



SCHOOL of
GRADUATE STUDIES
EAST TENNESSEE STATE UNIVERSITY

East Tennessee State University
**Digital Commons @ East
Tennessee State University**

Electronic Theses and Dissertations

Student Works

12-2015

Assessing the Physical Security of IDFs with PSATool: a Case Study

Sulabh Bista

East Tennessee State University

Follow this and additional works at: <https://dc.etsu.edu/etd>



Part of the [Information Security Commons](#), and the [Systems Architecture Commons](#)

Recommended Citation

Bista, Sulabh, "Assessing the Physical Security of IDFs with PSATool: a Case Study" (2015). *Electronic Theses and Dissertations*. Paper 2605. <https://dc.etsu.edu/etd/2605>

This Thesis - Open Access is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

Assessing the Physical Security of IDFs with PSATool: a Case Study

A thesis

presented to

the faculty of the Department of Computer Science

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Computer and Information Sciences

by

Sulabh Bista

December 2015

Dr. Michael Lehrfeld, Chair

Dr. Phillip E. Pfeiffer IV

David Tarnoff

Keywords: Physical Security Assessment, Case Study

ABSTRACT

Assessing the Physical Security of IDFs with PSATool: a Case Study

by

Sulabh Bista

PSATool is a checklist-based, web-based application for assessing the physical security of Intermediate Distribution Frameworks. IDFs, or wiring closets, are an integral if often neglected component of information security. Earlier work by Timbs (2013) identified 52 IDF-related security requirements based on federal and international standards for physical security. PSATool refines Timbs' prototype application for IDF assessment, extending it with support for mobile-device-based data entry.

PSATool was used to assess 25 IDFs at a regional university, a college and a manufacturing corporation, with an average of 9 minutes per assessment. Network managers and assessors involved in the assessments characterized PSATool as suitable for creating assessments, usable by IT department personnel, and accurate, in terms of its characterizations of IDF status.

TABLE OF CONTENTS

	Page
ABSTRACT.....	2
LIST OF TABLES.....	5
LIST OF FIGURES	6
Chapter	
1. INTRODUCTION	8
2. LITERATURE REVIEW	10
2.1 Physical Security Requirements and PSATool v0.....	10
2.2 User Interface Guidelines for Web-Based Software.....	11
2.3 User Interface Guidelines for Mobile-Based Software.....	14
2.4 Case Study Method.....	18
3. METHODOLOGY	22
3.1 Application Development	22
3.1.1 PSATool.....	22
3.1.1.1 Model View Controller	23
3.1.1.2 Database Design.....	24
3.1.1.3 Focus on Usability	26
3.1.2 PSAToolApp.....	27
3.1.2.1 AngularJS, Ionic Framework and Cordova	28
3.1.2.2 REST API Design and Specification.....	29
3.1.2.3 Focus on Usability	30
3.1.2.4 Calculating Assessments' Durations	30
3.1.3 Traceability of Requirements.....	31
3.1.4 Accountability.....	32
3.1.5 Assessment Group and Assessment Management in PSATool.....	32
3.1.6 User Management in PSATool	33
3.1.7 PSATool Assessment Reports	33

3.2	Case Study and Data Collection	33
4.	RESULTS	35
5.	CONCLUSIONS	42
5.1	Recommendations for Future Work.....	42
5.1.1	Workflow Related Recommendations	42
5.1.2	System Design Related Recommendations	43
5.1.3	UI Related Recommendations	44
5.2	Summary	45
	WORKS CITED	46
	APPENDICES	48
	Appendix A: Questionnaire for Manager	48
	Appendix B: Questionnaire for Assessor.....	50
	Appendix C: PSATool Manual: User Management	51
	Appendix D: PSATool Manual: Managing assessments	54
	Appendix E: PSATool Manual: Generating reports	57
	Appendix F: PSAToolApp Manual	60
	Appendix G: REST API Specification	67
	VITA.....	70

LIST OF TABLES

Table	Page
1. PSATool database tables.....	25
2. Overview of data collected at a regional university.....	36
3. Overview of data collected at a college	38
4. Overview of data collected at a manufacturing corporation.....	40

LIST OF FIGURES

Figure	Page
1. PSATool assessment requirement management screen	23
2. Example of a Django data model.....	24
3. PSATool UI features.....	27
4. A screen from PSAToolApp.....	29
5. PSATool UI features.....	31
6. Traceability matrix accordion UI.....	32
7. Aggregate of all assessments	35
8. Regional university top 5 ranking.....	37
9. Regional university bottom 5 ranking.....	37
10. College top 5 ranking.....	39
11. College bottom 5 ranking.....	39
12. Manufacturing corporation top 5 ranking.....	41
13. Manufacturing corporation bottom 5 ranking.....	41
14. Link to user management.....	51
15. List of users.....	51
16. Assigning manager/assessor role to a user.....	52
17. Adding a new user	52
18. Additional information for a user	53
19. Listing assessment groups and creating a new group	54
20. Creating an assessment group.....	54
21. Managing requirements	55

22. Initializing assessments and add custom requirement	55
23. Adding a custom requirement.....	55
24. Updating a requirement's date/time	56
25. PSATool user who created a requirement	56
26. Source of a requirement.....	56
27. Reports page.....	57
28. Bar charts displaying IDF rank.....	58
29. List of IDFs and requirements	59
30. IDF specific report.....	59
31. Login screen with API URL selection.....	60
32. Assessment and IDF list.....	61
33. Changing the order of the IDF list.....	62
34. Selecting users involved in data gathering.....	63
35. List of requirements	64
36. Requirement gathering screen.....	65
37. Logout screen.....	66
38. Icons for requirement state.....	66

CHAPTER 1

INTRODUCTION

Physical security is an integral if often neglected component of information security. While various authorities provide broad guidelines for implementing and assessing physical security (Kwo-Shing Hong, 2003), a relative lack of software applications for helping with assessment makes these guidelines difficult to apply.

This lack of applications for supporting assessment is particularly evident for intermediate distribution frames (IDFs) (Timbs, 2013), IDFs, also called switch boxes, act as a distribution point for user cable and host network switches. IDFs typically house other appliances such as power backup units and humidity and temperature control units that help to assure the operational and physical integrity of the devices they contain. Because IDFs provide a physical point of access to a network, it is important to keep them secured.

The subject of this research, PSATool (Physical Security Assessment Tool), is a web and mobile-device-based application for assessing the physical security of IDFs. PSATool is the outgrowth of thesis research by Timbs (2013), who distilled physical security requirements from the NIST SP 800-53 (2009), NIST SP 800-53A (2010), and ISO/IEC 27002 (2005) standards into a 52-item checklist for assuring IDF security. Timbs' work included the creation of a prototype application for conducting checklist-based assessments.

PSATool enhances and extends Timbs' prototype. That prototype consisted of a paper-based checklist and an Access database. PSATool consists of a mobile application, PSAToolApp, that is configured to present assessors with Timbs' 52-item checklist. Users can modify and add requirements to this checklist, using a back-end, web-based application that supports the collection, management and reporting on assessment-related data. Assessments were

administered using a single code base that supports the creation of different database instances. This architecture gives institutions exclusive and complete control over their data without the need to maintain separate code bases.

PSATool's effectiveness was gauged with three case studies involving a regional university, a college and a manufacturing corporation. A total of 25 IDFs were assessed. PSATool reduced average time required to assess an IDF from 22 minutes in Timbs' work to 9 minutes. Surveys of the tool's users and their network managers were used to assess the application's effectiveness. Users' responses established PSATool's potential usefulness to network infrastructure managers as a tool for characterizing IDF security and its ease of use for assessors.

CHAPTER 2

LITERATURE REVIEW

PSATool is an extension of Timbs' work on physical security assessment (Timbs, 2013). PSATool's default requirements were taken from Timbs's work. Its user interface was based on principles and guidelines from sources like (Bhattacharya, 2011), (Nielsen, Ten good deeds in web design, 1999), and (Schniderman, 2005). Additional guidelines from sources like (Adipat & Zhang, 2005) were used to design the interface for PSATool's mobile device front end, PSAToolApp. PSATool and PSAToolApp were evaluated using the case study methodology described in (Iacono, Brown, & Holtham, 2011)

2.1 Physical Security Requirements and PSATool v0

Information security has focused mostly on cyber security while neglecting the physical security of a network's infrastructure. In (2013), Timbs describes a prototype application, here referred to as PSATool v0, that was meant to fill a void in tools for assessing the physical security of wiring closets, or Intermediate Distribution Frameworks (IDFs). PSATool v0 presents assessors with a 52-item checklist of physical controls derived from rules, requirements and recommendations from various sources, including U.S. federal agencies and the NIST SP 800-53, NIST SP 800-53A and ISO/IEC 27002 standards. Timbs extrapolated from these standards to obtain a model of an ideal IDF: an enclosed interior area that houses and protects network infrastructure. Timbs' ideal IDF would authorize, monitor and log all access to its interior. The IDF would be protected from environmental hazards and free from unrelated materials, including hardware and building supplies. Its communication wiring would be separate from power wiring to minimize interference. It would have multiple power lines that would provide power in case of

failure. It would support the use of remote disconnects to power down the closet: a concern of utmost importance for managing emergencies. The closet's temperature and humidity would be maintained using an HVAC (Heating, Ventilation, Air Conditioning) system.

In order to ensure that assessments could be completed in a reasonable amount of time, Timbs' checklist was limited to controls that sources indicated were essential for physical security. For simplicity, questions were limited to whether those controls were present. More detailed concerns about IDF security, including the controls' viability, robustness, and responsiveness, were excluded to reduce assessment time.

PSATool v0 provides a qualitative rather than a quantitative measure of IDF security. It was, however, used to rank IDFs by number of requirements passed.

PSATool v0 was implemented as a Microsoft Excel front end with a Microsoft Access back end. Excel was used to enter data that assessors recorded on paper checklists. Access was then used to pull this data into a relational database. PSATool v0 supported various reports, including reports of failed and passed and assessor information.

PSATool v0's effectiveness was established through a case study at the East Tennessee State University (ETSU). A paper version of the PSATool checklist was used to collect 52 itemized data from 135 wiring closets at ETSU. Each assessment required 22 minutes on average. ETSU's network administrator stated that the IDFs that passed more PSATool requirements were viewed as more secure.

2.2 User Interface Guidelines for Web-Based Software

In (2011), Bhattacharya describes a case study involving the application of usability guidelines to a flawed course website. Bhattacharya sought to redesign the website to improve information dissemination among students who were semi-skilled in computer and Internet use.

Bhattacharya used guidelines from Ivory and Megraw's Web Tango Methodology (2005) to gather usability related requirements. The methodology highlights eight quality features of a website: text elements, link elements, text formatting, link formatting, graphic formatting, page formatting, page performance and site architecture. It recommends putting a logo on each web page, providing a search feature, putting a heading or title on each page, and adhering to popular design practice.

The laws of Gestalt Theory provided guidelines for presenting images and graphics. They include the laws of balance (place visual weight to achieve balance), continuation (the eyes follow the direction shown in a visual image), closure (use closed shapes whenever possible), figure-ground (different foreground colors can represent different items), focal point (graphics should have a center of interest), isomorphic correspondence (an image's meaning varies with a user's experiences), proximity (proximity determines grouping), simplicity (viewers attempt to simplify images), and unity/ harmony (user will try to relate similar object in a design). This last law is helpful in generating menu groups, forming item groups and placing images on pages.

Colors were selected based on guidelines by Shneiderman and Plaisant (2005). These guidelines advocate the conservative use of color (do not overuse colors, use neutral colors), the use of limited number of colors (do not use a wide color palette), and adherence to common expectations about color code (red signals danger while green is safe).

Bhattacharya's redesigned website was evaluated using heuristic evaluation, the thinking aloud technique, and a user survey. Heuristic evaluation is based on nine usability heuristics identified by Nielsen and Molich (1990) : use simple and natural language; speak the user's language; minimize user memory load; be consistent; provide a view of a system's operation; provide clearly marked exits; provide shortcuts; generate relevant error messages; and prevent

user errors. The heuristics technique is inexpensive, can be done without prior planning and can be used in any phase of development. The authors note, however, that these simple-seeming heuristics are very difficult to implement in practice. Their use, moreover, can be affected by an evaluator's mindset, such as an inability to comprehend content or an interface.

The thinking aloud technique asks users to verbalize their thoughts while doing test-case-specified actions in a laboratory setting. This technique is inexpensive, effective and especially useful when combined with a retrospective think aloud technique. Its primary drawback is that some users find it difficult and inconvenient to verbalize when doing engaging activities.

Bhattacharya also used a survey to evaluate the redesigned system's usability. While heuristics are more effective in identifying major issues than surveys, surveys can identify more usability problems than the heuristics approach. Data from surveys can be useful and, when combined with heuristics, can produce satisfactory results.

The specific principles that Bhattacharya applied to her redesign were as follows:

- Web interface content influenced the choice of color, text-size, image content, and navigation menu group.
- Visible status information was provided in the form of breadcrumbs that showed the current page's relative location on the website as well as information that distinguishes downloadable files and external links.
- User control and freedom were supported with “cancel” buttons for downloading content.
- The need for recall was reduced through the use of a standard web layout format.
- Help and documentation were provided by adding a technical FAQ.

- Consistency and standards were assured by using a standard Windows platform for downloading and opening files. Hyperlinks to site pages also adhered to this principle.
- Flexibility and efficiency of use were achieved by distinguishing downloadable files from web page links.

Bhattacharya asked 16 users of the original site to complete six actions while thinking aloud about them. These actions mostly involved finding or interacting with website content. The redesigned site obtained a far better usability score than the original site: improvements for the tested tasks ranged from 16% to 96%. In a survey that followed the thinking aloud test, the redesigned system rated higher in aspects like proper use of color, font and image, resemblance of the interface to online course materials, ease of use and overall reaction to the interface. On a final, heuristics evaluation, the redesign scored higher on visibility of system status, match between system and the real world, consistency and standards, error prevention, flexibility and efficiency of use and aesthetic and minimalist design.

Although the redesign appeared to improve the site's usability, the result was statistically insignificant, due to the small sample population. The order of testing, moreover, may have affected the results. The results also attest to the known limitation of the thinking aloud technique: participants made more errors when thinking aloud (40% participants got confused).

2.3 User Interface Guidelines for Mobile-Based Software

Mobile applications have created new challenges for interface design. Mobile devices have smaller screen sizes, less memory and clumsier data entry interfaces than traditional guidelines for interface design assume. These limitations can cause information overload, due to the difficulty in locating information on a small screen; difficulties in providing visual hints to

users, due to limited memory; confusion about an action or path required to reach content; and difficulties in data entry due to small physical/soft keyboards. In addition, mobile device users are often distracted by other tasks like talking and walking (York & Pendharkar, 2004).

