Data-Driven Software Architecture for Analyzing Confidentiality

S. Seifermann, R. Heinrich and R. Reussner, "Data-Driven Software Architecture for Analyzing Confidentiality," *2019 IEEE International Conference on Software Architecture (ICSA)*, Hamburg, Germany, 2019, pp. 1-10, doi: 10.1109/ICSA.2019.00009.

1. Fichamento de Conteúdo

Um dos pontos mais importantes atualmente para todos os usuários é a sua confidencialidade dos dados, e caso seja quebrada, consequências em lei podem ocorrer. Muitas vezes, dados são vazados por um falha no esquema da arquitetura do software, visto que a confidencialidade não é abrangida nessa fase do desenvolvimento. O artigo propõe a inserção de fluxo de dados no modelo da arquitetura do software, assim podendo alegar as restrições na confidencialidade. Visto que o problema se concentra na arquitetura do software e nos dados que devem ser protegidos, é proposto o uso da arquitetura de software orientada a dados (DDSA) para que assim, o projeto seja elaborado com o foco nos 2 pontos mais críticos. Utilizando o Palladio e Prolog em conjunto, eles planejam utilizar a modelagem da arquitetura detalhadamente com base em lógica matemática para definir regras de negócio do programa. Visto que o Prolog é uma linguagem de lógica matemática onde apenas se definem os problemas e regras, o código fonte não seria um problema, já que a máquina determinaria a melhor maneira de solucionar o problema atendendo as regras estipuladas. A ideia foi testada e resultou como um sucesso nos 16 casos propostos pelos autores, onde todos os problemas de confidencialidade foram encontrados com sucesso e nenhum falso positivo foi reportado.

2. Fichamento Bibliográfico

- Arquitetura de software orientada a dados (DDSA, Data-Driven Software Architecture) é baseado em um paradigma de programação onde dados e os processadores desses dados são entidades primárias dele, e o comportamento do programa deve ser baseado no fluxo dos dados nele presentes.
- Prolog é uma linguagem de programação com o paradigma de programação de lógica matemática, onde você fornece uma descrição do problema a ser resolvido e indica como ele deve ser solucionado. Não possui tipos de dados como inteiros ou caracteres, as variáveis assumem um tipo chamado de “termo”.
- Palladio é uma linguagem para modelagem arquitetural de software, capaz de representar a estrutura básica da arquitetura, além de uma abstração do comportamento dos componentes presentes.
- Component-based Software Engineering é um ramo onde a ênfase está na decomposição dos sistemas em componentes funcionais e lógicos.

3. Fichamento de Citações

- “Software vendors must consider confidentiality in every development phase to ensure compliance. This is beneficial because the earlier vendors detect issues, the more cost efficient they can fix them. Architectural design is an early phase that helps cutting down costs for fixing issues significantly”
- “Control defines the problem-solving strategy. Logic defines the knowledge to solve the problem. A solver component automatically determines and applies the problem solving strategy. A programmer merely specifies the knowledge.”
- “the process (Component-based Software Engineering, CBSE) is not sufficient for building data-driven architectures and analyzing their compliance with confidentiality constraints because it misses information about data, data processing and confidentiality properties.”

Deep Learning in Software Engineering

X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, “Deep learning in software engineering,” arXiv, vol. abs/1805.04825, 2018.

1. Fichamento de Conteúdo

Deep learning é uma área muito comum nos últimos anos, porém sua fama não resolve seus problemas que continuam em aberto. O artigo busca investigar se as técnicas de *Deep learning* realmente estão auxiliando a engenharia de software em geral, e se está evoluindo ou apenas sendo utilizada. Para isso, 98 artigos sobre engenharia de software que continham técnicas de *Deep learning* foram analisados para avaliar o real ganho nas soluções propostas. Nota-se um cenário significativamente crescente em relação a *deep learning* em conjunto com a engenharia de software, 98 artigos foram analisados, porém já se encontram por volta de mais 150 artigos contendo tanto *deep learning* quanto a engenharia de software esperando a aprovação da banca. Para encontrar esses artigos, houveram critérios para a busca, como um conjunto de palavras-chave, estar na língua inglesa e dentro do tópico de ciência da computação dos periódicos. Com os resultados observamos claramente que *deep learning* é um conceito recente, visto que no período entre os anos 2000 e 2014, apenas 12 artigos foram publicados. Mas de 2015 até o primeiro trimestre de 2018, quando o artigo foi escrito, haviam 86 artigos já publicados, e tem facilitado por volta de 40 tarefas da engenharia de software, tanto no mercado quanto na univesidade.

