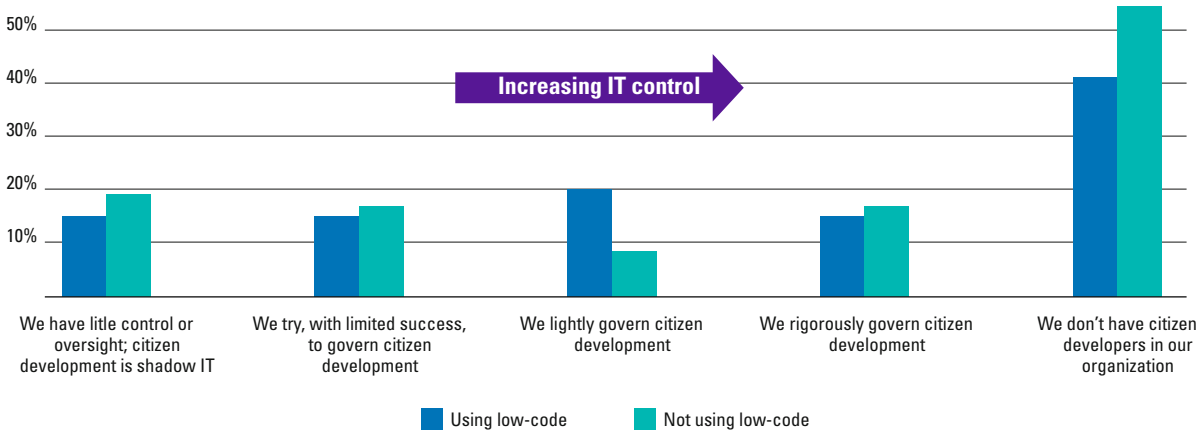


Figure 3. Use and governance of citizen developers ([OUTS18]).

organizations to focus on customer experience and innovation, rather than on application development and management.

Visual modeling promotes better understanding of requirements, cleaner designs, and more maintainable systems. The resulting models are abstractions that portray the essentials of the complex problem the software attempts to solve, with nonessential details filtered out, making the problem easier to understand. Models help us organize, visualize, understand, and create complex things. They are used to help us meet the challenges of developing software today and in the future. ([Quato2]).

But even in a low-code environment, maintaining a high quality, Sustainable Software is key in the era of digitization.

The use of low-code platforms can help to reach the goal of developing Sustainable Software

Citizen developers

Gartner [GART17] defined: "A citizen developer is a non-professional developer who builds applications for use by other people. Although they do not report directly to IT, they use tools such as low-code platforms that are sanctioned by IT. This oversight by IT distinguishes citizen development from so-called 'shadow-IT', which takes place without the knowledge or control of IT." The introduction of low-code platforms works as a catalyst for citizen development: the ease of use and abstraction from traditional programming that these platforms offer, almost invites tech-savvy business users to start building their own applications. With non-professional developers added to the equation, the quality of delivered applications may be in jeopardy.

The research by Gartner as well as a recent survey by OutSystems [OUTS18] show that approximately 50% of respondents claims not to have any citizen developers within their organization. Under organizations that already apply low-code platforms, 60% of all respondents confirmed citizen development. The actual share of citizen developers in the IT landscape might be even higher as it is questionable how many of these respondents were even aware of business-led development or subject to wishful thinking.

In the survey conducted by OutSystems, out of the 52% of respondents that claimed citizen development in their organization, 31.8% claimed to have little or no control or oversight in the development efforts and delivered quality of citizen developers, and identified a risk for the organization.

HOW TO KEEP SOFTWARE SUSTAINABLE

The use of low-code platforms can help to reach the goal of developing Sustainable Software. However, even with visual modeling, developing a system remains the work of people and as such, development is rarely perfect and errors can, and usually will, occur. For instance, like duplication of source code on traditional platforms, duplication of models leads to increased maintenance cost.

Similarly, even with visual models it is not very difficult to create "spaghetti code". One might even argue whether low-code development platforms – or fourth generation programming languages in general – don't hide too much from the software engineer, therefore making it easy to oversee options leading to unintended behavior of the application. For example, adding a list of available products to an application is relatively easy to implement using low-code platforms, but how many developers also realize they must put a limit on those views to prevent the application from fetching millions of rows?

Therefore, in short: yes, using low-code platforms can help create more Sustainable Software, but even these technologies do not present us the silver bullet when it comes to software quality as the programmer building the software is just a human.

