Usability Insights for Requirements Engineering Tools: A User Study with Practitioners in Aeronautics

Annex A: Scenarios

Hélène Gaspard-Boulinc
University of Toulouse - ENAC
Toulouse, France
helene.gaspard-boulinc@enac.fr

Stéphane Conversy
University of Toulouse - ENAC
Toulouse, France
stephane.conversy@enac.fr

ABSTRACT
This technical note is an annex of the 25th IEEE Requirements Engineering Conference 2017 article:

Gaspard-Boulinc H. and Conversy S. 2017. Usability Insights for Requirements Engineering Tools: A User Study with Practitioners in Aeronautics, in Proceedings of RE'2017, IEEE.

In the following sections, we present scenarios and usability insights for each identified context of use. We deliberately report details in the scenarios in depth, in order to provide materials for future research. The content and the chosen form may give the reader the wrong impression that the scenarios are anecdotal, whereas they rather reflect the realism of our empirical data. Scenarios are coded in italics. The usability insights are referenced in brackets throughout the scenario and detailed at the end of the scenario, ensuring that they can be traced up to their origin.

COLLABORATIVE EXPLORATORY DESIGN WITHOUT SPECIFIC TOOL

John is a requirement engineer and works on a new service for air traffic control.

Eliciting design questions by drawing (EDQ)

John uses a drawing editor, such as Visio, to build a global behavioral diagram. He uses this drawing as a means of reasoning to elicit the questions to be solved step by step in a comprehensive manner. Requirements engineers prefer to use drawing editors instead of dedicated system modeling tools for two reasons. In fact, they use the drawing as a means of reasoning, thus they want to change the drawings over time very quickly [EDQ_2]. They only draw simple boxes and arrows, and do not need any other tools that could come in the way [EDQ_1]. *In parallel, John writes the design questions in an Excel file, with a reference to the initial system requirements.* Since the user considers Visio and Excel as two separate applications, there is no direct link between the diagram and the requirements, and design questions listed in the spreadsheet: the links are only in the engineer's mind [EDQ_3 & 4]. *Sometimes, John does not use the drawing to discover the design questions, but writes them directly in the appropriate column on the chart, just by reading the requirement*. This means that there is no pre-definite sequence between requirement, drawing and design question, depending on requirement maturity [EDQ_5].

In summary, in order to elicit design questions raised by a set of initial system requirements, the system shall allow the requirement engineer to:

EDQ_1. Draw behavioral views of the system using boxes and arrows (effectiveness: no need to remember or comply with semantic meaning of notation)

EDQ_2. Change behavioral views of the system with a minimum number of actions (efficiency in editing actions to support exploration)

EDQ_3. Create a design question from an element (box or arrow) of the behavioral view with a minimum number of actions (efficiency to foster traceability)

EDQ_4. Edit and visualize a tabular list of design questions, linked to the initial system requirements (design: the tabular list of the design questions is a complementary view of the behavioral view)

EDQ_5. Edit requirement, drawing and design question in any order in any view (behavioral or tabular list) (effectiveness in the number of elicited design questions)

Sharing and discussing the design questions and drawings (SDQ)

Some time later, John sends the drawing and the Excel file by email in order to discuss them with the component suppliers during a meeting. Today, the meeting is by videoconference, because one of the component suppliers is based in UK [SDQ_1]. *The meeting goal for John is to present his drawing and the identified design questions to the component suppliers, and identify with them whether "they already have a solution that implements the function on their own side". During the meeting, the questions are addressed step by step through the Excel document, allowing them to build a common understanding of the problems and challenge the technically feasible solutions. The possible solutions are not recorded during the meeting but detailed and discussed by email afterwards*. There is no direct link between the email exchanges and the design

questions of the Excel document [SDQ_2]. *As a result of the electronic discussions, John modifies a system requirement (the defined performance is not achievable): he writes "WAS+ old statement" and "IS + new statement in the Word document."* He does not use the change-tracking mode of Word, because he wants to highlight content change only (not formatting change) and in a persistent way [SDQ_3].

In summary, in order to collaborate around design questions raised by a set of initial system requirements, the system shall allow the requirement engineers and the component suppliers to:

SDQ_1. Share the tabular list of design questions and the behavioral view in a co-located or distributed meeting (efficiency in sending and reading the data)

SDQ_2. Discuss each design question in any view (behavioral view or tabular list) (efficiency in editing and accessing discussions to foster traceability)

SDQ_3. Change a requirement statement in a visible and persistent manner (design: WAS+ old statement" and "IS + new statement) linked to a discussion (effectiveness: traceability of changes)

EXPERIMENTING REQUIREMENTS ANALYSIS WITH A SYSTEM MODELING TOOL

Alex is a requirement engineer and is involved in an R&D project in order to apply system modeling on crisis management.

