Personas: Moving Beyond Role-Based Requirements Engineering

Granville Miller
Microsoft Corporation
randymil@microsoft.com

Laurie Williams
North Carolina State University
williams@csc.ncsu.edu

Abstract

A primary vehicle for understanding the user in the context of the requirements for a system has been the role. For example, the role is captured through the use of actors in the use case diagram and use case descriptions. Recently, personas have been used in conjunction with scenarios in participatory design to go deeper into examining the different types of people who could play a role. A persona is an archetype of a fictional user representing a specific group of typical users. This paper expands the use of personas to scenario-based requirements engineering. Personas and scenarios are being used together for specifying requirements at Microsoft. The result of this combination has been a more comprehensive understanding of the target customers' behaviors to drive and refine our scenarios and subsequently our product development.

1. Introduction

In requirements engineering, a role is the specific behavior of an entity participating in a particular context [10]. Use case models capture roles in the 'actor' modeling construct. In use case models, an actor is an entity that interacts with the system for the purpose of completing an event [11]. Actors are used to describe how a system interacts with users and external systems within a use case description.

Roles alone may not allow analysts and consumers of a use case model to develop a deep understanding of the users of the system because the role is a rather homogeneous view of the users of the system. Within a given role, there can be many different types of users. For example, some people in the role may be "power users" while others may only have a superficial knowledge of the system. The view of a role as a homogeneous construct may not provide the system designers, developers, and testers enough information to make key decisions that could make the system more appealing to its users.

The lack of understanding of the user community has lead to the suggestion that we bring some subset of our users onsite [3]. Certainly, user involvement is an important element in the success of a project [22] and may be the best option for many projects. However, very few projects find themselves able to get dedicated users available for the duration of the project. Additionally, requirements analysts of mass-market, commercial software can have trouble finding representative users [9].

Recently, Grudin and Pruitt [9, 16] have suggested using *personas* in conjunction with scenarios as a participatory design technique and as a compromise between the actor and the on-site customer. In the context of product development, a *persona* is an *archetype of a fictional user representing a specific group of typical users*. Personas have hypothetical names, likenesses, occupations, friends, and other specific personal data. Posters with photographs of the personas and their information can be hung in places frequented by the analysts and development team [9]. The photographs are of models that have a likeness to an archetypical user.

The powerful psychological identification with and engagement between an analyst and a persona can bring about the inclusion of complex and realistic social and political aspects of the persona within a scenario. Additionally, the persona aids in communication among stakeholders. Instead of talking about a group of users in an abstract, impersonal, "middle ground" way, a persona represents a 'proxy' for the user group and can be discussed by name, such as "Would Dave use this feature?" [8]. The persona, therefore, provides a means to talk and reason about the group through the characteristics of one fictional individual, the persona.

Personas are currently being used by many of the development groups within Microsoft and have more recently been incorporated into the concrete software development processes included in Version 4.0 of the

Microsoft Solutions Framework (MSF)[1]. The most widely known of the groups using personas are the Windows and the Visual Studio development organizations. However, smaller groups such as Hotmail, MapPoint, and MSN Money also utilize this technique. In this paper, we will share experiences of the Visual Studio team.

The rest of this paper is organized as follows. In Sections 2 we will provide basic information about personas and scenarios. In Sections 3 and 4, we will discuss the use of personas for security requirements and exploratory testing. Section 5 presents information about the use of personas at Microsoft by the Visual Studio development organization in the context of the Microsoft Solution Framework. We present our conclusions in Section 6.

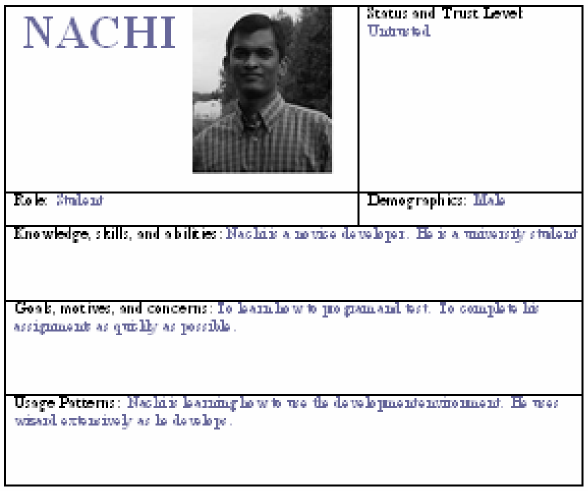
2. Personas and Scenarios

In this section, we provide background and explanatory information about personas and scenarios.

