A List of Pre-Requisites to Make Recommender Systems Deployable in Critical Context

E. Bouzekri[1], A. Canny[1], C. Fayollas[1], C. Martinie[1], P. Palanque[1],
E. Barboni[1], Y. Deleris[2], C. Gris[2]

[1] ICS-IRIT, University of Toulouse
118, route de Narbonne
31042 Toulouse, France
{Elodie.Bouzekri,Alexandre.Canny, fayollas,palanque,barboni}@irit.fr
[2] AIRBUS Operations
316 Route de Bayonne
31060 Toulouse, France
{Yannick.Deleris,Christine.Gris}@airbus.com

Abstract. In the academic area, recommender systems have received a lot of attention in the recent years (see for instance the increasing success of the RecSys conference). This success has also reached industry and the general public via large platforms such as Amazon or Netflix. The recommender systems present multiple benefits that can be attributed to automation by means of the migration of function from the operator to the Recommender System itself. Depending on the type of recommender system, these functions can cover: support to perception of information, support to identification of potentially relevant elements, support to the selection of one element amongst a list... Despite their widespread use, their development has not reached the level of maturity required for the deployment in the context of critical command and control systems. This position paper identifies some pre-requisite for making recommender systems deployable in critical context of critical systems.

Keywords: Automation, recommender systems, automation, operator tasks, dependability, certification.

1 Introduction

Recommender Systems (RS) are nowadays widely used in the area of consumer electronics and home entertainment. They are exploited by large companies (such as Amazon in the area of e-commerce [17]) and used by millions of users (e.g. 93 million for Netflix [1]). The main target for RS designers and developers has been accuracy as demonstrated in [22]. More recently, focus has moved to other perceived quality of use such as user experience and specific attributes of this quality factor [19]. Despite all these efforts, engineering recommender systems follows craft processes and remain far away from software engineering practices. Contributions such as [22] address reliability but only in the sense of reliability

of users (in the information they provide to the system) and [29] only consider recommender systems in software engineering i.e. how a recommender system could help software engineers.

As for the development of RS, several platforms have been proposed throughout the years starting with Lenskit [12]. Other open source platforms for developing RS have been proposed such as **??** or [14] focusing on the integration of various algorithms implementing the main types of RS (e.g. item-based, knowledge-based, collaborative ...). Assessing which platform produces better results has also been identified as a challenge that RiVal [30] is meant to address.

The software engineering aspects of RS thus remain mainly an untouched problem whose challenges range from requirements and specification to validation and verification and are not covered by generic programming platforms.

This position paper intent to highlight the pre-requisite related to the software engineering of recommender systems to be deployed in a critical context. Next section introduces briefly the main characteristics of recommender systems and highlights their role as automation of users' tasks. The following introduces both the challenges raised by critical systems engineering and the tools and methods that have to be used to ensure adequate levels of dependability. Through this list, we elicit a list of requirements that recommender systems should meet to be eligible for deployment in critical context. Last section presents the ECAM concept of civil aircrafts and highlights how its functioning could be extended to cover more functions of recommender systems.

2 Brief Introduction to Recommender Systems

This section presents the characteristics of recommender systems and makes explicit their role as automation of users' tasks.

2.1 What are recommender systems

Recommender Systems (RSs) are "software tools and techniques providing suggestions for items to be of use to a user or a group of users" [6, 15, 28]. The RSs provides support for the process of decision-making for a user or a group of users. Recommendations are predictions of the most suitable items based on user's preferences [28]. An "Item" is the general term referring to what the system recommends to users [28]. They also argue that a RS normally focuses on a specific type of item (e.g., product, news or command) and that the user interface has to be designed accordingly.

Recommender systems implement one of the following recommendation methods [5]:

- Memory-based: heuristics that make rating predictions based on all rated items by the users.
- "Model-based": a machine-learning algorithm updates a model from rated items to make rating predictions.

Types of RSs Recommender systems can implement several types of filtering, following the sources of information and algorithm used for the filtering. These filtering can be: collaborative filtering (e.g. [13, 20]), content-based filtering (e.g. [9]), knowledge-based filtering (e.g. [13]) and hybrid filtering (e.g. [16, 26]) or demographic filtering (e.g. [9]) that predict absolute value of ratings.