Editing a system model (ESM)

She is editing the model of Figure 1. *She wants to add an output on the rectangles of the right border and access the corresponding class to modify all instances. However, in order to position the output graphically on the rectangle side, she has to perform this operation for each rectangle: "*this is box by box, macros are missing*". In addition, since the rectangles are too close to the window border, she has to move each rectangle on the left.* This is a basic usability problem known as viscosity (resistance to change), and can be related to the user preference for drawing editors (see EDQ_2 in previous section) [ESM_1]. *Then Alex wants to refine the requirement. She double clicks on the corresponding graphical object, triggering the display of a pop-up window with 11 tabs*: "we have the feeling that properties are classified in reverse order we need" [ESM_2]. *The work on requirements is not linear: Alex is not able to fill in the properties at once. She did not work on the system model in the previous week, since she was involved on another project. She does not remember the properties she has already filled, and the properties she has to complete. She browses the different already modeled elements in order to display the properties window and check the many tabs.*

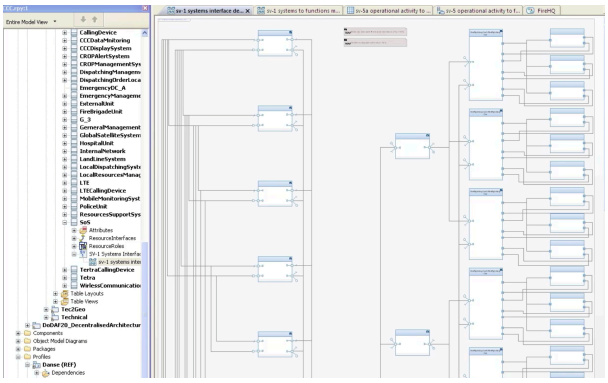

Figure 1: An example of system model

This represents a loss of time remembering the work done, and the work to be done, which can be worse if several engineers work on the same model [ESM_3].

In sum, in order to edit a model of a system, the system shall allow the requirement engineer to:

ESM_1. Modify graphical properties of a set of objects with a minimum number of actions (efficiency in editing actions to foster modeled alternatives)

ESM_2. Search and edit properties by key word from a graphical element (efficiency of search action)

ESM_3. Retrieve the completion status of properties from a graphical element (efficiency in filtering)

Communicating an existing model

Alex wants to explain the model to the project stakeholders. She produces a video that integrates an illustrative description of the different actors, since she considers that the resulting model is difficult to read (see ***Erreur ! Nous n'avons pas trouvé la source du renvoi.****) as a standalone product*: from left to right, rectangles indiscriminately represent town neighborhoods, fireman headquarters, districts and firemen. The issue of visual representation has already been identified by previous research. Our scenario reinforces the need to modify SE notations as proposed by Moody in order to improve their readability.

CES_1 In order to support model reading, the system shall allow any engineer to discriminate the different graphical elements in the model with a minimum number of actions (efficiency in reading)

REQUIREMENT REFINEMENT AND MANAGEMENT WITH A SPECIFIC TOOL

The team is composed of ten engineers. Claire, Fred and Phil are members of the team. They use the DOORS software (see a screenshot in ***Erreur ! Nous n'avons pas trouvé la source du renvoi.****) to edit and manage requirements. They have two screens in order to display DOORS in full screen.*

Writing consistent requirements (WCR)

Claire is writing a requirement: she is constantly switching between the data model and the DOORS tool in order to use the right words in the requirement statement. The domain concepts to be used are defined within a data model, using

another dedicated tool. Therefore, the data model is not integrated into the main tool. [WCR_1]. *An internal rule specifies that the domain concepts must be in italics in a well-formed textual requirement, in order to ease the reading afterwards. To comply with this rule, Claire manually changes the character formatting of all domain occurrences* [WCR_2]. *Fred is a newcomer in the project and creates a requirement stating "interact with the flight sequence timeline", while an existing requirement already states "manipulate the flight sequence timeline".* The consequence is that two different requirements express the same need. Requirement engineers usually rely on their expertise on the system to detect this situation. Due to long service life and inevitable staff turnover, this is not a sustainable solution [WCR_3].

