Agile Dev
Better Software
DevOps WEST
A TECHWELL EVENT

BT7

Agile Techniques
Thursday, June 7th, 2018, 1:30 PM

Mobbing for Test Design: Connecting with Your Colleagues' Test Ideas

Presented by:

James Fogarty & Jeff MacBane
TechSmith

Brought to you by:

TECHWELL

350 Corporate Way, Suite 400, Orange Park, FL 32073
888-268-8770 ⋅ 904-278-0524 - info@techwell.com - https://www.techwell.com/

James Fogarty

TechSmith

James Fogarty is a passionate software testing practitioner. He works at TechSmith, the makers of Snagit, Camtasia, and other visual communication software applications. He works with software testers, developers, and business professionals to level up the software quality by designing tools, teaching automation, and involving the entire development team in testing and improving software quality. He's been testing digital cameras and imaging and commercial software for sixteen years. James cofounded the Lansing Area Software Testers Meetup, a group dedicated to improving software quality and the software community in the greater Lansing area.

Jeff MacBane

TechSmith

Jeff MacBane is a software tester at TechSmith, the makers of Snagit, Camtasia, and other visual communication software applications. He has thirteen years of software testing experience in insurance, legal, medical, and commercial software organizations. Jeff has a passion for testing software and continuing to learn about agile and Scrum practices. He has presented at Mid-Michigan Agile Group and Lansing Area Software Testers Meetups.

Mobbing for Test Design: Connecting with Your Colleagues' Test Ideas

With Jeff MacBane and James Fogarty

DevOPS West 2018

@jmacbane @zithica007


TechSmith®
TechSmith Snagit™
TechSmith Camtasia™
TechSmith Relay®
@jmacbane @zithica007

Our Agenda

- **Presentation Takeaways**
- **How we discovered Mob Test Design and Mob Testing, a 5 step process**
    - Step 1 - Testing By Yourself
    - Step 2 - Pair Testing
    - Step 3 - Pair Testing using SFDIPOT
    - Step 4 - Mob Test Design with SFDIPOT
    - Step 5 - Mob Testing using SFDIPOT
- **Go through a couple of exercises**
    - Exercise # 1: - Mob Test Design using SFDIPOT
    - Exercise # 2: - Mob Testing using SFDIPOT
- **Has quality improved?**
- **Recap**
- **Questions**

Your Takeaways

- **An understanding of Mob Test Design, and Mob Testing**
- **A resource packet**
  - Has cheat sheets for Pair Testing, SFDIPOT, Mob Test Design, and Mob Testing
- **You will be able to run your own Mob Test Design and Mob Testing sessions using SFDIPOT**
- **What are your expectations from today's session?**

How We Discovered Mob Test Design and Mob Testing?

@jmacbane @zithica007

Step 1 – Testing By Yourself

What is Testing By Yourself?

- You're the only software tester on a team

What can happen if you only Test By Yourself?

- Little to no interactions between teams
- Knowledge silos are formed

When can you Test By Yourself?

- You can test with two or more people
- You want the whole team involved with testing

Step 2 – Pair Testing

What is pair testing?

- When two people test software side by side. One person controls the mouse and keyboard, the other person takes notes, discussing the test cases and asking questions.

Why should you use Pair Testing?

- Pair testing's a great way to quickly find problems in software, share knowledge about testing and the software under test, get to know team members, and most importantly, it's fun!

When should you Pair Test?

- Anytime you have the opportunity you should Pair Test. You can

Step 3 – Pair Testing Using SFDIPOT

What is SFDIPOT?

- From the Heuristic Test Strategy Model (HTSM)
- A mnemonic to help you generate test cases

Why should you use SFDIPOT with Pair Testing?

- A way to structure your test cases / ideas
- Helps generate test cases / ideas when you're stuck

How is SFDIPOT broken out?

- Let's find out…

SFDIPOT - Mnemonic To Generate Test Cases

- **Structure -** Everything that comprises the physical product
- **Function -** Everything that the product does
- **Data -** Everything that the product processes
- **Interface -** Every conduit by which the product is accessed or expressed
- **Platform -** Everything on which the product depends (and that is outside your project)
- **Operations -** How the product will be used
- **Time -** Any relationship between the product and time

SFDIPOT – Using A Car Stereo

- **Structure –** The volume button
- **Function –** Facilitates playing music from radio stations
- **Data –** Process digital music and outputs to your speakers
- **Interface –** The clock
- **Platform –** Vehicle manufacture harness to house it
- **Operations –** Consumer will use it to listen to music
- **Time –** How long is it on after the car shuts off

Step 4 – Mob Test Design Using SFDIPOT

What's Mob Test Design?