In addition, other recommender systems predict relative preferences for users and not absolute values. Jerbi et al. [18] presents a preference-based recommender system that uses the past queries of the user to generate recommendations.

The RSs include more and more criteria to improve the quality of recommendations and take into account the context [6]: real-time recommendations (e.g. the one used by Amazon [21]), location awareness (e.g. [10]) and recommendation for a group of users (e.g. [8]) for example.

Tasks when interacting with a RS Recommender systems provide support for particular types of user goals. The tasks to be performed in order for the user to reach their goal are both generic (i.e. supported by most recommender systems e.g. "browse recommendations") and specific (i.e. only supported by a given recommender system e.g. "find director of the recommended movie"). List of generic tasks are provided in the literature [28] and can be seen as generic requirements for recommender systems in order to support users goals. When using a RS, users target the accomplishment of the following goals:

- Find some good items / Find all good items
- Find a recommended sequence of items
- Find a recommended bundle of items
- Browse the proposed list of items
- Look in detail at one recommended item
- Annotate in context the item under consideration
- Improve my profile / Express myself
- Help others / Influence others

2.2 Recommender systems as partly-autonomous interactive systems

Recommender systems execute functions that users were previously performing on their own. Fig. 1 depicts the functions performed by the RSs with respects to the stages of human information processing:

- *Sensory processing stage*: RS filters, ranks and highlight recommended items. Localization of the information from the recommender system might deeply affect that sensing.
- *Perception/working memory*: RS reminds items to the users by duplicating them or recalls items by repeating them. In addition, RS filters items and presents only the relevant items for the user or the group of users. Human errors such as interference, overshooting a stop rule... [27] and thus should be avoided (and if not possible shall be detected, recovered or mitigated).

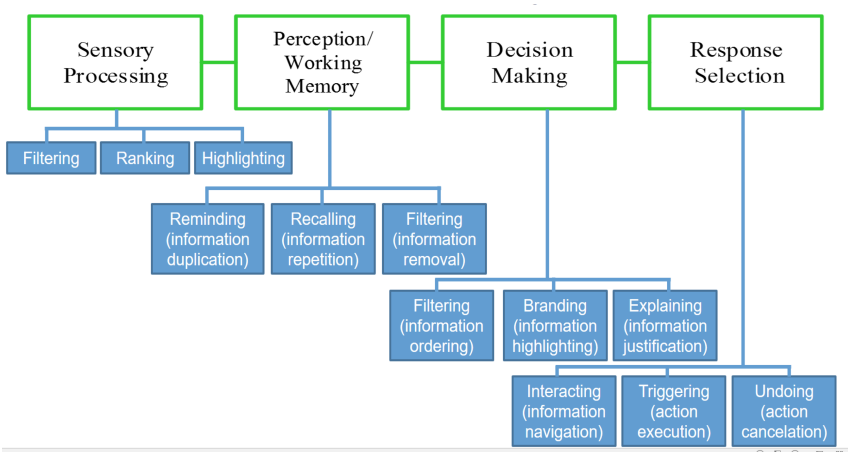

Fig. 1. RSs automation of Four-stages model of Human information processing.

- *Decision-making*: RS filters recommendation by ordering information about items presented. It brands one or several recommendations by highlighting information that are interesting for the user.
- *Response selection*: RS enables the user to select one of the recommendation.

2.3 Recommender Systems in Critical Context

Currently, there is no recommender system deployed in critical contexts. However, as detailed above, recommender systems present multiple benefits that can be attributed to automation by means of the migration of function from the operator to the Recommender System itself. Depending on the type of recommender system, these functions can cover: support to perception of information, support to identification of potentially relevant elements, support to the selection of one element amongst a list... These automation means could be useful in supporting operators' activities in critical context. For instance, in the avionics domain, aircrafts pilots have to manage all of the aircraft systems through the cockpit (the flight deck). The cockpit is a really complex environment and the use of automation is mandatory to support the pilots' activities. The use of recommender systems in this context may, for instance, help the pilots in choosing an action from a multitude. The example of how a recommender system may be useful in an aircraft cockpit is detailed in the last section of this paper.