2.1. Personas

A persona contains information about a fictitious, archetypical person who holds an interest in the system. The descriptions of personas hold information about the persona's knowledge, skills, and abilities. They also hold information about their goals, motives, and concerns. Finally, the persona description will often describe the usage patterns that a persona would have of the system. A sample persona of Mort, a Visual Basic developer used in Visual Studio scenarios, is shown in Figure 1. A template with directions for documenting the information on a persona appears in the appendix.

The idea is that the personalization of a role via the persona psychologically makes a longer lasting impression on the extended development team (business analysts, project managers, architects, developers, and testers). As members of this team ask questions about the users, they can look to the persona as a method of reasoning about the solution. Ultimately, any decisions must be validated by the real user community. However, the intent is for personas to keep business analysts, developers, and testers from becoming stymied by the many design and usability decisions that are not directly specified by the requirements.



NACHI

Status and Trust Level: Untrusted

Role: Student

Demographics: Male

Knowledge, skills, and abilities: Nachi is a novice developer. He is a university student

Goals, motives, and concerns: To learn how to program and test. To complete his assignments as quickly as possible.

Usage Patterns: Nachi is learning how to use the development environment. He uses wizard extensively as he develops.

Figure 1: Nachi, a Visual Basic developer

The depth of information instilled in a persona can vary. Simple personas can reflect superficial knowledge, goals, and usage patterns. Others can reflect a deep psychological profile of the user base, and there are many levels of detail in between. The amount of information gathered to create the persona involves an implicit balance between the resources required to create the persona and the ultimate ease and accuracy of the decisions made based upon the persona details.

Finding personas is similar to finding actors. There are often several roles that will interact with a system. Once these roles have been determined, we can further refine these roles by providing personas. There are typically three to five types of people per role [12]. The personas for a product are created by writing a description for each of the targeted types of users.

Personas and actors are not the same. An actor is a homogeneous role, and any descriptions of it are couched in terms of the role rather than the people who play the role. A closer concept to a persona is an agent. An agent is someone who plays a role like a persona but does not consolidate the information of many users [5]. For example, consider a developer using an integrated development environment (IDE). A 'role' view could treat all developers in the same manner. Yet, some developers like to build systems by looking at and following examples of code while others like to write algorithms from scratch. Some developers are interested in building systems as rapidly as possible while others strive for maintainability. These differences cannot be accounted for by a single role, the developer.

[1] http://msdn.microsoft.com/vstudio/enterprise/msf/default.aspx
MSF version 3.0 is a framework for processes. Version 4.0 will continue this heritage while delivering two specialized processes for software development.

Personas were initially introduced by Cooper [8] as a usability concept. Grudin and Pruitt [9, 16] have implemented personas as a participatory design technique with Microsoft MSN Explorer and Windows development teams over a period of approximately three years. Personas were used to provide guidance on product design and development decisions as well as to prioritize requirements via a weighted priority matrix. Each persona is assigned a weight according to the proportion of the market each represents. Each requirement is assigned a value for each persona:

- -1: The persona is confused, annoyed, or in some way harmed by the requirement
- 0: The persona does not care about the requirement
- 1: The feature provides some value to the persona
- 2: The persona loves the feature or the feature does something wonderful for the Persona even if the persona does not realize it

A weighted sum for each requirement is computed. Requirements with the highest weighted sum are given the highest priority in development.

The following benefits were noted in the Microsoft experience [9, 16] of using personas with mass-marketed software:

- Personas create a focus on users and work contexts.
- Personas utilized the minds of the team members to extrapolate from partial knowledge of people to create a coherent whole, more completely than with generic "actors."
- The act of creating personas made their assumptions more explicit.
- Personas provided a medium for communication ("Patrick cannot use the search tool on your web page.")
- Personas were used by testers in their test scripts and activities.

However, Grudin and Pruitt [9, 16] experienced that getting the right set of personas and a comprehensive portrait of a persona is challenging. They also caution against over-reusing personas and of overusing the persona technique. Additionally, one study of the use of personas for participatory design [18] in the telecom industry demonstrated that technology, market, and competition issues might dominate over the issues surfaced by personas.

