Hungarian Association of Agricultural Informatics
European Federation for Information Technology in Agriculture, Food and the Environment

Journal of Agricultural Informatics. Vol. 10, No. 1
journal.magisz.org

Improving the architecture of agricultural knowledge processing systems using design patterns

Stefan Nadschläger[1], Jussi Nikander[2], Dagmar Auer[3]

INFO
Received 05 Apr 2019
Accepted 09 Sep 2019
Available on-line 30 Sep 2019
Responsible Editor: M. Herdon

Keywords:
Disease Pressure, Software Quality, Knowledge Processing Systems, Design Patterns

A B S T R A C T

Use of Knowledge Processing in agriculture has continuously increased, since the first era of knowledge-based systems. Such software systems are used for support in detailed tasks, such as determining the amount of fertilizer in precision farming, as well as on high-level decision support, such as 'what to plant in the next growing season'. Unfortunately, these software systems often have shortcomings in software quality. Applying design patterns is a recognized means to achieve better systems in terms of efficiency, flexibility, and quality. In this paper, several software design patterns are mapped to the context of knowledge processing systems in agriculture. Furthermore, additional patterns are identified and described. The need for patterns with focus on particular aspects of knowledge processing in agriculture is addressed, and an implementation is introduced as a proof of concept.

1. Introduction

Knowledge Processing Systems (KPS), especially Decision Support Systems (DSS) are now a widespread technique to organize and digitally apply knowledge in different domains. In contrast to Knowledge Management Systems (KMS), KPS use algorithms (e.g. Rete or ReteOO) to process existing knowledge in order to generate new knowledge or insights, whereas KMS deal with storage and organization of knowledge. In the domain of agriculture, KPS have been in use for a long time (Jones 1993), (Cox 1996), (Fountas et al. 2015), and are used in a large number of different applications, such as disease management (Perini & Susi 2004), (Kukar et al. 2018), or fertilization planning (Koch et al. 2004), (Zheng et al. 2013), (Nikander et al. 2018). Systems have been developed as stand-alone applications, as well as integrated into existing Farm Management Information Systems (FMIS) (Sørensen et al. 2011).

Furthermore, farmers are yet to fully accept KPS as part of the farming process (Lindblom et al. 2017), (Syropoulou 2017). There are a number of reasons for the farmers' attitudes. Typical complaints include the number of systems already in use, the lack of desired functionality and integration with other systems, bad user interfaces, or incompatibility with existing farming processes (Jørgensen et al. 2007a), (Jørgensen et al. 2007b), (Mackrell et al. 2009), (Evans, Terhorst & Kang 2017). In addition, the long delay in adapting software to changing market situations and unreliable outcomes (errors, wrong results, etc.) negatively influence the acceptance of KPS by farmers. As history in software development has shown, a bad software architecture is typically the reason for the last two points (fixing bugs takes longer and the risk to introduce new ones is higher).

[1]Stefan Nadschläger
Johannes Kepler University Linz, Institute for Application Oriented Knowledge Processing
snadschlaeger@faw.jku.at
[2]Jussi Nikander
Aalto University, Department of Built Environment
jussi.nikander@aalto.fi
[3]Dagmar Auer
Johannes Kepler University Linz, Institute for Application Oriented Knowledge Processing
dauer@faw.jku.at

The aim of a work package in the EU 7th framework project (CLAFIS 2014) was to implement a KPS for the use case "disease pressure calculation". Research on existing KPS in agriculture has revealed that a concrete architecture of such a system has not yet been documented. Thus, it was not possible to start the system on a previously proven software architecture. It was to be assumed that concepts from software architecture in the domain of classical enterprise applications could be reused, but due to different data structures (different concepts than classical business objects to represent knowledge) and the use of complex processing algorithms (usually not found in classical business applications), the development process was more of a trial and error style.

One has to assume that based on missing materials on KPS development also all the other projects were developed in a trial and error style, respectively no inference of quality and functionality of previously implemented systems could be made. Due to the missing proven basic software architecture, many resources in every project will be wasted on creating a sound basic architecture.

To provide suitable documentation of a working KPS software architecture to allow other software developers to start developing new systems based on this knowledge, the following research questions have to be answered: Can existing best practices in software development be applied in KPS development? What is a good way to document them? Does this improve the development and architecture of a KPS?