2. Fichamento Bibliográfico

- *Deep learning* é uma das bases da inteligência artificial, derivado do *Machine learning* e tem ajudado os computadores a realizar tarefas como classificar, reconhecer, detectar etc.
- Relatórios de *bugs* são documentos criados na fase de testes de um software, onde ele alega um *bug* encontrado no programa, como foi feito para se encontrar o *bug* e descrever o erro ocorrido.
- Um *AutoEncoder* é uma rede neural treinada com o objetivo de receber uma entrada de dados, conseguir realizar uma compressão desse valor e replicá-lo na saída.

3. Fichamento de Citações

- “For typical SE tasks, deep learning helps SE participators extract requirements from natural language text, generate source code, predict defects in software, etc. As an initial statistics of research papers in SE in this study, deep learning has achieved competitive performance against previous algorithms on about 40 SE tasks.”
- “To identify informative sentences, researchers utilize AutoEncoder to encode the words in bug report sentences in an unsupervised way. Since the hidden state of AutoEncoder provides a compressed representation of the input data, the weights of words in a bug report can be measured by calculating how much information of a word is reserved in the hidden states.”
- “we find that deep learning attracts little attention in SE for a long time, i.e., only less than 3 papers are published each year before 2015. The reason may be that although deep learning performs well on image processing, speech recognition, etc., it takes time for the practitioners and researchers in SE to validate deep learning on domain-specific SE tasks.”
- “As a complex and almost opaque model, several factors limit the practicability of deep learning, including the effectiveness, efficiency, understandability, and testability. These issues may influence the development of deep learning in SE.”

Enhancing the Student Learning Experience in Software Engineering Project Courses

M. Marques, S. F. Ochoa, M. C. Bastarrica and F. J. Gutierrez, "Enhancing the Student Learning Experience in Software Engineering Project Courses," in *IEEE Transactions on Education*, vol. 61, no. 1, pp. 63-73, Feb. 2018, doi: 10.1109/TE.2017.2742989.

1. Fichamento de Conteúdo

Grande parte das instituições de ensino começam a lecionar o desenvolvimento de software de maneira extremamente focada no desenvolvimento ágil. Porém nem sempre essa pode ser a maneira mais adequada de se ensinar a construir softwares para programadores novatos. Vemos o método de reflexão semanal de desempenho (RWM, *Reflexive weekly monitoring*) ser abordado no artigo, alegando que este método de início seria mais efetivo e coordenado, além de os alunos se sentirem mais satisfeitos e prestativos com a sua equipe. O RWM busca principalmente desenvolver as competências dos alunos, como a vontade de aprender, adaptação a ferramentas entre outras. O RWM, como o próprio nome sugere, consiste em encontros semanais com um monitor que auxilia a equipe a coordenar o projeto, embora o monitor não se responsabilize por nenhuma tarefa direta. Como resultado, foi observado que o RWM auxilia no desenvolvimento das *soft skills*, mas não necessariamente a produtividade dos alunos. Por mais que saiam mais satisfeitos com os resultados, eles não se dedicam mais do que o esperado aos projetos, aparentemente eles apenas ficam acomodados.

