A Framework for Extending Usability Engineering

API Usability Essentials: Extending Usability via Component-Based Platform

Muhammad Bilal Munir
Computer and Software Engineering Department
Bahria University Islamabad, Pakistan
bilalmunir80@gmail.com

Arif Mushtaq
Computer Science Department
Bahria University Islamabad, Pakistan
mushtaq_arif@yahoo.com

Abstract — Application Programming Interface (API) in software development acts as an important milestone for software productions. It is believed that API usability impacts upon ease-in-use, operationability and acceptability among its audience. Likewise, an ever increasing need for extending and integrating Usability Engineering (UE) has become vital for the success of software products. Earlier researches within this domain do not address API's usability via a component-based framework approach. The proposed framework emphasizes on consolidated formulation of various usability and quality models to derive chunks of dimensional variables. Further the paper highlights API usability practices and heuristics applied in API development process and discusses API product's artifacts component to be used in deriving further product-related components to support enhancing usability.

Keywords – Extending Usability, API Usability, Usability Engineering (UE) and Software Engineering (SE) Integration, Component-Aided Usability, Model Consolidation.

I. INTRODUCTION

Extending usability engineering to software development processes that are deficient in addressing usability and quality integration carries a great deal of importance; not doing so in general can be a major reason for software failure [1][2][3]. Addition of usability-guided features to a software design is a source of providing quality, user satisfaction, ease-in-use and fault tolerance [2][10][14]. Working on API-Usability holds a great deal of importance and it aids in enhancing software success through filtering adversely affecting features, hence supporting the software design with a stable base to deliver an optimized usability impact [4][8][12]. The APIs act as raw materials to the software development processes so subjecting them to usability practices is a useful way of identifying dysfunctions before its consumption cycle begins. Likewise, adapting to API-usability practices ensures prevention from design anomalies that may occur at later stages, thus giving software products greater reliability and acceptability rates [6][19]. The API usage in software development is cost beneficial if they are written in a method that targets generic, dynamic, testable and reusable approach, this way APIs and dependent software can undergo maturity with relatively lesser resources and effort [4][9][11][20].

With the objective to resolve quality and usability issues, a usability and quality integration framework - *Features Relative Usability Illustration Technology* (**FRUIT**) is proposed. The proposed model focuses on defining and delivering the manageability and applicability of usability engineering disciplines over a component-definitions based platform. The proposed FRUIT-framework is aided with various establishment and utilization details. This is to give an insight and coverage of how it addresses to usability engineering disciplines via quality and usability models' consolidated approach, in combination with component constructs' vocabulary. This model's major emphasis is on moving a step ahead of ad-hoc usability disciplines' practicing which it does by integrating organizational processes through its provisions that allows usability and quality experts to manage, guide and control from a centralized platform.

II. LITERATURE REVIEW

A. API Usability

Application programming interface (API) acts as interfaces between software modules and developers that provide a baseline to work across software development process. Tulach [19] describes an API in terms of a contract: "The API is everything that another team or application can depend on: method and field signatures, files and their content, environment variables, protocols, behavior." Reusable API can be considered as a potential support to line of software products from a developer's perspective. Since APIs work in the background of Graphical User Interfaces (GUI) they provide a direct means of transferring whatever can be offered via design, highly usable APIs make the integration of usability practices to UI easier and faster [4][6][19].

A very few API usability design guidelines have been discussed in literatures compared to User Interface GUI designs. According to Arnold, the APIs might not appear to be User Interfaces (UIs) when dealt with; they rather act as a human-computer interface to software developers who write an application. Integration of third-party technologies into software solutions requires these technologies to be stronger usability-wise. Henning notifies that consequences of using improperly written APIs can be serious to software developers, as it would be time consuming in grasping them, hence delaying their proper usage. Thus, if an API is improperly designed and not filtered through usability disciplines, it becomes difficult to determine whether the time and effort saved is worth the investment done on API. Since APIs are used as building blocks for software development, their usability will be beneficial for delivering

the initial software's requirements and across entire software development lifecycle [2][6][10][11][19].

