Automating UI Guidelines Verification by Leveraging Pattern Based UI and Model Based Development

Satya Viswanathan
SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
satya.viswanathan@sap.com

Johan Christiaan **Peters**
SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
Johan.christiaan.peters@sap.com

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.
ACM 978-1-60558-930-5/10/04.

Abstract

In large enterprises different teams work on different parts of a big software application. Therefore, retaining user interaction paradigms and concepts becomes important. However, during the development of a large software product, these principles and paradigms get progressively diluted, due to trade-offs, differences in interpretation, communication errors and many other reasons. In order to remain true to design rationale and communicating them to a wider audience/consumers, often User Interface (UI) Style Guide are created. The style guide attempts to sensitize and educate its consumers about design principles and document some of these design rationales for references.

However, the *usability, usage and adoption* of these UI guidelines within an organization are topics frequently discussed and debated in several forums for years.

Post the 'design and definition phase' of software development lifecycle, UI designers are often required to do 'quality checks' as the UIs get developed. Despite painstakingly defining every interaction to its finest level of granularity, in practice the guidelines are often not followed or interpreted incorrectly.

The method of manually inspecting the 'implemented' user interface for compliance to UI guidelines has the following pitfalls:

- Highly effort and time consuming
- Outcome is often inaccurate, unreliable and sub-optimal in quality
- Findings are too late in the process to be fixed.
- Not an efficient process for tracking issues to resolution

This case study talks about the challenges we faced with our UI Style guide and how we tackled them. Based on internal user research and design thinking we defined an approach of *better integrating* UI style guide into the software design and development process. We leveraged the benefits of pattern based UI approach and a model based development environment to achieve compliance to our UI guidelines by:

- Providing tools to automate verification of UI guidelines in the model based development environment
- Redefining the development process to support UI verification early-on during the design and development process

Keywords
Graphical User Interfaces, Usability Evaluation automation, Style Guides, Standards & Guidelines, Model based user interface design, Usability

ACM Classification Keywords
Categories and subject descriptors: H.5.2 [Information interfaces and presentation (e.g., HCI)]: User Interfaces---Evaluation / methodology, Style Guides

General Terms
Design, Documentation, Experimentation, Human Factors, Reliability, Standardization

The Basic Framework
While developing our large business software suite we need to cover a huge number of business scenarios, uses cases, thus needing many different screens. Our final application has about 4000 screen. Starting in 2001, we adopted the approach of Christopher Alexander [1] and developed a generic UI pattern model [2, 3]. Our expectations from adopting a pattern based model were:

- Reduce production costs: Once a UI pattern has been developed it just can be re-used and only needs be configured according to the business process thus simplifying and accelerating the development process.
- Increased UI consistency across different business scenarios
- Reduced learning curve for end users

A pattern based UI approach also introduced model based development environment. We decided to de-couple the creation of screens from the development of the underlying business logic. With this combination, we could do a lot of upfront product definition, working on prototypes, before actual implementation started.

The idea of using patterns for UI design has been described already by Borchers [4]. To begin with, we went down the conventional path of defining generic UI patterns for our product too: e.g. object Work Lists (OWL), Contextual Navigation Pattern, Identification Region, Forms, Tables, and so on. Each pattern served a specific function. For example, the OWL shows a list of all objects the user is responsible for. The interactions within these UI patterns were standardized at a generic level. E.g. 'selection' of an object in an Object work list; a toolbar that provided generic, possible actions that can be done on a selected object etc.

Apart from these UI patterns, we defined some generic 'work spaces/ window types' (we called them Floor plans) in which these patterns could be used.

User Centered Design

Contrary to expectations, this basic framework did not limit 'creativity' and 'user centeredness' of the product. In fact the time designers saved by following a pattern based approach allowed them to do more upstream and core design tasks. Extensive user research provided vital information for extending the base framework; refining the generic 'patterns' (based on the understanding of how users perform their tasks) or defining new patterns (catering to new use cases)

A combination of UI controls and floor plans created user centric patterns that can be further reused. E.g. including a 'search control' with the OWL became a standard pattern to allow the users to search within the list of objects.