- A way to generate test case ideas before or shortly after development begins

Why should you use Mob Test Design?

- Generate test case ideas and workflows
- It can generate questions to bring back to stakeholders

What are our experiences with Mob Test Design using SFDIPOT?

- Let's take a look…

Our Experiences With Mob Test Design Using SFDIPOT

- Anyone can participate in a Mob Test Design session
- Test case ideas are more nebulous and workflow orientated
- Questions come up due to different perspectives and can change the design

How do you setup a Mob Test Design session?

- In 3 easy steps…

Setting Up A Mob Test Design Session

Running A Mob Test Design Session

After A Mob Test Design Session

Questions With Mob Test Design Sessions?

Exercise # 1 – Mob Test Design Using SFDIPOT

Goal

- Form into groups and use Mob Test Design with SFDIPOT to write down test case ideas and questions. We'll be using a new scenario 'Car Stereo V 1.0'

Steps

- Form into small groups of 3 – 4 people
  - Choose a person to write down test case ideas and questions
  - Choose another person to facilitate
  - Everyone on the team suggests test case ideas
- Read the 'Car Stereo V 1.0' scenario

Share Your Test Case

Ideas and Questions

Step 5 – Mob Testing Using SFDIPOT

What is Mob Testing

- Several people get together to test out software for a period of time

Some of our experiences

- Don't have more than 5 people in a group
- Developers and QA can participate together

Why should you use Mob Testing?

- You want a larger variety of testing perspectives to exercise the software being tested

Exercise # 2 – Mob Testing Using SFDIPOT

Goal

- Each group will use Mob Test Design with SFDIPOT to write down test cases and questions. We'll be using a new scenario 'Car Stereo V 2.0'

Steps

- Choose a different person to write down test cases, questions and facilitate
- Everyone on the team suggests test cases
- Read the 'Car Stereo V 2.0' scenario


Share Your Test Cases
and Questions

Differences Between Mob Test Design and Mob Testing

	Mob Test Design	Mob Testing
When should you use it?	Before or shortly after development	Software is ready to test / end of development
Who should use it?	Anyone in the company	QA / members of your team
Why should you use it?	Generate more nebulous test case ideas / workflows	Generates more specific test cases / small workflows
Two Benefits foe each way	Injects quality at the beginning of development Generates test case ideas and brings up workflow / design questions	More perspectives testing the software Generates test cases for QA or the team to use

Has Quality Improved?

Here are some of the ways quality has changed for us:

- Everyone is accountable for testing
- Pair Test with developers as they're developing software
- Mob Test Design sessions helps root out missing workflows
- Have Mob Testing Sessions with stakeholders
- QA isn't a bottle neck, we don't need to test everything

Presentation Recap

- **How we discovered Mob Test Design and Mob Testing in 5 steps**
  - Testing By Yourself
  - Pair Testing
  - Pair Testing using SFDIPOT
  - Mob Test Design using SFDIPOT
  - Mob Testing using SFDIPOT

- **Practical example for Mob Test Design and Mob Testing using SFDIPOT**

Any Questions?

@jmacbane @zithica007

Resources

Contact Jeff:

Twitter: @jmacbane

LinkedIn: jmacbane

Contact James:

Twitter: @zithica007

LinkedIn: j-fogarty

Additional Resources

Heuristic Test Strategy Model SFDIPOT PDF	https://bit.ly/2ccCGOJ
PowerPoint Slides and Workshop Packet	http://bit.ly/2018devopswest

Mobbing for Test Design: Connecting with Your Colleagues' Test Ideas

With Jeff MacBane and James Fogarty

DevOPS West 2018

Resources

Contact Jeff: **Twitter:** @jmacbane **LinkedIn:** jmacbane	**Contact James:** **Twitter:** @zithica007 **LinkedIn:** j-fogarty
Heuristic Test Strategy Model SFDIPOT PDF	https://bit.ly/2ccCGOJ
SFDIPOT Example Mind Map	https://bit.ly/2fXXoE1
PowerPoint Slides and Workshop Packet	http://bit.ly/2018devopswest

---

Workshop Packet Table of Contents

Pair Testing Cheat Sheet

Objective: Explain pair testing, why you'd want to use it and how to run a paired testing session.

What is Pair Testing?

- Testing software by two team members. One team member is in control of the mouse and keyboard, the other one is taking notes, documenting test cases and asking questions.

Why would I use Pair Testing?