Mobile interface design should address these problems and improve a mobile user's experience. Research has identified best practices for mobile interface design like providing clear and concise menu labels, avoiding long lists of choices, and using navigational structures to support access to content. Other research has focused on improved data entry methods like speech and stylus input.

In (2005), Adipat and Zhange present guidelines for mobile application development that include considerations related to user preferences. The guidelines are divided into four major areas: user, context, information presentation and data entry methods.

User related guidelines emphasize interactions with users. An interface should allow users to customize presentation elements like font size, type, and color; media objects; and content depiction (e.g., showing images instead of just texts). The design should accommodate user disabilities and reduce cognitive load: e.g., manage split attention with additional feedback through vibration and audio.

Information overload can be addressed by following these four guidelines:

- Categorize information and present it in hierarchies that allow users to drill down for more information.
- Put more information closer to the top of the hierarchy.
- Design interfaces to guide users to focus on a specific part of a screen at a time.
- Provide a search function for quick content discovery.

Disabled users require additional hints to navigate an interface and access relevant content. Guidelines for blind users include the following:

- Provide audio and tactile modes for input and presenting information: e.g., audio-to-text modes of input and screen readers for output.
- Provide audible confirmation for user actions
- Provide a function that reminds users of their current position and choices in navigational hierarchies: e.g., “You are at the contact list page”, “Say search anytime to search for contacts”.

Guidelines for colorblind users include the following:

- Don’t use color to convey information: e.g., avoid using red to signal dangerous choices and green for safe choices.
- Use high contrast foreground and background colors
- Provide preferences to set text and background color

User interfaces for the hearing impaired should emphasize the visual display and substitute tactile feedback for audio feedback.

Context related guidelines address the need to let users control programmed responses to an application's environment, including location, ambient light, and noise. Users should have options for turning services off and on like location reporting services or automatic brightness controls. Users should receive feedback about changing an interface setting. Privacy should be respected and care taken to avoid displaying undesirable information. Context-aware information should only be displayed when a user needs it. A user should be able to prioritize notifications and control the visibility of interfaces generated by context-aware services.

Information presentation guidelines address techniques for presenting large volumes of information on a small screen. They include adaptive interfaces, the use of multiple modalities to present information, and visualization, which combines graphics with techniques like summarization, scrolling, zooming, and using focus and context to highlight relevant data and dim peripheral information.

One-dimensional data like text can be organized for efficient navigation using hierarchies and menus, with important content positioned near the tops of menus. Menu items should be properly labeled and breadcrumbs provided for navigational support. Horizontal scrolling should be avoided and consistent layout and visual design throughout the application should be encouraged. Error messages should be clear and meaningful. Proper selection of input fields (date input field for dates, phone input field phone number) is also desirable.

Two-dimensional structures can be used to present hierarchical data. Zooming techniques can be used to drill down for more detail. Similarly, techniques like focus and context technique improve data presentation for 2-D information.

Three-dimensional data presentation has special use in virtual reality and gaming. Three-D data presentation should allow a user to view an entire object and provide zoom.

Information presentation can be improved with adaptive interfaces and multimodal displays. Adaptive interfaces adapt to a user's use patterns. These interfaces should be controllable, predictable and non-obstructive. Multimodal displays provide multiple input and/or output methods for disseminating information. These displays' interfaces should allow users to choose a mode based on the operating environment: e.g., noise level, privacy concerns.

Data entry method guidelines present best practices for data input. A soft keyboard should be used when a user can attend to a mobile screen and when the screen is large enough to

display one. Handwritten input can be used for devices that support it for users who can write legible text. Voice is the input method of choice for blind users. Voice can also be a good option when privacy is not a concern during data entry.

Adipat and Zhang suggest security, privacy, and their effect on user interface design as subjects for future research. Other topics include exploring LED displays, pixel-based visual notification, and interface design for ultra-mobile devices (e.g. smart watches), since guidelines for mobile devices might not apply to these devices.

2.4 Case Study Method

In (2011), Iacono, Brown, and Holtham present guidelines for developing case studies in the context of a study of electronic marketplaces (e-marketplaces) in the steel industry. A case study is an empirical investigation of a contemporary phenomenon in its real-life context, especially when the boundaries between phenomenon and context are unclear. The main idea of case study research is to analyze evidence objectively, eliminate alternative interpretations and produce a compelling case. The method is flexible, produces diverse research outcomes and supports all types of philosophical paradigms. Case studies can be categorized as exploratory (study of a poorly defined problem), descriptive (study of a phenomenon's characteristics) or explanatory (detailed explanation of a topic including 'why and how it happens'). A case study can also be intrinsic, instrumental or collective, based on the number of cases studied.

The case study method is a qualitative method for developing and testing theory. It focuses on the meaning of real-life phenomena and not on their frequency. Case studies emphasize the characteristics of the entities under observation and the variables used to collect data. It can be a challenge to identify the variables that influence the phenomenon being studied. Equally challenging is to determine the range of data to collect, the selection and number of case

sites and type of analysis to do. Hence it is important to follow a proper procedure for designing case study research.

The case study process starts with definition of a research question. The variables to study are identified using a literature review. Data is collected through fieldwork. The tentative results obtained from the fieldwork are then compared with the data and theory proposed in the literature. This comparison determines if the theory is consistent with the available data. This comparison is crucial, as it can identify anomalies in the data and gaps in the theory. The study should include critical, extreme, revelatory cases with transparently observable phenomena. The use of multiple cases adds additional support for a theory's validity. Experimental analysis alongside a case study also provides additional validity.

Case studies collect data through various sources and in various formats. Triangulation techniques should be used to enhance the data's reliability and validity. Internal validity is concerned with whether the variables chosen directly affect the outcome. External validity is concerned with generalizing findings beyond the cases. Case studies need to be externally valid for theoretical propositions, as case studies do not deal with statistically significant sample sizes.

Since the case study method is derived from natural science, issues arise when applying it to Management and Information Systems. The controlled observation of real-world settings is difficult, as is a replicable real world setting for case study validation. Findings are often difficult to generalize as case studies involve statistically insignificant datasets and every case is unique. Finally, controlled deductions are difficult to make. To address these problems, research communities, by consensus, treat qualitative analyses as scientific when they follow established methodologies for case studies. These methodologies, which are based on methods derived from

natural science, roughly prescribe that observations and deductions be controlled and that research be replicable and generalizable.

Ionoco et al. justified the use of a case study in their work as follows:

- E-marketplaces could not be studied outside of their natural setting; data needed to be derived from the real companies that attempted to implement an e-marketplace.
- The study had to focus on events and decisions related to the time when the e-marketplace was in use.
- The study had to be done in the real environment with real subjects.
- Finally, there were no historical studies on effect of e-marketplace on steel industry.

The authors' study started with the formulation of a research question. Several authors suggested variables that might affect the viability of electronic marketplaces. Combining these authors' observations produced a conceptual framework, the Model. A hypothesis was proposed along with a predicate to test the hypothesis. The Model was field-tested and the outcome used to refine the Model. The study included seven cases, some descriptive, with multiple levels of analysis. Data was collected using unobtrusive techniques (no interviews) to minimize disruption. Emerging issues were analyzed and explored by telephone when necessary.

The authors argued their experimental design followed best practices for assuring their study's validity, reliability and replicability. The research featured a clear research question, an a priori specification of constructs, a clean theoretical slate, and a theory of interest, with predictions from this theory and rival theories. It used multiple cases, with clearly defined protocols for researching individual and multiple cases, including a multi-level strategy for analyzing results. The authors used best practices related to data collection and analysis, including maintaining logical chains of evidence, doing empirical testing and time series

analysis, and developing comparisons with this literature and across their cases. Finally, their research followed prescribed methods of designing case studies for Management and Information System.

The authors conclude by observing that case studies can be used for rigorous qualitative research that meets the criteria of validity, reliability and reliability.

CHAPTER 3

METHODOLOGY

This research sought to develop an effective application for assessing physical security. The work involved three major tasks: developing two applications for administering data collection and collecting data, using these application to collect data, and gauging the applications' effectiveness through user feedback.

3.1 Application Development

This work began with the development of two related applications. The one, a front end web application with a back-end database, allows managers to plan and manage assessments, assign assessors to assessments, and generate reports from assessments. The other, a mobile application, allows assessors to gather data from IDFs and transmit it to the secured database.

3.1.1 PSATool

PSATool is a web-based application based on the Model View Controller (MVC) architecture¹. MVC is a software architecture pattern that divides an application into a model, which stores data; a view, which displays data in the model; and a controller, which relates the model and the view, manipulating and processing data for storage and display.

PSATool was implemented using the Python based Django² MVC framework. Figure 1 shows a requirement management screen from PSATool.

¹ <https://en.wikipedia.org/wiki/Model-view-controller>

² <https://www.djangoproject.com>

3.1.1.1 Model View Controller

PSATool's data model defines entities for users, assessment groups, requirements, and collected data. Django's Object Relational Mapper (ORM) uses these definitions to generate controller-level application program interfaces for persisting content in PSATool's database. Data model definitions in PSATool consist of about 250 lines of code. Figure 2 shows a Django data model used to store IDF related data. The data model is defined as a Python class, IDFData.

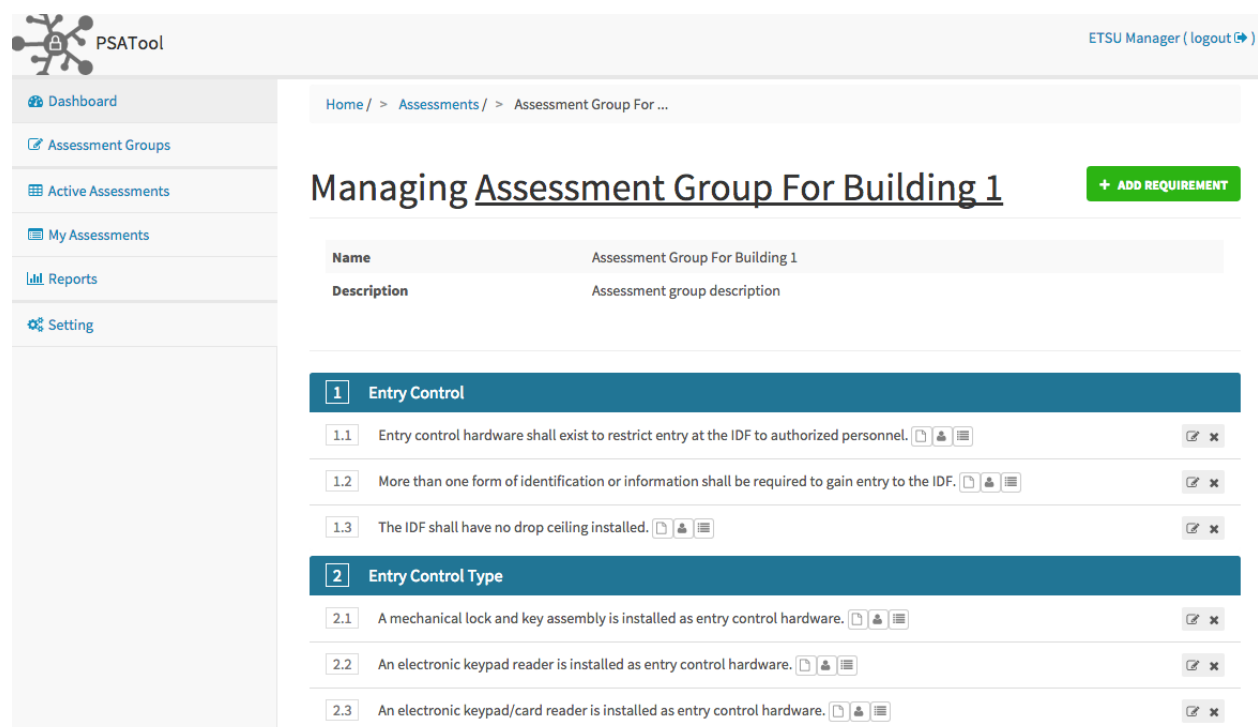


Figure 1: PSATool assessment requirement management screen

PSATool's views are defined using Django-compliant HTML markup files, using JavaScript and Cascading Style Sheets (CSS) to control formatting. The views determine how model data is formatted and displayed. They define web browser based interactions: for example, the pop-ups that appear when a user hovers over an icon. Finally, they determine styling (color and size of text, layout and arrangement of HTML content) via CSS.

PSATool's controllers render views based on data models and forward them to a web browser. Controllers also validate data, manage database transactions and format data: i.e., convert between data model and textual representations of content. Finally, controllers use Django modules for authentication, authorization, HTML form generation and user messaging and feedback.

```
class IDFData(ModelBase):
    """
    Defines data collected while performing assessments.

    assessment -- the Assessment this data belongs to
    requirements -- relates to a Requirement via IDFRequirement. (many-to-many realationship with extra data

    max_length = 100 is equivalent to saying VARCHAR(100). It is a constraint for databases. Django will com
    syntax suitable for actual database being used. See https://docs.djangoproject.com/en/1.7/ref/models/fields/
    explanation of all the options used to describe a field (like verbose_name, null=True, blank=True etc)
    """
    assessment = models.ForeignKey('Assessments', editable=False)
    idf_name = models.CharField(max_length=100, verbose_name='IDF Name')
    thv_assessment = models.TextField(verbose_name='THV Assessment', null=True, blank=True)
    thv_control = models.TextField(verbose_name='THV Control', null=True, blank=True)
    requirements = models.ManyToManyField('AssessmentRequirement', through='IDFRequirement', editable=False)
    performed_by = models.ManyToManyField('users.User', null=True, through='IDFDataPerformedBy')

    def __str__(self):
        return self.idf_name
```

Figure 2: Example of a Django data model

3.1.1.2 Database Design

PSATool uses the MariaDB database³, an open source version of MySQL⁴. Because PSATool uses Django's ORM layer to access and manage this database, any Django-compatible database could be used in place of MariaDB.

The PSATool database supports a workflow for managing and reporting data from assessments. PSATool managers can create, define requirements for, and assign assessors to assessment groups. Assessors use checklists, optionally supported by texts and pictures, to

³ <https://mariadb.org>

⁴ <https://www.mysql.com>

collect data regarding the fulfillment of requirements. The PSATool database's tables (Table 1) store this data, along with supporting data for interfacing with PSAToolApp.