A literature study on how to implement a KPS revealed a general lack of materials (descriptions, books, articles, papers, etc.) on architecture and best practices. In order to verify this assumption, an expert interview in the form of a questionnaire was conducted with authors of recently published papers on KPS in agriculture (see section 4.1). Moreover, a study was carried out amongst software developers on the need for more materials and the best format to provide them for the development of KPS (see section 4.2).

The design and implementation of complex software architectures in classical enterprise systems (as defined in (Fowler 2003)) seems to have commonly accepted solutions. These are documented in the many materials (books, design patterns, best practices, etc.), explaining how to implement a system from the ground up, and how to overcome difficulties, concerning special software architecture quality criteria (loose coupling, high cohesion, etc.). KPS differ in architecture and complexity (algorithms, data structure, etc.) from enterprise systems, but have to deal with similar problem aspects. The differences between these two types of systems make it difficult for software developers to reuse existing concepts from enterprise application development. Hence, existing materials have to be analyzed and it has to be discussed how to transfer and reuse proven concepts to KPS development.

Applying KPS to the domain of agriculture creates its own unique problems. The majority of primary agricultural producers in Europe are small or micro-enterprises. In 2013 the average farm size in the EU was 16.1 hectares, with significant variation between different EU countries (Eurostat 2015). Furthermore, the extremely large number of small farms in Romania is undoubtedly distorting the EU-wide average. Nevertheless, from this, we can deduce that the most common type of agricultural enterprise in the EU is still a family farm, where the workforce consists of the farmer and their family. Though it should also be noted that the majority of EU agricultural area is utilized by farms larger than 100ha, nevertheless, the farmers' ICT tools need to be carefully constructed to be easy to learn and use in order to be most useful for the users, as they cannot be assumed to be experts in using software systems.

However, farming is becoming an increasingly data-driven activity, where different devices and systems, both on-farm and elsewhere, are used in order to manage the farming process (Fountas et al. 2015). For example, the ISO 11783-10 standard used for tractor and machinery data networking covers tractor-FMIS connectivity, and thus integrate farm machinery with management systems. There are also proprietary solutions that accomplish this, such as the John Deere JdLink or the Valtra Connect. The number of variety of available software is rapidly increasing, and the field is currently extremely heterogeneous and becoming more so. Therefore, the design of KPS also has to consider that the software ecosystem is rapidly changing and that while different tools are likely to be more closely integrated in the future, the means this integration will be done is not yet clear. It is not enough to integrate high-sophisticated processing algorithms. They have to be integrated into a high-quality software architecture that supports established standards, to ensure that maintenance and extension tasks can be executed within a reasonable timeframe and at a reasonable price.

A well-established concept to communicate best practices to software developers is *design patterns*. Because it is familiar to most software developers, it was chosen as an appropriate means to describe a KPS software architecture. In software engineering, many materials on the development of high-quality enterprise applications do exist (books, best practices in architecture, design patterns, etc.). Therefore, a comparison of the main characteristics of enterprise applications and KPS will show which design patterns can be transferred to the context of KPS development. To avoid reinventing the wheel, existing design patterns used in the context of enterprise application development are analysed, selected, and transferred to the context of KPS development. Gaps are filled with the identification of new design patterns.

The rest of this paper is organized as follows. Section 2 describes the concept of software design patterns and presents three contributions made to answer the research questions. Section 3 contains the proof-of-concept example applying the patterns from section 2. Section 4 is about a preliminary interview with creators of KPS systems in agriculture and a study conducted with software engineers to evaluate the impact of patterns. Finally, a summary is given in section 5.

2. Software Design Patterns for KPS

Software design patterns are descriptions of best practices and long-time experience of developers. They are written using a *template*, which defines a structure containing the key points of a design pattern description. The template contains every element required to support a developer to solve a specific problem, without the need for profound knowledge in the domain. It can also be extended. The three main parts of a design pattern are: (1) *context*, (2) *problem* and (3) *solution*. The context defines the circumstances when a design pattern can or should be used. The problem section is a precise description of the exact problem and the solution section describes how to solve this problem. Often these three sections are followed by *consequences* (*benefits* and *liabilities*). Further sections could be implementation examples, known uses, related design patterns, etc. A good example of a non-software specific design pattern, including explanations for each template section is described by (Wellhausen & Fiesser 2012) in the pattern DOORLOCK.

