Managing “Testing Cycles” efficiently

Yury Makedonov

(416) 481-8685
yury@ivm-s.com
http://www.softwaretestconsulting.com

1

Introduction

- Sometimes the testing of a product consists of several so called “Testing Cycles”.
- To manage the Testing Cycles we have to better understand their nature.
- In this presentation we discuss the different reasons for why these Testing Cycles can happen and how to handle them.

Agenda

The following types of “Testing Cycles” will be discussed:

- Real Testing Cycles
- Defect Fixing Cycles
  - Code Fixing Cycles
  - Design Fixing Cycles
  - Requirements Fixing Cycles
- Death March Cycles

Real Testing Cycles

- Sometimes we group all test cases according to their timing or hierarchy, e.g.:

  Cycle 1: Transactional test cases (invoices, purchase orders, etc.)

  Cycle 2: Weekly cheque run

  Cycle 3: Month-end reports and batches
- This approach is typically used in complex System Integration Testing projects.
- We plan and manage these testing cycles using standard project management techniques.

Testing Cycles in an Agile environment

- The “Testing Cycle” term is not used in:
  - Extreme Programming,
  - Scrum,
  - Lean Software Development
  - etc.
- In case of an Agile process:
  - Testing is happening concurrently with development
  - A team starts another **iteration** regardless of how many defects are discovered.

Testing Cycles in Agile environment

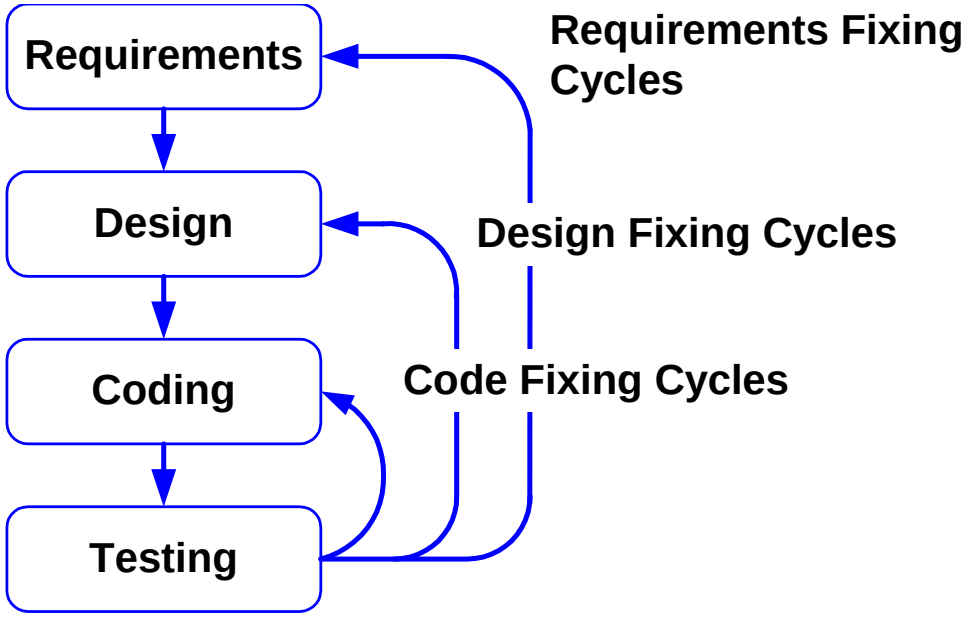
- New iteration might be devoted to new features or to defect fixing, but the term “Testing Cycle” is not used.
- The development backlog consists of:
  - New features to implement
  - Defects to fix
- Defects are treated the same way as new features.
- These processes have no need for special “Testing Cycles”.

It looks like these guys have everything under control.

Testing Cycles in Waterfall SDLC

- Typically we hear about “Testing Cycles” in projects which use Waterfall SDLC or its derivatives.
  - Testers discover and report defects
  - Developers fix these defects and send a new release for testing
- Apparently these are “Defect Fixing Cycles”!
- So, typically the “Testing Cycles” term is used in a Waterfall development process to describe different “Defect Fixing Cycles”.

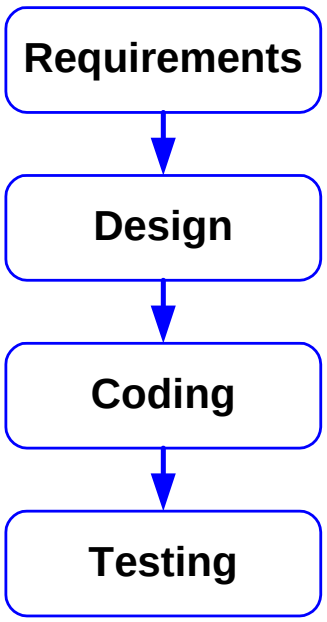
Different Defect Fixing Cycles



Defect Fixing Cycles vs. Testing Cycles

- Why do these “Testing Cycles” happen in Waterfall but not Agile processes?
- Why do managers use such imprecise and misleading terms?
- Typically managers are not stupid.
  They are apparently trying to reach certain goals.
- What are these goals?