This leaves us with one final question: how do we ensure continuous sustainability of the (low-code) software?

Hygerth presents possible actions that organizations can undertake to enhance and secure their software sustainability: make use of testing tools, and address the lack of understanding of quality within the organization by creating a "quality culture" ([Hyge16]).

The attentive reader might notice that these actions are dependent on each other. "Give a man a fish, and he will be hungry again tomorrow; teach him to catch a fish, and he will be richer all his life." ([Loane1911]). The same goes for tooling for developers to chase for high-quality software: without the understanding of what high quality means to the organization and what level is to be achieved, a tool is nothing more than a gizmo without purpose.

Tooling

Nowadays a lot of different tools are available to developers. Some tools cover the functional testing of software where others focus more on evaluating and enhancing the quality of the actual coding of an application.

Kyte and Van der Zijden [Kyte15] mention a few of these tools or services that, according to them, can contribute to realizing more sustainable, higher quality software. A few examples are CAST Software, Parasoft, SonarQube, Software Improvement Group and Omnext. All these tools or services aim to, fully automated or not, analyze the application source code and evaluate its quality in terms of maintainability based on industry standards and best practices (such as the ISO/IEC 25010 standard for software quality).

Static analysis tooling uses quality models to (automatically) assess the application source code. Although most of these models are based on the ISO/IEC 25010 standard for software quality and have matured over the years (despite still under constant change due to new insights in software engineering), many of these are tailored for a specific programming platform or technology. The rise of low-code platforms has caused the industry to re-evaluate the way we look at software quality: ideally one would adopt a quality model that fits the low-code platforms but also allows to compare to more 'regular' technologies to make use of the knowledge built up over decades.

As the aim is to create high quality, easy to maintain and easy to adapt software, tooling can be of great help for developers to double check their work in a very time and cost-efficient way and make their lives a bit easier while enhancing the quality in one go.

Quality Model: ORQA

The ORQA (Omnext Risk and Quality Assessment) model assesses application code (or the low-code equivalent) against rules on maintainability, security, performance and reliability, following the definitions from ISO/IEC 25010 standard.

The ORQA model is somewhat different from other well-known quality models as it combines technology-specific industry good practices. Through the years, Omnext gathered specific "do's and don'ts" for each technology and created so-called Best Practice Rulesets. These rulesets are kept up-to-date by an open workgroup of practitioners that periodically discusses good practices.

One thing that makes the ORQA model different from other quality models is that it shows results in relation to the size of the application. Generally, the size of an application can be measured by counting the source lines of code (SLOC). For low-code applications, the number of checks against the Best Practice Ruleset is used to determine size. In other words, the model counts the number of times each Best Practice rule is being checked during the analysis. Despite not being an exact sizing measure, it does correlate with application size: the larger the application, the more individual Best Practice rule checks will be executed.

To make sure the model and the ORQA score are easily understandable and interpretable, they are based on a model that can best be described as the 'Exam Grading Principle'. Violations on the Best Practice Rulesets will decrease the perfect score with a certain number of penalty points. These penalty points are based on two metrics, the Impact Score (the extent to which a found Best Practice violation impacts the behavior of the application in in production) and the Effort Score (the effort and difficulty to resolve or mediate the violation), which are being explained further on in this document. Eventually, the ORQA score can have a value between 0% and 100%. The closer to 0, the higher the risk in your application. The closer to 100, the lower the risk in your application.

Understanding of quality

The use of tooling to address rather technical issues such as overly complex applications is a great way to enhance software sustainability which can be implemented on an organizational level. Moreover, the insight gained through the dashboards and reports generated by this tooling helps the organization (both developers, as well as IT and business managers) to understand the quality of their software and specifically the hidden cost in terms of technical debt – the very same debt that hinders the flexibility of the software.

The key argument against it is that it simply takes quite a lot of time to check and double-check your implementation and make sure that not only the functional requirements are being met, but that also the non-functional ones, such as maintainability and adaptability, are given appropriate attention. As changes are often expected to be implemented fast, this is time that some developers just do not have.

This often triggers a discussion within (IT) organizations on what is deemed to be important quality aspects for the business or a specific system. The outcomes of these discussions are not only valuable for developers to determine how they should spend their time and effort, but caters to the entire IT delivery factory. Business analysts are more focused on gathering the requirements for the quality aspects that matter most; architects and designers tailor the system to the right requirements ensuring the system can deliver to the selected quality aspects; testers verify that the system is suitable for usage by the business not only in terms of delivering the right functionality, but also the non-functional aspects.