Design patterns are generally not considered to be invented, but instead, they are identified (Bass, Clements & Kazman 2013). The "rule of three" defines that a problem solution has to be used in at least three independent projects to become a design pattern. However, it should be emphasized that not every repeated solution needs to be recognized as a design pattern. A pattern tries to solve a non-trivial problem in a specific context. It can be used and defined in any domain (software engineering, education, cooking, etc.). The benefit of using software design patterns is that they are widely known amongst software developers and they represent concise documentation of best practices that can easily be passed on to other developers.

So far, no sufficient design patterns are available for KPS development. Davis, Gamble & Kimsen (2004) identified two design patterns for KPS. However, they only solve very specific problems and do not support the base architecture. In order to increase the number of relevant patterns in the context of KPS, existing patterns in enterprise applications and object-oriented design will be analysed on their usability in KPS. The result of this research is a collection of design patterns (existing and new ones) to be able to quickly implement a good base architecture for KPS.

In the following, existing design patterns are written in small caps (for example, "VISITOR", a design pattern from (Gamma et al. 1995)).

2.1. Enterprise Application Design Patterns

Enterprise applications (EA) are systems used for critical areas in business and have a high demand for *maintainability* and *extensibility*. In the domain of EA a large number of patterns can be found. Therefore, EAs are compared with KPS to identify common characteristics. Special aspects that often make the development of EA complex, like *cloud*, *parallel computing*, *internal/external integration* of other systems, etc. are also relevant for KPS. Design patterns in the context of these systems have to be analysed if they can also be applied in other contexts. The result of this analysis is a small list of relevant patterns for KPS development.

Comparing the architecture of KPS and EA, six common aspects could be identified: *Data Access*, *Internal Integration*, *External Integration*, *Parallelism*, *Cloud* and *Server* (see (Nadschläger & Küng 2016) for details on the comparison). For every aspect, a set of requirements (non-functional, as well as functional) was defined helping to select suitable design patterns. These design patterns were analysed in detail on their potential to solve problems in the KPS domain.

The result of the comparison of enterprise applications with KPS and the pattern analysis is a small collection of design patterns, grouped by the identified aspects. The context was adapted to the domain of KPS, thus developers of KPS can choose from a now much smaller, but specific set of patterns to solve problems. Table 1 lists an overview of these identified and organized design patterns. Every row contains the name of the patterns, usable for solving problems of a specific aspect (table column titles) common to both system types (KPS and EA).

Table 1. Some selected design patterns from the pattern collection.

Data Access	**Internal Integration**	**External Integration**	**Parallelism**	**Cloud**	**Server**
DAO	Blackboard	DTO	Task Parallelism	Cache-Aside	Request – Response
Repository	Shared Repository	Message Bus / Channel / Router / Bridge	Pipeline	Circuit-Breaker	Command / Document Message
RDG	Dependency Injection	Point-to-Point / Datatype Channel	Fork / Join	CQRS	
Active Row	Listener / Observer	Portal / DO / MOM / SO / Presentation Integration	SPMD	Event Sourcing	
Data Mapper	Whiteboard	Publish-Subscribe	Actors	Sharding	
TDG	Pipes and Filters	Shared Database	Map-Reduce	Materialized View	
Gateway		File Transfer	Master / Worker	Retry	

2.2. Design Pattern Language

Selecting suitable design patterns from a large unordered list (like the one in Table 1) is difficult. It does not contain any hint on when to choose which pattern. A design pattern *language* is a collection of patterns, including *relationship* information between the patterns. The relationship shows how the patterns are related to each other and give a hint on the order of their selection and implementation. Generally, patterns in a language form a set that can be used together to fully solve a larger problem: In this case, to create the base architecture of a KPS. Design patterns for the essential components were identified, described, and their relationships are shown in (Figure 1) so that they form a design pattern language to create a KPS software architecture.

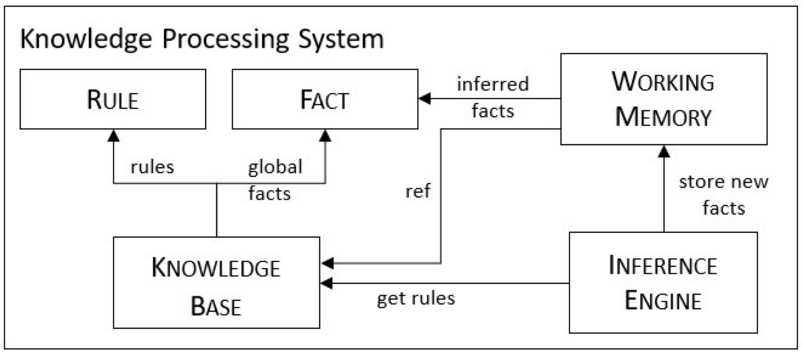

Figure 1. A design pattern language for the base architecture of a KPS (Nadschläger & Küng 2018).