Table 1: PSATool database tables

Table Name	Content
Authentication and authorization related	
users_user	User information, including name, password, email, along with access level (e.g., staff, superuser)
authtoken_token	Temporary authentication token for PSAToolApp
Master list of requirement related	
dashboard_mastercontrolclasslist	Master list of PSATool's built-in control classes
dashboard_masterrequirementlist	Master list of PSATool's built-in requirements list
dashboard_masterguidancedocument	Source document for master requirements.
dashboard_traceabilitymatrix	Maps master requirements to their guidance documents including page number
dashboard_requirementoptions	Yes, No, Not Applicable and Not Assessed
Assessment management related	
dashboard_assessmentgroup	Assessment group data, like name and description
dashboard_assessmentcontrolclass	An assessment group's control class
dashboard_assessmentrequirement	An assessment group's requirements
dashboard_assessments	Data on active assessments
dashboard_assessments_assigned_to	Assignment of an assessment to an assessor user
Assessment data collection related	
dashboard_idfdata	Data on IDF like names, THV assessment and control
dashboard_idfdataperformedby	Users involved in assessing an IDF
dashboard_idfrequirement	An IDF's specific requirements. Textual comments and file system locations of uploaded pictures are also stored here.
dashboard_assessmentduration	The duration required to assess each IDF requirements.

Managing Image Uploads

PSATool follows recommended practice for image management, storing images outside the database and limiting databases to image metadata (Wiles, 2012). This makes the database easier to backup and improve access speed; database reads are slower than file-system reads.

3.1.1.3 Focus on Usability

The design of PSATool's user interface (UI) used guidelines from (Bhattacharya, 2011) and (Nielsen, Ten good deeds in web design, 1999). Each page includes a logo that links to an application's home page. Multiple navigation bars, including top and left bars (Bhattacharya, 2011) (Burrell & Sodan, 2006), have been used and their content determined by user access level. A fixed palette of colors was used (Bhattacharya, 2011) (Schniderman, 2005) with blue denoting positive action (creating an assessment, click to expand for more information) and grey denoting side-effect-free actions (go back buttons). Icons and tooltips provide visual clues about a UI element's actions. Breadcrumb trails provide navigation cues and indicate system status (Bhattacharya, 2011) (Nielsen & Mack, Heuristic Evaluations, 994). Accordion style UI elements allow users to expand and explore contents of possible interest.

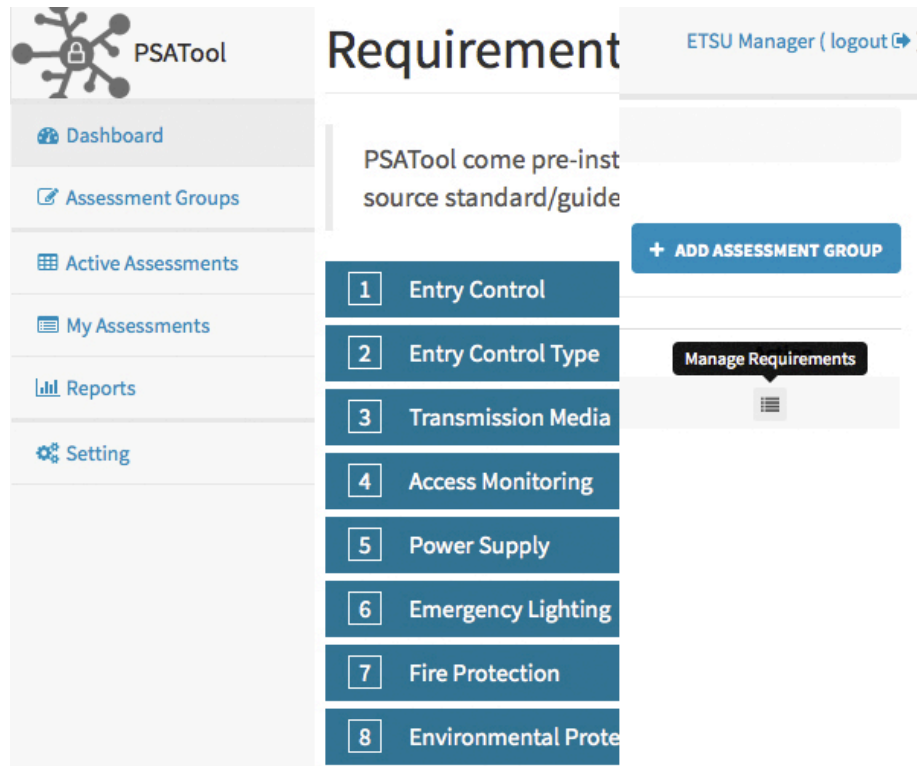


Figure 3: PSATool UI features

Figure 3 shows some of these features. From left to right, the figure shows PSATool's cross-web-page side menu-bar and one of its accordion based lists, along with the interface's consistent color in buttons and support for tooltips.

3.1.2 PSAToolApp

PSAToolApp (Figure 5) is a mobile/tablet-based application that allows an assessor to collect data from IDFs and transmit it to PSATool's back-end application. PSAToolApp was developed using AngularJS⁵, Ionic Framework⁶ and Cordova⁷. In theory, it can run on Android,

⁵ <https://angularjs.org>

⁶ <http://ionicframework.com/>

⁷ <https://cordova.apache.org/>

iOS and Windows Phone. To date, PSAToolApp has only been tested in Android. It is available in the Google Play store⁸.

Similar to PSATool, PSAToolApp is an MVC based application. Unlike PSATool, PSAToolApp lacks a persistence layer for storing model data. Rather, it uses a representational state transfer (REST) based API to communicate with the main PSATool application. This API forms the basis for storing and retrieving data in PSAToolApp.

3.1.2.1 AngularJS, Ionic Framework and Cordova

Instead of developing PSAToolApp as a native application for Android, iOS and Windows Phone separately, PSAToolApp was developed using web technologies. This approach allows the same code base to be used to generate application binaries for different platforms, including iOS, Android and Windows Phone. The application logic is written in AngularJS. The user interface is written in the Ionic Framework. Cordova provides various runtime and libraries for accessing mobile device features like a camera. Cordova also generates binaries suitable for specific device platforms: i.e., Java packages for Android, Objective C binary for iOS and Common Language Runtime for Windows phone.

⁸ <https://play.google.com/store/apps/details?id=com.psatool.psatoolapp>

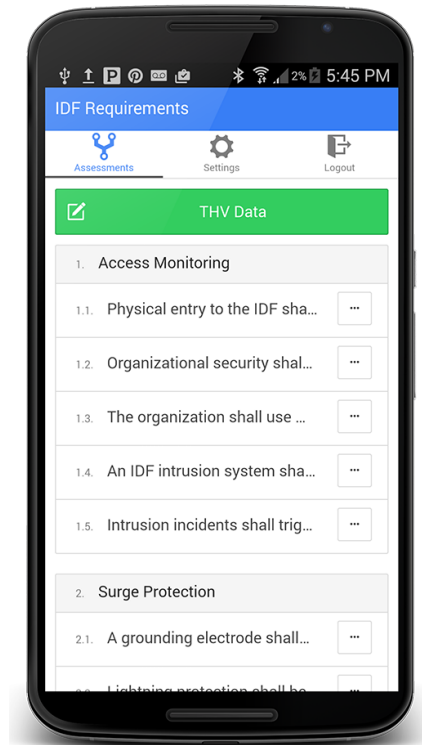


Figure 4: A screen from PSAToolApp

3.1.2.2 REST API Design and Specification

PSAToolApp communicates with PSATool using a REST⁹ API. REST APIs use HTTP operations to invoke data storage and retrieval operations at a server. For example, HTTP GET fetches server data; HTTP POST stores data; and HTTP OPTIONS retrieves all HTTP server-supported actions. REST also makes it easy to use JavaScript Object Notation (JSON) as a data transfer format. JSON is native to the JavaScript programming language, used in PSAToolApp.

PSAToolApp makes HTTP requests to a PSATool server using HTTP GET and POST. It uses JavaScript Object Notation (JSON) for data exchange, in order to leverage AngularJS's support for parsing JSON and generating content in a format similar to that used by PSATool's data dictionary (see Appendix G for REST API documentation).

⁹ https://en.wikipedia.org/wiki/Representational_state_transfer

3.1.2.3 Focus on Usability

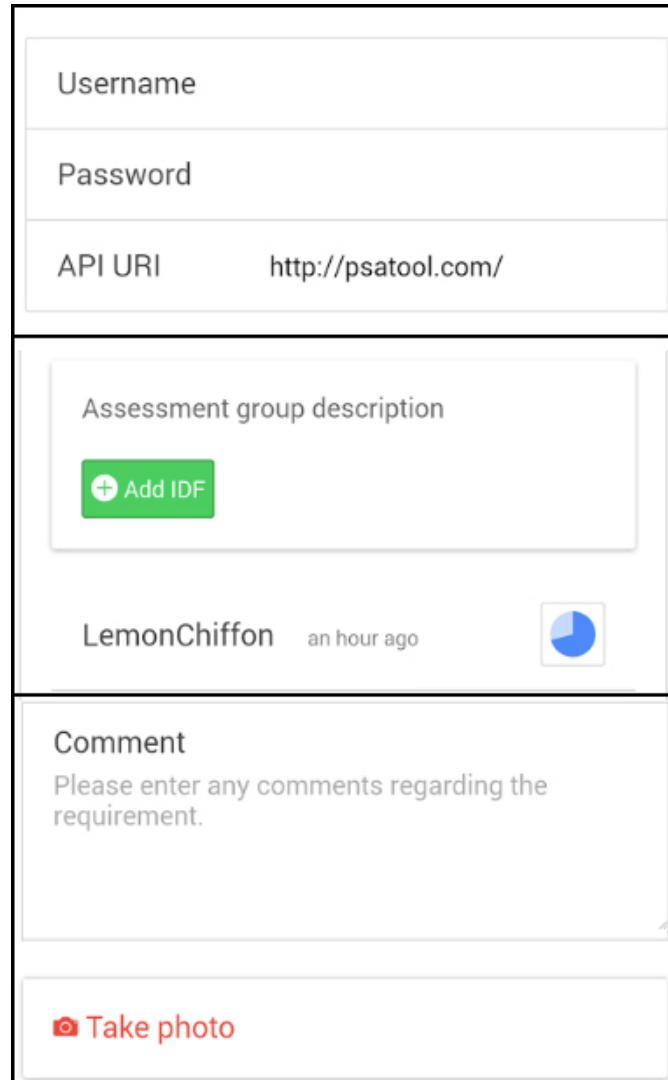
PSAToolApp's UI was designed for usability, based on guidelines from (Adipat & Zhang, 2005) and other best practices for mobile application design. The Ionic Framework used in PSAToolApp provides various UI templates that adhere to best practices.

In keeping with recommendations from Adipat and Zhang (2005), PSAToolApp allows users to change PSATool's server URL and to customize how lists are sorted. This includes support for sorting IDFs based on time of last update or percentage of requirements assessed. Users receive feedback following connection failures, authentication failures, and other errors. Small pie charts provide quick visual feedback for the percentage of requirements assessed for each IDF. Modal dialogs prompt users for actions and screen transition effects provide for gradual changes in screen information. PSAToolApp supports a soft keyboard and the uploading of photos in support of assessment. (See Appendix F: "PSAToolApp manual" for PSAToolApp user documentation)

Figure 5 shows some features of PSAToolApp's UI. From top to bottom, the figure shows support for selecting PSATool server URL, the use of a dynamic pie chart icon to show number of requirements assessed, and support for adding comment and uploading photo.

3.1.2.4 Calculating Assessments' Durations

PSAToolApp automatically records the time taken to assess requirements based on a user's interaction with the app. When a user enters a screen that collects requirement data, PSAToolApp starts a timer. Saving requirement data stops the timer and sends the timing data to the PSATool server along with other requirement related data. PSATool uses the timing data thus collected to calculate the time cost of assessments.



The image shows a mobile application interface for PSATool. It is divided into three main sections. The top section contains three input fields: 'Username', 'Password', and 'API URI' (with the value 'http://psatool.com/' pre-filled). The middle section is titled 'Assessment group description' and features a green button with a plus icon and the text '+ Add IDF'. Below this, the user 'LemonChiffon' is shown with the timestamp 'an hour ago' and a blue circular profile picture. The bottom section is titled 'Comment' and includes the placeholder text 'Please enter any comments regarding the requirement.' Below the comment area is a red button with a camera icon and the text 'Take photo'.

Figure 5: PSATool UI features

3.1.3 Traceability of Requirements

PSATool's interface presents information about the 52 requirements that are prepackaged with PSATool. The 'Dashboard' section and the 'Assessment Groups' management page list each requirement and document its origin, including the documents and page numbers from which it was derived.

Figure 6 shows part of PSATool's traceability matrix. It shows that requirement "1.1" under "Entry Control" was derived from multiple documents like section 9.1 of NIST SP 800-53

and sections PE-2.1 and PE-3.1 of ISO/IEC 27002. This matrix helps to establish the validity and usefulness of PSATool's default requirements.

Requirement to source document mapping

PSATool come pre-installed with a set of 52 requirements. The following list categorizes these requirements and provides a mapping to its source standard/guideline. Click on the items to expand.

1 Entry Control			
1.1	Entry control hardware shall exist to restrict entry at the IDF to authorized personnel.	NIST SP 800-53	9.1.2
		ISO/IEC 27002	PE-2.1, PE-3.1
		FEMA	SAND2007-5591.24.1.1.5
		O'Connor et al, 2007	p.155
1.2	More than one form of identification or information shall be required to gain entry to the IDF.	NIST SP 800-53	9.1.2.b
		ISO/IEC 27002	PE-2(2).1
		O'Connor et al, 2007	p.155
1.3	The IDF shall have no drop ceiling installed.	NIST SP 800-53	9.1.1
		(Baker, 2005)	426.3.1.3
2 Entry Control Type			

Figure 6: Traceability matrix accordion UI

3.1.4 Accountability

PSATool records the identities of users who modify assessment groups (addition and modification of requirements in a group) and participate in assessments using PSAToolApp.