- Pair testing's a great way to quickly find problems in software, share knowledge about testing and the software under test, get to know team members, and most importantly, it's fun!

Benefits:

- Quickly find problems in the software
- Share Knowledge about testing and the software under test
- Get to know team members (your team or another person's team)
- Expose other team members to testing

Running a Paired Testing Session:

1. Choose a tester or developer who you want to partner with in a pair testing session

2. Pick a suitable feature or set of features to test

- **Tips**: Don't try to test a whole application in one session – it's too much.

3. Planning your pairing session

- Schedule a 1 – 2 hour time to pair test (1.5 hours is optimal)
- Choose an environment that is suitable for two people to test together at one machine.
- Be sure you can work without interruptions and are free to talk to each other.
    - **Tip:** Pick a location away from distraction such as a conference room or office.

4. Running your pair testing session

- One team member (the Driver) is in control of the keyboard and mouse.
- The second team member thinks out-loud, asks questions, and makes notes / test cases.

5. Evaluate outcomes

- What worked well?
- What would you improve from the session?

More Information:

Pair Testing: How I Brought Developers into the Test Lab	https://bit.ly/2d0cmrX
Katrina The Tester – Pair Testing	https://bit.ly/2cElAEW
Pair Testing	https://bit.ly/2d5yPTs

SFDIPOT - Mnemonic to Generate Test Case Ideas

- **Structure -** Everything that comprises the physical product
- **Function -** Everything that the product does
- **Data -** Everything that the product processes
- **Interface -** Every conduit by which the product is accessed or expressed
- **Platform -** Everything on which the product depends (and that is outside your project)
- **Operations -** How the product will be used
- **Time -** Any relationship between the product and time

Example of SFDIPOT – Using Software

- **Structure –** Software CD, Box, etc.
- **Function –** Installing and Uninstalling the software
- **Data –** Displaying data, reading / writing to a Database
- **Interface –** Command line, GUI, Icon, Browser
- **Platform -** .NET, Java, DLL's
- **Operations –** User can add two numbers and the result is displayed
- **Time –** Leaving the software open for days doesn't cause memory leaks

Example of SFDIPOT – Using A Car Stereo

- **Structure –** The volume button
- **Function –** Plays music from radio stations
- **Data –** Process digital music and output to your speakers
- **Interface –** The clock
- **Platform –** Vehicle manufacture harness to house it
- **Operations –** Consumer will use it to listen to music
- **Time –** How long is it on after the car shuts off

Mob Test Design Cheat Sheet

Objective: Explain Mob Test Design, how to setup your own and what to do after one.

What is Mob Test Design?
Mob Test Design is an activity in which you take the requirements for your feature / product breaking them down into test ideas using the SFDIPOT mnemonic from the HTSM. You and partner(s) brainstorm test case ideas. Below is an explanation of SFDIPOT.

SFDIPOT mnemonic used in Mob Test Design sessions to generate test case ideas:

- **Structure -** *Everything that comprises the physical product*
- **Function -** *Everything that the product does*
- **Data -** *Everything that the product processes.*
- **Interface -** *Every conduit by which the product is accessed or expressed*
- **Platform -** *Everything on which the product depends (and that is outside your project)*
- **Operations -** *How the product will be used*
- **Time -** *Any relationship between the product and time*

Why would I use Mob Test Design?
The purpose of a Mob Test Design session is to generate and categorize test case ideas. It can generate questions to bring back to stakeholders that need clarification and possibly design changes to the product / feature. Doing test design as an individual is unique from a single perspective. Including other people such as testers, developers, and stakeholders, opens the test design to broader perspectives and ideas can be generated from other people's ideas.

Who can use Mob Test Design?

- Anyone can use Mob Test Design such as:

Software Testers	**Developers**	**Management**	**Stakeholders**
Internal Users	**External Users**	**Marketing**	**User Experience**

When should I have a Mob Test Design?