2. Fichamento Bibliográfico

- A reflexão semanal de desempenho (RWM) é uma metodologia de desenvolvimento que consiste na reunião semanal com um monitor para acompanhar de perto o seu projeto e orientar onde for possível.
- O gerenciador de requerimentos de software (SRM, Software Requirement Manager) é um repositório disponível apenas para os alunos envolvidos no projeto e tem a finalidade de organizar e compartilhar com a equipe todos os arquivos do projeto.
- O monitor apresentado no RWM é como um monitor de faculdade, ele está lá para auxiliar a equipe com suas próprias ideias e o conhecimento e experiência já adquiridos, mas não necessariamente ele é parte da equipe nem é um profissional renomado da área.

3. Fichamento de Citações

- “Several mechanisms have been reported in the literature to enhance students’ experience when using agile methods. However, less research has been done on supporting the learning process when using disciplined development strategies.”
- “The results indicate that RWM helps student teams improve their coordination, sense of belonging to a team, and effectiveness, but not necessarily their productivity. Monitored teams tend to be more productive during the first half of the project, but this changes when they realize that they have the project under control.”
- “The RWM method is led by monitors who perform weekly monitoring sessions with a software development team to facilitate members’ understanding of their own performance and project status. The monitor is not a coach, project manager, or Scrum master.”
- “The weekly RWM sessions take place throughout the project, although they are more useful during the early stages. Students should not perceive the sessions as an evaluation, but rather as a formative activity that allows them to improve individually and as a team, based on their previous experiences and without direct consequences on their course grade.”

Gamifying research in software engineering

M. Nowostawski, S. McCallum and D. Mishra, Gamifying research in software engineering, Computer Applications in Engineering Education, 26(5), pp. 1641–1652, 2018.

1. Fichamento de Conteúdo

Um dos maiores desafios no curso de engenharia de software é conseguir engajar os alunos assim como nos outros cursos semelhantes, visto que os outros são cursos mais práticos do que teóricos. Então como despertar o interesse dos alunos em um curso com foco mais teórico e gerencial? O artigo propõe a gamificação da educação da engenharia de software. A gamificação vem crescendo nos últimos anos em diversos ramos do ensino, não é de se surpreender que cursos relacionados a informática também estão começando a utilizar essa metodologia. O artigo quer demonstrar como a gamificação pode apresentar melhoras no ensino da engenharia de software, tanto em qualidade quanto em interesse dos alunos. Para isso, eles criaram uma metodologia de ensino a parte, chamada "*The Game of Reading and Discussion*". Durante 4 anos (2015-2018) eles colocaram essa metodologia em prática e consistia basicamente em fazer os alunos lerem artigos, elaborarem perguntas sobre o artigo e após isso, realizar uma reflexão do artigo lido e avaliar a reflexão de algum outro aluno. Outra ideia utilizada por eles foi um tipo de debate, a fim de treinar as habilidades de argumentação e contra-argumentação dos alunos. E ao final os números mostram que os alunos estão mais engajados nas aulas, porém não estão certos se os efeitos disso seriam duradouros para anos seguintes de estudo dos alunos.

2. Fichamento Bibliográfico

- Gamificação é um método de engajamento onde se utilizam recursos, mecânicas e características de jogos para despertar o interesse do povo, normalmente devido ao despertar o espírito competitivo, seja com os outros, ou consigo mesmo.
- *Self Determination Theory* é uma teoria que fala sobre a motivação própria através da essência do ser humano em seu crescimento próprio e, subconscientemente, suas necessidades psicológicas. Existem diversos estudos que unem essa teoria com a metodologia de gamificação.
- *Inquiry-based learning* consiste na aprendizagem através de perguntas. Em vez de professores apenas levarem informação aos alunos, eles fazem perguntas significativas, mas simples, a fim de fazer que o próprio aluno consiga pensar na resposta certa de maneira intuitiva.

3. Fichamento de Citações

- "The satisfaction of learners' needs for autonomy, competence, and relatedness facilitate them to move along the learner's autonomy continuum from dependence to autonomy."
- "The basic idea of IBL is asking meaningful questions, which may be established by students, teachers, or by negotiation among them. Inquiry based learning is often described as a cycle which implies five phases: ask, investigate, create, discuss, and reflect."
- "One of the key features of good educational programme is engaging students with the teaching material."
- "The students are required to postulate an argument, and then present it, in oral or written form. After formulating their own argument, students have to prepare counterarguments for other students' statements."