The patterns of this pattern language cover the elementary components of so-called expert systems (the most basic form of KPS). It contains the patterns RULE and FACT. A RULE consists of two parts: the IF and the THEN part, a FACT is static knowledge. Both of them are needed to store knowledge in a system processable manner. The patterns define the structure of the data to become knowledge, for example: IF date = 25.12.2018 THEN bring a present. This is a RULE, containing FACTs (25.12.2018, present).

Knowledge is stored in a KNOWLEDGE BASE. This pattern describes a way to store knowledge that is optimized for RULEs and FACTs and also supports adding and removing knowledge at runtime. It describes a software component with methods to add, remove, and find knowledge at runtime and persist it in any kind of data store.

The heart of the knowledge processing system itself is defined by the pattern INFERENCE ENGINE. It defines the structure for integrating knowledge processing algorithms that work on knowledge stored in the KNOWLEDGE BASE. This pattern describes how RULEs, FACTs, and the KNOWLEDGE BASE correctly interact with the algorithms. Some algorithms also need a means to temporarily store information at runtime. This is described in the pattern WORKING MEMORY. In contrast to KNOWLEDGE BASE, knowledge stored in the WORKING MEMORY will be lost when the system breaks down or gets restarted.

By applying these five essential patterns, a robust and extensible architecture for a KPS can be designed and implemented. It leads to a form of software architecture that is described in major books on software design, like (Bass et al. 2013), and that is commonly accepted to be of high quality. The design patterns also include a description of *how* and *when* to implement the components, which is missing in a high-level overview of KPS components.

2.3. Object-Oriented Design Patterns

KPS described in publications are often developed in an object-oriented style (e.g., using Java or C#). As a further step, existing frameworks (expert system shells and rule engines, especially *JBoss Drools*, *JavaDON*, *OpenRules,* and *NxBRE*) where analysed for the usage of object-oriented design patterns. A special focus was placed on the patterns from the „Gang-of-Four" collection (Gamma et al. 1995), containing patterns for object-oriented programming.

The result of this analysis revealed that the frameworks mentioned above do not contain many design patterns. This leads to difficulties when applying or extending these frameworks, as every developer first has to analyse their code to know how to use them correctly. A search for further frameworks supports this assumption, see Table 1 in (Nadschläger & Küng 2017) for details. Many of them originated from research and have been abandoned.

Figure 2 shows identified patterns (boxes with a thick border) and the components in which they can be used (boxes with thin border).

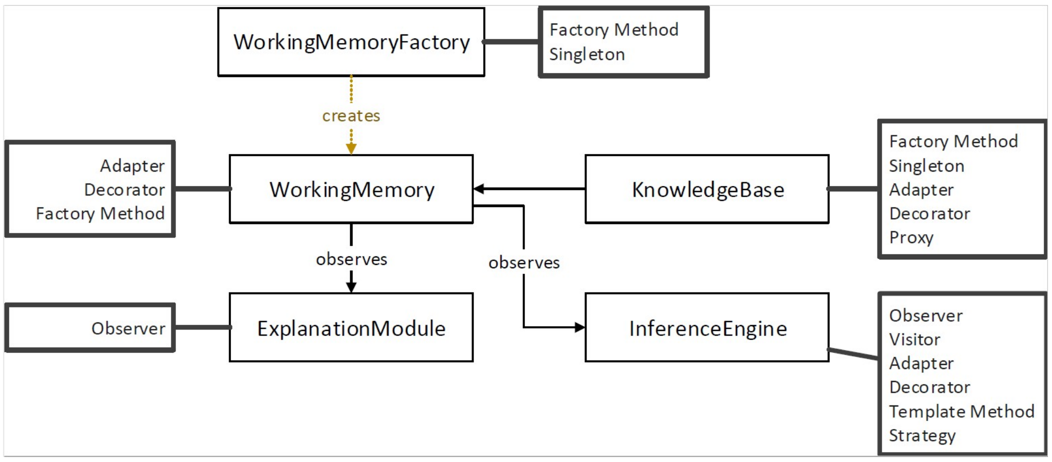

Figure 2. Overview of applied object-oriented patterns (Nadschläger & Küng 2017).