When it comes to developing Sustainable Software, experience is key

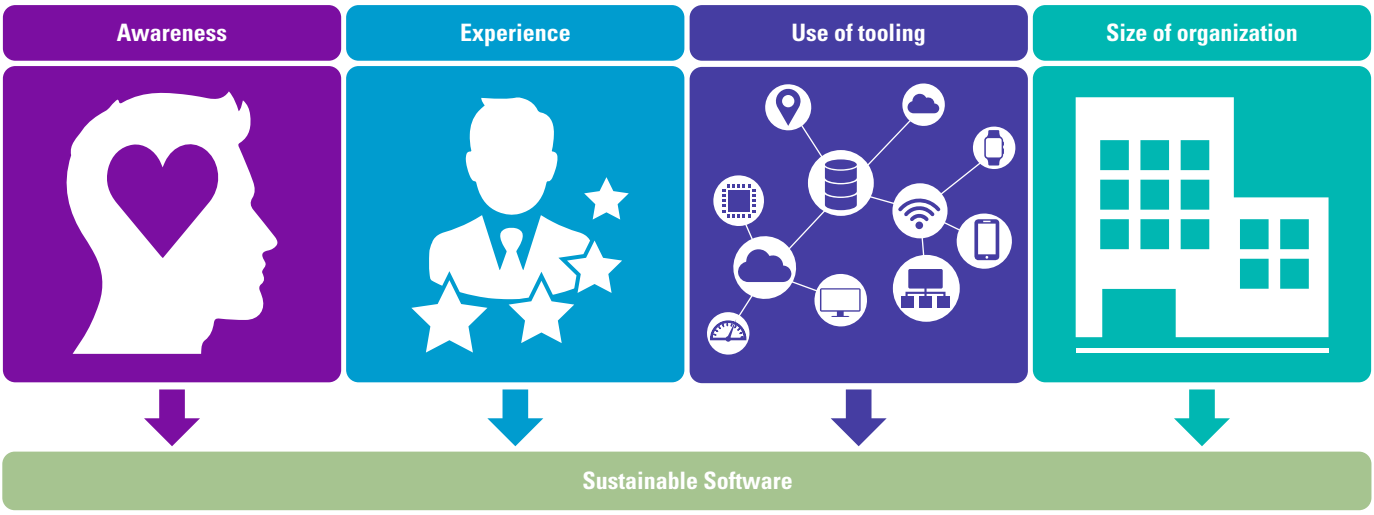


Figure 4. Factors impacting sustainability of software.

SUSTAINABLE SOFTWARE DEVELOPMENT

To realize such a quality culture, people within an organization must be trained and educated in order to overcome a lack of understanding. Furthermore, people do not just need to be educated to be made aware of the effects of software sustainability, but also trained in using certain development or requirement-setting methods that may increase software quality, maintainability and sustainability. Furthermore, people simply need to gain experience; this can be achieved through training as well.

Research [Vrie17] on this topic shows that "quality culture" within an organization is characterized by four aspects: awareness of the importance of software sustainability, experience (with software development, red.), use of tooling and size of the organization. These factors help organizations to realize more Sustainable Software, and are therefore able to adapt to their changing environments. What is interesting though is the fact that the strength of the relationship between the different variables and software sustainability differs.