B. API Usability Essentials

The International Organization for Standardization (ISO, 1998) defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [19]. Ineffective, inefficient, and unsatisfying API usability adversely affects software product's cost, since the developers will have to deal with all sorts of usability problems that could have been tackled at an earlier stage. Cooper in 'the hidden costs of bad software' notifies that if API usability is not addressed thoroughly, it can lead to a series of cost issues that will give birth to decreased productivity and increased customer support costs, moreover dissatisfied end user [3][4][6][11][19].

The working features of well-designed API should remain consistent no matter what programming languages it has to work with. Code usability concerns at this stage would be to follow standard coding practices, which lead to removal of complexity for better understanding and quicker adaptation to its use by developers without wastage of precious time on learning it, rather allow focusing on software development. Bloch, Henning and Zibran in general emphasize on various elements of a good API that bring the factors of API usability into perspective [19]. Forthcoming elements are derived from the commonly stated practices by the three: Modularization and grouping of similar types of information for easy referencing across code, version numbering updatable implementations according to recommended standards, virtual-machine integral code, cross-platform compatible easy-to-use reusable design, extensive and correct use of design patterns, templates and starter kits make developers life easier. Implementing hardware specifications oriented compact versions of APIs with well documented support and scenario-based examples helps optimize development time, cost and resources. Information designs for components and UI-Controls should feature ease-of-use, organized accessibility, data connectivity, understandable event-handling, look-and-feel and well designed typography from usability perspective [7][16][19][20].

C. API Usability and Technical Writing

Studying API usability involves computer science and software engineering disciplines. It involves writing program code, knowing the user, developing scenarios and tasks, defining usability goals, observing users, and all the other usability research methods. "Everybody on the team needs to take responsibility for improving the user experience" and "anybody can do usability" – Nielsen [19]. The API designers have considerable responsibility for the usability of their design. Furthermore usability support can come from other sources as well, such as technical writers of APIs can also easily adapt to these disciplines. Therefore, allocating technical writing resources to usability tasks at an earlier design stage can support design's usability along with API's documentation which may result in improving the end users' experience with the resulting product [9][10][17][18][20].

D. API Usability Heuristics

The usability of an API can be determined by the use of cognitive dimensions; Green, Blandford, Church, Roast, and Clarke describe Twelve cognitive dimensions, that provide specific descriptions of API usability design issues, that tell how well a developer's preferences relate with the properties of a particular API, along with highlighting the importance of knowing the user and the API using these dimensions in parallel [19]. Microsoft has made use of cognitive dimensions in evaluating the usability of the Microsoft .NET Framework and other APIs [19]. The measurements that result from use of cognitive dimension analysis are not necessarily numerical or even necessarily absolute measurements, instead they act as a ratings scale vocabulary. Green and Petre suggest that the cognitive dimensions framework can be used by usability engineers without any formal training in cognitive psychology [19]. According to Nielsen, usability tests with good usability methods tend to find more issues compared to poorly defined methods, even usability tests with poor methods still identify more usability issues than no testing whatsoever. In order to streamline API project's requirements, design, development and evaluations with usability practices, the usability engineers must work with design teams, derive API usability heuristics aided with checklist in collaboration with the cognitive dimensions framework [6][10][13][15][19].

III. Extending Usability Engineering Via Component Based Platform

The previous section covered the significance of API usability and why it should be processed through usability disciplines. Robert Watson observes from literature that API usability has received notably lesser attention which may be because usability engineering discipline is lesser equipped with well-defined sets of tools and technologies as compared to software engineering [19]. In order to step out of this ad-hocism usability engineering needs to treat itself as its own subject and address to itself as a product via an organized set of software apparatus or a framework. This may enhance its applicability and give it readiness and adaptability to various working scenarios. Such a framework should act as quality and usability focused production-oriented metadata.

FRUIT: This framework is targeted at enhancing usability engineers' work by encouraging them to accomplish their quality and usability related activities over a component based platform. This would help reduce task objective time due to structural nature of component-based referential. Usability Engineers can formulate product's feature using various usability component apparatus that aids in control management, decision making, analysis and design etc. An organization benefits the most when extending component-oriented techniques outside processes other than production. This led to formulation of fundamental production stances under context of quality and usability disciplines. The FRUIT runs in parallel with organizational processes. It revolves around production and manufacturing processes such as software analysis, design, development, testing, feedback and maturity etc. It largely addresses the respective processes' activities via its discipline of derived

components vocabulary constructs and it updates its components' definitions across the software's maturity lifecycle. All this is done to integrate quality and UE practices within SE, thus allowing controlled impact delivery via the component constructs. These components act as company's products' technical specification dictionary providing means of advancements to processes guided-workflow, product's features, production information, standard specifications, evaluations techniques, readings on various analytical reports etc. Component vocabulary can be used in producing technical documents that provide coverage on various product physical dimensions, relating them with their quality attributes. Generically defined component forms can be used to manage various customizations that can help mark the products' and processes' present and absent standards and features in quick and concise manner [6][16][14][15][19].