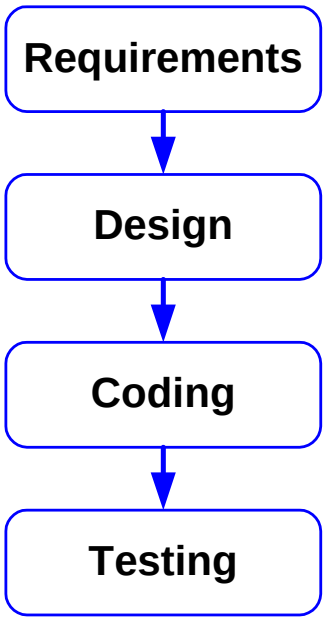
Waterfall SDLC:



According to this model :

- The development of requirements is finished before the start of a design phase.
- Design is finished before the start of a coding phase.
- Coding is finished before the start of testing.

Waterfall SDLC:



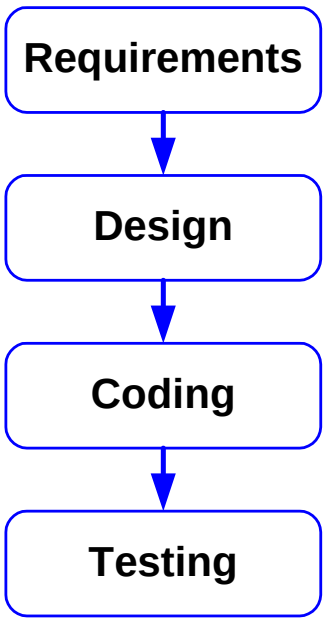
These tasks are used for:

- Budgeting,
- Planning,
- Status reporting

from the very bottom of the organization to the very top.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 11

Waterfall SDLC:

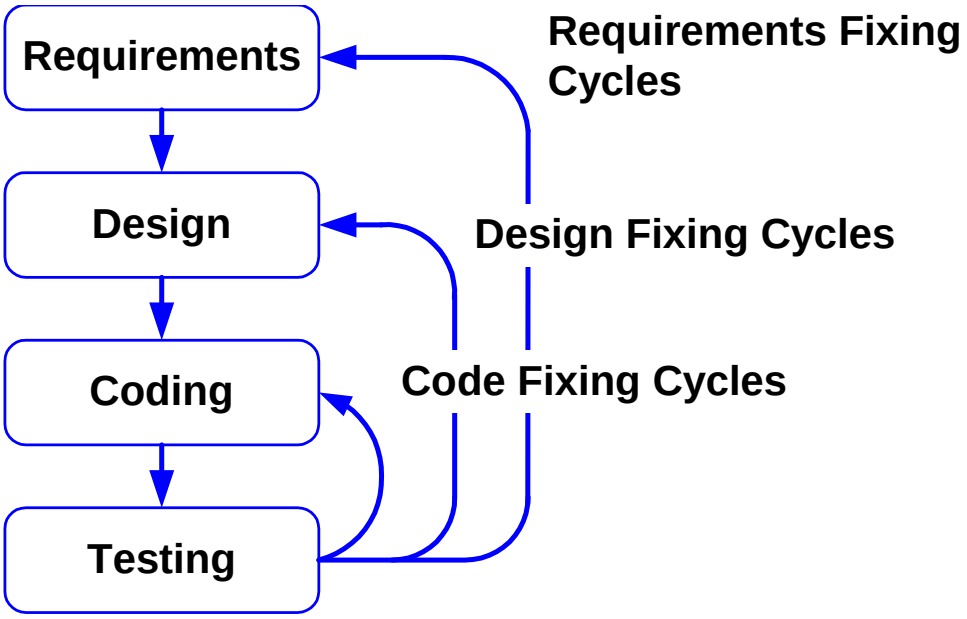


“In theory, there is no difference between theory and practice. In practice, there is...” :

- In real life this theoretical model sometimes doesn’t work.
- Managers just pretend that their projects follows this model.
- They use this Waterfall terminology to maintain the illusion that they are in complete control of a project.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 12

Different Defect Fixing Cycles



Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 13

Terminological “mind tricks”