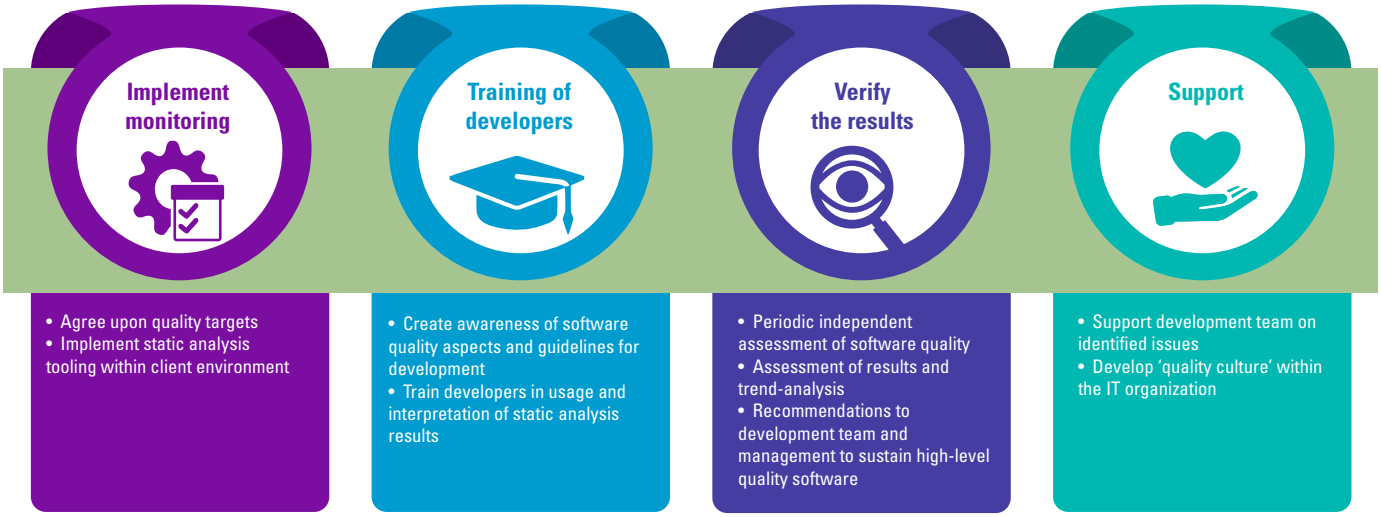


Figure 5. Supporting a quality culture by a QA Walk-along.

The strongest relationship is found between experience and software sustainability. When it comes to developing Sustainable Software, experience is key. Both awareness and the use of tooling show a less strong relationship, but still positive and significant. Based on these results the conclusion is that if organizations want to enhance their software sustainability, they should invest in raising the experience of the people within their organization as this has the highest impact.

The research also uncovered an unexpected fourth correlation between software sustainability and organization size: size has a negative effect on software sustainability. Smaller organizations are more likely to create more Sustainable Software than their larger counterparts.

Low-code platforms support the digital transformation. Nonetheless, even in a low-code environment there is room for human error.

Whereas the last factor is a given, the first three factors can be controlled by the organization. As mentioned earlier, a "quality culture" can be developed by training. Awareness of the importance of software sustainability, experience with software development, and adoption of supportive tooling are all within reach of the organization. If there are no champions in-house to lead the software engineers towards these targets, a QA Walk-along approach by a skilled vendor can kick-start the process.

With a QA Walk-along, a vendor can help the organization set the target quality level, select and implement the right tooling, and train the software developers in the usage of the tooling and interpretation of the results. Furthermore, by periodic reviews of the software quality and analysis of trends in the quality, there is a constant focus (and therefore awareness) on delivering high quality software. It is important that such a walk-along approach is set up in such a way that developers feel "supported" instead of attacked. The vendor needs to be a partner to the organization to support developing the quality culture needed to survive the era of digitization.

CONCLUSION

Digitization is no longer a choice, but a necessity. In our modern, fast-paced society, keeping up with customer expectations can be difficult as their demands shift. It is no longer enough to just build the right software; you must also deliver superior customer experience quickly and with high quality to survive in the digital world.

Organizations face two major challenges today – offering exceptional customer experience and keeping up with the fast-changing technological developments. To deliver quality solutions on time and in line with customer expectations, software quality must be seen as intrinsic to the product and not as an afterthought.

Low-code platforms support the digital transformation by enabling rapid application development, while also promoting better understanding of requirements, cleaner designs, and more maintainable systems. Nonetheless, even in a low-code environment there is room for human error and maintaining high quality. Sustainable Software is still key in this era of digitization.

Research shows that four factors impact the sustainability of software: awareness of the importance of software sustainability, experience with software development, use of tooling, and the size of the organization. Whereas the latter is a given, the first three factors can be controlled by the organization. By means of a QA walk-along, the right toolset can be embedded in the IT development factory; software engineers are trained and gain experience in the usage of the tooling and awareness of good practices on developing Sustainable Software is created. All this is needed to support further growth of the last factor: the size of the organization.

References

[Bank16] Bankewitz, M., Aberg, C., & Teuchert, C. (2016). *Digitalization and Boards of Directors: A New Era of Corporate Governance?* Business and Management Research, 5(2), p 58.

[Cale13] Calero, C, Moraga, M. A. and Bertoa, M. F. *Towards a software product sustainability model*, WSSSPE'1: First workshop on Sustainable Software for science: practice and experiences, SC'13, 17 November 2013, Denver, CO, USA.