In order to write consistent requirements, the system shall allow the requirement engineer to:

WCR_1. Select and edit words of the data model during requirement statement edition (efficiency in editing action + effectiveness: the words are compliant with the data model)

WCR_2. Facilitate highlighting of words of the data model in requirement statement (efficiency in editing action)

WCR_3. Detect requirement similarities during requirement statement edition (effectiveness: no redundancy).

Conducting coverage analysis (CCA)

Now, Claire wants to read and verify the refined requirements written by Phil, in order to check compliance and coverage. Starting from a system requirement, she selects an attached yellow arrow. Whatever the number of refined requirements, the representation is the same arrow, which conveys a feeling of flat structure, reported as unclear by participants [CCA_1]. *The selection of the arrow triggers the display of a menu with the file paths (ex: /Project/Baseline/Evolutions/R347), which does not help Claire to understand the requirement content* [CCA_2]. *Then Claire selects the refined requirements one by one in the menu,* making difficult the verification of the global compliance with the initial requirement [CCA_2]. *Moreover, Claire has to remember which requirement she has already read in the list, in order to move to the next item* [CCA_4]. *Sometimes, in order to deal with this situation, Claire generates a matrix using the DOORS.* However, when the refined requirement is too long, the initial requirement is not displayed anymore, which implies a loss of time going back and forth in the matrix [CCA_3]. *Claire has detected that a refined requirement must be clarified, but she cannot perform the change directly on the matrix.* S*he has to find the requirement in the DOORS tool, from the requirement identifier,* which again implies a loss of time [CCA_4].

In summary, in order to conduct coverage analysis, the system shall allow the requirement engineer to:

CCA_1. Visualize the overview of requirements without any action (efficiency in overview understanding) (treemap as an example of design)

CCA_2. Access the requirements of a given level with a minimum number of actions from the overview visualization (efficiency of navigation) (treemap as an example of interaction design)

CCA_3. Read all the refined requirement statements and the initial requirement statement always displayed in its entirety (efficiency in reading) (treemap as an example of design)

CCA_4. Access direct actions on a requirement from the overview visualization: validate, reject, comment, modify (efficiency) (treemap with right click actions as an example of interaction design).

CCA_5. Identify the requirements not refined at a glance on the overview visualization (efficiency in anomaly detection) (black requirement on the treemap as an example of design).

Managing the content of system releases (MSR)

There are simultaneously four releases under definition for the system, chronologically named V9, V9.1, V9.2 and V9.3. This is a usual situation, which allows the steering committee of the project to prioritize the numerous evolutions requested by the users of the system and the ever-changing regulatory context. Within the RE team, Phil establishes and maintains the matrix Figure 2 *for each system release under study. He has listed the evolutions horizontally at the top, and the system components vertically. If the evolution has an impact on the component, Phil reports "1" in the corresponding cell. The steering committee complains about this visualization, but Phil is puzzled:* "I do not understand, you can find any information you want in this matrix"[MSR_1].

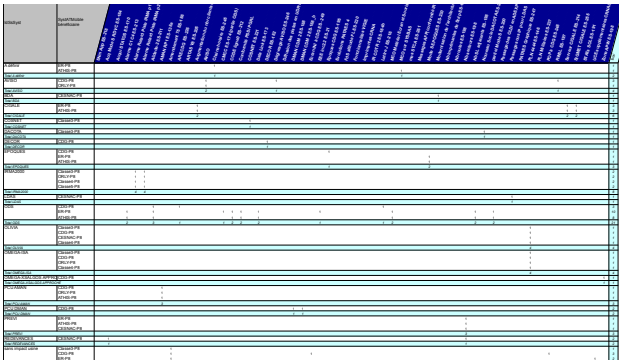

Figure 2: A matrix built by a requirement engineer

The RE team has concurrently evaluated the impact of 10 main evolutions (from E-A to E-J) on the system. The E-A evolution is planned in the V9 release, while the E-D evolution is planned in the subsequent V9.1 release. Claire is modifying the R1 requirement, linked with E-A and applicable in V9, while Fred is concurrently modifying R1, linked with E-D and applicable in V9.1. Since V9.1 is subsequent to V9, E-A change of R1 is inserted in the E-D changed R1. Configuration management services integrated into the tool allow the two requirement engineers to be aware of and manage the concurrent modifications. *Meanwhile, the steering committee decides to change the*