A. Models' Consolidation Approach

The FRUIT, defines a hybrid discipline which converges the use of various consolidated quality models aligned with combinations from consolidated usability models. This is done to overcome one model's deficiency through support of the others in producing a stance for refining the quality and usability of an undergoing production. For quality models consolidated form the three models namely DIM – Dromey, ISO, Macall are sufficient. In order to include a financial perspective in models consolidation approach for cost-aided extension, the combination of usability cost-benefit models would be useful. Three cost-benefit analysis models can be considered, namely BERK – Bevan, Ehrlich and Rohn, Karat. These three can be used aided with combination of famous usability models namely SINE – Shackle, ISO, Neilson, and Eason.

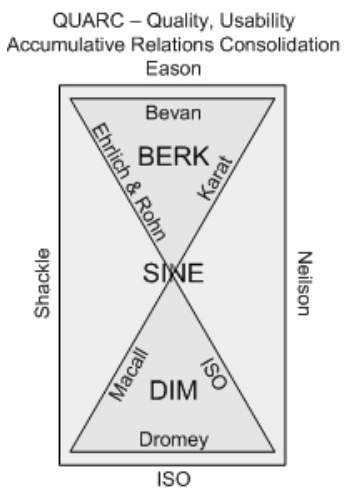

FIGURE 1 - USABILITY HOURGLASS

The models combinations are termed as (**QUARC** – Quality, Usability Accumulative Relations Consolidation**)** and are used to define dimensional variables' chunks whose instances can further address the various observed portions in a design as per requirements. These variables group chunks would therefore carry usability perspectives guided quality impacts across the entire product's lifecycle. Different usability models will help produce variety of crucial design perspectives from the same set of variable groups on a common design platform to ensure controlled design features integration. The usability hourglass (Figure-1) divides the consolidated approach into three groups; DIM and BERK - *the primary and secondary* consolidations partially targeting organization's processes and products. Their major focus is on formulation of cost-effective usability integration with the organization's quality stances and production processes. Moreover, it gives them perception and dynamic adaptivity towards requirement-influenced application within usability disciplines.

B. Component's Categories, Types and Compositions

The FRUIT, component constructs' may have several sources. A primary source of these components is obtained from studying the decomposed forms of generic variables' generation groups (**VERBS** – Variables Engineering Relations' Basic Sets). While they are maintained using collections sets that fall under given sets of general categories (**PLANC** – Processes Lineage Associated Nine Categories) based on what processes they occur from. These categorical distributions may vary according to organization's processes infrastructure and activities distribution. Each category can be divided into several phases depending on direct or indirect relations with production processes. In order to populate an initial level of component constructs and related materials per product, the VERBS are used to divide the under-development system. In order to begin with it may contain nine generic variables' groups, namely: Dimensions (Quality, Usability etc.), Users (Roles, Complexity, Domain), Interaction (UI, System etc), Processes (Workflows, Tasks, Activities etc.), Products (Design, Features etc.), Environments (Tasks, Scenarios, Modes etc.), Technology (System, Inherited etc.), Integrity (Solution, Processes etc.), Resources (Organization's, Client's etc.). The PLANC can be used to categorize the component constructs based on what processes they are acquired from. The given process categories are: Production (Analysis, Design, Development, Testing etc.), Quality Management, Research and Development, Marketing and Promotion, Procurement and Supplies, Human-Resource, Finance, Manufacturing, Business and Sales.