3 Critical Systems Engineering

This section highlights the software engineering and dependability context of critical systems development. We use this presentation as a mean for identifying 12 requirements to address when engineering recommender systems for a critical context.

3.1 Dependability for critical systems

Building dependable critical systems is a cumbersome task that raises the need to identify and treat the threats that can impair their functioning. In the perspective of identifying all of those threats, Avizienis et al. [7] have defined a typology of all the faults and errors that can impair a computing system. This typology leads to the identification of 31 elementary classes of faults. Fig. 3 presents a simplified view of this typology. It makes explicit the two main categories of faults (top level of the figure): i) the ones occurring at development time (including bad designs, programming errors,...) and ii) the one occurring at operation times (right-hand side of the figure including user error such as slips, lapses and mistakes as defined in [26]). This Figure organizes the leaves of the typology in five different groups, each of them bringing a different issue that has to be addressed:

- *Development software faults (issue 1)*: software faults introduced by a human during the system development.
- *Malicious faults (issue 2)*: faults introduced by human with the deliberate objective of damaging the system (e.g. causing service denial or crash of the system).
- *Development hardware faults (issue 3)*: natural (e.g. caused by a natural phenomenon without human involvement) and human-made faults affecting the hardware during its development.
- *Operational natural faults (issue 4)*: faults caused by a natural phenomenon without human participation, affecting the hardware and occurring during the service of the system. As they affect hardware, they are likely to damage software as well.
- *Operational human-errors (issue 5)*: faults resulting from human action during the use of the system. These faults are particularly of interest for interactive system and the next subsection describe them in detail.

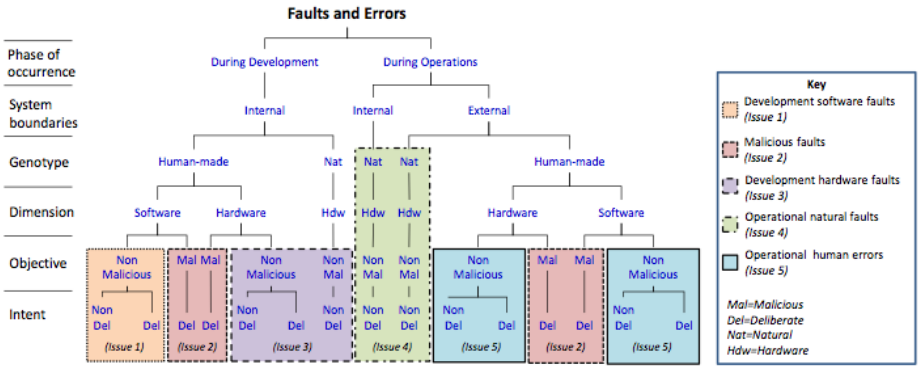

Fig. 2. Typology of faults in computing systems (adapted from [7]) and associated issues for the resilience of these systems.

We consider that development hardware faults (that are more on the electronic side of computing) and malicious faults (that are a separated concern in the avionics domain for now) are beyond the scope of this paper. Each remaining branch of this classification of faults calls for specific engineering methods, processes and tools to be used and followed for developing recommender systems to be used in a critical context.

3.2 Regulation for critical systems

Several types of standards rule the development and operation of critical systems. In this article, we focus on standards that define processes and methods for the development of interactive software applications in aeronautics.

Development Assurance Level	**Failure condition categories**	**Description of the failure conditions**	**Failure rate (failures/hour)**
A	Catastrophic	Failure conditions that may cause a crash	Extremely improbable 10^{-9} +fail safe
B	Hazardous	Failure has a large negative impact or performance, Or reduces the ability of crew to operate the plane	Extremely remote 10^{-7}
C	Major	Failure is significant, but has lesser impact than hazardous	Remote 10^{-5}
D	Minor	Failure is noticeable, but has lesser impact than Major	Probable 10^{-3}
E	No safety effect	No impact on dependability	Any range

Table 1. Development Assurance Level for civil aircraft.