priorities of the evolutions: E-A becomes lower priority and will be consequently applicable in the later V9.2 release.
The consequence is that the chronology between E-A and E-D evolutions is modified: "we have to unknit the work done". *Phil has to unravel the work done on the R1 requirement to obtain consistent baselines for V9, V9.1 and V9.2 releases of the system: modify R1 applicable in V9 (remove change due to E-A and the related trace), modify R1 applicable in V9.1 (remove change due to E-A and the related trace, check change due to E-D and the related trace), and finally modify R1 applicable in V9.2 (integrate change due to E-A with the related trace). Phil has to proceed for all the requirements impacted by the E-A evolution: three weeks of work are required to deal with 182 requirements* [MSR_2]. This is a typical requirement management task, which is considered daunting by the engineer. However, this is a mandatory task in order to keep requirement baselines consistent for certification purpose. The required effort is huge and tedious, while the Agile process movement praises parallel development and greater requirement flexibility.
In summary, in order to manage system release flexibility, the system shall allow the requirement engineer and the steering committee to:
MSR_1. Visualize the evolutions under study and the corresponding impacted components (efficiency in selecting and reading)
MSR_2. Reschedule evolutions over time in a minimum number of actions (efficiency in tracing).

PROTOTYPES AND DESIGN WALKTHROUGH RESULTS

Before detailing the results of the design walkthroughs, we briefly describe the prototypes that we designed and showed to 7 participants.

ReqMap, ReqSpector and ReqChord

After exploring several visualizations (such as the collapsible tree and the rotating cluster layouts), we have worked on the Treemap visualization: the entire system occupies the global area, subdivided into rectangles that represent the requirements (see Figure 3).

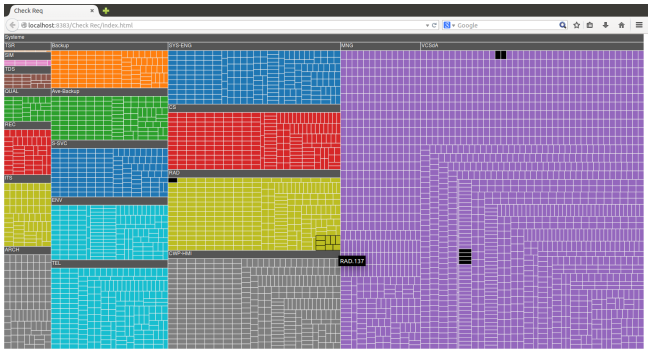

Figure 3: ReqMap: the treemap visualization for requirements

The space allocated to each category is related to the number of requirements. Requirements that have not yet been refined are coded in black, showing the "requirements hole". The refined requirements of a same system requirement form a group on the treemap and the ID system requirement is displayed in a tooltip (in the center of Figure 3). One can navigate through the structure of requirements by selecting the title bar and can then focus on a level. At the lowest level, we display the requirement text at the top and the refined requirements below (one rectangle for each refined requirement). In addition, we make some actions (verify, reject, modify, comment) available on each requirement by right clicking.
We have also designed an interactive inspector for multiple objects, named ReqSpector, and integrated it into the treemap visualization (Figure 4).

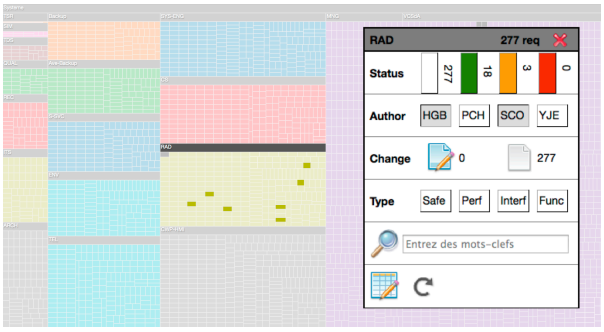

Figure 4: ReqSpector, the (magnified) requirement inspector

The inspector can be triggered on the system level, on the category level or on the requirement level, by clicking on the text label: it shows all property values used by the set of corresponding requirements. For instance in Figure 4, the inspector displays all requirement property values for the RAD category (e.g. author: HGB, PCH, SCO, YJE). By toggling values, one can finely tune the filter: in the figure, only the requirements written by the toggled authors (HGB, SCO) are highlighted. One can access the tabular visualization of the resulting filtered requirements by clicking on a tabular icon at the bottom of the inspector. We have integrated a search function in the inspector tool to support consistency checking, as described in WCR_3. For instance, if one enters "flight sequence timeline", all requirements that contain the group of words are highlighted on the treemap and can then be displayed in a tabular visualization to support comparison and detect similarities.
Finally, the chord diagram visualization is an alternative graphical method to display matrices. We have adapted it in Figure 5 as an alternative to the matrix in Figure 2. System evolutions (in red), system components (in blue), and operational sites (in green) are arranged radially around a circle. The relationships between them are drawn as arcs connecting the data together (left on Figure 5). The selection of a given evolution highlights the system components that implement it (right on Figure 5).