- “Testing Cycles” are just terminological “mind tricks”.
- Why are managers willing to lose efficiency through the use of such imprecise and erroneous language?
- Why are managers playing these games?
- Are these games innocent or not?
- We won’t discuss why such tricks are sometimes used by manager–magicians in mature, hierarchical organizations.
  It’s outside the scope of this presentation.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 14

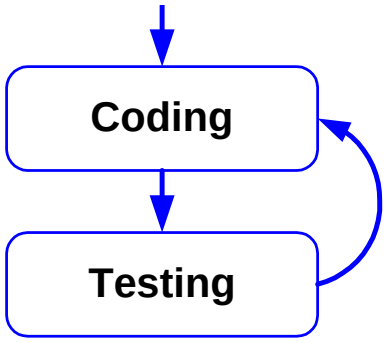
Survival of terminological “mind tricks”

- Do not fight for better and cleaner language at any cost.
  Remember that the managers already made their choice.
- Let’s instead discuss what testers should do to better handle this situation.

Code Fixing Cycles

1 – Code Fixing Cycles

Code Fixing Cycles



- Testers are just doing their job – they report defects and verify whether they are fixed or not.
- Everything is simple unless there is a request to plan such Testing Cycles in advance:
  - Number of these cycles
  - Duration of these cycles

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 17

Planning Code Fixing Cycles

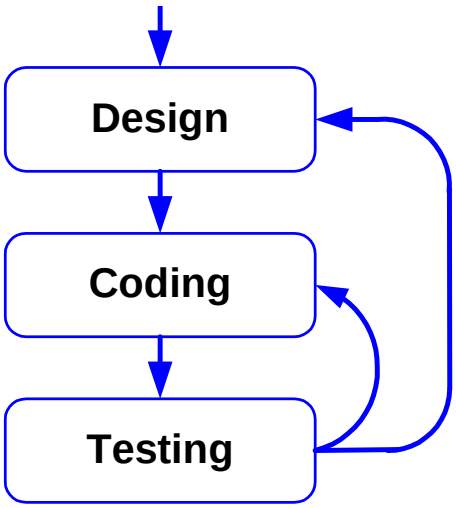
- Testers can’t create such plans because it was not them who created these defects.
- This is just a game! Give some reasonable data:
  - Use numbers from previous releases to estimate the duration and the number of these testing cycles.
  - Use your best guesstimates for a new project.
  - Clearly describe your assumptions.
- There is no value in spending a lot of time working on such plan.
- Pass the ball back to the developers - send your estimate back to developers and the PM for review and approval.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 18

Design Fixing Cycles

2 – Design Fixing Cycles

Design Fixing Cycles

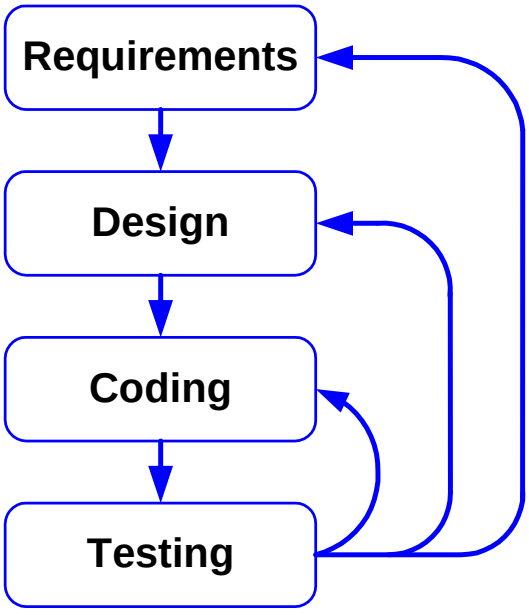


- They are rather similar to code fixing cycles.
- The risk is that a major design defect may require a significant redesign and a lot of coding to repair it.
- You just can’t plan for this in advance.

Requirement Fixing Cycles

3 – Requirement Fixing Cycles

Requirement Fixing Cycles



- These are typically the most complex cycles to handle.
- We have to understand the root cause of these requirement changes.
- New test cases are required.

Requirement changes

- **In a mass market** company the requirements for a specific release are typically pretty solid.

  All new ideas are incorporated into the requirements for the next release to avoid disruption, and to decrease the time to market for the release under development.
- Requirements are typically much less stable in case of custom software **when a real customer is present**.

  The problem is the most severe when a customer doesn’t have a mature software acquisition process and doesn’t have recent software acquisition experience.

Root cause of Requirement Fixing cycles

Let’s take a look how requirements were developed:

- The Business Analyst (BA) talked to users and stakeholders.
- Users and stakeholders explained what they needed.
- The Business Analyst got a **perception** that he understood these users.