3. Application of Design Patterns in an Agricultural KPS

The majority of a KPS system consists of the knowledge, the rules, and the processing algorithms. This also applies to KPS in agriculture. The patterns presented in the previous section were identified during the development of an agricultural system and should generally handle the common parts, also of KPS in other domains. Nevertheless, the proof-of-concept has been done especially in the agricultural domain.

In the context of the CLAFIS (2014) project, a KPS was designed and implemented for disease pressure calculation on cereals. The system was designed as a module of a larger cloud-based service (Nikander et al. 2017) and had an expert system for disease management at its core. It, therefore, accessed static knowledge[4] (farmer, fields, crop, weather data, etc.) and applied special formulas (implemented in the form of rules) to calculate the risk of a disease outbreak on a given field at a given point in time.

A detailed analysis of further product development, including a business plan, revealed that the lack of materials on correct design and implementation of such systems leads to major future problems in maintenance and extension of the system, thus indirectly impacting the acceptance by farmers (having to wait longer for updates, system not usable on acute market changes, possible higher prices due to increased development).

The prototype of the disease pressure calculation system was developed by applying the patterns described in section 2. It shows how a solid architecture for KPS can be created using design patterns. Complex environments, like server, cloud, integration of external services, etc., can also be handled with this concept.

Knowledge is stored by applying the patterns FACT and RULE. Data needed in disease pressure calculation (the concrete knowledge processing algorithm) is stored in the form of facts (weather data, field data, etc.). The formula is represented as a set of rules. Both are stored in a KNOWLEDGE BASE. The engine evaluating the rules is implemented by applying the INFERENCE ENGINE pattern. The knowledge processing algorithm needs to be able to temporarily store deduced facts, which happens in the WORKING MEMORY.

The prototype also shows many similarities with the identified aspects presented in section 2.1 (execution in a cloud environment, need for parallelism, integration of external systems, etc.). The application is designed to run as a *server* and provides its functionality via a webservice to others. Moreover, it runs in a *cloud environment* and applies patterns from this aspect's collection to increase stability and performance. The algorithm processes many data. Therefore, patterns from the aspect of *parallelism* have been selected. The data partially comes from external services (for example, weather

[4] The term „static knowledge"means data that can easily be stored in database, for example. It does not need further computational processing or intepretation to be used.

data) that have to be integrated (*external integration*) and the internal structure is designed in a loosely coupled way to be able to exchange data sources, data sinks, and algorithms (*internal integration*).

The prototype was implemented in the Java programming language, an object-oriented language, and therefore also design patterns from the catalogue introduced in section 2.3 could be used.

The system could be implemented by relying on approved architectural concepts that automatically ease the communication amongst developers and results in a loosely coupled system. Reduced communication and learning time also cuts down the time needed to create and add new features. The overall time needed to write the source code and communicate the design of the software architecture is reduced.

4. Interview and Study

After the initial research on agricultural KPS software architecture in the (CLAFIS 2014) project, an expert interview has been performed via an online questionnaire to find out if there is a need for more material. Therefore authors of recent publications on agricultural KPS were asked to answer questions on technologies and the quality of their systems. The results are presented in section 4.1.

In order to evaluate (1) if the new material on KPS development (presented in section 2) is sufficient to allow software developers to implement such a system, (2) if the system has a high degree of quality and (3) if design patterns are a suitable form to document this knowledge, a study has been conducted with software developers. The process and results are described in section 4.2.

4.1 Agricultural Knowledge Processing System Interviews

In order to better understand the influence of the quality of KPS in agriculture, interviews with authors of papers that recently published their work on KPS in agriculture (Rupnik et al. 2018), (Oliver et al. 2017), (Rossi et al. 2014), (Clarke et al. 2017) and (Pérez-Expósito et al. 2017)) were conducted via a questionnaire with a combination of open and closed questions. They concerned the current state of their projects, the technologies they used, their quality estimation, and their opinion on the influence of software quality on the acceptance by farmers.

The questionnaire was answered by authors of five different systems (one author per system). All of these systems were started initially and did not rely on previous versions of KPS (but they used existing processing algorithms). Most of them were developed using web-related technologies and languages (JavaScript, PHP, HTML5, CSS, etc.). PostgreSQL and MySQL were used as database management systems. One project was developed as a research project only, one as a business product only, the others targeted both groups. Three were not familiar with the concept of software design patterns, the other two even used some during development. Three of these projects are currently in development, two have already been abandoned.