3.1.5 Assessment Group and Assessment Management in PSATool

An assessment group is used for easier management and grouping of assessments. For example, a different assessment group can be created for each of an organization's buildings. Assessors can then be assigned to an active assessment for an assessment group. The assessors

can gather requirements for their assigned buildings. (See Appendix D, "PSATool manual: Managing assessments")

3.1.6 User Management in PSATool

PSATool supports two types of users, Managers and Assessors. Managers can create assessments, modify requirements, assign assessors to assessments, and gather requirements. Assessors can gather requirements for assessments that managers have assigned them to do. (See Appendix C: "PSATool manual: User Management")

3.1.7 PSATool Assessment Reports

PSATool can generate two different types of assessment reports. Summary reports present an overview of an assessment and rank IDFs based on passed and failed requirements. IDF-specific reports provide details about requirements that an IDF passed or failed. (see Appendix E: "PSATool manual: Generating reports")

3.2 Case Study and Data Collection

PSATool's effectiveness was assessed using case studies at a regional university (ETSU); a regional college; and a manufacturing corporation. Different types of assessment institutions were selected to have a variety in the collected data.

A multi-tenant setup of PSATool web application was hosted at ETSU. Unique domain names were created for the three institutions with all domain names referencing the same server. PSATool used different databases for different domain names, thereby isolating data from different institutions.

A manager level account was created for each institution. The manager user created assessor users and assessment groups and assigned assessors to assessments. The assessors then used PSAToolApp to collect requirement data from IDFs.

PSAToolApp provides an option to specify the web server to which PSAToolApp should connect. Assessors were provided with the web server name for their institution.

Managers and assessors were provided with a feedback form (Appendix A, Appendix B) and asked to evaluate the usefulness and effectiveness of PSATool and PSAToolApp. Their feedback was used to assess the effectiveness of PSATool and generate ideas for future work.

CHAPTER 4

RESULTS

PSATool was used at three institutions to collect requirement data from IDFs. A total of 25 IDFs were assessed during the case study. The assessors from these institutions each completed a questionnaire regarding the effectiveness and usefulness of PSATool software.

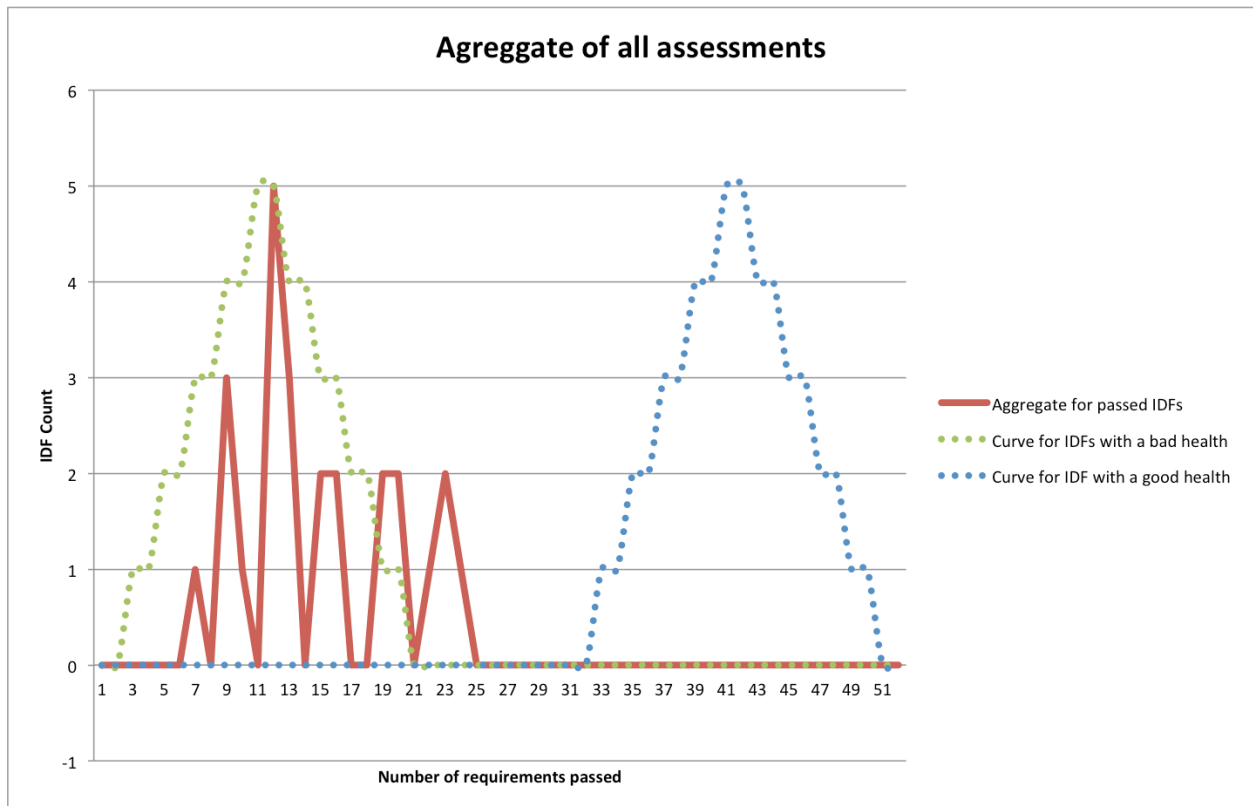


Figure 7: Aggregate of all assessments

Figure 7 shows the overall health of all IDFs assessed for this study. The x-axis is the 'number of requirements passed' and the y-axis is 'IDF Count', or the count of IDFs that passed 'the number of requirements'. The 'aggregate of passed IDFs' curve is a plot of the data collected during the assessment. PSATool's reporting tool generates a similar graph on the fly while an overview report is requested.

In Figure 8, the ‘Curve for IDFs with bad health’ represents data from a fictional institution with bad IDF health. This curve is skewed towards the left, which denotes that most IDFs have failed most requirements. The ‘Curve for IDFs with good health’ represents data from a fictional institution with good ID health. This curve is skewed towards the right, which denotes that most IDFs have passed most requirements.

Table 2: Overview of data collected at a regional university

Regional university	
Number of IDFs assessed	6
Maximum number of requirements passed by an IDF	23
Least number of requirements passed by an IDF	11
Maximum number of requirements failed by an IDF	32
Least number of requirements failed by an IDF	21
Average time required for all assessments	17.68 minutes

Table 2 summarizes data from the six IDF assessments at ETSU. The best performing IDF passed 23 and failed 11 requirements while the poorest IDF passed 11 and failed 21 requirements. Average time per assessment was 17.68 minutes. The ‘Passed top rank’ (Figure 8) shows the top 5 best performing IDFs while the ‘Failed top rank’ (Figure 9) shows the top 5 worst performing IDFs.

ETSU’s assessor noted that PSAToolApp was easy to use and follow through the requirements. The assessor mentioned that simplifying the wording on the requirements would improve the understandability. ETSU’s network manager noted that the reports were useful enough to support IDF upgrade proposals for higher management. The manager also suggested some UI changes related to providing serial numbers in reports.

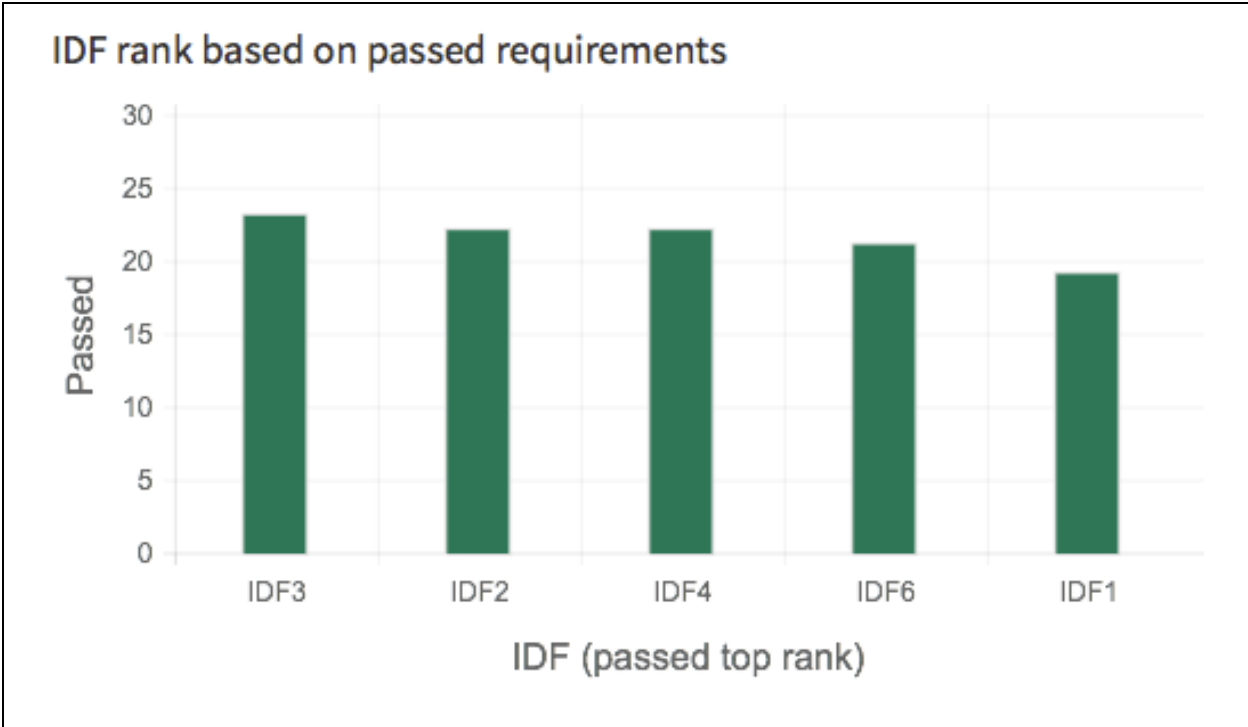


Figure 8: Regional university top 5 ranking

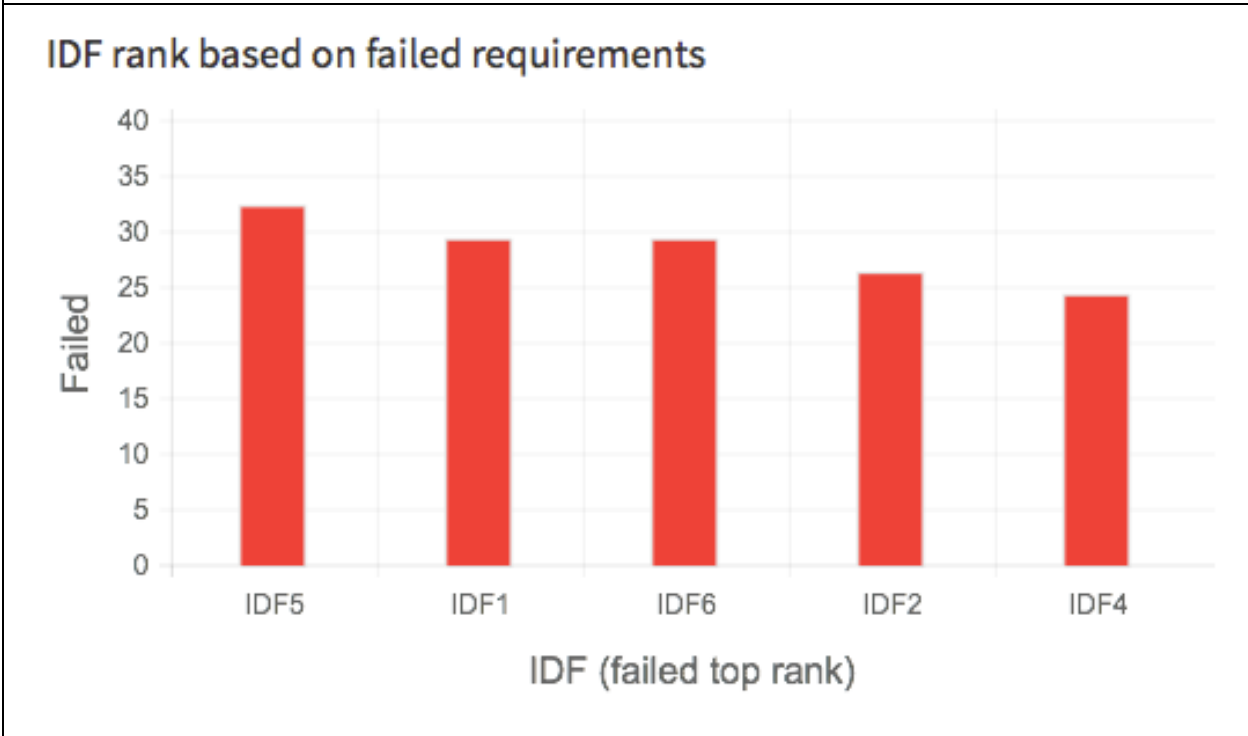


Figure 9: Regional university bottom 5 ranking

Table 3 summarizes data from the ten IDF assessments at the regional college. The best performing IDF passed 19 and failed 33 requirements while the poorest IDF passed 6 and failed 46 requirements. Average time per assessment was 4.01 minutes. The use of small, wire-rack enclosure IDFs at this college significantly reduced the assessment time. The ‘Passed top rank’ (Figure 10) shows the top 5 best performing IDFs while the ‘Failed top rank’ (Figure 11) shows the top 5 worst performing IDFs.

Table 3: Overview of data collected at a college

A college	
Number of IDFs assessed	10
Maximum number of requirements passed by an IDF	19
Least number of requirements passed by an IDF	6
Maximum number of requirements failed by an IDF	46
Least number of requirements failed by an IDF	33
Average time required for assessment	4.01 minutes

The college's assessor noted that PSAToolApp was easy to use and intuitive. The assessor added that extra hints about requirements would help in understanding them. The college's network manager noted that the traceability matrix was very useful in establishing the requirements' credibility. The manager thought the tool was well designed for its purpose and 100% complete.