2.2. Scenarios

The relationship between the user and the system is characterized in a scenario as follows:

> *The scenario identifies the person as having certain motivations toward the system, describes the actions taken and some reasons why these actions were taken, and characterizes the results in terms of the user's motivations and expectations.* [4]

There is a clear need to examine the user's desire in this definition. Personas are complimentary to a scenario-based approach to requirements engineering. Specific names of the personas are used in the description of the scenario. Since there are many types of scenarios, the term scenario does not have a commonly accepted definition [20]. For our purposes, a *scenario* describes the system's behavior through a *sequence of concrete interactions with its users who are trying to achieve some goal* [1]. The sequence contains detailed interactions that illustrate one of nearly infinitely many ways of interacting with a nontrivial system. Scenarios reflect a concrete path or set of steps toward a goal.

Methods of determining the scenarios usually involve finding goals [17]. A goal is a statement of the desired problem that a user needs to be solved [1]. A scenario can be expressed as a series of actions and system responses (called a transaction in a use case model) that attempt to reach the goal. During this attempt, obstacles may prevent the goal from being reached [15]. As the persona attempts to reach his or her goal, the scenario records the specific steps that are taken. The combination of the motivations, knowledge, and goals of the persona lead to their behavior. This behavior is exactly what a scenario is intended to capture.

In contrast to use cases, scenarios provide specific details of the interactions between a system and its users. These details can provide value and challenges. Patterns which would be discovered through an abstraction process may be lost in the detailed information and thus never be surfaced [20]. Additionally, many more scenarios may be required to cover the same area as a use case. In fact, taken to the extreme, attempting to cover every facet of a product may result in scenario explosion [20]. This condition is caused by writing too many scenarios instead of only the relevant ones. On the other hand, an advantage of scenarios is that they focus on real interactions in a testable way, causing analysts to address the "devil in the details" [20]. Abstract requirements models may go unquestioned whereas a scenario approach requires an understanding of the assumptions of the model [20].

Understanding how to create a sufficient set of scenarios is one of the difficult problems in requirements engineering [20]. Sutcliffe [20] characterizes the issues associated with this coverage problem as follows:

1. Steps can be left out of scenarios due to assumptions resulting from implicit or tacit knowledge.
2. Individual views of the problems encountered may make it difficult to distill the views into a common set of problems.
3. Scenarios must cover not only the "sunny day" situations but also the situations where things go wrong.
4. Abnormal examples are often forgotten or exaggerated. Problems encountered most recently or frequently are likely to be recalled but those less frequently encountered may not.

Creating a sufficient set of scenarios requires that we create an accurate representation of the solution to the current problem.

The use case approach addresses Sutcliffe's issues three and four by using a goal-oriented approach. This approach follows three steps:

1. Find the actors (or roles)
2. Find the use cases (or goals of the actors)
3. Write the use case descriptions (or all of the paths in an abstract way that could lead to the resolution or attempted resolution of the goals).

We start by building the "sunny day" scenario first [20]. These scenarios explain the easiest way to achieve some goal of the system. Once this is in place, we can gather the alternatives and exceptions and fill in the abnormal situations. However, this goal-oriented approach does not focus on building an understanding of users or their roles

Personas offer a similar mechanism to actors in determining the goals of the system and are similarly concrete. In many ways the persona/scenario-based approach parallels the role/use case-based approach:

1. Find personas
2. Find "sunny day" scenarios of the personas
3. Write the scenarios for both the "sunny day" situations and the alternatives

However, the introduction of the persona can make it easier to understand whose goals the system is attempting to satisfy. The result is a better method of dealing with issues three and four of the coverage problem.

The suggested solution to the problem of the missing steps or the individual variation (Sutcliffe issues one and two of the coverage problem) is to create a set of common scenarios and use these as "hooks" for questions later about different individual strategies for using the system [20]. The idea is that the set of scenarios can be further refined through a series of on-going user reviews. However, each of these reviews may result in changes to existing scenarios without an understanding of origin of the changes. Moreover, some of these changes may be conflicting and thus result in confusion among those who did not interact with the users.