Kanban in software engineering: A systematic mapping study

Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M. (2018). Kanban in software engineering: A systematic mapping study. Journal of Systems and Software, 137, 96–113.

1. Fichamento de Conteúdo

O Kanban vem amplamente sendo usado em empresas de desenvolvimento ágil, devido a sua simplicidade e o resultado entregado quanto a gestão de tarefas. Mas determinar o real valor que o Kanban agrega a uma empresa não é trivial, o artigo se dispõe a avaliar o valor entregado por esse sistema toyotista. O problema vem do fato que poucos estudos são realizados sobre o Kanban, vemos apenas breves citações do modelo em artigos. Apenas 23 artigos até 2018 apresentam informações relevantes sobre o Kanban, enquanto a grande maioria dos artigos apenas o aponta como um auxiliar no sucesso das empresas, sem muitos detalhes. Assim é proposto um estudo de mapeamento sobre o estado da arte do Kanban apontando seus benefícios e desafios no uso, além de abrir oportunidades para estudos futuros. Para esse estudo temos 3 fases principais, sendo elas, planejamento, condução e documentação. O planejamento se resume em encontrar os artigos a serem estudados e definir o escopo do projeto, a condução consiste em determinar dados relevantes que foram encontrados nos arquivos, e a documentação seria a escrita do artigo próprio. Ao final é dito que o Kanban, apesar de trazer muitos benefícios, ele por si só não é o suficiente para garantir seu sucesso.

2. Fichamento Bibliográfico

- Kanban é um sistema desenvolvido pela Toyota para controle e gestão de tarefas, consiste em 3 colunas (*To do, doing, done*) onde são dispostas as descrições das tarefas mediante sua situação atual e são remanejadas a medida que a tarefa assume outro estado.
- Estado da arte diz respeito ao atual nível de conhecimento sobre o tema sendo discutido no artigo.
- *Lean* é uma filosofia de gestão toyotista, que busca eliminar todo o desperdício de qualquer tipo de recurso, resolver problemas sistematicamente e mudar a forma de pensar ao gerenciar um negócio.

3. Fichamento de Citações

- “Annual ‘State of Agile’ reports show that the use of Kanban increased from 31% to 39% in 2015 and from 39% to 50% in 2016.”
- “A clear finding emerging from this systematic mapping study is the need to (i) increase the number of rigorous academic studies on Kanban, (ii) be explicit about the validity threats to that study and how these were mitigated, and (iii) build on cumulative knowledge.”
- “While the opportunities to conduct high quality research of Kanban in software engineering are limitless, its theoretical development will be limited if future research on Kanban does not adopt a tradition of cumulative building of knowledge.”

Lean and agile software process improvement in traditional and agile environments

Poth, A., Sasabe, S., Mas, A., & Mesquida, A. L. (2019). Lean and Agile software process improvement in traditional and Agile environments. Journal of Software: Evolution and Process.

1. Fichamento de Conteúdo

Muitas empresas tradicionais já possuem um padrão de projeto bem definido, porém é necessário se adaptar as novas necessidades do mercado. O artigo busca mostrar que, mesmo em empresas com um padrão bem antiquado, é possível melhorar a sua qualidade de software caso tenha cabeça aberta para utilizar o desenvolvimento ágil e o *lean*. O grande sucesso na área de informática utilizando dessas duas técnicas, incentivou Poth, Sasabe, Mas e Mesquida a pesquisarem sobre o uso dessas técnicas em conjunto. Podemos observar algumas semelhanças no desenvolvimento ágil e o *lean* quando vemos os conceitos do *lean*, como eliminar qualquer recurso que não agregue valor a entrega final, decisões tomadas futuramente, empoderar o time e enxergar as entregas como um todo, já que ao final elas produzirão um software completo. Para concluir o artigo vemos que a adaptabilidade das empresas tradicionais ao novo modelo de mercado ficou bem facilitado com diversos *frameworks* e profissionais voltados para ajudar a fazer essa transição, e que dependendo do contexto aplicado, o retorno é significativamente melhor.