The authors were also asked to personally estimate the quality of their software. On a scale from 1 (very bad) to 5 (very good), two authors considered their system as average quality (rated as 3), and the three others as good quality (rated as 2). Unfortunately, the rating was only from the viewpoint of the application user, and thus missing insight into the actual source code. All five agreed on the assumption that the software quality increases the acceptance by the users and that it would also support and improve their research.

An interesting result was revealed by the answers to the question how long the actual implementation of the system took (in percentage of the overall project duration): 1 x 40%, 2 x 50%, 1 x 70% and 1 x 90%. Obviously, there is a potential to reduce development time to keep more resources for research on the farmers’ needs.

Even though the number of respondents is low, they support the assumption that more materials on KPS development would increase the quality of the system, as there is obviously room for improvement. Moreover, the development time has potential to be decreased, especially by relying on design patterns for KPS.

4.2 Knowledge Processing System Developer Study

To identify the need for more materials on the development of KPS and the effectiveness of the concept of *design patterns*, a study was carried out. The study is split into a preliminary questionnaire and an implementation task. The participants had to implement a simple KPS in the form of an expert system with rules and facts to build the functionality of an automated teller machine. As research methodology action research was chosen (see (Sjøberg et al. 2007) for an explanation and its application in software engineering).

The target test group of the study was designed to contain educated and experienced developers. In order to exclude distortive submissions by participants not complying to these criteria, a questionnaire was sent to eleven developers (from a software development company and university). It contained questions on their education, general work experience and their understanding and familiarity of the development of KPS, Rule Engines and the application of design patterns. The answers were used to find participants having a comparably high level of education and long-time work experience.

After all eleven participants answered the questionnaire, they were provided with a task description to implement a KPS (in Java, C# or JavaScript). Therefore, they were given a starter template already containing a simple inference algorithm, to allow the participants to focus solely on the software architecture and the quality of the system. The criteria for an optimal result were defined beforehand and the amount of source code that had to be written was small enough to allow the participants to do several iterations (e.g., for improvements, changes). The architecture never got large enough so that a restructuring would have been impossible in acceptable amount of time. After two months, they were requested to hand in the best solution they could come up with. The developers did not get any additional material in advance, but had to rely on their experience in software development and some individual research.

An analysis of the resulting systems revealed that they greatly differed in design and implementation, some systems were not executable at all. The architecture also showed some major design violations (e.g., open-closed principle) and had a strong coupling between the components.

After evaluation of the handed-in results, the participants were provided with the materials in the form of design patterns, presented in section 2 and asked to continue the implementation. Again, after two months, they were requested to hand in their results. The evaluation of the systems of the continued task showed that the software architecture became sound and a common base architecture could be identified. Also, the understanding of the task seemed to improve. The result of this study shows that there is, in fact, a need for more materials on how to develop a KPS. Due to the participants' familiarity in the domain of software engineering, design patterns seem to be a suitable means to provide these materials to developers.

5. Summary and Conclusion

Research on KPS in agriculture and also in other domains, has revealed that the software architecture of these systems has not yet been documented detailed enough to allow software developers to build new systems on approved architectures. A further difficulty in KPS development is the additional complexity introduced by hard to understand processing algorithms and different data structures unfamiliar to non-experts. The knowledge on how to implement KPS must be documented to be reused in new projects. The opportunity for improvement has been supported by an expert interview in the form of a questionnaire, conducted with authors of recent publications on agricultural KPS systems.

The software architecture of EAs is well documented. In a first step, these systems were compared with KPS to identify best practices in EA that can also be applied in KPS development. Most of these best practices are documented in the form of so-called design patterns. The concept of design patterns is familiar to software developers and is successfully used to document long-time experience for many years. Therefore this concept was chosen as a suitable documentation format.

The results of research on design patterns for KPS are (1) a set of smaller collections of design patterns from the domain of EA, (2) a list of object-oriented design patterns whose usage in existing

KPS frameworks was analyzed, as well as (3) a KPS specific pattern language, consisting of five essential patterns to create a basic software architecture.

To find out if the documentation in the form of design patterns is sufficient and if they improve the ability to implement high-quality KPS architectures, a study was conducted with eleven software developers. The study has shown that this new material can improve the ability to develop KPS.