Regulation for software development DO-178C [3] defines Development Assurance Levels (DAL) for commercial software-based aerospace systems. These levels correspond to failure condition categories defined by certification authorities such as EASA (European Aviation Safety Agency) or FAA (Federal Aviation Administration). Table 1 presents the five Development Assurance Levels associated with their failure condition category and its description (summarized from the EASA CS-25 standard [4]). As presented in the first row of Table 1, a failure having catastrophic consequences (failure condition column) must not occur more often than once per 10-9 hours of functioning (failure rate column).

According to the standard DO 178-C, the first two rows of Table 1 (colored in grey) correspond to so-called **critical** systems while systems in the lower rows

are called **non critical**.

Requirement1: DAL level of recommender system must be identified.

Requirement2: Development methods used for the recommender system must be adequate with the identified level of DAL.

It is important to note that DAL levels can also be ensured by the availability of redundant systems of a lower DAL. This means that systems not developed following DAL A constraints could be used for systems with potentially catastrophic failure conditions, provided it exists redundant systems of a lower DAL.

Interactive systems for flight crew The Certification Specification 25 (CS 25) standard [4] specifies, in its section 1302 named "*Installed systems and equipment for use of the flight crew*", that the cockpit must allow the crew to perform safely all of their tasks and that the cockpit must not lead to error prone behaviors. In addition to requirements, the CS 25 standard provides a list of Acceptable Means of Compliances (AMC), which are acceptable means of showing compliance with the requirements. Fig. 3 depicts an excerpt of these means of compliance for systems that contain automation (see [4], page 2-F-20).

Requirement3: Tasks performed by the operator using the recommender system should be explicitly described.

Requirement4: The functions of the recommender system should support all the tasks identified.

Requirement5: The presentation and interaction with the recommender system must not be error prone.

Requirement6: As the recommender system automates flight crew tasks, the design of this specific automation shall follow guidelines on automation.

Requirement7: The automation behavior should be as dependable as the other part of the recommender systems.

3.3 Model-based approaches for dealing with faults during development

Model-based approaches and in particular formal model-based approaches provide support for the design and analysis of interactive systems. They are a mean to analyze in a complete and unambiguous way the interactions between a user and a system [23]. Several types of approaches have been developed [11], which encompass contributions about formal description of an interactive system and/or formal verification of its properties. For example, developing a system of a DAL A or B now requires the use of formal description techniques according to DO-178C supplement 330 [2]. This supplement defines the overall analysis process (depicted in Fig. 4) that developers of aircraft systems must apply.

In this process, presented in Fig. 4, software engineers have to rely on the software requirements ("shall" statement) to develop a model of the system and, independently, they have to express the same requirements in the format of CTL properties. A formal analysis model integrates the system model and the properties. A model checker takes the analysis model as input and generate a counterexample if a property of the system does not hold.

Requirement8: High-level requirements for the recommender system shall be formally described.

Requirement9: Behavior of the recommender system shall be described using formal methods.

Requirement10: Compatibility between behavioral descriptions and high-level requirements shall be checked using verifications techniques.

3.4 Approaches for dealing with natural faults during operations

The issue of operational natural faults must be addressed, more particularly when dealing with the avionics domain as a higher probability of occurrence of these faults [31] concerns systems deployed in the high atmosphere (*e.g.*, aircrafts [25]) or in space (*e.g.*, manned spacecraft). As the operational natural faults are unpredictable and unavoidable, the dedicated approach for dealing

Displays for Automated Systems

Automated systems can perform various tasks with minimal crew interventions, but under the supervision of the flight crew. To ensure effective supervision and maintain crew awareness of system state and system "intention" (future states), displays should provide recognisable feedback on:

- Entries made by the crew into the system so that the crew can detect and correct errors.
- Present state of the automated system or mode of operation. (What is it doing?)
- Actions taken by the system to achieve or maintain a desired state. (What is it trying to do?)
- Future states scheduled by the automation. (What is it going to do next?)
- Transitions between system states.

The applicant should consider the following aspects of automated system design:

- Indications of commanded and actual values should enable the flight crew to determine whether the automated systems will perform according to their expectations;
- If the automated system nears its operational authority or is operating abnormally for the conditions, or is unable to perform at the selected level, it should inform the flight crew, as appropriate for the task;
- The automated system should support crew coordination and cooperation by ensuring shared awareness of system status and crew inputs to the system; and
- The automated system should enable the flight crew to review and confirm the accuracy of commands constructed before being activated. This is particularly important for automated systems because they can require complex input tasks.