2.3. Individual variation between scenarios

We can examine the idea of individual variation between scenarios by creating scenarios for creating a unit test in an IDE. One scenario might read:

The developer selects the "new unit test" option and enters the unit test class name, "MoneyTest." The system responds by creating the new unit test class.

The developer enters the new unit test method, "TestCurrencyExchange" with two currency amounts and two currency types. The system creates the unit test method stub inside the unit test class. The default test failure logic "throw new Exception ("Unit test is not implemented");" is inserted.

The developer replaces the default test failure logic with the appropriate unit test logic. The system notes any compiler errors and displays the test logic..

This scenario would most likely be implemented through a wizard that walks a developer through the process of creating a unit test. Wizards are a great way to learn how to write unit tests. However, they become cumbersome when developers become experienced with writing unit tests. Advanced developers or "power users" of the IDE will write these unit tests the same way that they write normal classes. A scenario for a more experienced user could be as follows:

The developer enters the unit test class construct, several test methods and their appropriate logic. The text reads "…". The system notes any compiler errors and displays the test logic.

These two scenarios are related in that they have the same goal and that one is close to providing a subset of the other. However, without both scenarios, one of the two options may not be enabled by the system. In other words, both scenarios must be included in the sufficient set to cover the needs of a "developer." The role "developer" is overloaded in these examples. These two scenarios may appear as individual variation when, in fact, there are two different types of developers. The goal is the same for both scenarios, "to create a unit test." However there are significant differences in the implementation of the two scenarios.

Let's define two personas who wish to create a unit test. The first is Nachi who is a student, a novice developer. Nachi's counterpart is the persona Mark. Mark is an expert developer and has been working in the field for fifteen years. It is easy to associate each of these personas with one of the "Create Unit Test" scenarios. We would expect Nachi and Mark to exhibit individual variation. We can also use Nachi and Mark to drive the discovery of these alternative scenarios. The fundamental premise of this model is based on the understanding that the combination of knowledge, interactions, and goals leads to behavior.

The difference between Nachi and Mark is their knowledge. They may have similar goals. They may even have similar interactions with the system. However, their resulting behavior may be dramatically different. While the system responds in a deterministic way, we must account in our scenarios for this different behavior.

Figure 2 displays an annotated use case diagram. The actor is the `Developer` and the `Developer` has a `Create Unit Test` use case. This use case can be made more specific through the creation of scenarios. The figure shows two scenarios, an `Expert Scenario` and a `Novice Scenario`, as discussed earlier in this section. By replacing "the developer" with "Nachi" and "Mark," a greater understanding of the needs of the user can emerge.

The knowledge level of a generic "developer" actor is unclear. We could certainly fix the knowledge level of the developer actor, but the resulting scenarios would be incomplete. Since the knowledge level of an actor may not be the same as one of the many personas that plays the role, the scenarios generated may be very different. Since there is one actor but multiple personas, the actor model may fail to achieve coverage or a sufficient set of scenarios to be representative of the current problem.

Knowledge is just one aspect that is captured in a persona. Depending on the depth reached in a persona, there may be deeper understandings of the motivations and behavior beyond the simple interactions to consider. This depth is achieved through the constant refinement of the personas over a longer period of time.

Once we have created a persona, we can utilize lifestyle snapshots or "a day in the life of" a persona to uncover additional scenarios [2]. Lifestyle snapshots are another means of goal elicitation. Often, if we look at the system from different perspectives, we can understand it more thoroughly. Each lifestyle snapshot will determine a number of candidate scenarios. They provide further context for the goals of the system.

3. Security Requirements

Scenarios are good at describing and analyzing functional behavior of the system. However, there are better ways to describe the nonfunctional or quality of service (QoS) requirements[2]. One form of these QoS requirements center on security. Other QoS requirements focus as constraints on scenarios such as performance, load, and stress requirements.

Application security has been a recent focus. Seventy-five percent of system security breaches have occurred at the application level [14]. Threat modeling is a systematic method for determining the vulnerabilities of a system [21]. Two keys to understanding these vulnerabilities are the system's assets and the adversaries that might want to modify, steal, access, or manipulate them.