As experience in software development has shown, good software architecture positively influences further development, maintenance, and extensibility of the system and thus reduces the costs of a system. Providing documentation of best practices allows developers to build on experience, adding the chance to further improve the system and therefore indirectly influencing the acceptance by the farmers.

The material on design patterns for KPS has initially been developed and applied in an agricultural KPS focusing on disease pressure calculation on cereals. In this proof-of-concept, most of the design patterns presented in this paper have been applied.

Also, in further studies, design patterns proved to be a good way to document best practices and experience in KPS development. An initial lack of materials has been addressed by providing pattern collections and a new design pattern language. The application of the patterns has only been evaluated in the context of agriculture so far, but their use needs to be evaluated in other domains in the future.

References

Bass, L., Clements, P., Kazman, R. (2012): Software Architecture in Practice, Pearson Education.

CLAFIS (2014): CLAFIS. Available from: http://www.clafis-project.eu [04 July 2018].

Clarke, N., Bizimana, J.-C., Dile, Y., Worqlul, A., Osorio, J., Herbst, B., Richardson, J.W., Srinivasan, R., Gerik, T.J., Williams, J., Jones, C.A., Jeong, J. (2017): Evaluation of new farming technologies in Ethiopia using the Integrated Decision Support System (IDSS), Agricultural Water Management, v. 180, pp. 267–279. https://doi.org/10.1016/j.agwat.2016.07.023

Cox, P. G. (1996): Some issues in the design of agricultural decision support systems', Agricultural systems, 52(2-3), pp. 355–381. https://doi.org/10.1016/0308-521X(96)00063-7

Davis, L, Gamble, R. F., Kimsen, S. (2004): A patterned approach for linking knowledge-based systems to external resources, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(1), pp. 222–233. https://doi.org/10.1109/TSMCB.2003.811862

Eurostat (2015): Farm structure statistics. Available from: https://ec.europa.eu/eurostat/statistics-explained/index.php/Farm_structure_statistics [18 February 2019].

Evans, K.J., Terhorst, A., Kang, B. H. (2017): From data to decisions: helping crop producers build their actionable knowledge. Critical Reviews in Plant Sciences, 36(2), pp. 40–50. https://doi.org/10.1080/07352689.2017.1336047

Fountas, S., Carli, G., Sørensen, C. G., Tsiropoulos, Z., Cavalaris, C., Vatsanidou, A., Liakos, B., Canavari, M., Wiebensohn, J., Tisserye, B. (2015): Farm management information systems: Current situation and future perspectives. Computers and Electronics in Agriculture, vol. 115, pp. 40–50. https://doi.org/10.1016/j.compag.2015.05.011

Fowler, M. (2003): Patterns of Enterprise Application Architecture, Pearson Education.

Gamma, E., Johnson, R., Helm, R., Vlissides, J. (1994): Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley.

Jones, J.W. (1993): Decision support systems for agricultural development. Systems approaces for agricultural development, pp. 459–471. https://doi.org/10.1007/978-94-011-2840-7_28

Jørgensen, L. N., Noe, E., Langvad, A. M., Jensen, J. E., Ørum, J. E., Rydahl, P. (2007): Decision support systems: barriers and farmers' need for support, EPPO bulletin, 37(2), pp. 374–377. https://doi.org/10.1111/j.1365-2338.2007.01145.x

Jørgensen, L. N., Noe, E., Nielsen, G. C., Jensen, J. E., Ørum, J. E., Pinnschmidt, H. O. (2007): Problems with disseminating information on disease control in wheat and barley to farmers. Sustainable disease management in a European context, pp. 303–312. https://doi.org/10.1007/978-1-4020-8780-6_9

Koch, B., Khosla, R., Frasier, W. M., Westfall, D. G., Inman, D. (2004): Economic feasibility of variable-rate nitrogen application utilizing site-specific management zones. Agronomy Journal, 96(6), pp. 1572–1580. https://dx.doi.org/10.2134/agronj2004.1572

Kukar, M., Vračar, P., Košir, D., Pevec, D., Bosnić, Z. (2018): AgroDSS. A decision support system for agriculture and farming. Computers and Electronics in Agriculture. https://doi.org/10.1016/j.compag.2018.04.001

Lindblom, J., Lundström, C., Ljung, M., Jonsson, A. (2017): Promoting sustainable intensification in precision agriculture: review of decision support systems development and strategies, Precision Agriculture, 18(3), pp. 309–331. https://doi.org/10.1007/s11119-016-9491-4