Fig. 3. Excerpt of the Book 2 of EASA Certification Specification 25.

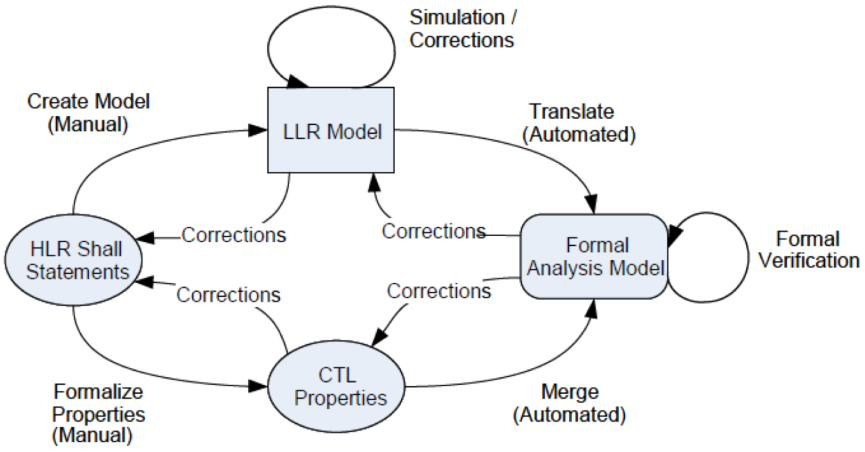

Fig. 4. Analysis process from DO-178C supplement 330.

with them is fault-tolerance [7] that can be achieved through specialized fault-tolerant architectures (such as the COM-MON architecture [32]), by adding redundancy (e.g. [33] or diversity using multiple versions of the same software or by fault mitigation (reducing the severity of faults using barriers or healing behaviors [24]).

Requirement11: Fault-tolerant mechanisms shall be embedded in the implementation of the recommender system (at least to support detection of faults).

Requirement12: Effective fault-tolerance of the recommender system shall be checked using, for instance, fault-injection techniques.

3.5 Summary of the Identified Requirements

In this section, we identified 12 requirements to address when engineering recommender systems for a critical context. These requirements cover both the certification aspects (more particularly in the avionic domain) and the dependability aspects of critical systems. They also cover properties specific to interactive systems (such as usability aspects). It is important to note that the some of these requirements are specific to recommender systems (e.g.requirement 6) while others (e.g.requirement 12) are generic for interactive applications deployed in the cockpit. Finally, while these requirements might not be exhaustive, they depict a first draft of the issues that have to be addressed to deploy recommender systems and call for methods from the area of software engineering, dependable computing and human-computer interaction to address them.

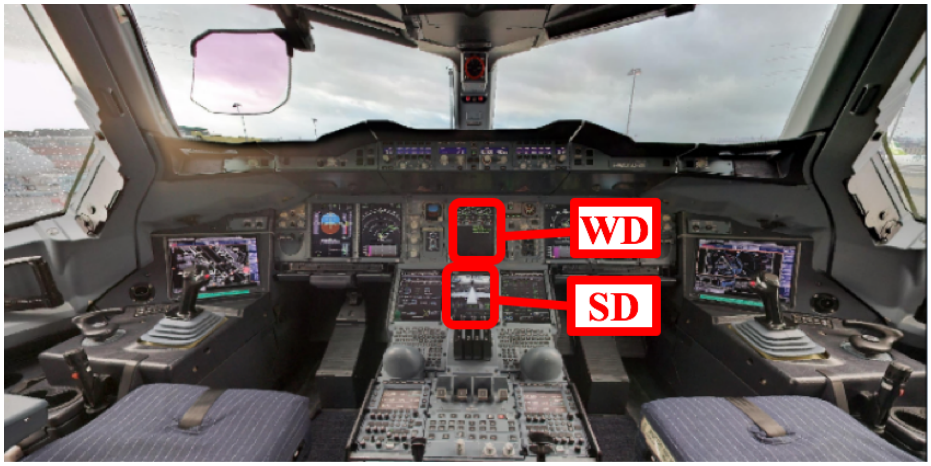

Fig. 5. Warning and system displays in A380 cockpit.