- You can have a one at any point in the development
- The earlier in development you use it the more nebulous your test case ideas will be
  - For Example:
    - Prior to or shortly after development begins
      - Test case ideas are more nebulous, expect a lot of variety in test ideas
    - When a feature is being developed
      - Test case ideas follow the workflow and tend to be more specific
    - After development is done
      - Test case ideas are focused on specific aspects of the software and small workflows
      - Good when lots of bugs have been reported

- **PRO Tips when brainstorming test case ideas with SFDIPOT**
  - During a session, one person facilitates /answers questions and one person records notes
    - The facilitator can help keep the test case ideas on track
  - The SFDIPOT is a guide to help generate test case ideas
  - Have a timer to help keep the session on point and time boxed
  - There is no bad test case idea this is a brainstorming session
  - It's ok if you don't have an answer to a question during the session
  - Starting with 'Function' from the SFDIPOT mnemonic makes it easier to start generating test case ideas
  - The earlier you can have one the better
    - It's easier to generate test case ideas

- Having more than one Mob Test Design Session, with different participants, will generate different ideas
- It's ok to end the session early if you've exhausted your test ideas

Process: Conducting a Mob Test Design Session

STEP 1: Preparing for a Mob Test Design Session

- Have a firm understanding of the product / feature being developed
  - Meet with the stakeholders to discuss how the feature / product functions
  - Familiarize yourself with the user stories, mockups, acceptance criteria, etc.
- Find a partner to facilitate or write notes during the session
- Find people to participate in the session
  - Groups of 3 -4 work best
- Schedule a meeting with your partner and participants
  - Usually 1 – 2 hours works
  - The more complex the product / feature the longer a session should run
- Create a short summary of the product / feature to discuss with your partner
- Collect the user stories / acceptance criteria for the product / feature

STEP 2: During a Mob Test Design Session

- Thank your participants for their time
- Bring a copy of the user stories / acceptance criteria to the session
- Give a quick overview of the product / feature with your partner
  - Make sure enough context is given to the participants
- Determine who's going to facilitate and take notes between you and your partner
- Be sure to use a separate area to record test case ideas and questions
- Start a timer (leaving 10 minutes left in the session)
- Begin brainstorming and recording the participant test case ideas / questions
- Take 10 minutes at the end to wrap up
  - Finish writing any test case ideas / questions that came up
  - Thank your participants and let them leave the session
  - Have a short retrospective, with your partner
    - How the session went and were there any surprises
  - Save your notes so you can access them later

STEP 3: After the Mob Test Design Session:

- Share the questions with your stakeholders
  - Get answers to your questions from the stakeholders
  - If needed, ask the stakeholders to inform the team of any changes in the product / feature based on the answers to your questions
- Share you test case ideas with the stakeholders and your team
  - Using your test case ideas / questions to come up with themes (if possible)
  - Your test case ideas and or questions may trigger more changes in the product / feature
  - If changes need to be made, update the user stories, scenarios, and acceptance criteria
  - Record your test case ideas, themes, in your test plan(s)

More Information:

Heuristic Test Strategy Model SFDIPOT PDF	https://bit.ly/2ccCGOJ
SFDIPOT Example Mind Map	https://bit.ly/2fXXoE1

Mob Testing Cheat Sheet

Objective: Explain Mob Testing and how-to setup and run your own Mob Testing Session.

What is Mob Testing?
Mob Testing is an activity where you get a group of software testers / developers from your team together to come up with test cases and test the software. It occurs when the software is ready to test. You can use SFDIPOT to help drive the session or if you become stuck generating test cases.

SFDIPOT mnemonic used in Mob Testing sessions to generate test case ideas:

- **Structure -** *Everything that comprises the physical product*
- **Function -** *Everything that the product does*
- **Data -** *Everything that the product processes.*
- **Interface -** *Every conduit by which the product is accessed or expressed*
- **Platform -** *Everything on which the product depends (and that is outside your project)*
- **Operations -** *How the product will be used*
- **Time -** *Any relationship between the product and time*

Why should you conduct Mob Testing Session?
The purpose of Mob Testing sessions is to generate, document, and execute test cases for your software that's ready to be tested. Having other people allows you to have the more perspectives for testing. This can generate questions to bring back to stakeholders for clarification and possible design changes to the software.

Who can participate in Mob Testing Session?

- Software testers and developers in the company

When do you setup a Mob Testing Session?

- When the software you want to test is ready for testing

Process: Conducting a Mob Testing Session

STEP 1: Preparing for a Mob Testing Session

- Have a firm understanding of the software being developed
  - Familiarize yourself with the user stories, mockups, acceptance criteria, etc.
- Find a partner to facilitate and write down test cases / notes during the session
- Who will perform the testing?
  - You can have others suggest what to do and you perform the actions
  - Each person can take turns testing for short periods of time
- Find people to participate in the session
  - Groups of 3 -4 work best
- Schedule a meeting with your partner and participants
  - Usually 1 – 2 hours works
  - The more complex the product / feature the longer a session should run

STEP 2: During a Mob Testing Session

- Thank your participants for their time
- Give a quick overview of the software being tested making sure to give enough context
- Time your session and start it (leaving 10 minutes left in the session to wrap up any final notes)