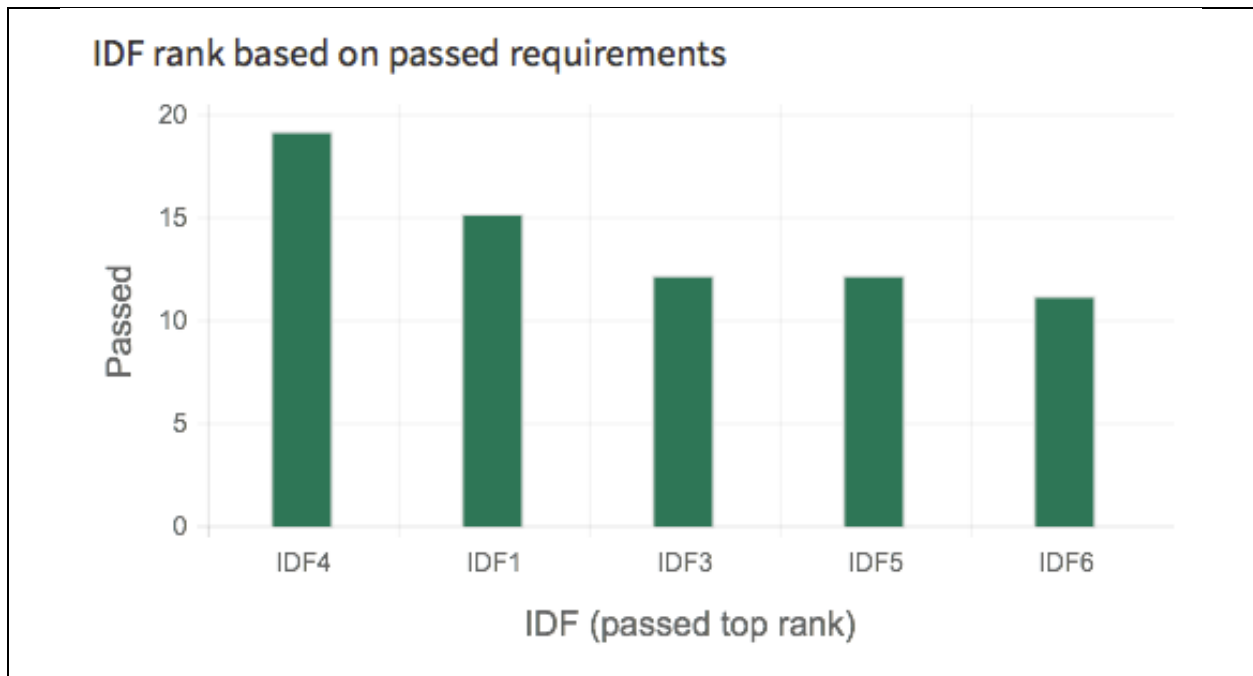


Figure 10: College top 5 ranking

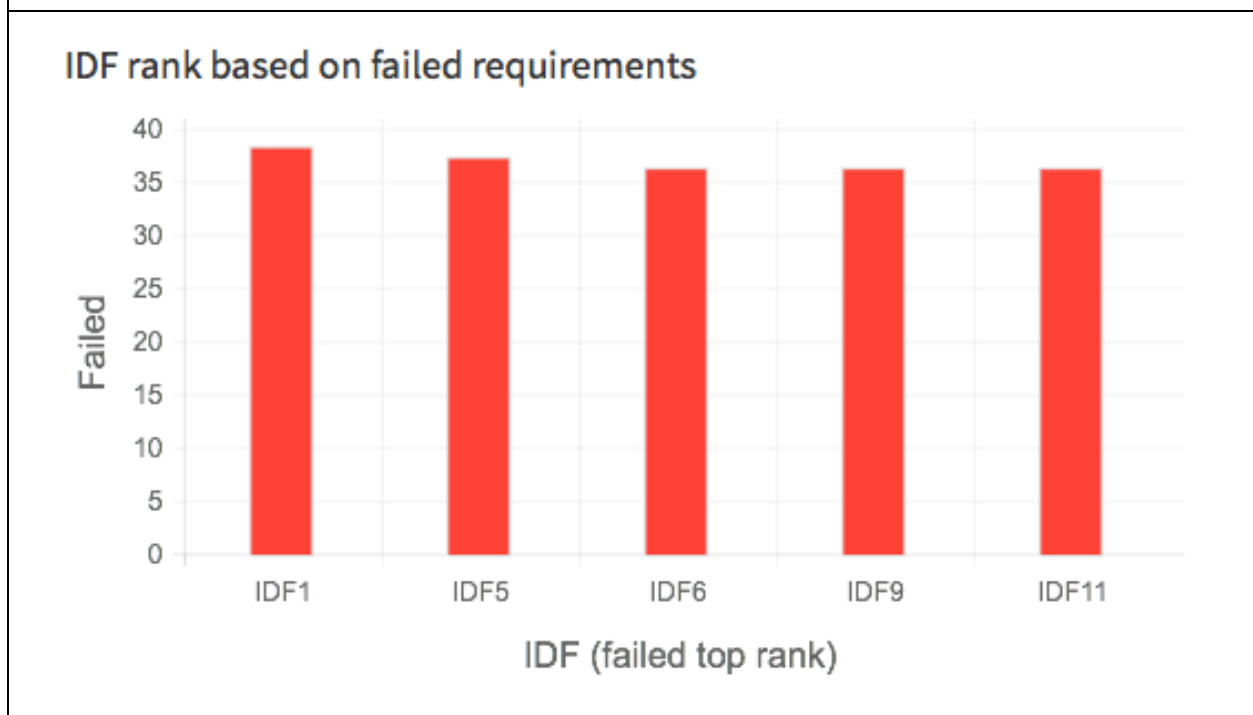


Figure 11: College bottom 5 ranking

Table 4 summarizes data from the nine IDFs at the manufacturing corporation. The best performing IDF passed 18 and failed 32 requirements while the poorest IDF passed 11 and failed

38 requirements. Average time per assessment was 8.56 minutes. The ‘Passed top rank’ (Figure 12) shows the top 5 best performing IDFs while the ‘Failed top rank’ (Figure 13) shows the top 5 worst performing IDFs.

Table 4: Overview of data collected at a manufacturing corporation

A manufacturing corporation	
Number of IDFs assessed	9
Maximum number of requirements passed by an IDF	18
Least number of requirements passed by an IDF	11
Maximum number of requirements failed by an IDF	38
Least number of requirements failed by an IDF	32
Average time required for assessment	8.56 minutes

The manufacturing corporation’s assessor noted that PSAToolApp was easy to maneuver around and efficient for quick assessments. The assessor suggested that the requirements could be broken down more with further definition and explanation to improve understandability.

The manufacturing corporation's network manager noted that PSATool's functionality was easy to remember and recognize after setting up a test assessment. The manager also noted that the reports were easy to understand and navigate. The manager was satisfied with the qualitative view of the IDFs presented by PSATool, noting that PSATool reports could be used to prioritize improvements to IDFs.

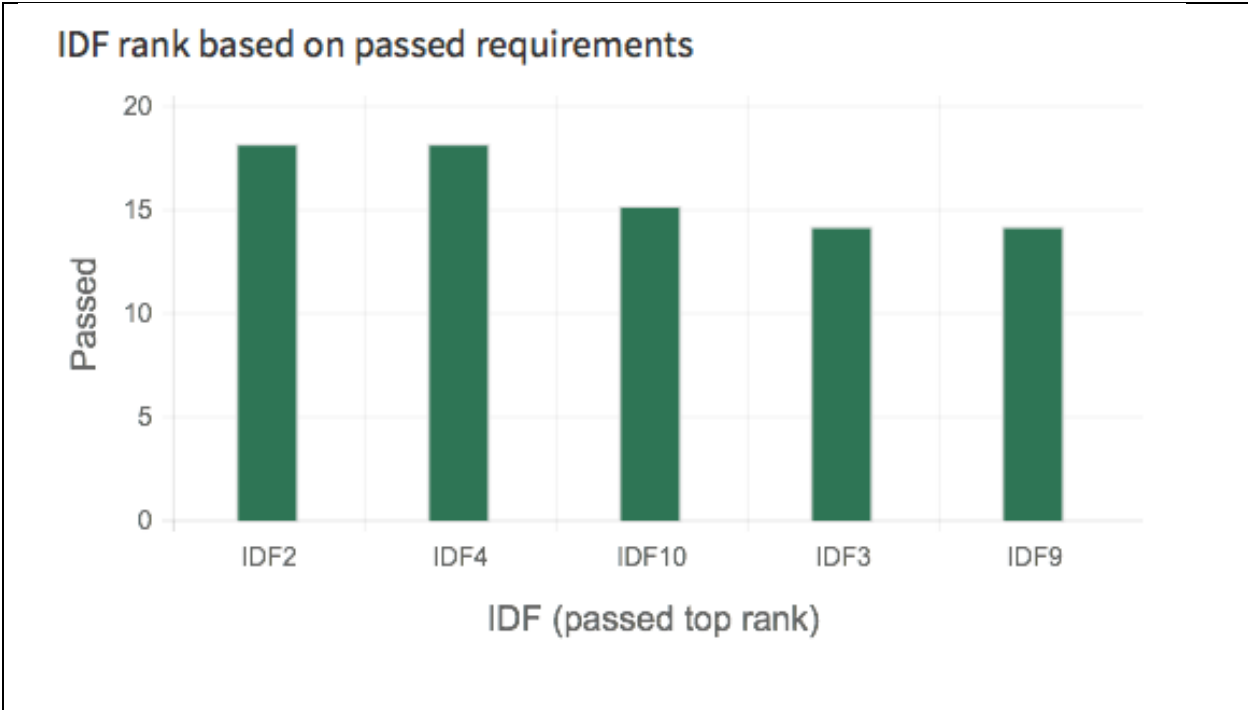


Figure 12: Manufacturing corporation top 5 ranking

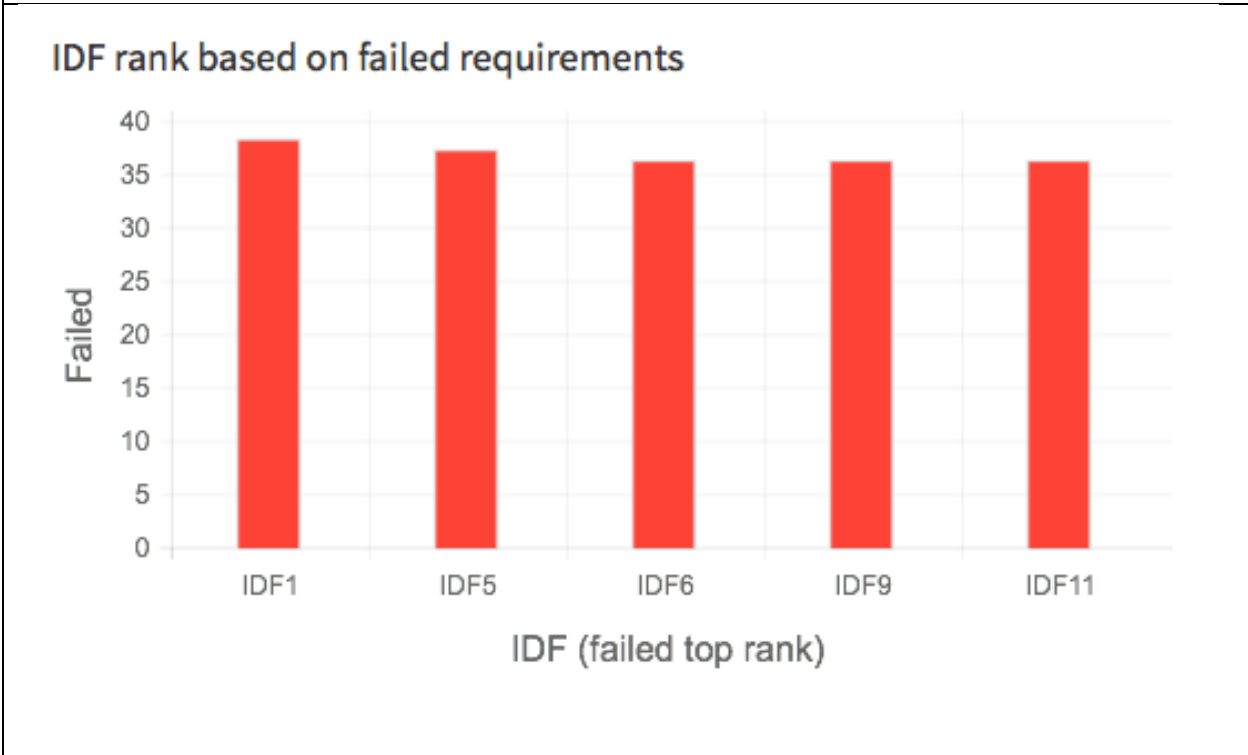


Figure 13: Manufacturing corporation bottom 5 ranking

CHAPTER 5

CONCLUSIONS

PSATool proved to be an effective tool for creating assessments, managing requirements and generating reports for assessment of IDFs. This was demonstrated by the case studies at the three institutions.

PSATool provided an easy and intuitive workflow for creating and managing assessments. PSATool allowed managers to create assessor users and assign them assessments. PSATool-generated reports provided relevant overviews of the health of assessed IDFs. The reports provided details about failed and passed requirements for all IDFs including images taken during the assessment. The quality of the IDFs as reflected by PSATool corresponded to the managers' qualitative views of the IDFs. Managers noted that printed reports from PSATool could be included in work orders or used to inform upper management about security issues and prioritize improvements to IDFs.

PSAToolApp made it easy for assessors to record requirements-related data. Assessors noted that the ability to supplement observations with photos from mobile devices could be useful for later, detailed investigations of issues. The use of a mobile device allows data to be sent to a server immediately, in one step, without any extra data entry.

5.1 Recommendations for Future Work

5.1.1 Workflow Related Recommendations

The following features should be considered for future versions of PSATool and PSAToolApp:

1. *Support for custom control classes.* PSATool groups requirements into control classes. PSATool should provide an interface for modifying existing control classes and adding new classes. This feature was suggested by an assessment manager.
2. *Support for managing requirements' provenances.*, PSATool should provide an interface for managing the associations between the 52 default requirements and their sources or associating custom requirements with their own sources.
3. *Support for cross-group reports.* PSATool should support the generation of reports for multiple assessments within a single assessment group and comparisons of assessments done at different times.
4. *Support for demo mode.* PSATool should support a step-by-step demo mode for new user. A UI pattern called ‘walkthrough’ can be used for this purpose and bootstrap-tour¹⁰ can be used to implement such pattern in PSATool.
5. *Support for standard conformance.* PSATool should support reports that relate assessment data to individual standards like ISO 27001 and NIST.
6. *Categorization of requirements.* Requirements should be further categorized into requirements for IDF robustness and requirements for IDF data security.

5.1.2 System Design Related Recommendations

The following design changes should be considered for future versions of PSATool and PSAToolApp:

¹⁰ <https://github.com/sorich87/bootstrap-tour>

1. *Improvements to the REST API.* PSATool's REST API should be rewritten using a standard guideline for REST interfaces. Possible sources for such a guideline could be (Jansen, 2012) or (Hirsch, 2015).
2. *Improvements to PSATool's accountability tracking.* PSATool's support for logging should be expanded to log and track all manager-level actions including creating assessments and adding requirements.
3. *Support for concurrent assessments.* PSAToolApp should support the concurrent use of multiple mobile devices to assess an IDF. PSATool and PSAToolApp could be redesigned to use a WebSocket¹¹ based multicast architecture for passing requirement data between multiple mobile devices. Source code from (Bista, 2015) could be used as a reference system to implement a WebSocket based multicast architecture. (Agrawal, Starobinsku, & Trachtenberg, 2002) could be used for strategies for data synchronization. This feature was suggested by assessors who experimented with using 2 mobile devices to assess a single IDF.
4. *Improvements to database efficiency.* PSATool's data model could be efficiently represented using tree like database tables as discussed in (Celko, 2007). The modified pre-order tree traversal (MPTT) discussed by Celko has been implemented as a Django library¹². Future versions of PSATool could use this library for efficient database design.