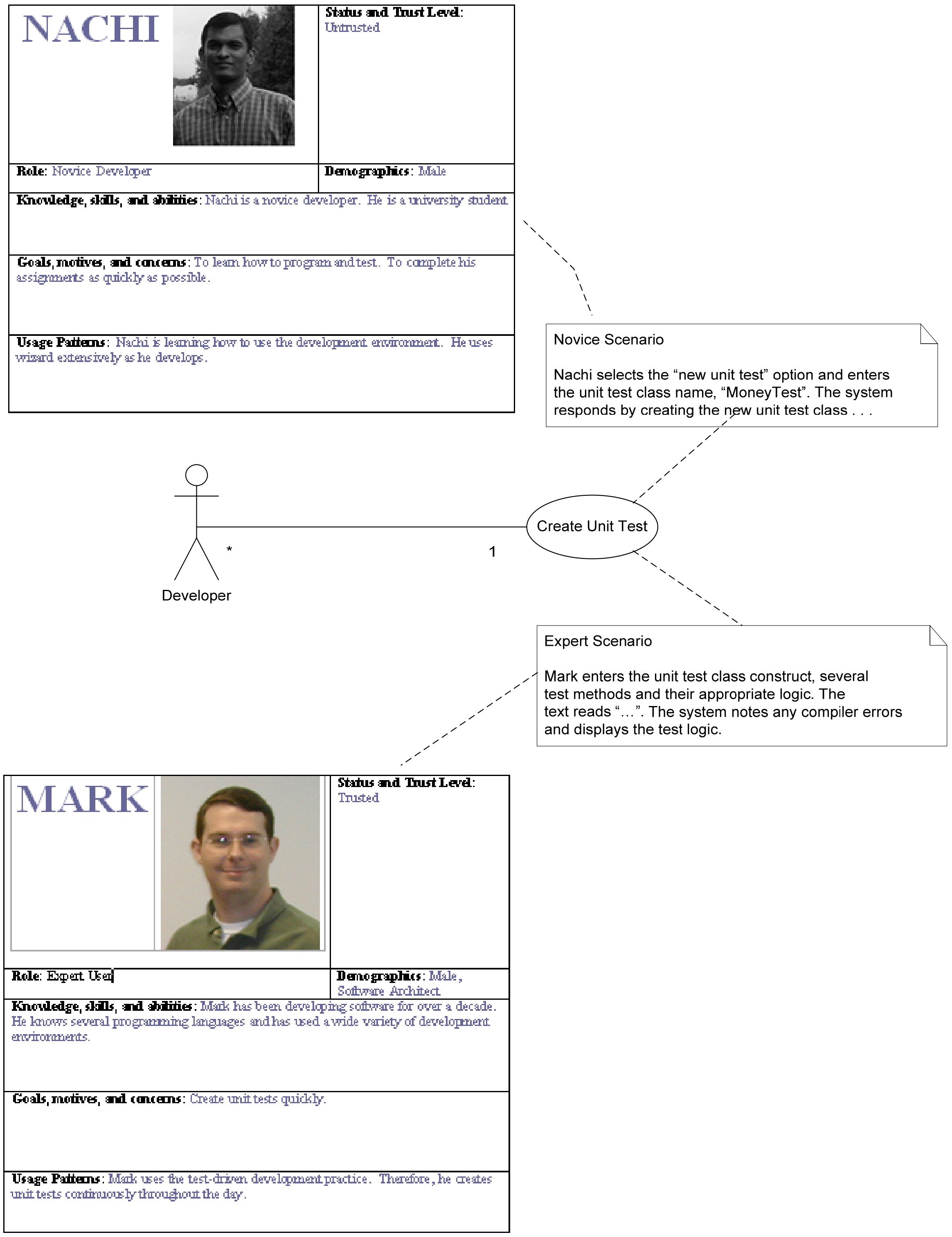
Threat modeling looks at the system from the outside-in because this is the approach that an adversary would take [21]. Key elements to understanding this view are the entry points, assets, and trust levels. An entry point is a location where data or control transfers between systems. Assets are the tangible or intangible resources that the adversary might wish to steal. Trust levels are the privileges or credentials assigned to protect the assets.

Threat modeling moves through the architecture of a system to determine threats and vulnerabilities. A vulnerability is a condition in a system, or in the procedures affecting the operation of the system, that makes it possible to perform an operation that violates the explicit or implicit security (or survivability) policy of the system [6]. From a threat model, we obtain a list of these vulnerabilities. These vulnerabilities become security requirements. Understanding the importance of an asset to an adversary is the key to determining an acceptable level of security.