2. Fichamento Bibliográfico

- O processo de melhoria de software (SPI, software process improvement), como o próprio nome diz é um processo que busca melhorar a qualidade na produção do seu software, por meio de técnicas diferentes, otimização de tempo e custo entre outros benefícios.
- *Lean* é uma filosofia de gestão toyotista, que busca eliminar todo o desperdício de qualquer tipo de recurso, resolver problemas sistematicamente e mudar a forma de pensar ao gerenciar um negócio.
- KAIZEN é uma ferramenta utilizada para se implementar a melhoria dentro de uma empresa, que está ligada coma melhoria contínua, assim como o *lean*.

3. Fichamento de Citações

- “The KAIZEN improves quality of product, process, and capability of work by closing a feedback-loop after implementation of work on such objects and repeating the Plan-Do-Check-Act (PDCA) cycles step-by-step in an incremental way towards improvement goals. Necessary steps for solving quality problems were evolved to the Deming cycle, or PDCA cycle, which is a tool for conducting quality improvement in Japan.”
- “The Japanese automotive industry introduced the TQC/TQM approach for improving their production system of cars. Toyota created its own Toyota Production System (TPS) with the philosophy of removing waste (“Muda”) via KAIZEN. The “Just in time” concept and the “Kanban” method were invented and practiced by Taiichi Ohno. TPS realized a car production system to fit the Japanese automotive market situations and showed flexibility in adapting to the customer requirement changes.”
- “Agility can be defined as the ability to create and to respond to change in order to create value in turbulent business environment. It is based on several business principles like continuous innovation, product adaption, shortening delivery times, adjustment of people and processes and getting reliable results.”
- “To apply the Lean/Agile approach in a traditional environment has been usual in recent times. However, companies have had to carry out an important transformation to adapt their way of working to this new approach. Even though there exist many guidelines and frameworks for agile adoption, organizations have problems with the selection of the most convenient method and with the general initiation of the Agile Transformation Process.”

Lean Learning – Applying Lean Techniques to Improve Software Engineering Education

R. Chatley and T. Field, "Lean Learning - Applying Lean Techniques to Improve Software Engineering Education," *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, Buenos Aires, 2017, pp. 117-126, doi: 10.1109/ICSE-SEET.2017.5.

1. Fichamento de Conteúdo

Existe um vão significativo enquanto aprendemos a produzir softwares, esse intervalo está entre a fase universitária o que Chatley e Field chamam de "mundo real". Por sua vez, os autores planejam criar um programa de estudos q reduza esse vão de aprendizado focando nas ferramentas, técnicas e erros mais comuns presentes no mercado de trabalho. Como eles utilizaram uma metodologia *lean* de ensinamento, prezaram fortemente por um *feedback* rápido e ciclos iterativos curtos, para assim manter o contato com os 150 alunos da universidade Imperial. Eles decidiram utilizar essa abordagem ágil para familiarizar os alunos com o mercado atual, onde sua grande maioria desenvolve *softwares* com metodologias ágeis. Também defendem fortemente a ideia de que só se aprende de verdade com o trabalho prático, a teoria é apenas uma base, o ensinamento real vem do aluno realizar a tarefa proposta. A abordagem com os estudantes para introduzi-los ao mercado é interessante, visto que incentivam o uso de técnicas já utilizadas no mercado e que funcionam, como a programação em pares e ideias semelhantes a *Hackathons.*