5.1.3 UI Related Recommendations

The following UI enhancements should be considered for future versions of PSATool:

¹¹ <https://en.wikipedia.org/wiki/WebSocket>

¹² <https://github.com/Django-mptt/Django-mptt/>

1. Serial numbers should be added to requirements listed in individual IDF reports. This would require adding serial numbers to the requirements on the database level.
2. Archiving and pagination should be implemented for all list based pages like list of assessment groups and list of IDFs.
3. A virtual walkthrough or a visual representation of an ideal IDF should be included in the reports.

The following enhancements should be considered for future versions of PSAToolApp:

1. Specific syndromes rather than generic error messages should be provided following loss of Internet connectivity,
2. As per PSATool's assessors, the UI should provide
 - a. a button to add an IDF directly from the requirement list screen.
 - b. descriptive texts and examples should be added for all requirements.

5.2 Summary

PSATool and PSAToolApp successfully automated and extended the proposed strategy for IDF assessment as described in (Timbs, 2013). PSATool and PSAToolApp provide a software-based workflow for creating, managing, assessing and reporting assessments of IDFs. The application supports the 52 requirements as proposed in (Timbs, 2013). Processes like report and graph generation were automated. The application's effectiveness and usefulness were demonstrated with case studies at ETSU, a regional college and a manufacturing company. Average assessment time for IDFs was reduced significantly and users expressed their satisfaction with the tool's usability and data.

WORKS CITED

- Adipat, B., & Zhang, D. (2005). Interface Design for Mobile Applications. *AMCIS 2005 Proceedings* , 494.
- Agrawal, S., Starobinsku, D., & Trachtenberg, A. (2002). On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices . *Network, IEEE* , 22-28.
- Bhattacharya, P. (2011, 8). *A Case Study of the Effects of a Web Interface Redesign Based on Usability Guidelines*. Retrieved 11 09, 2014, from ETSU Digital Commons:
<http://dc.etsu.edu/etd/1320/>
- Bista, S. (2015, April 1). *paintcollaborate*. Retrieved September 27, 2015, from GitHub:
<https://github.com/sul4bh/paintcollaborate>
- Celko, J. (2007). *Trees in SQL*. Retrieved July 23, 2015, from ibase.ru:
<http://www.ibase.ru/devinfo/DBMSTrees/sqltrees.html>
- Hirsch, B. (2015, February 1). *White House Web API Standards*. Retrieved September 27, 2015, from GitHub: <https://github.com/WhiteHouse/api-standards>
- Iacono, J. C., Brown, A., & Holtham, C. W. (2011). The use of the Case Study Method in Theory Testing: The Example of Steel eMarketplaces. *The Electronic Journal of Business Research Methods* , 57-65.
- Ivory, M. Y. (2005). Evolution of web site design patterns. *ACM Trans.Inf.Syst.* , 463-497.
- Jansen, G. (2012, November 15). *Lessons learnt from designing the Red Hat Enterprise Virtualization API*. Retrieved September 27, 2015, from Thoughts on RESTful API Design: <http://restful-api-design.readthedocs.org/en/latest/intro.html>
- Kwo-Shing Hong, Y.-P. C.-H. (2003). An integrated system theory of information security management . *Information Management & Computer Security* , 243-248.

- Nielsen, J. (1999, 10 3). *Ten good deeds in web design*. Retrieved 11 09, 2014, from <http://www.useit.com/alertbox/991003.html>
- NIST. (2011, September). *NIST Special Publication 800-30 Revision 1*. Retrieved January 20, 2014, from National Institute of Standards and Technology Information Technology Laboratory: http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf
- NIST. (2004, February). *Federal Information Processing Standards Publication*. Retrieved January 20, 2014, from National Institute of Standards and Technology Information Technology Laboratory: <http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf>
- NIST. (2012, May 27). *NIST Special Publication 800-53 Revision 3*. Retrieved January 20, 2014, from National Institute of Standards and Technology Information Technology Laboratory: http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf
- NSIT. (2010, June). *NIST Special Publication 800-53A Revision 1*. Retrieved January 20, 2014, from National Institute of Standards and Technology Information Technology Laboratory: <http://csrc.nist.gov/publications/nistpubs/800-53A-rev1/sp800-53A-rev1-final.pdf>
- Schniderman, B. &. (2005). *Designing the user interface: Strategies for effective human computer interaction (Fourth Ed.)*. Boston: Pearson Addison Wesley.
- Timbs, N. H. (2013, 12). *Physical Security Assessment of a Regional University Computer Network*. Retrieved 11 09, 2014, from ETSU Digital Commons: <http://dc.etsu.edu/etd/2280>
- Wiles, F. (2012, 5 1). *Three things you should never put in your database*. Retrieved 7 20, 2015, from REVSYS: <http://www.revsys.com/blog/2012/may/01/three-things-you-should-never-put-your-database/>

APPENDICES

APPENDIX A: QUESTIONNAIRE FOR MANAGER

Name (First, Last) _____

Name of organization _____

Position(eg: Supervisor, Sysadmin) _____

Work Phone _____

Email _____

Date _____

PSATool Quality: *Please...*

characterize PSATool's usability

characterize PSATool's understandability

rate, on a scale of 1 (poor) to 7 (excellent), PSATool's

understandability: _____ **usability:** _____

state how the tool could be improved

PSATool Assessment Quality

How many personnel hours did the assessment take?

What costs did the assessment incur?

Please characterize

the tool's completeness

the format and content of the tool's output

the assessment's usefulness

how, if at all, this assessment has affected your view of IDF Security

how, if at all, this assessment might affect your organization's practice

how the assessment procedure could be improved

Please rate the following aspects of the assessment provided by PSATool on a scale of 1 (poor) to 7 (excellent)

Usefulness: _____

Completeness: _____

Please rate the following aspects of the assessment provided by PSATool on a scale of 1 (poor) to 7 (excellent)

Effect on your view of IDF security: _____

Likely effect on organizational practice: _____

Likelihood that a future assessment will include PSATool: _____

APPENDIX B: QUESTIONNAIRE FOR ASSESSOR

Name (First, Last) _____

Name of organization _____

Work Phone _____

Email _____

Date _____

PSAToolApp Quality: Please...

characterize PSAToolApp's usability

characterize PSAToolApp's understandability

rate, on a scale of 1 (poor) to 7 (excellent), PSAToolApp's

understandability: _____ **usability:** _____

list any issues or annoyances you encountered while performing assessments

state how the tool could be improved

APPENDIX C: PSATOOL MANUAL: USER MANAGEMENT

PSATool supports two types of users, Managers and Assessors. Managers can create assessments, modify requirements, assign assessors to assessments, and gather requirements. Assessors can gather requirements for assessments that managers have assigned them to do.

Commands for managing users are accessible via `http://<psatool domain>/admin`. This URL is guarded by a login screen that accepts manager credentials only.

Click on Users to list the system's users (Fig. 14).

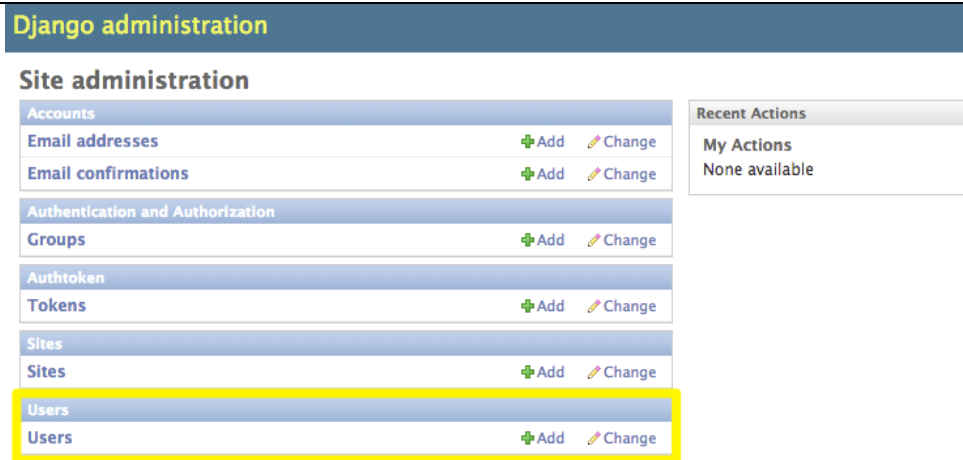


Figure 14: Link to user management

Click on a user to view/change the user's attributes (Fig. 15).

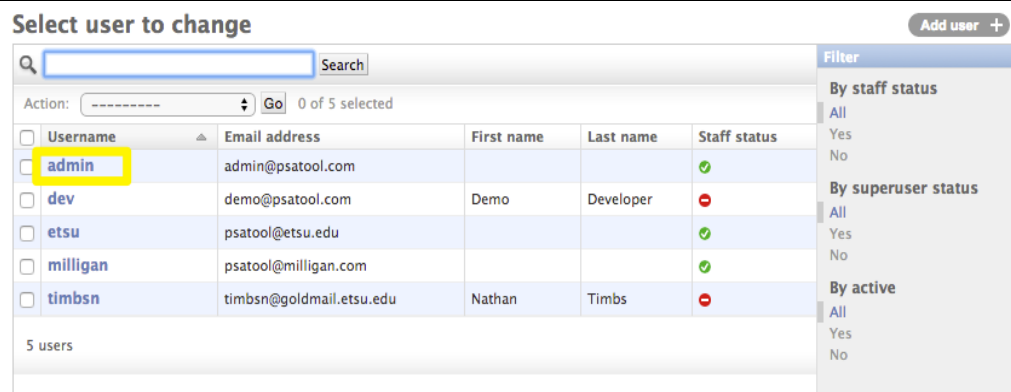


Figure 15: List of users

Check 'Staff status' or 'Superuser status' to assign manager role to a user. Users without Staff status or Superuser status are assessor users. (Fig. 16)

Change user History

Username:
 Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 12000 **salt:** t991V5***** **hash:** LFBoyi*****
 Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:

Last name:

Email address:

Permissions

☒ **Active**
 Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☒ **Staff status**
 Designates whether the user can log into this admin site.

☒ **Superuser status**
 Designates that this user has all permissions without explicitly assigning them.

The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Figure 16: Assigning manager/assessor role to a user

Add a new user by clicking on Add User in users list (Fig. 17).

Select user to change Add user +

Search

Action: Go 0 of 5 selected

<input type="checkbox"/>	Username	Email address	First name	Last name	Staff status
<input type="checkbox"/>	admin	admin@psatool.com			✓
<input type="checkbox"/>	dev	demo@psatool.com	Demo	Developer	✗
<input type="checkbox"/>	etsu	psatool@etsu.edu			✓
<input type="checkbox"/>	milligan	psatool@milligan.com			✓
<input type="checkbox"/>	timbsn	timbsn@goldmail.etsu.edu	Nathan	Timbs	✗

5 users

Filter

By staff status

All

Yes

No

By superuser status

All

Yes

No

By active

All

Yes

No

Figure 17: Adding a new user

While adding a new user, click on ‘Save and continue’ to add additional details about a user (Fig. 18).

•

Home > Users > Users > Add user

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

Password confirmation:
Enter the same password as above, for verification.

[Save and add another](#) [Save and continue editing](#) [Save](#)

Home > Users > Users > test

✓ The user "test" was added successfully. You may edit it again below.

Change user

[History](#)

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 12000 **salt:** 39qUeY***** **hash:** wgjA46*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using [this form](#).

Personal info

First name:

Last name:

Email address:

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ **Staff status**
Designates whether the user can log into this admin site.

☐ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

Figure 18: Additional information for a user

APPENDIX D: PSATOOL MANUAL: MANAGING ASSESSMENTS

An assessment group is used for easier management and grouping of assessments. For example, a different assessment group can be created for each of an organization's buildings. Assessors can then be assigned to an active assessment for an assessment group. The assessors can gather requirements for their assigned buildings.

Only managers can configure assessment groups. This is done via the Assessment Groups link in the side navigation (Fig. 19).

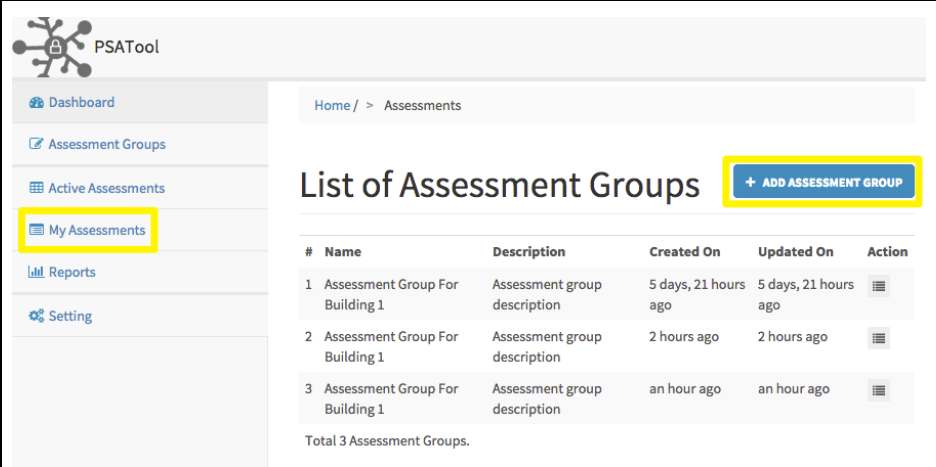


Figure 19: Listing assessment groups and creating a new group

New assessment groups can be created by clicking ‘Add Assessment Group’ (Fig. 20). Provide a name and description for what this group denotes (e.g. building name and motivation for assessment)

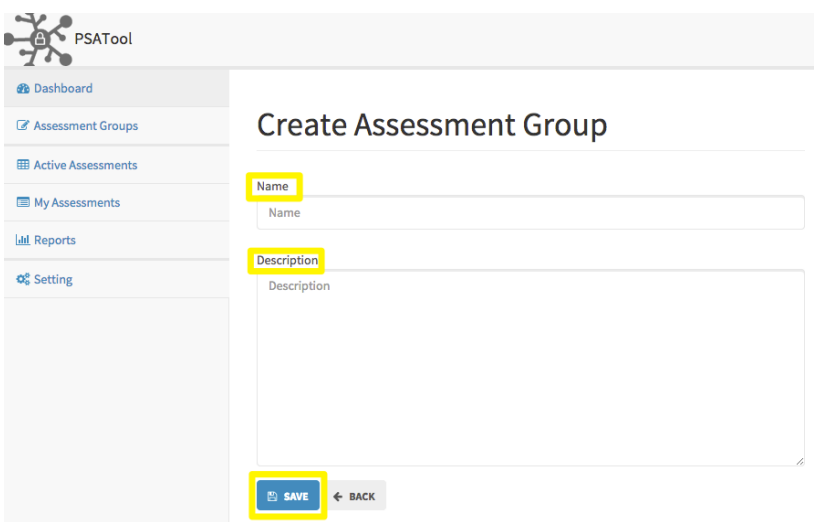


Figure 20: Creating an assessment group

After creating a group, assessment requirements should be associated with that group (Fig. 21). Click the ‘Manage Requirements’ icon to initialize and (possibly) modify a group's requirements. PSATool automatically adds 52 requirements when initializing a group's requirements. Click on ‘Initialize Requirements’ to load these 52 requirements (Figure 22).