To determine the importance of an asset, it is important to understand the adversary. Personas can be used to build this understanding. The result is two forms of personas, the *favored persona* and the *disfavored persona.* A favored persona is a user of the system that utilizes the system in the intended manner. The disfavored persona, or adversary, attempts to bypass security measures and obtain a system's assets.

Trust levels can be applied to all personas, as shown in the persona template in the appendix. These trust levels represent a set of rights given to a persona based upon the system's knowledge of that persona [21]. A trust level is a group in which the persona belongs, such as a "normal user" or "administrator." An adversary may attempt to elevate his trust level through bypassing a system security mechanism.

[2] The term "quality of service requirement" is used at Microsoft because "nonfunctional requirements" sound like the requirements are broken.



NACHI

	Status and Trust Level: Untrusted
Role: Novice Developer	**Demographics**: Male

Knowledge, skills, and abilities: Nachi is a novice developer. He is a university student.

Goals, motives, and concerns: To learn how to program and test. To complete his assignments as quickly as possible.

Usage Patterns: Nachi is learning how to use the development environment. He uses wizard extensively as he develops.

Novice Scenario

Nachi selects the "new unit test" option and enters the unit test class name, "MoneyTest". The system responds by creating the new unit test class . . .

Expert Scenario

Mark enters the unit test class construct, several test methods and their appropriate logic. The text reads "…". The system notes any compiler errors and displays the test logic.

MARK

	Status and Trust Level: Trusted
Role: Expert User	**Demographics**: Male, Software Architect

Knowledge, skills, and abilities: Mark has been developing software for over a decade. He knows several programming languages and has used a wide variety of development environments.

Goals, motives, and concerns: Create unit tests quickly.

Usage Patterns: Mark uses the test-driven development practice. Therefore, he creates unit tests continuously throughout the day.

Figure 2: Annotated Use Case Diagram

Trust levels can be used in scenarios to describe how a persona should be treated when he or she attempts this elevation of privilege. QoS requirements can fill in details about how the system should protect its assets against the misuse by adversaries. These scenarios and QoS requirements should also look "innocent" misuse by favored personas as well.

The use of roles in security analysis began almost ten years ago with the advent of role-based access control (RBAC) [19]. In RBAC, a role is defined as a job function within the organization that describes the authority and responsibility conferred on a user assigned to the role. The use of personas can augment the RBAC-type role by further delineating favored and disfavored people in various roles.

4. Exploratory Testing

Requirements drive most forms of testing. However, there is a form of testing called exploratory testing which is designed to find new requirements as well as bugs. *Exploratory testing* is a *systematic method of testing that applies heuristics instead of being driven by requirements* [13]. The idea of exploratory testing is to "think outside of the box" and to look for problems outside of the normal validation testing used to verify a scenario's functionality.

The goal of exploratory testing is to create test cases and execute them at the same time. The understanding gained by running a test leads to new tests. Thus, exploratory testing seeks to maximize creativity in the determination of a system's problems through testing. Exploratory testing sessions are usually bounded by time. They also tend to concentrate on a single area or facet of the system.

One of the heuristics for exploratory testing is to adopt a mindset of a persona and look for areas of the product that would cause problems for that persona. The goal is to find missing requirements, poor existing requirements, or bugs that fall outside of the normal requirements. The tester must consider the knowledge, goals, and usage patterns of the persona when working through the goals of the system.

5. Use of Personas at Microsoft

As discussed in Section 2, development groups at Microsoft have been utilizing personas and scenarios since the mid 1990s. These personas appear on posters in the halls of the buildings in which the developers reside. Each development group shares its persona experiences with the other groups. Scenarios which reference personas are found in the requirements documents for each group. The scenarios are implemented and become part of the functionality of the a specific product/release. However, personas live and are refined from release to release.

5.1. Using personas and scenarios to build development tools

The Developer Division is a several hundred person line of business which builds developer tools at Microsoft. In the Developer Division, we have developed three personas that represent the developer role. These personas were developed through on-going interactions with our user community. Each persona was carefully refined over a six-month period. Like the other groups in Microsoft, these personas appear on posters that hang on the walls of our building. These three personas appear in our scenarios and aid us in seeing the scenario unfold through the user's eyes. Since the personas respectfully represent these users, their names are used in the scenario document rather than a role name.