The FRUIT, components can belong to various types (**TIDES** – Type-wise Infrastructure Domain Elaborative Stylistics) depending on detail levels per product to provide sufficient coverage on various design aiding aspects. Derivations are done with quality perspectives at components' roots. Product's constituent component constructs type-wise elaborations may be done based on following: Benchmarks, Metrics, Performance Factors, Metadata, Features, Expert Concepts, Design Patterns, Requirement Elaborations, Templates and Checklists, Milestones, Progress Measurements, Qualitative and Quantitative Studies, Verification and Validation Procedures, Standards and Design Guidelines, Rules and Principles, Tools and Technologies, Artifacts and Deliverables, Process Workflows, Documentation, Profiles and Manuals, Results and History etc. The FRUIT, components can be used to provide a compositional vision per product using derivations with quality perspectives at their roots to provide coverage on various design aiding aspects. Depending on what type of information FRUIT components cover their compositional layout can be elaborated using following Specifiers (**CASTS** – Composition Associated Structuring Technical Specifiers): i.e. Dimensions, Constraints, Attributes, Procedures, Parameters, Collections, Relations, Resources, Definitions, Inputs/Outputs, Evaluators, Properties, Triggers, References,

Structures, Behaviors, Versions, Techniques, Values and Content etc [6][16][14][15][17][19].

C. FRUIT Component Constructs' Classifications

The FRUIT, aids quality and usability practices by making use of four types of building blocks which offer simplicity in being addressed due to their component like structure. Various aspects of a product's design representations that form structured components introduce ease in addressing product definitions across heuristics. For conducting UE practices these constructs can be stored in a hierarchical form and expressed in various formats. Approaches for deriving component constructs on various production and related processes can be top-down and bottom-up depending on the availability of information. The FRUIT component information can be classified into following four groups.

VITAL – *Variables' Information Technical Associational Linguistics***:** This is the first component constructs collection and a new collection per product is built, in case no previous one exists in the FRUIT repository. This otherwise will require updating of the previous ones. The dimensional variable group's instance fragments undergo binding with rest of the eight variable groups' derivations. This allows perceivable quality and usability heuristics to group and settle under them. In order to make dimensional variables group address all necessary quality and usability scenarios they should be bound separately to all broken-down subsystem parts. **COVER** – *Component Oriented Vocabulary Entity Relations***:** This is the most important of all components construct collections. This is because it holds all the consolidated work across the organizations products' related component format data, which acts as usability guiding quality principle information. Moreover, it also acts as holder to product's design formulation-related components which allow simplified establishment and application of analytics based on usability and quality practices. Products' VITAL design definitions and performance progress can be tracked and measured against COVER. The more refined the COVER is, the better its definitions can be targeted and organized under their respective dimensional variables. Such components act as superset or maxima of definitions, which can be used to express existing products over their definition scale or serve the means to carry on new concepts in development. **CCCCC** – *Constructed Component Cluster Customized Catalogue***:** The third important component constructs collection CCCCC or 5C is a derived components apparatus to perform per product quality and usability design feature calculations. The COVER collection contributes derivations to 5C which may contain component related stances acquired directly or indirectly from product-related organization's processes. These building blocks contain the actual product definition specific to components that can be juiced out after they are connected to heuristics and usability metrics. Source component dictionary COVER can be updated in case a component has completely changed its structure during development strategy affliction. **PROOF** – *Product Relative Object-Oriented Formulations***:** Many organizations jump to this definition as it tends to cut out predecessor stages and allow more control over the quality and usability integration of disciplines within product development lifecycle. In a direct design practice the designers jump to PROOF without leaving any initial component design constructs and residues in COVER collections, thus it isolates it from other processes. The PROOF constructs are a combined form of 5C from multiple processes supporting the production process, while leaving out unwanted components that will not directly aid the final formulation. This final stage is where VITAL, COVER and 5C from various organizational processes deliver per product PROOF. Usability dimensional variable chunks' derived from performance metrics can be set against these constructs and its constituent elements as a whole. This product's building block can be primarily described via its artifacts' dimensions composition.

D. FRUIT Component Constructs' Formats

The FRUIT component constructs can be expressed and represented in various formats (**FORTS** – Formation Oriented Relations Technical Specifications). It will guide usability, quality and software engineers in conducting their respective tasks. Some of the formats can be given in following forms: Tables, Information Packages (Data Warehousing), Trees, Formal Notations, Venn Diagrams, Metrics-based Visualizations, Clusters and Scales etc.