STEP 3: After a Mob Testing Session

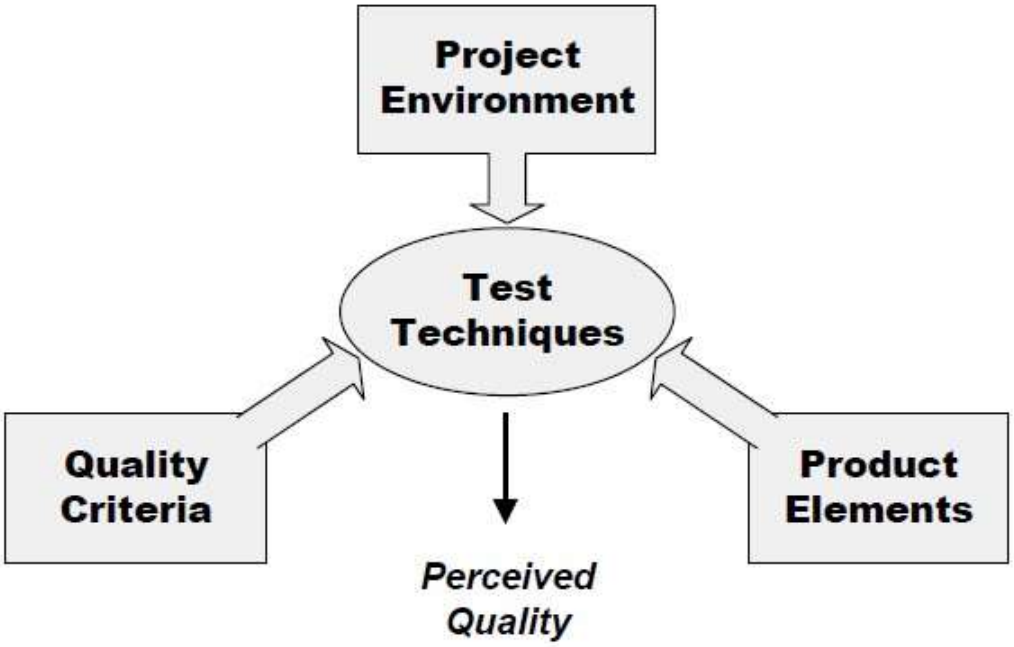
- You and your partner have a short debrief to go over your questions / notes
  - Group your test cases into sections (use SFDIPOT if needed)
  - Share any questions from your session with your stakeholders getting answers to them
- Start planning another Mob Testing session, as needed

More Information:

Heuristic Test Strategy Model SFDIPOT PDF	https://bit.ly/2ccCGOJ

Copyright 1996-2015, Satisfice, Inc.

Heuristic Test Strategy Model



The **Heuristic Test Strategy Model** is a set of patterns for designing a test strategy. The immediate purpose of this model is to remind testers of what to think about when they are creating tests. Ultimately, it is intended to be customized and used to facilitate dialog and direct self-learning among professional testers.

Project Environment includes resources, constraints, and other elements in the project that may enable or hobble our testing. Sometimes a tester must challenge constraints, and sometimes accept them.

Product Elements are things that you intend to test. Software is complex and invisible. Take care to cover all of it that matters, not just the parts that are easy to see.

Quality Criteria are the rules, values, and sources that allow you as a tester to determine if the product has problems. Quality criteria are multidimensional and often hidden or self-contradictory.

Test Techniques are heuristics for creating tests. All techniques involve some sort of analysis of project environment, product elements, and quality criteria.

Perceived Quality is the result of testing. You can never know the "actual" quality of a software product, but through the application of a variety of tests, you can make an informed assessment of it.

General Test Techniques

A test technique is a heuristic for creating tests. There are many interesting techniques. The list includes nine general techniques. By "general technique" we mean that the technique is simple and universal enough to apply to a wide variety of contexts. Many specific techniques are based on one or more of these nine. And an endless variety of specific test techniques may be constructed by combining one or more general techniques with coverage ideas from the other lists in this model.

Function Testing
Test what it can do
1. Identify things that the product can do (functions and sub functions).
2. Determine how you'd know if a function was capable of working.
3. Test each function, one at a time.
4. See that each function does what it's supposed to do and not what it isn't supposed to do.

Domain Testing
Divide and conquer the data
1. Look for any data processed by the product. Look at outputs as well as inputs.
2. Decide which particular data to test with. Consider things like boundary values, typical values, convenient values, invalid values, or best representatives.
3. Consider combinations of data worth testing together.