Mackrell, D., Kerr, D., Von Hellens, L. (2009): A qualitative case study of the adoption and use of an agricultural decision support system in the Australian cotton industry: The socio-technical view. Decision Support Systems, 47(2), pp. 143–153. https://doi.org/10.1016/j.dss.2009.02.004

Nadschläger, S., Küng, J. (2016): A pattern collection for knowledge processing system architecture. Proceedings of the 21st European Conference on Pattern Languages of Programs, ACM. http://dx.doi.org/10.1145/3011784.3011796

Nadschläger, S., Küng, J. (2017): Analysis of GoF Design Patterns used in Knowledge Processing Systems. Proceedings of the 22nd European Conference on Pattern Languages of Programs, ACM. https://doi.org/10.1145/3147704.3147711

Nadschläger, S., Küng, J. (2018): Towards a pattern language for knowledge processing systems: expert systems. Proceedings of the VikingPLoP 2017 Conference on Pattern Languages, ACM. https://doi.org/10.1145/3158491.3158506

Nikander, J., Huitu, H., Koistinen, M., Laajalahti, M., Niemëläinen, O., Kaivosoja, J. (2018): Application of Drone-Based Maps in Planning Farm Operations for Sward Management in Silage Production. European Agricultural Engineering Conference EurAgEng, Wageningen.

Nikander, J., Linkolehto, R., Jäger, M., Pesonen, L., Ronkainen, A., Suokannas, A. (2017): Prototype Environment for integrating and sharing Farm Things and associated data. Advances in Animal Biosciences, 8(2), pp. 645–649. https://doi.org/10.1017/S2040470017000425

Oliver, D. M., Bartie, P. J., Heathwaite, A. L., Pschetz, L., Quilliam, R. S. (2017): Design of a decision support tool for visualising E. coli risk on agricultural land using a stakeholder-driven approach. Land Use Policy, Vol. 66, pp. 227–234. https://doi.org/10.1016/j.landusepol.2017.05.005

Pérez-Expósito, J. P., Fernández-Caramés, T. M., Fraga-Lamas, P., Castedo, L. (2017): VineSens: An Eco-Smart Decision-Support Viticulture System, Sensors, 17(3). https://doi.org/10.3390/s17030465

Perini, A., Susi, A. (2004): Developing a decision support system for integrated production in agriculture, Environmental Modelling & Software, 19(9), pp. 821–829. https://doi.org/10.1016/j.envsoft.2003.03.001

Rossi, V., Salinari, F., Caffi, T., Bettati, T. (2014): Addressing the implementation problem in agricultural decision support systems: the example vite.net(R). Computers and Electronics in Agriculture, vol. 100. https://doi.org/10.1016/j.compag.2013.10.011

Rupnik, R., Kukar, M., Vračar, P., Košir, D., Pevec, D., Bosnić, Z., (2018): AgroDSS: A decision support system for agriculture and farming. Computers and Electronics in Agriculture. https://doi.org/10.1016/j.compag.2018.04.001

Sjøberg, D. I., Dybå, T., Jorgensen, M. (2007): The Future of Empirical Methods in Software Engineering Research. Future of Software Engineering, IEEE Computer Society, pp. 358–378. https://doi.org/10.1109/FOSE.2007.30

Sørensen, C. G., Pesonen, L., Bochtis, D. D., Vougioukas, S. G., Suomi, P. (2011): Functional requirements for a future farm management information system, Computers and Electronics in Agriculture, 76(2), pp. 266–276. https://doi.org/10.1016/j.compag.2011.02.005

Syropoulou, P., Symeonidou, M., Tekes, S., Wawer, R., Kazantzidis, A., Crnojevic, V. (2017): Developing an intelligent ICT system for environmentally optimized irrigation management in agriculture. Journal of Agricultural Informatics, 8(2), pp. 22–32. http://dx.doi.org/10.17700/jai.2017.8.1.375

Wellhausen, T., Fiesser, A. (2012): How to write a pattern?: a rough guide for first-time pattern authors. Proceedings of the 16th European Conference on Pattern Languages of Programs, ACM. https://doi.org/10.1145/2396716.2396721

Zheng, Y. J., Song, Q., Chen, S. Y. (2013): Multiobjective fireworks optimization for variable-rate fertilization in oil crop production. Applied Soft Computing, 13(11), pp. 4253–4263. https://dx.doi.org/10.1016/j.asoc.2013.07.004