FORTS for API-Product Artifacts' PROOF: Component collection derivations related to API can contain tools and technology, environment and platform, cognitive dimension, API marketing formulas specific definitions etc. Following component elaborates an API product's six artifacts dimensions answering what an API shall contain, whether one is passing an API design concept or referring to artifacts of an existing one. *Data Storage Artifacts* consist of databases, XML-Files, Excel-Sheets, Objects, Archives, and Operational Systems Sources etc. The *Business Layer Artifacts* consist of DALs (Data Access Layer), Drivers, Data-Sets, and Data-Sources etc. The *Class Libraries Artifacts* consist of code modules, integrated help, and class hierarchies etc. The *Referenced Technologies Artifacts* consist of external libraries/technologies, components, UI-controls, ActiveX, DCOM, OLE2, services and assemblies etc. The *Resources Cataloged Artifacts* consist of documents: help-guides, configurations-templates, manuals, and default documents, multimedia: images, icons, A/V, animations; themes: CSS, skins, scripts etc. Finally the *Presentation Layer Artifacts* consist of SDK: templates, starter kits, application backend/frontend connectivity logic etc. Furthermore, how each of these dimension's contents can be referenced by usability e.g., in case of class-libraries code modules via 12-cognitive dimensions, how this component can be used as a core reference point to all addressable details in the form of derived components upon its expansion.

E. FRUIT Maturity Lifecycle Stages

PROOF construct can be used as a scale to measure existing API product artifacts' definitions or deliver a concept model for a new API. This PROOF construct can be used as a standard to elaborate API-Artifacts measurements. Since there are no source components for derivation of this

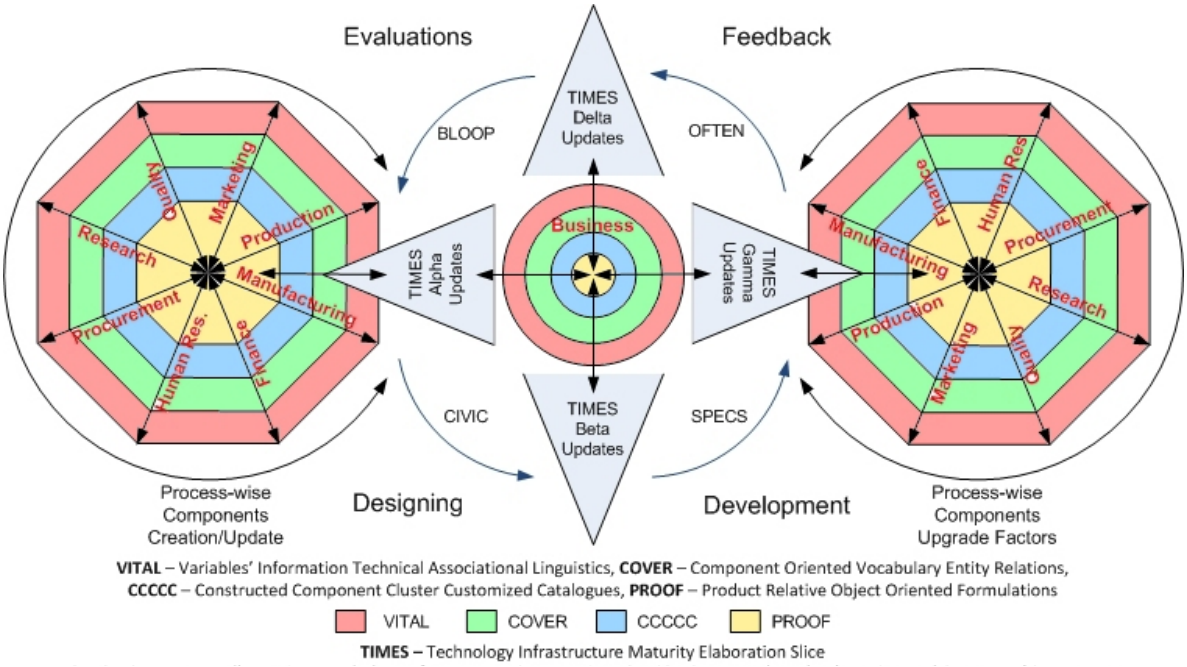

FIGURE 2 - FRUIT MATURITY LIFECYCLE STAGES

PROOF construct, using bottom-up decomposition approach of its dimensions and elements can lead to construction of VITAL, COVER and 5C component collections relevant to only the production process. The FRUIT, maturity lifecycle is distributed into four major stages that are guided using divisions called *Technology Infrastructure Maturity Elaboration Slice* – **TIMES** that act as carriers to PROOF constructs across the four maturity lifecycle phases. Relevant components constructs can be funneled into TIMES as final production stance, which are then used to aid controlled development across maturity phases in a manner which justifies quality and usability practices. Components undergo changes over time based on updates from user feedback on product deficiencies or motivations.