The Style Guide

The principles behind the basic framework; the patterns; their usage; best practices and examples were documented in our UI Style guide. This enabled all the stakeholders – UI designers, solution managers, and technical authors, developers – to have a common understanding of the design principles and to apply patterns and floor plans in a same way.

In principle, patterns were a set of guidelines/ best practices based on our understanding of our users and how they performed their generic tasks. However, owing to the complexities of integrated enterprise software, dealing with vital business data, it was not enough for us to define the 'patterns' at a design level, targeted at only the UI designers.

We had the following additional layers of complexities:

- The UI style guide was consumed by UI designers, UI developers and in some cases, the business analysts. Unlike the Yahoo Design Patterns library (targeted primarily at designers), we needed to provide further details, addressing needs of multiple consumers.
- Every design pattern was a consolidation of multiple 'controls.' The interactions, behavior and visual design of these were defined centrally.
    - Often, we had dependencies on 'interaction possibilities' with a control, based on the type of business content; the business process.

UI Designers could define 'variants' of 'usage' of these patterns depending on their end user needs and use cases. However, it posed a challenge downstream when we moved into the 'production' phase.

The development communities found it hard to comprehend the finer points of the style guide, because they had not been part of the initial the user research and the resulting situational scenarios. This was difficult due to our distributed development set up. The following examples illustrate aspects of UI guidelines that were not understood by developers:

- If sub-views are used, there should be at least 3. If there are 2, consider redesigning the screen. Avoid using more than 7 sub-views. (Developers were not sensitive to such numbers/ heuristics of +/- 7)
- Provide sub-views only if the selected view has additional sub-categories that warrant a separate navigational element. (Developers had different decision parameters of warranting a separate navigation element. For example, they would decide based on the complexities of having separate navigational elements in the backend architecture/code.)
- Multiple 'flavors' of 'Save' buttons – to address different use cases: There were times when the end user wanted to simply edit (master) data and 'Save'. Other times, the end user desired to create/edit data and expected to 'trigger a workflow' in the background. In such scenarios, the UI guideline recommended to name the button with an appropriate 'action' label instead of a simple save. (In the backend, the Developers could program either buttons to behave in the appropriate manner – the change in label was not important for them)

This led to the development community rejecting the document as a reference and applying their own logic/creativity to the User Interfaces they were developing – resulting in 'inconsistencies' across thousands of developed User Interfaces.

- During the course of development, the teams ran into various constraints and exceptions which called for different design proposals. Owing to the model based development environments, most often, these constraints and exceptions were limitations on a framework/landscape level. However, (due to various reasons) each team took different approaches to work around them.
- The User Experience (UX) team was now posed with an additional challenge to ensure these 'work around designs' were consistent. We were forced to document the design proposals for exceptional cases into our Style Guide hoping that every team facing similar issues will use a common approach.
- The development community had created their own 'instant' (UI) fixes for their technical constraints. By this, we learnt, that the UI style guide needed to include, "what NOT to do" aspects too.
- The (already) 500 pages Style Guide grew even bigger – turning it unusable.

The Production Phase

This phase of software development tends to be rather 'industrial' in nature – thousands (in the range of 4000) of screens had to be produced, by thousands of developers, hundreds of stakeholders working across multiple geographical locations. This called for increased process orientation, high degree of standardizations and repeatable processes. Also, such large scale production demands a strong quality assurance & control processes.

While our (UX team) intentions were noble, the UI Style Guide was not a widely read document by developers and/or other consumers.

- It is not considered a relevant document for their core tasks – coding, testing etc.
- It did not speak the language they understand – the syntax, the terminologies etc. Instead spoke of 'design principles'; recommendations/best practices – things that are seemingly farfetched for the community
- It did not instantly provide them with answers they seek – answers to their constraints and exceptions
- The voluminous style guide was hard to comprehend

A good number of inconsistencies arose during the transformation of wireframes into implemented screens due to various reasons.

- Technical constraints
- Globalization requirements
- Specific business process requirements
- Requirement based on specific end user needs
- Assumptions made by stakeholders as the UI Guidelines were too generic.
- Compliance to legal standards

Towards the end of the production/development phase, we found ourselves in a quality nightmare. There were too many inconsistencies in our user interfaces – some of them of course were seemingly justifiable, but many were not.