Stress Testing
Overwhelm the product
1. Look for sub-systems and functions that are vulnerable to being overloaded or "broken" in the presence of challenging data or constrained resources.
2. Identify data and resources related to those sub-systems and functions.
3. Select or generate challenging data, or resource constraint conditions to test with: e.g., large or complex data structures, high loads, long test runs, many test cases, low memory conditions.

Flow Testing
Do one thing after another
1. Perform multiple activities connected end-to-end; for instance, conduct tours through a state model.
2. Don't reset the system between actions.
3. Vary timing and sequencing, and try parallel threads

Scenario Testing
Test to a compelling story
1. Begin by thinking about everything going on around the product.
2. Design tests that involve meaningful and complex interactions with the product.
3. A good scenario test is a compelling story of how someone who matters might do something that matters with the product.

Claims Testing
Challenge every claim
1. Identify reference materials that include claims about the product (implicit or explicit). Consider SLAs, EULAs, advertisements, specifications, help text, manuals, etc.
2. Analyze individual claims, and clarify vague claims.
3. Test the veracity of each claim about the product.
4. If you're testing from an explicit specification, expect it and the product to be brought into alignment.

User Testing
Involve the users
1. Identify categories and roles of users.
2. Determine what each category of user will do (use cases), how they will do it, and what they value.
3. Get real user data, or bring real users in to test.
4. Otherwise, systematically simulate a user (be careful—it's easy to think you're like a user even when you're not).
5. Powerful user testing is that which involves a variety of users and user roles, not just one.

Risk Testing
Imagine a problem, then look for it.
1. What kinds of problems could the product have?
2. Which kinds matter most? Focus on those.
3. How would you detect them if they were there?
4. Make a list of interesting problems and design tests specifically to reveal them.
5. It may help to consult experts, design documentation, past bug reports, or apply risk heuristics.

Automatic Checking
Check a million different facts
1. Look for or develop tools that can perform a lot of actions and check a lot things.
2. Consider tools that partially automate test coverage.
3. Consider tools that partially automate oracles.
4. Consider automatic change detectors.
5. Consider automatic test data generators.
6. Consider tools that make human testing more powerful

Project Environment

Creating and executing tests is the heart of the test project. However, there are many factors in the project environment that are critical to your decision about what particular tests to create. In each category, below, consider how that factor may help or hinder your test design process. Try to exploit every resource.

Mission. *Your purpose on this project, as understood by you and your customers.*

- Do you know who your customers are? Whose opinions matter? Who benefits or suffers from the work you do?
- Do you know what your customers expect of you on this project? Do you agree?
- Maybe your customers have strong ideas about what tests you should create and run.
- Maybe they have conflicting expectations. You may have to help identify and resolve those.

Information. *Information about the product or project that is needed for testing.*

- Whom can we consult with to learn about this project?
- Are there any engineering documents available? User manuals? Web-based materials? Specs? User stories?
- Does this product have a history? Old problems that were fixed or deferred? Pattern of customer complaints?
- Is your information current? How are you apprised of new or changing information?
- Are there any comparable products or projects from which we can glean important information?

Developer Relations. *How you get along with the programmers.*

- *Hubris:* Does the development team seem overconfident about any aspect of the product?
- *Defensiveness*: Is there any part of the product the developers seem strangely opposed to having tested?
- *Rapport:* Have you developed a friendly working relationship with the programmers?
- *Feedback loop:* Can you communicate quickly, on demand, with the programmers?
- *Feedback*: What do the developers think of your test strategy?

Test Team. *Anyone who will perform or support testing.*

- Do you know who will be testing? Do you have enough people?
- Are there people not on the "test team" that might be able to help? People who've tested similar products before and might have advice? Writers? Users? Programmers?
- Are there particular test techniques that the team has special skill or motivation to perform?
- Is any training needed? Is any available?
- Who is co-located and who is elsewhere? Will time zones be a problem?

Equipment & Tools. *Hardware, software, or documents required to administer testing.*

- *Hardware:* Do we have all the equipment you need to execute the tests? Is it set up and ready to go?
- *Automation*: Are any test tools needed? Are they available?
- *Probes*: Are any tools needed to aid in the observation of the product under test?
- *Matrices & Checklists*: Are any documents needed to track or record the progress of testing?

Schedule. *The sequence, duration, and synchronization of project events*

- *Test Design:* How much time do you have? Are there tests better to create later than sooner?
- *Test Execution*: When will tests be executed? Are some tests executed repeatedly, say, for regression purposes?
- *Development:* When will builds be available for testing, features added, code frozen, etc.?
- *Documentation*: When will the user documentation be available for review?