Following are the maturity stages that relate to Design, Development, Feedback and Evaluations respectively. Maturity Stage-one component constructs are related to requirements elicitation, design and analysis processes etc. are built or reused while experiencing updates; termed as *Components Illustration Vocabulary Infrastructure Construction* – **CIVIC.** Maturity Stage-two component constructs are related to development processes etc. are built or reused while experiencing updates; termed as *Structured Product's Engineered Constructs' Sequences* – **SPECS.** Maturity Stage-three component constructs are related to maintenance and feedback processes etc. are built or reused while experiencing updates; termed as *Observed Feedback's Tracked Evaluation Notes* – **OFTEN.** Maturity Stage-four component constructs related are to evaluation and optimization processes etc. are built or reused while experiencing updates; termed as *Business Leveraging Objects' Optimization Process* – **BLOOP**.

The TIMES can be affiliated with the four maturity stages of FRUIT; this can be done by subjecting the component constructs to update batches based on their update frequency. These updates affect the four component construct collections i.e. VITAL, COVER, 5C and PROOF. Alpha Updates deal in component constructs updates that are managed in requirements elicitation stage; these are most frequent updates and may happen rapidly. Beta Updates deal in component constructs updates that are managed based on usability practices during designing, development and testing; these are lesser frequent than CIVIC stage changes. Gamma Updates deal in component constructs updates that are managed during and after deployment using users feedback; these have longer update frequency based on user response time. Delta Updates deal in component constructs updates that are managed after bugs reports and complete symptoms evaluations for upgrading, removing or introducing newer stable features; these have least frequent updates usually per version.

F. Features Implementative Relations Modeling Solutions – FIRMS

The FRUIT should be implemented to act as a components featuring 'Production Oriented Metadata' (or a Usability Targeted Component Collection Recipes Book) that not only contains recipes for targeted systems' definitions but also on how its production and related processes' models can be customized and controlled to deliver that kind of product. The FRUIT-framework contributes in providing readiness to application of usability and quality disciplines and can be applied via software i.e. (**MAIDS** – Manageability, Applicability Integration Definitions' System). The MAIDS provide a complete set of apparatus to manage and report the progress on development and application of component constructs in reference to ongoing engineering projects. The MAIDS, projects therefore apply FRUIT-framework in form of small sized FIRMS. Each FIRMS-solution's project activities should be connected to a centralized PLANC-wise system in order to interconnect and integrate its offerings with other ongoing

projects. Lifetime of every MAIDS-FIRMS project begins and runs in parallel with any product's development processes. The proposed model will help in avoiding ad-hoc workflow practices in usability engineering and help in extension of usability and quality concerns of organizational processes by integrating combined disciplines over a components oriented platform. To simplify each MAIDS FIRMS-solution's project activities, achievable task sets (**ARROW** – Accumulated Relations Relative Organized Workload**)** can be established, which act as organized task units that meets targets alongside all the FRUIT-framework's maturity stages. The component constructs clusters can be best stored in structural form, as components may require frequent updating of the centralized components database.

IV. CONCLUSION

All usability featuring software solutions should address usability integration not only in their interface development stages but at earlier stages where raw development material i.e. APIs are subjected to usability practices, alongside this all development and production-related processes. In order to formulate a hybrid discipline that merges various usability and quality practices over a unified platform a solution that helps in overcoming each model's deficiency through support of others is essential. Such solution will act as a mode of connection between parallel disciplines that represent various infrastructure processes to work in a multidisciplinary form. The FRUIT is therefore a strong means of extending and merging usability-guided quality definitions across various organizational processes. It uses component constructs' collections as a means of establishing a usability-guided connection between people, processes and products. It is best if the framework is established generically in relevance to various organizational productions' contexts. This platform is not only a source for providing relations among various disciplines but it can be used in formally specifying and observing system in a statistical manner. The FRUIT-framework should be derived in a way that covers components ranging across necessary processes. The MAIDS-FIRMS apparatus that applies FRUIT-framework provides a means for production oriented metadata platform which is a step out of ad-hoc usability and quality discipline's practice, it provides a stable means of extending and integrating usability across organizational processes' infrastructure [6][14][16][19].