2. Fichamento Bibliográfico

- *Lean learning* se baseia na metodologia de desenvolvimento ágil *Lean*, com os princípios de eliminar o desperdício de tempo, eliminar recursos extras que seriam desnecessários, simplicidade e buscando a satisfação do usuário.
- Kanbam é um método de gerenciamento de processos com fundamentos Toyotistas e no *lean*, buscando evitar o desperdício em geral.
- Das 3 metodologias de ensino citadas por Chatley e Field, a "*transmission*" é a técnica que funciona como uma palestra, onde um profissional tenta transmitir seus conhecimentos a vários estudantes

3. Fichamento de Citações

- "Students should learn by doing and, wherever possible, software engineering principles should be assessed in the context of practical work, rather than by regurgitating material taught or extracted from text books. "
- "It seems futile to teach software engineering before learning to program, so the first year of Imperial's degree places a great deal of emphasis on programming "in the small". "
- "At the end of each term students complete an anonymous on-line survey. This feedback has been instrumental in guiding the evolution of our programme and, where appropriate, we include selected comments in the paper, both positive and negative, as part of the narrative."
- "One further change that made a big difference to both student learning and marking load is to encourage students to pair-program, which has been shown to be highly effective in a classroom environment ."
- "Each group has a different brief, but all are aiming to build a piece of software that solves a particular problem or provides a certain service for their users."

Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering

Staron, M., Meding, W., Soder, O., Back, M., 2018. “Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering”. Foundations of Computing and Decision Sciences 43 (4), 281–303

1. Fichamento de Conteúdo

Um dos grandes pilares do desenvolvimento ágil é a integração contínua, que consiste em integrar o código das entregas de forma mais frequente através de *builds* e testes automatizados. O artigo fará um estudo de caso com 2 empresas que trabalham com mais de 100 desenvolvedores seguindo a metodologia ágil e os principios do *lean*, onde ambas produzirão um sistema embarcado e depois medirão o quanto os testes e revisões automatizados e manuais influenciam a velocidade da integração. Ambas as empresas utilizarão o *Jenkins* para a integração contínua e o *Gerrit* para as revisões automatizadas. Também realizarão um *workshop* para identificar fatores que expliquem os resultados obtidos. Foram calculados dados como a média, valor mínimo, máximo e *outliers* de variáveis como o número de linhas de código por integração, número de revisões feitas e o tempo para cada revisão. Um dos problemas mais comuns no ramo da engenharia de software são os prazos normalmente muito curtos para as entregas ocorrerem, a solução então seria uma melhora no desempenho durante as revisões. O artigo cita fatores influentes para melhorar a efetividade da equipe, como por exemplo, boa organização dos repositórios, lembretes a equipe e a proximidade do time de desenvolvimento. Ao final de tudo também é dito que outros estudos de caso seriam interessantes para poder afirmar com mais certeza os pontos concluídos, além de tentar explorar outras possíveis métricas.

2. Fichamento Bibliográfico

- *Branch* é um desdobramento do código fonte primário ou até mesmo de um *branch* anterior, muito utilizado para o versionamento de softwares.
- Um repositório é o local onde o código fonte do projeto se encontra, todos os membros da equipe devem utilizá-lo para dar manutenção ao código e atualizar a versão final.
- Uma API é uma interface provida por algum serviço para que você possa integrá-lo ao seu sistema, funcionam como uma ponte de comunicação entre aplicações, mas sem revelar informações críticas ao usuário.

3. Fichamento de Citações

- “One of the challenges in achieving high development speed is the combination of automated and manual activities – certain activities need to be done manually, e.g. code reviews.”
- “Which factors are important when optimizing speed of reviews and speed of integration in continuous software development? In particular, we focus on the ability of modern companies to quantify, monitor and improve the speed of reviews and integrations software development in ways alternative to velocity.”
- “A recent study of advantages and disadvantages of having a single repository, conducted at Google, has shown that distribution of code repositories has significant advantages in the flexibility while the monolithic repository has significant advantage in terms of reuse of APIs.”
- “We have found that the definition of speed is a very good alias for the measurement of duration and is practically the most usable one. Since time is the only parameter in this measure, this measure of speed cannot be manipulated by changing the prerequisites.”