Test Items. *The product to be tested.*

- *Scope:* What parts of the product are and are not within the scope of your testing responsibility?
- *Availability*: Do you have the product to test? Do you have test platforms available? When do you get new builds?
- *Volatility:* Is the product constantly changing? What will be the need for retesting?
- *New Stuff:* What has recently been changed or added in the product?
- *Testability*: Is the product functional and reliable enough that you can effectively test it?
- *Future Releases*: What part of your tests, if any, must be designed to apply to future releases of the product?

Deliverables. *The observable products of the test project.*

- *Content*: What sort of reports will you have to make? Will you share your working notes, or just the end results?
- *Purpose:* Are your deliverables provided as part of the product? Does anyone else have to run your tests?
- *Standards:* Is there a particular test documentation standard you're supposed to follow? *Media:* How will you record and communicate your reports?

Product Elements

Ultimately a product is an experience or solution provided to a customer. Products have many dimensions. So, to test well, we must examine those dimensions. Each category, listed below, represents an important and unique aspect of a product. Testers who focus on only a few of these are likely to miss important bugs.

Structure. *Everything that comprises the physical product.*

- *Code:* the code structures that comprise the product, from executables to individual routines.
- *Hardware:* any hardware component that is integral to the product.
- *Non-executable files:* any files other than multimedia or programs, like text files, sample data, or help files.
- *Collateral:* anything beyond software and hardware that is also part of the product, such as paper documents, web links and content, packaging, license agreements, etc.

Function. *Everything that the product does.*

- *Application:* any function that defines or distinguishes the product or fulfills core requirements.
- *Calculation:* any arithmetic function or arithmetic operations embedded in other functions.
- *Time-related:* time-out settings; daily or month-end reports; nightly batch jobs; time zones; business holidays; interest calculations; terms and warranty periods; chronograph functions.
- *Transformations:* functions that modify or transform something (e.g. setting fonts, inserting clip art, withdrawing money from account).
- *Startup/Shutdown:* each method and interface for invocation and initialization as well as exiting the product.
- *Multimedia:* sounds, bitmaps, videos, or any graphical display embedded in the product.
- *Error Handling:* any functions that detect and recover from errors, including all error messages.
- *Interactions:* any interactions between functions within the product.
- *Testability:* any functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.

Data. *Everything that the product processes.*

- *Input:* any data that is processed by the product.
- *Output:* any data that results from processing by the product.
- *Preset:* any data that is supplied as part of the product, or otherwise built into it, such as prefabricated databases, default values, etc.
- *Persistent:* any data that is stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
- *Sequences/Combinations:* any ordering or permutation of data, e.g. word order, sorted vs. unsorted data, order of tests.
- *Cardinality:* Numbers of objects or fields may vary (e.g. zero, one, many, max, open limit). Some may have to be unique (e.g. database keys).
- *Big/Little:* variations in the size and aggregation of data.
- *Noise:* any data or state that is invalid, corrupted, or produced in an uncontrolled or incorrect fashion.
- *Lifecycle:* transformations over the lifetime of a data entity as it is created, accessed, modified, and deleted.

Interfaces. *Every conduit by which the product is accessed or expressed.*

- *User Interfaces:* any element that mediates the exchange of data with the user (e.g. displays, buttons, fields, whether physical or virtual).
- *System Interfaces:* any interface with something other than a user, such as other programs, hard disk, network, etc.
- *API/SDK:* Any programmatic interfaces or tools intended to allow the development of new applications using this product.
- *Import/export:* any functions that package data for use by a different product, or interpret data from a different product.

Platform. *Everything on which the product depends (and that is outside your project).*

- *External Hardware:* hardware components and configurations that are not part of the shipping product, but are required (or optional) in order for the product to work: systems, servers, memory, keyboards, the Cloud.
- *External Software:* software components and configurations that are not a part of the shipping product, but are required (or optional) in order for the product to work: operating systems, concurrently executing applications, drivers, fonts, etc.
- *Internal Components:* libraries and other components that are embedded in your product but are produced outside your project.

Operations. *How the product will be used.*

- *Users:* the attributes of the various kinds of users.
- *Environment:* the physical environment in which the product operates, including such elements as noise, light, and distractions.
- *Common Use:* patterns and sequences of input that the product will typically encounter. This varies by user.
- *Disfavored Use:* patterns of input produced by ignorant, mistaken, careless or malicious use.
- *Extreme Use:* challenging patterns and sequences of input that are consistent with the intended use of the product.