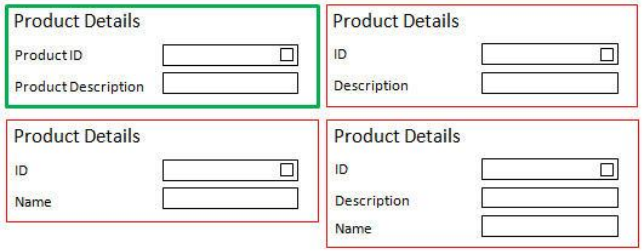

Figure 1. Inconsistency example: The box in 'green' illustrates the prescribed/desired labels for the fields in a Form. The boxes in 'red' illustrate the different variations we had.

The 'Clean Up Mission'

We (UX team) along with others in the team, became full time quality controllers. We did UI wall walks, scenario comparisons, manual UI reviews, UI testing on test systems, issue reporting, issue tracking, escalations…all of which can be categorized as 'quality control' tasks – not 'design' tasks.

The production/development phase is also a very unstable one. As the development framework and landscape issues got resolved centrally, we had new situations to tackle. E.g. our guidelines for work around designs need to be updated almost continuously.

- Issue categories were understood differently by different people.
- The quality control team came in too late. This team could only test once the developers completed development, published their models and the product was 'officially' in 'testing' phase.
- Until the start of the 'testing phase' our primary task was only identifying and documenting UI issues.

- Finally, during the 'correction phase' developers were overloaded with a long list of issues that included UI inconsistency issues plus other technical issues.
- Invariably, against the light of technical issues, UI issues were placed much lower in priority, often resulting in them not being resolved.

At one point, the entire organization was spending too much time and effort on such tasks and rework than on our respective core tasks.

The '(in) consistency' issue

The definition of consistency and to what level this consistency should be attained is often a debatable topic. It is a struggle to define a basis for consistency that does not come in the way of ease of use and consistency that does not stifle innovation and good user centered design. Considering we were faced with a high number of 'UI inconsistency' issues, it was important for us to define what 'inconsistency' meant for us and where to draw the line.

We refer to Jonathan Grudin's article on 'The Case against User Interface Consistency' [5] for the three categories of 'consistencies' described by him.

- Internal Consistency: Consistent interaction paradigm within a product/across product lines (e.g. Window handling, key functions/buttons, terminology, layout etc) Initial learning in particular, but also ease of use and perceived quality are reported to benefit from internal consistency.
- External Consistency: Consistent with user's conceptual model (of previously used similar systems). Full external consistency may not generally be possible. "Transfer of training" is then a key concern.
- Consistent with metaphoric correspondence of design features in the world beyond computer systems. Since real-world domains are part of everyone's experience, analogies may be significant aids to initial learning and recall

Our UI Style guide combined design guidelines, consistency recommendations and rules relating to all consistency categories, however a closer look at our 'UI inconsistency' issues, revealed that majority of our issues were related to internal inconsistencies.

Why were internal inconsistencies so important for us?

Our product provides integrated enterprise software dealing with vital business content and processes spanning multiple functional areas – Supply Chain Management; Financials; Supplier Relationship Management; Customer Relationship Management; Human Resources. Different groups spread across different locations focus on developing their functional area. Typically the team members are functional experts of their respective areas.

Our user research data indicates that our end user often multi-task – work across these functional areas. Example, a single end user could be responsible to work with Financials as well as some parts of Human Resources areas.

'Consistent interaction paradigms' are therefore very important to make this integration between functional areas feel seamless and retain a strong brand value.

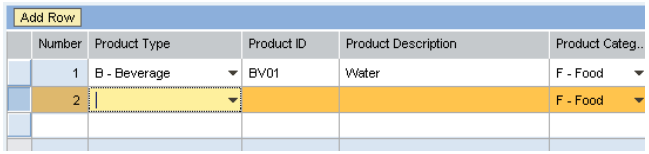

Figure 2. Inconsistency example: We had multiple variations of how 'Add Row' feature worked. In some cases, the new row was created to be displayed as the first one; other cases, it was displayed to be the last row but was selected by default.