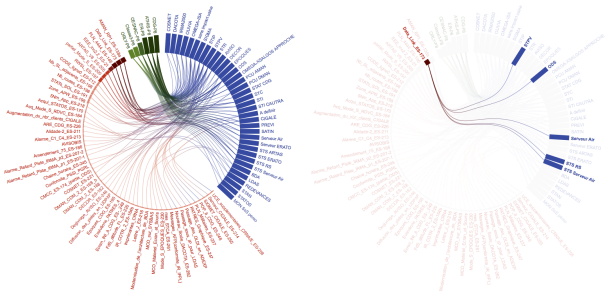

Figure 5: ReqChord: the chord diagram visualization for managing the content of system release

We have presented ReqMap, ReqSpector and ReqChord to seven participants as a shared visualization between the different engineers. Participants' feedback allowed us to (1) reinforce the identified usability insights with examples of design and (2) discover new usability insights.

Consolidation of Usability insights on Conducting Coverage Analysis

Participants were enthusiastic about ReqMap and we analyzed the reasons why the treemap visualization is adequate: it allows them to see the overview of the requirements, focus on a level, and read the system requirement and the refined requirements at the same time for compliance checking [CCA_1 CCA_2 CCA_3]. The available actions on requirements were evaluated as helpful for refinement of requirement while reading [CCA_4]. The link with the data model was also addressed through discussions on the right clicking actions [WCR_1]. The black hole feature was evaluated as worthwhile for anomaly detection at a glance, in comparison with the tedious matrix reading [CCA_5]. The next step for participants was to deal with the identified anomalies. This is compliant with the visual information-seeking mantra "overview first, zoom and filter, then details on-demand" [**Erreur ! Nous n'avons pas trouvé la source du renvoi.**].

Discovery of usability insights on Filtering (IFR)

We conducted discussions about anomaly detection and processing thanks to ReqSpector, which led to new usability insights. Participants' feedback shows that the status (modified-commented-rejected) filter and the direct feedback on the treemap were valuable to get an overview of the verification work completion [IFR_1.3]. The "author" property was evaluated as a valuable filter, because crosschecking is frequent in final verification [IFR_1.2]. Tabular visualization of filtered requirements was evaluated as helpful to process the filtering results step by step and work directly on the tabular visualization [IFR_2]. Through participants' discussions, it appears that the search feature, firstly designed for consistency checking, can also support impact analysis in order to identify the nature and number of requirements that must be modified to implement an evolution [IFR_1.1]. Other filters have been identified through discussions such as the requirement type or the change status [IFR_1.3][IFR_1.4]. However, we do not claim to provide a full list of filters here, only the ones that are necessary to conduct coverage analysis. Meanwhile, participants reported that the DOORS tool has filtering features, but they need to ask for the filter from an administrator. The filter is then available in a list. This is why it can be quicker to export the data in an Excel file and use the Excel column filters.

In sum, the system shall allow the requirement engineer to filter requirements interactively with immediate feedback on the overview visualization, at any level (efficiency in filtering) (inspector and treemap as an example of design):

IFR_1.1 By keywords, in order to process consistency checking and impact analysis

IFR_1.2 By author, in order to process crosschecking.

IFR_1.3 By completion status, in order to resume the work and evaluate the work progress.

IFR_1.4 By type (safety, performance, functional, interface), in order to prioritize the checking process (safety requirements are mandatory first)

IFR_2. In order to handle filtered requirements, the system shall allow the requirement engineer to display requirements on-demand after interactive filtering in an editable tabular visualization (efficiency in handling anomalies).