We find that scenarios are more effective at getting to the system details. Additionally, personas have helped us find the missing steps in a scenario during the gathering of requirements. When missing steps are discovered during development, personas help the program manager and developers find the needed functionality. There are often discussions about what a persona would do in a given situation. Decisions can be made in real time without extensively rewriting the scenarios because there is a shared understanding between all of the development team members of the people who will be using the system.

While this mechanism helps us make decisions more rapidly, we recognize that user interaction cannot be replaced by these personas. We are constantly verifying our new products through external design reviews, usability labs, and beta releases. The feedback from these sessions is reflected in both the personas and the scenarios.

Microsoft Visual Studio, our flagship product, has a very large user community. This product is an IDE that supports three different programming languages and a variety of development styles. A single developer role could not do justice to this large a community. With the addition of Visual Studio Team System[3], we have expanded beyond the developer into the extended development roles of project management, business

[3] Microsoft Visual Studio Team System is an extension to the Visual Studio integrated development environment that covers the entire software development lifecycle from requirements through test and deployment.

analyst, architect, and tester communities. We have created new personas to fill these extended development roles. The personas for these roles are not nearly as detailed as the three developer roles, partly because they are much newer. One of the ways that we have dealt with the lack of detail is to create fewer personas. In other words, until we have enough information to create a distinct, memorable persona, we use the persona like a role. Once additional information is obtained, we add personas to represent the individual variation.

Another consideration of how many personas to create is how different are each of the personas. If the resulting scenarios are very similar, there is no need to spend the time to create personas for each type of person. Another consideration is if a user transitions from one persona to another. For example, does a “Nachi” become a “Mark” with experience? How quickly does this occur?

5.2. Microsoft Solutions Framework

As a result of our experience, we are deeply embedding the use of personas in our new, externally-available software development processes. Microsoft is creating two of these processes geared toward two different communities. MSF Agile is directed toward the agile software developers while MSF Formal is geared for those wanting the rigor of the Software Engineering Institute’s Capability Maturity Model Integration [4] (CMMI) [7].

Both MSF Agile and MSF Formal are scenario-driven development processes for building .NET, Web, Web Service, and other object-oriented applications. Both also directly incorporate practices for handling QoS requirements, such as performance and security. Finally, they utilize a context-driven approach to determine how to operate the project. MSF Agile and MSF Formal will be made available as part of Version 4.0 of MSF.

Personas and scenarios play a central role in driving the MSF software development processes. All of the downstream development and testing activities are driven by these central work products. Even the architectural and other requirements elements are deeply connected to the scenarios. For example, QoS requirements constrain the functionality of scenarios in many areas including security, performance, stress, load, and platform. Architecture uses the scenarios as a basis when creating the form for the functionality of the system. Tests are created based on the functionality described in the scenarios.

In MSF Agile, scenarios are constrained to be small enough to drive short development iterations. Since an iteration is typically between two and six weeks, each scenario must be small enough to be completed by a small team in that period of time. Typically, several scenarios are finished in an iteration. This is usually not possible with larger vehicles such as a use case. MSF Agile utilizes a scenario list to capture the backlog of scenarios on the project. New scenarios can be added to the backlog as they are discovered. Just enough of the scenario is written to allow a rough order of magnitude estimate to be created.

Scenarios are also robust enough that they can be combined with traditional requirements in MSF Formal. A formal approach requires an estimate of the amount of time necessary to complete the project with some level of accuracy. Scenarios provide a decent level of information to provide veterans the ability to determine these estimates. Such a capability is often required for fixed-priced contracts.

6. Conclusion

Personas and scenarios can be combined the way that use cases and actors have been. Through the use of personas/scenarios at Microsoft for several years, we have found that personas have led to a more comprehensive understanding of the target customers' behaviors to drive and refine our scenarios and subsequently our product development. In isolation, personas are strictly a usability concept. But usability does not drive the architecture, development, and testing elements of a software development process. Instead, requirements drive these elements. The addition of personas unifies concepts from threat modeling for architecture and drives certain types of exploratory testing the way that scenarios drive validation tests.