Our Objectives

Our primary goal was to substantially reduce the effort spent within the organization in verifying the user interfaces for internal consistency and compliance to our UI Style Guide. We had several aspects to address:

- Define a methodology to proactively avoid/correct UI problems early on in the development process
- Leverage the benefits of a model-based development environment – provides additional context (meta-model) that can be used by verification tools to check guidelines with increased accuracy (e.g. guidelines distinguishes look-and-feel for a table control that lists a set of tasks from a table control that lists a set of items).
- Reduce the need for developers and designers to remember every little detail of the UI Style Guide.
- Improve overall internal consistency
- Improve compliance to UI Style Guide
- Reduce the MANUAL effort spent by everyone in order to achieve consistencies and compliance.

Our Approach

Given the situation and its scale of operation, we opted to experiment with automation to meet our objectives.

We wanted to explore the possibilities of creating a check/verification tool that could be integrated into the development tool used by our developers. The basic expectation was to run an automated check against some UI rules directly within the development tool and generate a report that illustrates the results.

We were aware that design guidelines cannot be completely automated. Hence it was important for us to make a distinction and define those that could be. As a first step, we separated the 'standards' (overall, generic pattern descriptions) from the 'guidelines' (application specific rules and recommendations).

Within these guidelines were consistency rules and recommendations that related to internal and external consistencies. Since external consistencies are more dependent on specific use cases, user needs, we decided to exclude external consistency guidelines from automation. We trusted that our upfront product definition phase following user centered design process and later, usability testing would address these sufficiently.

We laid our focus on internal consistency guidelines and further filtered out 'recommendations' from the 'rules'

We classified those guidelines, violations of which would cause serious break down to end user's task completion efficiency and effectiveness as 'rules.' Using the same principle, we defined priorities to rules.

Priorities were assigned to the rules in order to decide how critical it is to fix a violation of the rule. If, due to a violation of a rule, a user will not be able to complete a task successfully (for e.g. a missing 'close' button) it was marked as 'High'. If the user can complete the task, albeit with some cognitive effort it was rated as 'Medium' (for e.g. 'close' button not placed in the correct position of the screen). If the violation does not impact user's performance but affects the overall emotional appeal or perceived usability, these rules were marked as 'Low (for e.g. duplicate header or title).

We could define approximately 350 distinct rules.

The development group used these rules as source for developing a formal method of verification using a rule engine. Owing to our model based development environment and de-coupling of UI from its underlying logic, it was also important to distinguish rules that check the presentation model versus rules that check the navigation model. Guidelines for the presentation model such as layout (font, size and position), grouping (forms and sections) could be verified in the development environment and would ideally not require verification at runtime. Guidelines for the navigational model could only be checked at runtime (e.g. check if a hyper link on an employee name actually opens a popup of the correct type). The verification tools were built to verify both kinds of guidelines.

The low hanging fruits

- Since we made a distinction between checks on 'presentation model' Vs check of 'navigation model' the checks on presentation model could be done earlier.
- Developer's dependency on the quality team reduced as developers could now trigger UI verification themselves by a click of a button in their development environment; they could view a report that showed UI 'inconsistency' issues with their ratings; these issues could be fixed by the developers during development.
- The quality assurance group too could do these checks early on by triggering verifications centrally on nightly builds.
- Reduced overhead of time, effort and communication delays faced during manual testing.
- Reduced the burden of learning granular details of the UI guidelines; its updates. Rolling out new updates of these guidelines became easy – we only needed to update the rules/ add new rules/ remove old rules.
- Enhanced 'reporting' capabilities increased transparency of the entire process, early on.

Essential Learning

- (Most significant) Automated inconsistencies checking, freed designers to invest their efforts on higher level design improvements, user research and user validations rather than focusing on guideline compliance. This allowed the UI designers to once again focus on their 'core' tasks.
- The combination of 'pattern based UI approach' and 'model based development environment' provides a framework to cover more number of guidelines.
- Development productivity and quality of end product significantly improved with a proactive methodology to fix UI issues.
- It is important to make distinctions between:

- Standards versus detailed product specific guidelines.
- External inconsistencies guidelines versus internal inconsistencies guidelines.
- Checks on Presentation model versus checks on navigation model

After applying all these filters, approximately 60% of our UI Style guide could be categorized into 'internal consistency' and be translated into rules to be automated.