Root cause of Requirement Fixing cycles

In reality the users and the Business Analyst were speaking different languages:

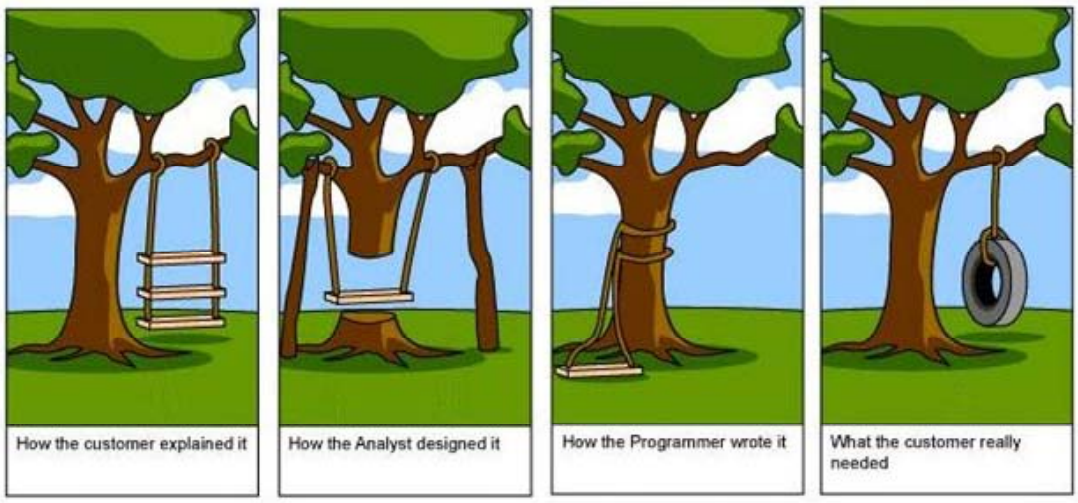
- they used different meanings of the same English words,
- considered different contexts,
- made different assumptions.

As a result the Business Analyst significantly misinterpreted the application users’ needs and created a flawed requirements document.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 25

When a product was delivered

Customer: “This is not what we wanted!”



Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 26

A case study of Requirement Fixing cycles

- It was a new application for the sales department (~100 people) of a division of a big company.
- The goal of this department was to configure and sell telecommunication products.
- The new application was to replace an existing legacy Client/Server application.
- The modern web application was to have a sexy web interface and more functionality.
- This department did not have an established software acquisition/implementation process.
- The vendor was a start-up company trying to establish itself. 10-15 developers were working on this project initially.

A case study of Requirement Fixing cycles

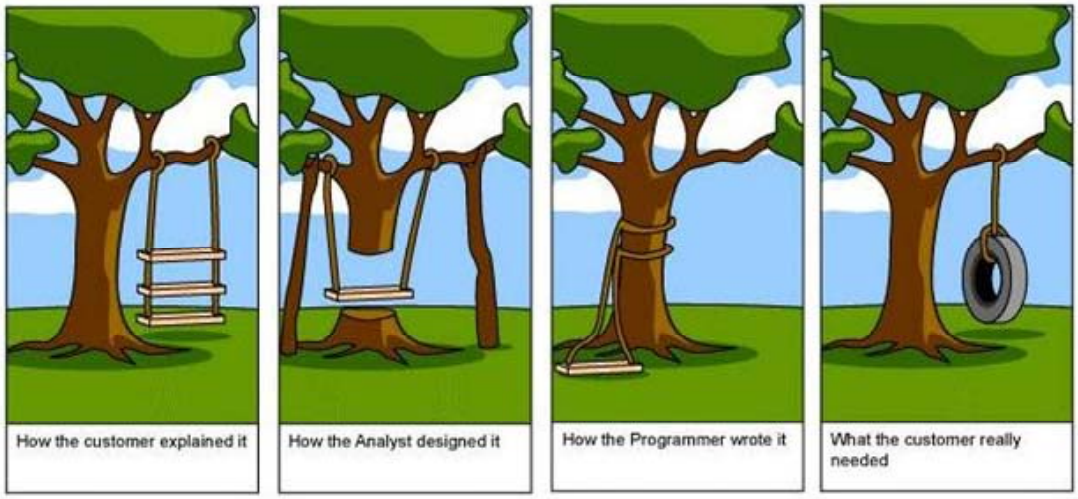
This project followed the classic review and sign-off approach:

- Requirements were reviewed and signed off
- Design of the system was reviewed and signed off
- Specifications were reviewed and signed off
- Development was finished and tested by the vendor

The first attempt to deliver the application to the customer failed. The customer’s reaction was:

- “This is not what we wanted!”