V. REFERENCES

[1] Holzinger, Andreas. "Usability Engineering Methods for Software Developers." *Communications of the ACM* (ACM) Vol. 48, No. 1 (January 2005): p. 71-74.

[2] Juristo, Xavier Ferré and Natalia, Helmut Windl, and Larry Constantine. "Usability Basics for Software Developers." *IEEE Software* (IEEE), 2001: p. 22-29.

[3] Samadhiya, Durgesh, Su-Hua Wang, and Dengjie Chen. "Quality Models: Role and Value in Software Engineering." *International Conference on Software Technology and Engineering (ICSTE).* IEEE, 2010. p.320-324.

[4] Bloch., J. "How to design a good API and why it matters." *OOPSLA.* ACM, 2006. p.506–507.

[5] Hou, Daqing, and Lin Li. "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions." *International Conference on Program Comprehension.* IEEE, 2011. p.91-100.

[6] Clarke, Steven. *Making Software - How Usable Are Your APIs?* Edited by Andy Oram and Greg Wilson. O'Reilly Media, 2011.

[7] Ko, Andrew J., and Yann Riche. "The Role of Conceptual Knowledge in API Usability." *Visual Languages and Human-Centric Computing.* IEEE, 2011. p.173-176.

[8] McLellan, Samuel G., Alvin W. Roesler, and Joseph T. Tempest. "Building More Usable APIs." *IEEE Software* (IEEE), 1998: p.78-86.

[9] Nasehi, Seyed Mehdi, and Frank Maurer. "Unit Tests as API Usage Examples." *Software Maintenance (ICSM).* IEEE, 2010. p.1-10.

[10] Nielsen, Jakob. "Enhancing the Explanatory Power of Usability Heuristics." *CHI, Human Factors in Computer Systems.* 1994. p. 152-158.

[11] Rajanen, Mikko, and Timo Jokela. "Analysis of Usability Cost-Benefit Models."

[12] Robillard, Martin P. "What Makes APIs Hard to Learn? Answers from Developers." *IEEE Software* (IEEE), 2009: p.27-34.

[13] Clarke, Steven, and Curtis Becker. "Using the cognitive dimensions framework to evaluate the usability of a class library." *Joint Conf. EASE and PPIG.* 2003. p.359–366.

[14] Scheller, Thomas, and Eva K¨uhn. "Measurable Concepts for the Usability of Software Components." *Conference on Software Engineering and Advanced Applications.* IEEE, 2011. p.129-133.

[15] Souza, Cleidson R. B. de, and David L. M. Bentolila. "Automatic Evaluation of API Usability using Complexity Metrics and Visualizations." *ICSE.* IEEE, 2009. p.299-302.

[16] Stylos, Jeffrey, and Brad Myers. "Mapping the Space of API Design Decisions." *Visual Languages and Human-Centric Computing.* IEEE, 2007. p.50-57.

[17] Stylos, Jeffrey, Andrew Faulring, Zizhuang Yang, and Brad A. Myers. "Improving API Documentation Using API Usage Information." *Visual Languages and Human-Centric Computing.* IEEE, 2009. p.119-126.

[18] Watson, Robert B. "Improving Software API Usability through Text Analysis: A Case Study." *International Professional Communication Conference, IPCC.* IEEE, 2009. p.1-7.

[19] Watson, Robert. *Usability of Complex Information Systems - Evaluation of User Interactions.* Edited by Michael J. Albers and Brian Still. CRC Press, 2011.

[20] Zibran, Minhaz F., Farjana Z. Eishita, and Chanchal K. Roy. "Useful, but usable? Factors Affecting the Usability of APIs." *Working Conference on Reverse Engineering.* IEEE, 2011. p.151-155