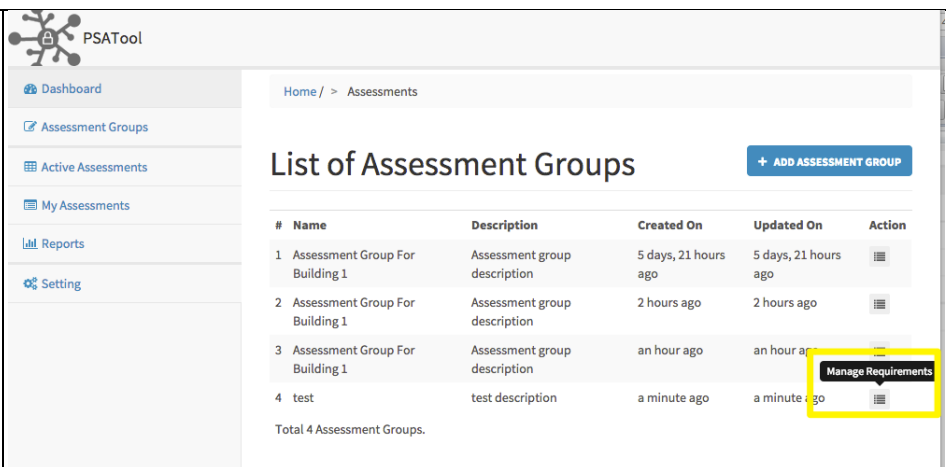


Figure 21: Managing requirements

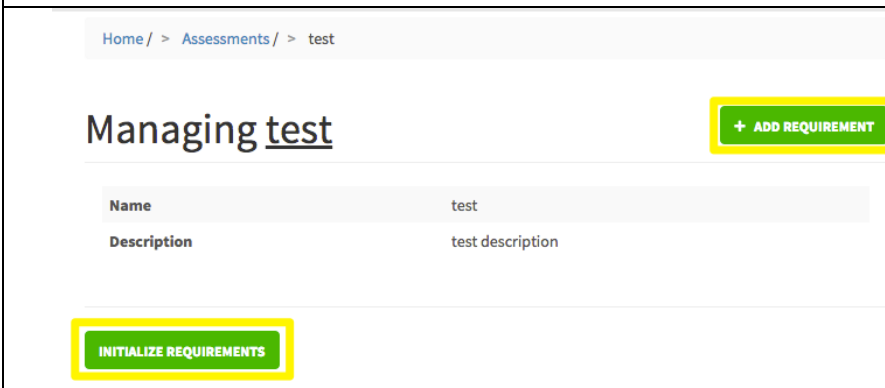


Figure 22: Initializing assessments and add custom requirement

PSATool also allows the addition of custom requirements. Click on ‘Add Requirement’ (Fig. 23) to create a custom requirement.

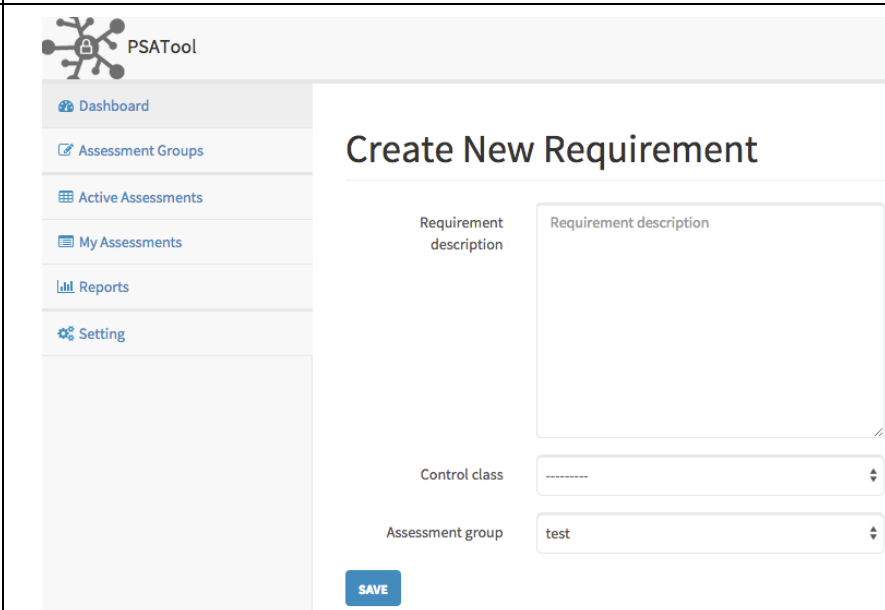


Figure 23: Adding a custom requirement

For the 52 requirements that PSATool initializes, icons provide access to information that includes the date of last update (Fig. 24) and the requirement's author (Fig. 25) and provenance (name of authority for a requirement) (Fig. 26) .

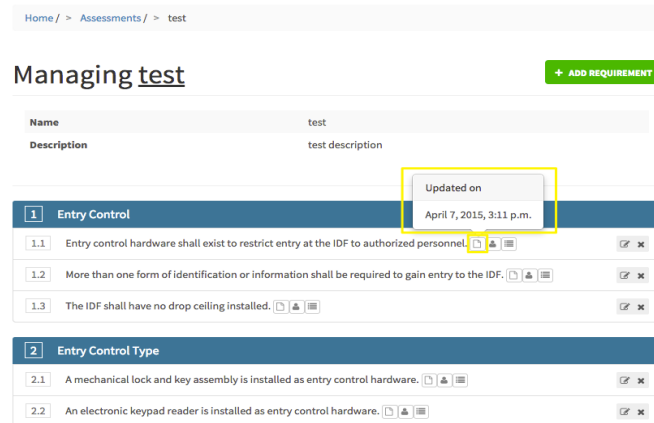


Figure 24: Updating a requirement's date/time

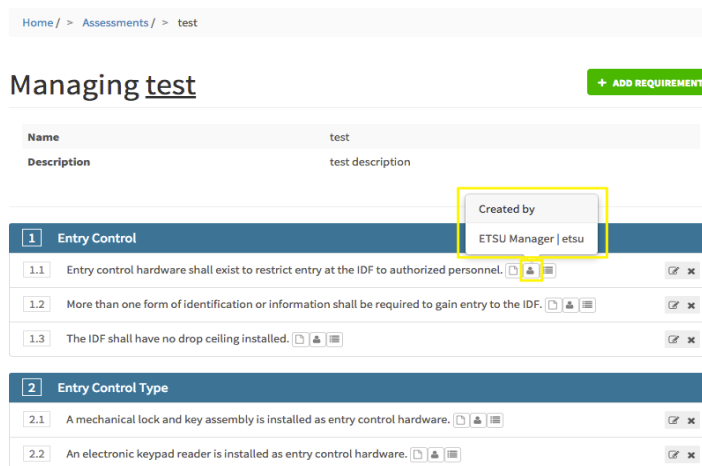


Figure 25: PSATool user who created a requirement

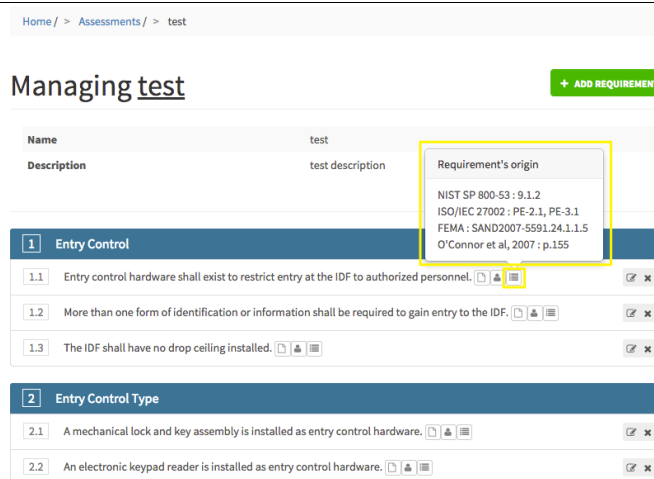


Figure 26: Source of a requirement

APPENDIX E: PSATOOL MANUAL: GENERATING REPORTS

Reporting is an important component of PSATool. After gathering data from IDFs, a manager can generate a summary report or an IDF-specific report. A summary report presents an overview of an assessment and ranks an IDF based on passed and failed requirements. An IDF specific report provides details about requirements that an IDF passed or failed.

Reporting can be accessed by clicking 'Reports' from the side menu. Selecting an assessment and clicking 'View Report' will display a summary report. (Fig. 27)

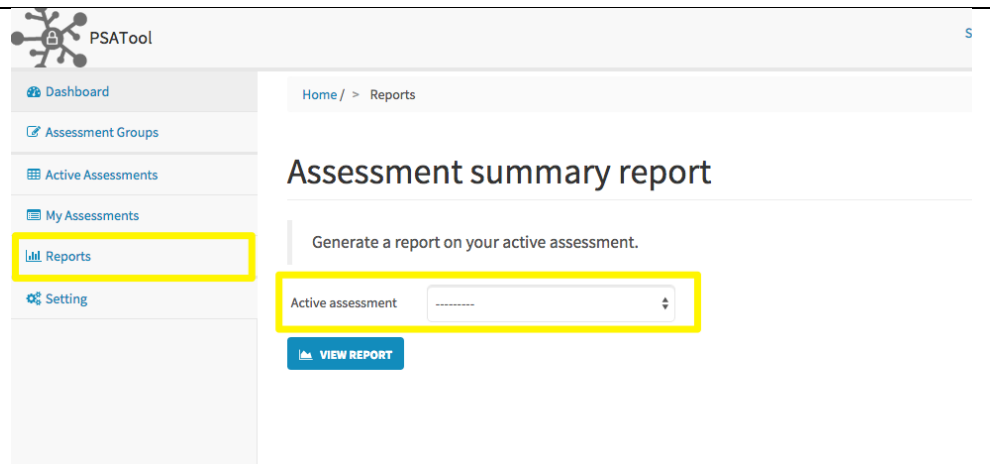


Figure 27: Reports page

A report can be printed using “print this report” (1) button.

The bar charts on a report display the rank of IDFs.

Rank according to passed requirement

(2) displays the top 5 ranked IDFs.

(3) displays the bottom 5 ranked IDFs. This graph is only displayed for assessment with more than 5 IDFs.

Rank according to failed requirement

(4) displays the top 5 ranked IDFs.

(5) displays the bottom 5 ranked IDFs. This graph is only displayed for assessment with more than 5 IDFs. (Fig. 28)

Assessment summary report

PRINT THIS REPORT 1

Assessment Name

Assessment Group For Building 1

Description

Assessment group description

Assessment created on

March 1, 2015, 5:35 p.m.

IDFs assessed

10

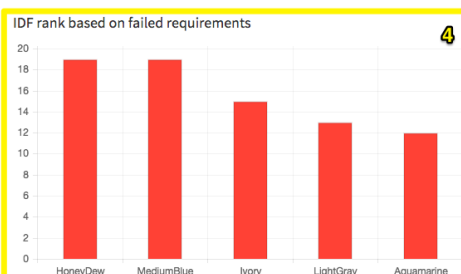
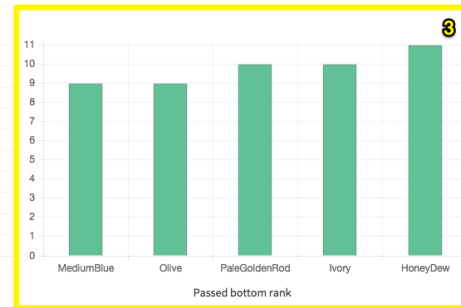
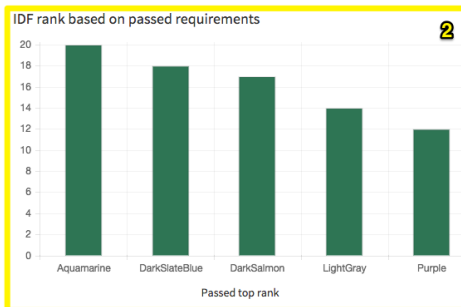


Figure 28: Bar charts displaying IDF rank


The summary report lists IDF's assessed, time required to assess them and all requirements along with the IDF's that passed each requirement.(Fig. 29)

IDFs Assessed				
Name	Last Assessed Date	IDF Assessment Duration	Assessor	
HoneyDew	April 15, 2015, 2:14 p.m.	0.56 minutes	Sulabh Bista sul4bh Dev Loper dev Deve Loper devdev YO LO yolo	
PaleGoldenRod	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
Aquamarine	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
Ivory	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
DarkSalmon	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
MediumBlue	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
DarkSlateBlue	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
Olive	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
LightGray	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	
Purple	March 1, 2015, 5:35 p.m.	0.0 minutes	Sulabh Bista sul4bh	

Requirements Assessed					
Entry Control					
#	Requirement	Passed IDFs	Failed IDFs	Not Applicable IDFs	IDFs not assessed
1.1	Entry control hardware shall exist to restrict entry at the IDF to authorized personnel.	DarkSalmon DarkSlateBlue	HoneyDew Aquamarine MediumBlue	Ivory Olive Purple	PaleGoldenRod LightGray
1.2	More than one form of identification or information shall be required to gain entry to the IDF.	DarkSalmon	Aquamarine MediumBlue DarkSlateBlue	PaleGoldenRod Ivory LightGray Purple	HoneyDew Olive
1.3	The IDF shall have no drop ceiling installed.	HoneyDew DarkSalmon MediumBlue DarkSlateBlue Olive LightGray	Aquamarine		PaleGoldenRod Ivory

Figure 29: List of IDF's and requirements

An IDF specific report can be accessed by clicking on an IDF name from the IDF list (see highlight in Fig. 3). The IDF specific report displays details about all passed, failed, not-applicable and unassessed requirements. This report can be printed using 'print this report' button. (Fig. 30)


PSATool
Sulabh Bista (logout)

[Dashboard](#)
[Assessment Groups](#)
[Active Assessments](#)
[My Assessments](#)
[Reports](#)
[Setting](#)

[Home](#) / > [Reports](#)

Report on IDF HoneyDew

[PRINT THIS REPORT](#)

Assessment Name Assessment Group For Building 1

Description Assessment group description

Assessment created on March 1, 2015, 5:35 p.m.