Discovery of Usability Insights on Managing the content of System Release (MSR)

The participants evaluated as helpful ReqChord in the context of communication with the steering committee [MSR_1]. They proposed to add extra information for a better decision support, such as requirement statistics of the evolution (number of new requirements, number of modified requirements, number of unchanged requirements). It could also improve the impact analysis of a new system versioning, by browsing from one evolution to another [MSR_3]. The participants imagined this visualization being used for communication within the RE team [MSR_4] to get a better awareness of the multiple evolutions under study. Finally, they were willing to use this visualization as a filtering tool, allowing the extraction of all the new/modified requirements related to an evolution, classified by system components "by drag and drop of the selection in a separate window"[MSR_5].

In sum, in order to manage system release flexibility, the system shall allow the requirement engineer and the steering committee to:

MSR_1. Visualize the evolutions under study and the corresponding impacted components (efficiency in selecting and reading, chord diagram as an example of evolution visualization design)

MSR_2. Reschedule evolutions over time in a minimum number of actions (efficiency in tracing).

MSR_3. Browse from one evolution to another with requirement statistics (new, modified, unchanged) for impact comparison (efficiency in browsing, chord diagram as an example of interaction design)

MSR_4. Share evolution visualization (effectiveness in team cognition)(chord diagram as an example of design)

MSR_5. Filter the initial requirements and the refined requirements by component (efficiency in filtering)(chord diagram as an example of interaction design)

DISCUSSION

The analysis of our scenarios allowed us to identify five aspects that significantly change over time during the process: traceability, requirement maturity, collaboration, system behavior representation and user satisfaction. The assessment of these five aspects reveals two opposite states over time, as shown Figure 6.

Creativity supported by flexible tools

At the early stage of the process (context of use 1), the requirements engineers rapidly face the problem of design, i.e. "the satisfactory matching of a system structure to a specific goal to be pursued by the system". Their work is collaborative and centered on design questions with component suppliers. They draw system behavior representations as a reasoning means to discover the design questions. General-purpose desktop tools are used to support this work: drawing editors or whiteboards to express behavioral views and elicit design questions, Excel or PowerPoint to record the design questions, email and videoconferencing to share them. Requirement engineers are very satisfied by this work: they consider the design as the core activity. They do not complain about tools since they use pliant tools i.e. flexible and responsive. However, the different files produced are stored in a common repository at best, but more frequently on personal computers. The requirement engineers do not spend time upgrading the official tools with their separate records. The consequences are dispersion and loss of information in terms of pre-traceability of requirements.

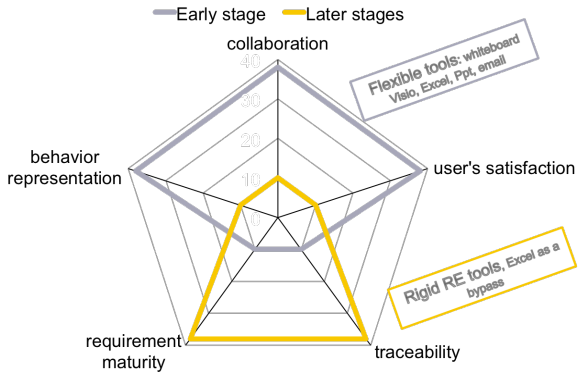

Figure 6: Two opposite states of RE activities over time

In order to improve traceability at the early stage of the process, RE tools should support the team to perform the tasks of context of use 1 currently performed with other tools: elicit design questions by drawing behavior representation of the system, share and discuss design questions, and create/update requirements from design questions. To this end, we have formulated the usability insights EDQ_1 to EDQ_5, SDQ_1 to SDQ_3 and ESM_1 to ESM_3.

Rigor supported by rigid tools

At later stages of the process (contexts of use 2 and 3), the requirement engineers' work is centered on requirement refinement and verification. The requirement engineers express much dissatisfaction at these stages: tasks are evaluated as routine and administrative, characterized by less creativity and more rigor. They use dedicated tools and Excel as a bypass to perform this work. As shown in our scenarios, dedicated tools constrain engineers to a rigid and inadequate workflow, yielding significant time loss. Moreover, the collaboration with the component suppliers through the tool-generated documents is not effective: the listing of the refined requirements conveys a piecemeal approach of the system, which makes the documents unreadable to anyone else but the writer. The consequences are loss of motivation and loss of time in establishing and maintaining traceability of an overwhelming number of requirements. To this end, we have formulated usability insights, with the help of prototypes, related to visualizations, (CCA_1 to CCA_5, MSR_1 to MSR_3), filtering and editing actions (IFR_1 to IFR_2, MSR_1, MSR_5).