Time. *Any relationship between the product and time.*

- *Input/Output:* when input is provided, when output created, and any timing relationships (delays, intervals, etc.) among them.
- *Fast/Slow:* testing with "fast" or "slow" input; fastest and slowest; combinations of fast and slow.
- *Changing Rates:* speeding up and slowing down (spikes, bursts, hangs, bottlenecks, interruptions).
- *Concurrency:* more than one thing happening at once (multi-user, time-sharing, threads, and semaphores, shared data).

Quality Criteria Categories

A quality criterion is some requirement that defines what the product should be. By thinking about different kinds of criteria, you will be better able to plan tests that discover important problems fast. Each of the items on this list can be thought of as a potential risk area. For each item below, determine if it is important to your project, then think how you would recognize if the product worked well or poorly in that regard.

Capability. *Can it perform the required functions?*

Reliability. *Will it work well and resist failure in all required situations?*

- *Robustness:* the product continues to function over time without degradation, under reasonable conditions.
- *Error handling:* the product resists failure in the case of errors, is graceful when it fails, and recovers readily.
- *Data Integrity:* the data in the system is protected from loss or corruption.
- *Safety*: the product will not fail in such a way as to harm life or property.

Usability. *How easy is it for a real user to use the product?*

- *Learnability:* the operation of the product can be rapidly mastered by the intended user.
- *Operability:* the product can be operated with minimum effort and fuss.
- *Accessibility:* the product meets relevant accessibility standards and works with O/S accessibility features.

Charisma. *How appealing is the product?*

- *Aesthetics:* the product appeals to the senses.
- *Uniqueness:* the product is new or special in some way.
- *Necessity:* the product possesses the capabilities that users expect from it.
- *Usefulness:* the product solves a problem that matters, and solves it well.
- *Entrancement:* users get hooked, have fun, are fully engaged when using the product.
- *Image:* the product projects the desired impression of quality.

Security. *How well is the product protected against unauthorized use or intrusion?*

- *Authentication:* the ways in which the system verifies that a user is who he says he is.
- *Authorization:* the rights that are granted to authenticated users at varying privilege levels.
- *Privacy:* the ways in which customer or employee data is protected from unauthorized people.
- *Security holes*: the ways in which the system cannot enforce security (e.g. social engineering vulnerabilities)

Scalability. *How well does the deployment of the product scale up or down?*

Compatibility. *How well does it work with external components & configurations?*

- *Application Compatibility:* the product works in conjunction with other software products.
- *Operating System Compatibility:* the product works with a particular operating system.
- *Hardware Compatibility:* the product works with particular hardware components and configurations.
- *Backward Compatibility:* the products works with earlier versions of itself.
- *Resource Usage:* the product doesn't unnecessarily hog memory, storage, or other system resources.

Performance. *How speedy and responsive is it?*

Installability. *How easily can it be installed onto its target platform(s)?*

- *System requirements:* Does the product recognize if some necessary component is missing or insufficient?
- *Configuration:* What parts of the system are affected by installation? Where are files and resources stored?
- *Uninstallation:* When the product is uninstalled, is it removed cleanly?
- *Upgrades/patches:* Can new modules or versions be added easily? Do they respect the existing configuration?
- *Administration:* Is installation a process that is handled by special personnel, or on a special schedule?

Development. *How well can we create, test, and modify it?*

- *Supportability:* How economical will it be to provide support to users of the product?
- *Testability:* How effectively can the product be tested?
- *Maintainability:* How economical is it to build, fix or enhance the product?
- *Portability:* How economical will it be to port or reuse the technology elsewhere?
- *Localizability:* How economical will it be to adapt the product for other places?

Car Stereo V 1.0

Use this for Testing by Yourself, Pair Testing, and Pair Testing with SFDIPOT to generate test cases and questions. Use a pen / pencil and sheets supplied to record your test cases and any questions.

Car Owner: Recently, you purchased a car. Now, you're taking some time to familiarize yourself with the features of the radio and how its functions work.

Car Stereo V 2.0

Use this for Mob Test Design using SFDIPOT to generate test cases and questions. Use a pen / pencil and sheets supplied to record your ideas and any questions.

Car Stereo Tester: Your company has been asked to develop a new car stereo with a software interface. You've been brought in to provide your testing insight on what should be considered when developing the car stereo.

Mobbing for Test Design Test Cases / Questions Sheets

Mobbing for Test Design Test Cases / Questions Sheets