4 An Illustrative Example of Recommender System in Aircraft Cockpit

Currently, there is no recommender system deployed in critical contexts. In the avionics domain, this is due (in particular) to the lack of means to ensure their dependability. However, the concepts underlying recommender systems (e.g. restriction of choices from a multitude) could be useful in supporting operators' activities. This section presents a system from large commercial aircraft that could be a good candidate to become a recommender system in future programs. This section first presents the current concepts of the ECAM (Electronic Centralized Aircraft Monitor) and then highlights what would be required for its engineering in order to embed recommender systems' philosophy.

4.1 Overview of the ECAM

The ECAM, in the Airbus family, monitors aircraft systems (e.g., the engines) and relays to the pilots data about their state (e.g., if their use is limited due to a failure) as well as the procedures that have to be achieved by the pilots to recover from a failure.

The ECAM is in charge of the processing of data from the monitoring of the aircraft systems and produces:

- The display of information about the status of the aircraft systems parameters. In the example of the Airbus A380, this display is done on the System Display (SD in Fig. 5).
- The display of alerts about system failures and procedures that have to be completed by the pilot to manage the detected warning. In the example of the Airbus A380, this display is done on Warning Display (WD in Fig. 5).

Fig. 6. Example of the display of warning messages on the Warning Display.

– Aural and visual alerts - also called attention getters (using several lights and loudspeakers in the cockpit).

Fig. 6 presents an example of the display of warning messages (one "APU FIRE" called red warning on line L1) and its associated recovery procedure on the Warning Display. In this example, the pilots are informed that a fire has been detected within the APU (Auxiliary Power Unit) system. The APU system provides bleed and electricity to the aircraft and consumes fuel. In case of failure, among others, the APU can trigger APU FAULT and APU FIRE alarms. The corresponding recovery procedure first indicates to the pilots that they have to land as soon as possible ("LAND ASAP" indication onL2). Then, they first have to push the "APU FIRE" pushbutton (L3) and shut off the "APU BLEED" service (L4). Then, if the fire is still present after 10 seconds (L5), they have to discharge an agent (L6) to stop the fire and to shut the APU off (L7).

If the Flight Warning System processes simultaneously several warning messages, it sorts them, in order to obtain a display order, according to three mechanisms:

– *Their relationship with others warning messages*: some warning messages may be inhibited in case of presence of others warning messages (for instance, the "APU FAULT" warning message is not displayed if the "APU FIRE" is already detected);
– *The current flight phase*: some warning messages are only displayed when the aircraft is in a given flight phase (for instance, flight management systems failures are not displayed after landing);
– *Their priority level*: a priority level is associated to each alert message in order to prioritize the more critical ones.

4.2 Flight Warning System as a Recommender System

The processing of warning messages (as presented above) is similar to the filtering activity of a recommender system. Indeed, only some of the potential messages are presented to the pilot according to the context (e.g. flight phases or other

messages). However, this filtering could be extended to other aspects (beyond context) such as feedback from other pilots who performed the same procedure (this is called collaborative filtering). Beyond that, FWS only presents one list of procedures at a time to the pilot. Procedures are sorted by order of priority and should be performed in that order by the pilot even though within a given procedure, options are offered. Exploiting recommender systems functionalities would allow presenting alternatives to the pilots that could select the most appropriate procedure to perform according to information that is not present in the system (for instance, in case of a LAND ASAP, the selection of the most practical airport for the airline to repair the aircraft).

5 Summary and Conclusions

To our knowledge, recommender systems have not been deployed in critical contexts. This position paper has first presented how the use of recommender systems in critical context could be helpful to support the operators' activities. The paper has then presented the issues that have to be addressed in order to deploy recommender systems in a critical context. We have made explicit those issues as a list of 12 requirements that call for methods in the area of software engineering, dependable computing and human-computer interaction to address them. Finally, the paper has presented an illustrative example of recommender system in an aircraft cockpit for supporting the pilots' activities while managing aircraft systems failures.

Acknowledgments

This work has been partially founded by the project SEFA IKKY (Intégration du KoKpit et de ses sYstèmes) under the convention number #IA-2016-08-02 (Programme d'Investissement Avenir).