[Coll15] Collin, J., Hiekkanen, K., Korhonen, J. J., Halén, M., Itälä, T., & Helenius, M. (2015). *IT leadership in transition - the impact of digitalization on Finnish organizations.*

[GART17] Wong, J. (2017). *Survey Analysis: Citizen Development Is Happening and IT Needs to Be More Engaged.* Gartner. 28 September 2017.

[Geus88] De Geus, A. (1988). *Planning as Learning.* Harvard Business Review, 66 (2), 70F74.

[Hirs92] Hirschorn, L., & Gilmore, T. (1992). *The New Boundaries of the Boundaryless Company.* Harvard Business Review, 4F16.

[Hyge16] Hygerth, H. (2016). *Sustainable Software Engineering: an investigation into the technical sustainability dimension.* Master of Science Thesis, Stockholm, Sweden.

[IEEE90] IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Std 610.12-1990. Dec. 1-84.

[ISO14] ISO/IEC-25000:*2014 Systems and software engineering* – Systems and software Quality Requirements and Evaluation (SQuaRE). (2014). Retrieved from http://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&start=6

[KPMG19] Özüm, A. (2019). *Why low-code is becoming more important.* KPMG. Retrieved from https://home.kpmg/nl/nl/home/social/2019/02/low-code-a-high-level-overview.html

[Kyte15] Kyte, A. & Van der Zijden, S. (2015). *Focus on Application Agility to Deliver Change with Velocity.* Gartner Research Report.

[Lin97] Lin, Z., & Hui, C. (1997). *Adapting to the Changing Environment: A Theoretical Comparison of Decision-Making Proficiency of Lean and Mass Organization Systems.* Computational & Mathematical Organization Theory, 3 (2), 113F142.

[Loane1911] Loane, M. (1911). *The Common Growth.* E. Arnold., Quote Page 139, Longmans, Green & Co, New York.

[Looi98] Looise, J., van Riemsdijk, M., & de Lange, F. (1998). *Company Labour Flexibility in The Netherlands: An Institutional Perspective.* Employee Relations, 5, 461F482.

[OUTS18] Warren, N. (2018). *What is Citizen Development? And, how can you govern Citizen Developers more effectively?* 20 August 2018. Retrieved from https://www.outsystems.com/blog/posts/citizen-developer/

[Quat02] Quatrani, T. (2002). *Visual modeling with rational rose 2002 and UML.* Addison-Wesley Longman Publishing Co., Inc.

[Rich14] Richardson, C., Rymer, J., Cullen, A. and Whittake, D. (2014). *New Development Platforms Emerge for Customer-Facing Applications*, Forrester.

[Scho03] Schoemaker, M. (2003). *Identity in Flexible Organizations: Experiences in Dutch organizations.* Creativity and Innovation Management, 12 (4), 191F201.

[Seac03] Seacord, R.C., Elm, J., Goethert, W., Lewis, G. A., Plakosh, D., Robert, J., Wrage, L., & Lindvall, M. (2003). *Measuring software sustainability.* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[Sund89] Sunday, D.A. (1989). *Software maintainability – a new 'ility'.* Reliability and Maintainability Symposium. Proceedings, Annual. 50-51.

[Tate06] Tate, K. (2006). *Sustainable Software development: An agile perspective.*

[Vent03] Venters et. al., *The blind men and the elephant: Towards an empirical evaluation framework for software sustainability*, WSSSPE'1: workshop on Sustainable Software for science: practice and experiences, SC'13, 17 November 2013, Denver, CO, USA.

[Volb96] Volberda, H. (1996). *Toward the Flexible Form: How to Remain Vital in Hypercompetitive Environments.* Organization Science, 7 (4).

[Vrie17] Vries, B.W. de (2017). *Software Sustainability – Why digitalization will force you to change your software game plan.* TIAS, Master of Business Administration Thesis, Tilburg, The Netherlands.

About the authors

Bryan de Vries MSc MBA is ultimately responsible for all business and operational activities of Omnext B.V. He fulfills this role by applying his experience in the field of Software Quality Assurance and change management. He advises organizations on software quality in general, application of the Omnext Fit Test tooling and supervises creating a "quality culture" in development teams.

Dennis Stam MSc CISA EMITA is a senior manager within the Technology group of KPMG Advisory. He specializes in the entire software production chain and is responsible in the Netherlands for the services concerned with software quality.