Click to expand

Passed requirements (11)

Failed requirements (19)

Not applicable (10)

Not assessed (12)

Figure 30: IDF specific report

APPENDIX F: PSATOOLAPP MANUAL

PSAToolApp is used by assessors for gathering requirement data from IDFs.

PSAToolApp can be installed on a mobile or tablet device. PSAToolApp requires a constant Internet connection because it communicates with the PSATool web server for retrieving and storing data.

PSATool is available from the Google Play Store

(<https://play.google.com/store/apps/details?id=com.psatool.psatoolapp>).

Assessors can log in using the PSAToolApp login screen (Figure 31). The login screen allows an assessor to enter an API end point. The assessor gets the API endpoint address from a PSATool manager. Once entered, API end point is saved and will be used for future logins.

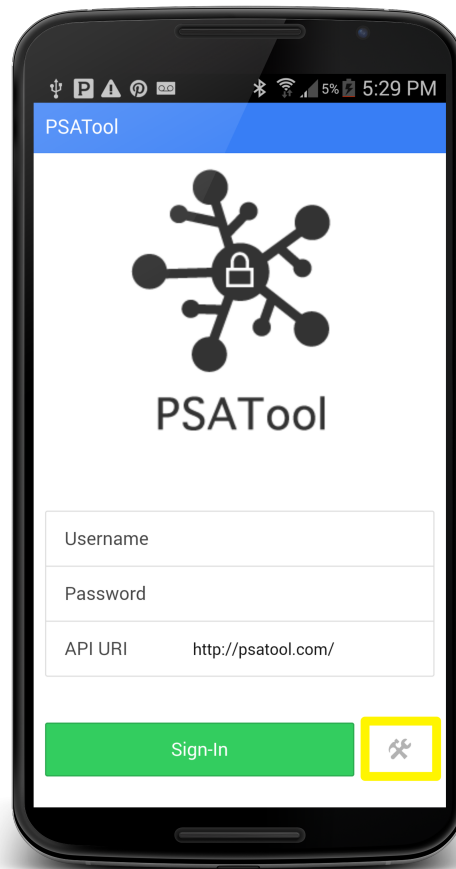


Figure 31: Login screen with API URL selection

After assessors are logged in, they can access a list of assessments assigned to them. (Fig. 32). The assessment list contains a list of IDFs and the percentage of requirements assessed in the form of pie chart.

New IDFs can be added by clicking on 'Add IDF'. Existing IDFs can be edited by clicking on the pie chart icon.

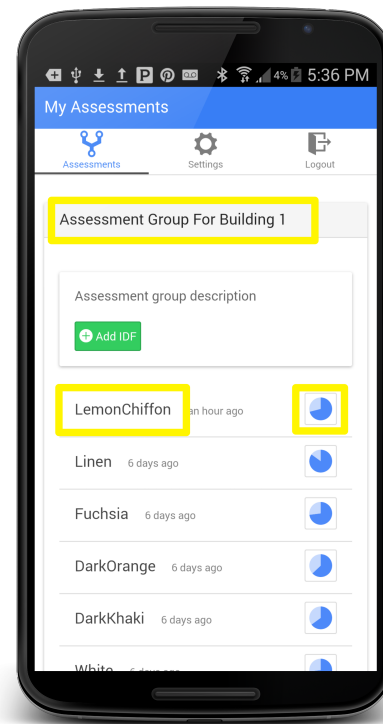


Figure 32: Assessment and IDF list

The order of IDFs can be changed using the Settings tab (Fig. 33)

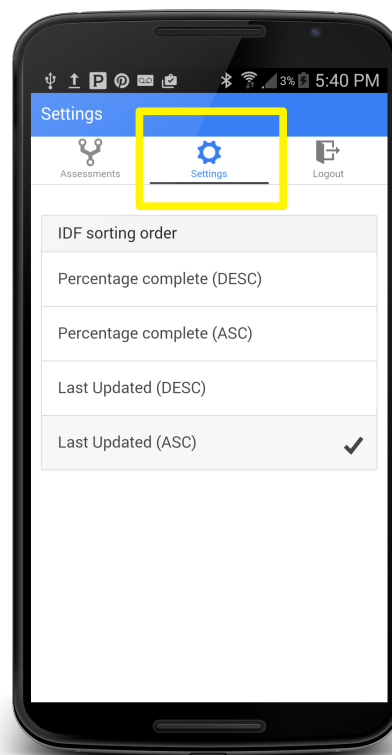


Figure 33: Changing the order of the IDF list

Adding or editing IDF data starts with selecting users involved in the gathering of requirements data (Fig. 34).

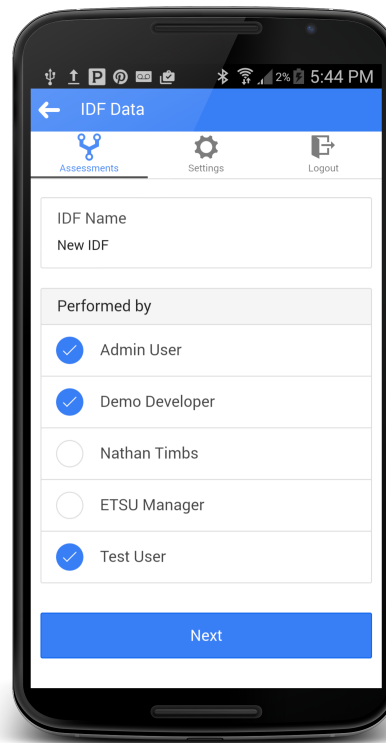


Figure 34: Selecting users involved in data gathering

A list of requirements is presented next (Fig. 35). A [...] button can be clicked to add data about specific requirement. THV data can be added clicking ‘THV Data’ button.

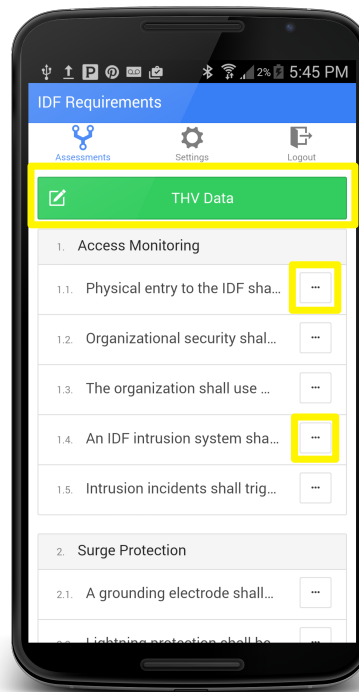


Figure 35: List of requirements

The screen to gather requirement data (Fig. 36) allows an assessor to select if the IDF being assessed meets a requirement, take a picture and add a comment. Clicking on “Save”

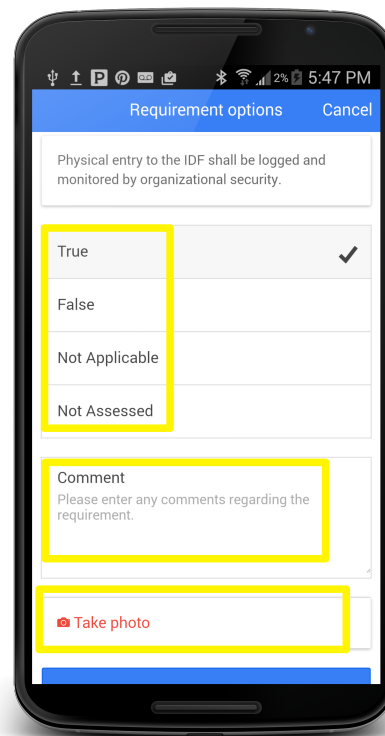


Figure 36: Requirement gathering screen

An assessor can log out from the app by using the Logout tab (Fig. 37).

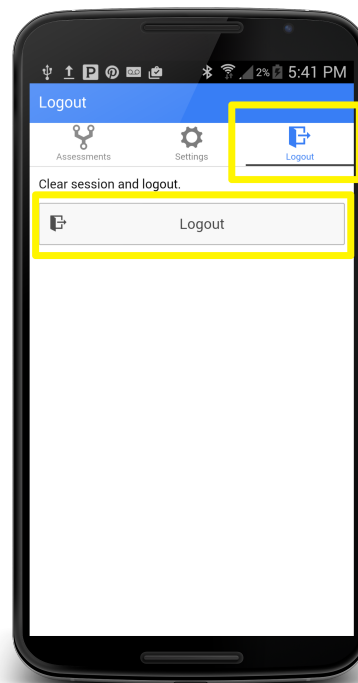


Figure 37: Logout screen

Various icons on the requirement list screen reflect a requirement's state (true, false, not applicable, not assessed) (Fig. 38)





Icon Convention	
True	
False	
Not Applicable	
Not Assessed	

Figure 38: Icons for requirement state

APPENDIX G: REST API SPECIFICATION

The following table documents PSATool's API for communicating with PSAToolApp. API functions return HTTP response code 200 on success, code 404 for requests that fail to generate data (non existent unique key) and 500 for requests that are incomplete or can't be fulfilled.

API name	Login
Description	Authorization of a user based on a username and a password
HTTP URL	<PSATool server url>/api/login/<username>/<password>
HTTP Method	GET
HTTP Response in JSON format	{ "username": <username>, "token": "<authorization token>",&br/> "is_staff": <boolean> }
Description of response	This function, on success, returns an authorization token. This token is associated to a user. Subsequent API requests use this token as an authentication criteria.
All other requests include an HTTP Authorization header with the value "Token <authorization token> from the above request. For example, if Login API returned the token zxybdkfj1234, the HTTP Authorization header contains the value Token zxybdkfj1234	
API name	My Assessments
Description	Fetches a list of assessments for the logged in user
HTTP URL	<PSATool server url>/api/assessments/sort/<sort criteria>/order/<sort order> sort criteria: updated_on, complete sort order: desc, asc
HTTP Method	GET
HTTP Response in JSON format	[{ "id": <assessment unique id>, "assessment_group": { "id": <assessment group unique id>, "created_on": <datetime>, "updated_on": <datetime>, "name": <name of assessment>, "description": <description of assessment> }, "assigned_to": [{ "id": <user unique id>, "username": <username>, "name": <user real name> }], "idfdata_set": [{ "id": <idf unique id>, "idf_name": <name of IDF>, "updated_on": <datetime>, "percentage_complete": <percentage of IDFs assessed> }], }]

	<pre> { "id": <idf unique id>, "idf_name": <name of IDF>, "updated_on": <datetime>, "percentage_complete": <percentage of IDFs assessed> }] }]</pre>
Description of response	The response lists all assessments assigned to a user, all users assigned to an assessment, and all IDFs under each assessment
API name	IDF Base Data
Description	Fetches a name for an IDF and a list of users who performed assessment on it
HTTP URL	<PSATool server url>/api/idf_base_data/<idf id>
HTTP Method	GET
HTTP Response in JSON format	<pre> { "idf_name": <name of idf>, "users": [{ "id": <unique user id>, "username": <username>, "name": <user real name> }, { "id": <unique user id>, "username": <username>, "name": <user real name> }] }</pre>
API name	Save IDF Data
Description	Create/update IDF name and list of users performing an assessment
HTTP URL	<PSATool server url>/api/idfdata/<assessment id>/<idf id>
HTTP Method	POST
POST Data	<pre> { "idf_name":<name of IDF>, "performed_by": [<user id>, <user id>] }</pre>
HTTP Response in JSON format	<pre> { "message": "IDF Data set.", "idfdata_id": <idf id> }</pre>
API name	Requirements
Description	Fetches a list of requirements, their attributes and THV data for a specific IDF
HTTP URL	<PSATool server url>/api/requirements/<idf id>
HTTP Method	GET

HTTP Response in JSON format	<pre> { "thv_assessment": <THV assessment text>, "thv_control": <THV control text>, "requirements": [{ "class_name":<requirement class name>, "requirements": [{ "id": <unique requirement id>, "text": <requirement text>, "option": <option value selected>, "comment": <comment text>, "media": <url to any associated media>, "duration": <time in milliseconds> }] }] } </pre>
API name	Save Requirement Data
Description	Update data for a specific requirement.
HTTP URL	<PSATool server url>/api/requirementdata
HTTP Method	POST
POST Data	<pre> { "comment": <comment for a requirement>, "id": <unique id of a requirement>, "media": <base64 encoded image data>, "option": <selected option for a requirement>, "text": <requirement text>, "duration": <time in milliseconds> } </pre>
HTTP Response in JSON format	<pre> { "message": "Requirement data saved", "requirement_data_id": <unique id of the updated requirement> } </pre>
API name	Requirement Options
Description	Fetches a list of available options and its related text
HTTP URL	<PSATool server url>/api/requirement-options
HTTP Method	GET
HTTP Response in JSON format	<pre> [{ "id": <unique id>, "name": "True" }, { "id": <unique id>, "name": "False" }, { "id": <unique id>, "name": "N/A" }, { "id": <unique id>, "name": "" }] </pre>

VITA

SULABH BISTA

Personal Data: Date of Birth: July 10, 1989

 Place of Birth: Dhangadhi, Nepal

 Marital Status: Unmarried

Education: BE in Computer Engineering, 2011

 Kathmandu University, Dhulikhel, Nepal

 MS Computer and Information Sciences, 2015

 Johnson City, Tennessee

Professional Experience: Software Developer, ETC; Johnson City, TN, 2014-2015

 Software Engineer (Remote), Scorewize; Los Angeles, CA

 2011-2013

 Freelance Software Developer; 2011-2013