- Several organizational challenges such as buy-in from management, coordinating deliverables and managing expectations of people were hard to handle. However, close and frequent interactions with different stake holders helped. We (the UX team) become the linking pin connecting different people and perspectives together. This also gave the User Experience group to play a persuasive and influential role in the process [6].
- An attempt at formalizing the guidelines into rules which could be tested revealed grey areas in the UI style Guide that were not clear and needed rewriting. These areas did not become apparent earlier.
- As user interface design guidelines are never exhaustive, it never covered all edge cases that needed an exception to following the rule. We previously struggled on documenting and updating such exceptions and edge cases into our UI style guide, with the risk that it may not be read/ maybe overlooked. Automation allowed us to directly enter these rules into the engine and making it available for verification.
- Increased and early transparency of UI inconsistency issues gave us sufficient lead time to resolve them appropriately
- This approach made it possible to instill a level of discipline without compromising on the flexibility required for creating user centric design solutions for our business cases.

Summing it up, the key achievement is that this approach allows us to be faster and better in our tasks – an increasingly growing demand in the industry.

Related Work

- We looked at the Yahoo Design Patterns Library extensively. Our goals were similar: increase consistency, predictability, and usability across applications. However, we soon realized the differences:
  - The Yahoo Design Patterns library documented design aspects of patterns and was targeted primarily at designers. Our UI Style Guide had varied consumers.

    Since Developers were big consumers of our Style Guide we needed a lot more details with respect to right 'usage' of these patterns. E.g.: *A team of Developers once experimented having Bread Crumb Navigation inside an Accordion view. Upon design review, this usage was considered inappropriate.*

    We would need to clearly specify the usage of Progress Bar in different types of windows – modal; non-Modal windows. Such details could have implications on data handling.
  - Owing to the different types of business content we handled, there were many 'conditional' aspects to our UI guidelines. These needed detailed specifications; instructions and often 'what not to do' aspects too.

- There are many web usability and accessibility verification tools in the market that address compliance to widely accepted set of guidelines (such as WCAG and Section 508 Web Accessibility Standards). However they do not provide a way to define and check organization specific UI guidelines
- Ivory and Herst's survey [7] indicates that most of the usability validation tools do quantitative analysis of the user interface based on end user's usage patterns. The analysis exposes fundamental issues in design arising mostly from low performance. While it may help improve the design, it cannot support a qualitative analysis of the user interface based on guidelines
- It has proven to be hard to formalize informal style guides written in free flowing text in design time tools. This opinion is frequently discussed and debated in all major forums including SIGCHI.
- There has been only one work so far [8] that leverages the benefits of a modeling environment to check for UI guidelines compliance.

Acknowledgements

The methodology was developed by the user experience group under the hat of innovative ventures. This couldn't have been a success without close collaboration with the development and quality groups and support from the management. We thank all who contributed for the success of this initiative.

References

[1] Alexander, C. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York, 1977.

[2] Waloszek, G. and Eberleh, E. *Introduction to User Interface Patterns at SAP*, 2003. http://www.sapdesignguild.org/community/design/patterns_sap.asp

[3] Arend, U. *User Interface Patterns Components for User Interfaces*, 2004. http://www.sapdesignguild.org/editions/edition8/patterns.asp

[4] Borchers, J. *A Pattern Approach to Interaction Design.* Willey Series in Software Design Patterns, Wiley, 2001.

[5] Jonathan Grudin: The Case Against User Interface Consistency, Communications of the ACM, October 1989, Volume 32

[6] Wilson, C. How Can Usability Practitioners Be More Persuasive?, Interactions, ACM Press (2007) 44 – 47

[7] Ivory, M.Y. and Hearst, M.A The State of Art in Automated Usability Evaluation of User Interfaces, ACM Computing Surveys, Volume 33, ACM Press (2001) 470 – 516

[8] Atterer, R. Model based Automatic Usability Validation NordiCHI 2008, ACM Press (2008) 1 – 14 (section 4)

Increasing Design Buy-In Among Software Developer Communities: CHI 2008, April 5–10, 2008, Florence, Italy ACM 978-1-60558-012-8/08/04.

Branding the Feel: Applying Standards to Enable a Uniform User Experience: CHI 2008, April 5 - 10, 2008, Florence, Italy ACM 978-1-60558-012-8/08/04