References

1. Netflix users statistics. `https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/`. Accessed: March 2017.
2. DO-333, Formal Methods Supplement to DO-178C and DO-278A. RTCA and EUROCAE, December 2011.
3. DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification. RTCA and EUROCAE, 2012.
4. CS-25 - Amendment 17 - Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes. EASA, 2015.
5. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

6. G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In F. Ricci, L. Rokach, and B. Shapira, editors, *Recommender Systems Handbook*, pages 191–226. Springer, 2015.
7. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, Jan. 2004.
8. J. Bobadilla, F. Ortega, A. Hernando, and J. Bernal. Generalization of recommender systems: Collaborative filtering extended to groups of users and restricted to groups of items. *Expert Syst. Appl.*, 39(1):172–186, Jan. 2012.
9. J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, July 2013.
10. D. Chatzopoulos and P. Hui. Readme: A real-time recommendation system for mobile augmented reality ecosystems. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 312–316, New York, NY, USA, 2016. ACM.
11. A. Dix. *Upside down As and algorithms - computational formalisms and theory.* HCI Models Theories and Frameworks: Toward a Multidisciplinary Science. Morgan Kaufmann, 2003.
12. M. D. Ekstrand, M. Ludwig, J. A. Konstan, and J. T. Riedl. Rethinking the recommender research ecosystem: Reproducibility, openness, and lenskit. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 133–140, New York, NY, USA, 2011. ACM.
13. A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger. Basic approaches in recommendation systems. In M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 15–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
14. Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 305–308, New York, NY, USA, 2011. ACM.
15. I. Garcia, L. Sebastia, and E. Onaindia. On the design of individual and group recommender systems for tourism. *Expert Syst. Appl.*, 38(6):7683–7692, June 2011.
16. M. A. Ghazanfar and A. Prügel-Bennett. A scalable, accurate hybrid recommender system. In *Third International Conference on Knowledge Discovery and Data Mining, WKDD 2010, Phuket, Thailand, 9-10 January 2010*, pages 94–98. IEEE Computer Society, 2010.
17. J. R. Hauser, G. L. Urban, G. Liberali, and M. Braun. Website morphing. *Marketing Science*, 28(2):202–223, Mar. 2009.
18. H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh. Preference-based recommendations for olap analysis. In *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '09, pages 467–478, Berlin, Heidelberg, 2009. Springer-Verlag.
19. B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell. Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):441–504, Oct. 2012.
20. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
21. G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.
22. P. Moradi and S. Ahmadian. A reliability-based recommendation method to improve trust-aware recommender systems. *Expert Syst. Appl.*, 42(21):7386–7398, Nov. 2015.

23. D. Navarre, P. Palanque, J.-F. Ladry, and E. Barboni. Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Trans. Comput.-Hum. Interact.*, 16(4):18:1–18:56, Nov. 2009.
24. S. Neema, T. Bapty, S. Shetty, and S. Nordstrom. Autonomic fault mitigation in embedded systems. *Eng. Appl. Artif. Intell.*, 17(7):711–725, Oct. 2004.
25. E. Normand. Single-event effects in avionics. *IEEE Transactions on Nuclear Science*, 43(2):461–474, Apr 1996.
26. C. Porcel, A. Tejeda-Lorente, M. A. Martínez, and E. Herrera-Viedma. A hybrid recommender system for the selective dissemination of research resources in a technology transfer office. *Inf. Sci.*, 184(1):1–19, Feb. 2012.
27. J. Reason. *Human Error*. Cambridge University Press, 1990.
28. F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011.
29. M. P. Robillard and R. J. Walker. An introduction to recommendation systems in software engineering. In M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 1–11. Springer, 2014.
30. A. Said and A. Bellogín. Replicable evaluation of recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 363–364, New York, NY, USA, 2015. ACM.
31. B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: A large-scale field study. *SIGMETRICS Perform. Eval. Rev.*, 37(1):193–204, June 2009.
32. P. Traverse, I. Lacaze, and J. Souyris. Airbus fly-by-wire - A total approach to dependability. In R. Jacquart, editor, *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, volume 156 of *IFIP*, pages 191–212. Kluwer/Springer, 2004.
33. Y. C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, Feb 1996.