Personas and scenarios have been used together at Microsoft for several years. However, this technique has not been described outside of Microsoft until recently. The release of two upcoming processes, MSF Agile and MSF Formal will be part of new additions to the Microsoft Solutions Framework in Version 4.0. These processes capture best practices from Microsoft for dealing with QoS requirements, security, and innovations in agile and formal software development.

[4] CMMI and Capability Maturity Model are registered trademarks of the Software Engineering Institute.

References

[1] I. F. Alexander and Neil Maiden, J.P. Wiley., "Scenarios, Stories, Use Cases Through the System Development Life-Cycle." New York: John Wiley & Sons, 2004.

[2] D. Anderson, "Lifestyle Snapshots: Solving the Context Problem for Wireless Design," http://www.uidesign.net/2000/papers/lifestylesnapshot.html, 2000.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, Mass.: Addison-Wesley, 2000.

[4] J. M. Carroll, "Introduction: The Scenario Perspective on System Development," in *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. M. Carroll, Ed. New York: John Wiley and Sons, 1995, pp. 1-17.

[5] J. M. Carroll, *Making Use: Scenario-Based Design of Human-Computer Interactions*. Cambridge, MA: MIT Press, 2000.

[6] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling Multistep Cyber Attacks for Scenario Recognition," DARPA Information Survivability Conference and Exposition (DISCEX'03), Washington, DC, 2003, pp. 284-292.

[7] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*. Boston, MA: Addison Wesley, 2003.

[8] A. Cooper, *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity*. ***[need city]***: Sams, 1999.

[9] J. Grudin and J. Pruitt, "Personas, Participatory Design, and Product Development: An Infrastructure for Engagement," The Participatory Design Conference, Malmö, Sweden, 2002, pp. 144-161.

[10] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Reading, Massachusetts: Addison-Wesley, 1999.

[11] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, England: Addison-Wesley, 1992.

[12] S. Johnson, "Buyer and User Personas," *ProductMarketing.com*, vol. 1, no. 4, pp. 8-9, Nov/Dec 2003.

[13] C. Kaner, J. Bach, Bret Pettichord, and J. P. Wiley., *Lessons Learned in Software Testing*. New York: John Wiley & Sons, 2001.

[14] T. Lanowitz, "Security at the Application Level: Are You Ready?" Gartner Application Development Summit: The Path to Modern AD, 2004, pp.

[15] C. Potts, "ScenIC: A Strategy for Inquiry-Driven Requirements Determination," 4th IEEE International Symposium on Requirements Engineering, Limerick, Ireland, 1999, pp. 58-65.

[16] J. Pruitt and J. Grudin, "Personas: Practice and Theory," Conference on Designing for User Experiences, San Franciso, CA, 2003, pp. 1-15.

[17] C. Rolland, G. Grosz, and R. Kla, "Experience With Goal-Scenario Coupling in Requirements Engineering," 4th IEEE International Symposium on Requirements Engineering, Limerick, Ireland, 1999, pp. 74-81.

[18] K. Rönkkö, M. Hellman, B. Kilander, and Y. Dittrick, "Personas is not Applicable: Local Remedies Interpreted in a Wider Context," Participatory Design Conference, Toronto, Canada, 2004, pp. 112-120.

[19] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38-47, 1996.

[20] A. Sutcliffe, "Scenario-based Requirements Engineering," 11th IEEE International Symposium on Requirements Engineering, Monterey, California, 2003, pp. 320-329.

[21] F. Swiderski and W. Snyder, *Threat Modeling*. Redmond, WA: Microsoft Press, 2004.

[22] The Standish Group International, "Extreme Chaos," http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf, 2001.

Appendix: The MSF Agile Persona Template

Name: Enter a respectful, fictitious name for the persona.	Status and Trust Level: Favored or disfavored and level of credentials
Role: Place the user group in which the persona belongs.	Demographics: Age and personal details optional
Knowledge, skills, and abilities: Group real but generalized information about the capabilities of the persona.	
Goals, motives, and concerns: Describe the real needs of the users in the user group represented by the persona. If multiple groupings exist, write a persona for each grouping.	
Usage Patterns: Write the frequency and usage patterns of the system by the persona. Develop a detailed understanding of what functions would be most used. Look for any challenges that the system must help the persona overcome. Note the learning and interaction style if the system is new. Does the persona explore the system to find new functionality or need guidance? Keep this area brief but accurate.	