Customer: “This is not what we wanted!”



Episode II - One more attempt

- The requirements were modified using the same review/sign off approach.
- The vendor implemented these updated requirements.

- To make the acceptance process more organized the customer brought in an external Test Manager to supervise the user acceptance testing.

Development of acceptance test cases

- End users were brought to the project team to develop acceptance test cases to verify the requirements.
- They started working together with “professional” testers and were trained on how to develop test cases.
- Users were asked to use their own language when writing these test cases. They ranked all existing requirements and started developing test cases for the most important requirements.

Development of acceptance test cases

- The first version of the test cases was incomprehensible for professional testers and developers.
- The test cases were modified and the second version could be understood by both testers and developers.
- Further on these acceptance test cases came to be used instead of the original requirements.
- These test cases still had many errors but nevertheless were much better than original and revised “requirements.”

Start of testing

- Users started testing as soon as acceptance test cases for the major functions were developed.
- The first round of testing was a complete disaster:
  - Most test cases failed.
  - It was impossible to even initiate the execution of many test cases - they depended on the successful executions of other test cases.
- Many “Severity 1” and “Severity 2” defects were recorded (high severity defects).

Change Control Board established

- To better manage this project a “Change Control Board” was established. It consisted of:
  - the project sponsor,
  - users of the application (acceptance testers) and
  - the developers’ managers.
- The Board approved a list of defects (including several missing functions) to be fixed.
- Only defects of the highest severities (“Severity 1” and “Severity 2”) were to be fixed.

Defect fixing

- The developers were not allowed to fix any defect or implement any new feature without the approval of this board.
- The developers were allowed to work only on defects of the highest priority and defects that prevented the execution of major test cases.
- The repair of not so severe defects was postponed despite the developers’ promises that some of them required “only 10 to 30 minutes to fix.”

Role of end users

- The vendor had daily builds delivered for testing.
- End users (acceptance testers) were available to clarify specific details of test cases (requirements) and defect reports for developers.
- Those clarifications were important because initially the developers had a lot of problems interpreting the test cases and defect reports.
- Development of new acceptance test cases continued.

Role of end users

- Why were end users more efficient at developing and executing test cases?
- They have working knowledge of their business – how to configure a product and create an order.
- They were able to use their knowledge better when creating and executing test cases than when talking to a business analyst and trying to understand his gibberish language.

Change Control Board’s role

- The Change Control Board discussed:
  - New test cases, which were essentially new requirements,
  - Progress of testing,
  - Fixed defects,
  - New discovered defects,
  - Defects reported as fixed by developers that were “failed” by acceptance testers.
- Priority of remaining defects and additional “new” defects were reviewed and revaluated every day.

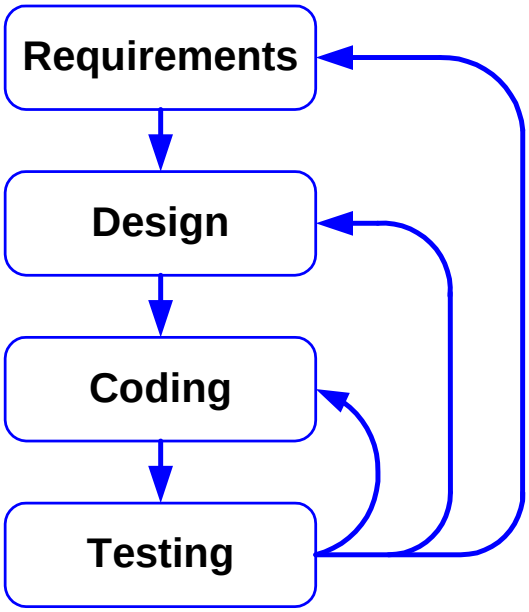
Requirement Fixing Cycles recommendations

- What allowed for the efficient management of “Testing Cycles“ on this project:
  - Active user involvement, instant feedback
  - Active involvement of the project sponsor, fast decision making.
  - Co-location, effective communication
  - Iterative and incremental development in a customer selected order.
- The same approach can be recommended for any project when significant changes in requirements are expected.

“Death March” Cycles

3 – “Death March” Cycles

Death March Cycles



- Developers and managers do not believe that they will meet a deadline.
- Developers just throw an unfinished product over the wall for testing, pretending that development was completed.
- Testers are pressured to confirm that this release is ready for production.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 41

Death March Cycles

- Most people do not enjoy this game.
- This game might be dangerous.
- You may develop adversarial relationships with developers and turn some of your developer-friends into enemies.
- You may even lose your mental or physical health.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 42

How to spot a Death March project

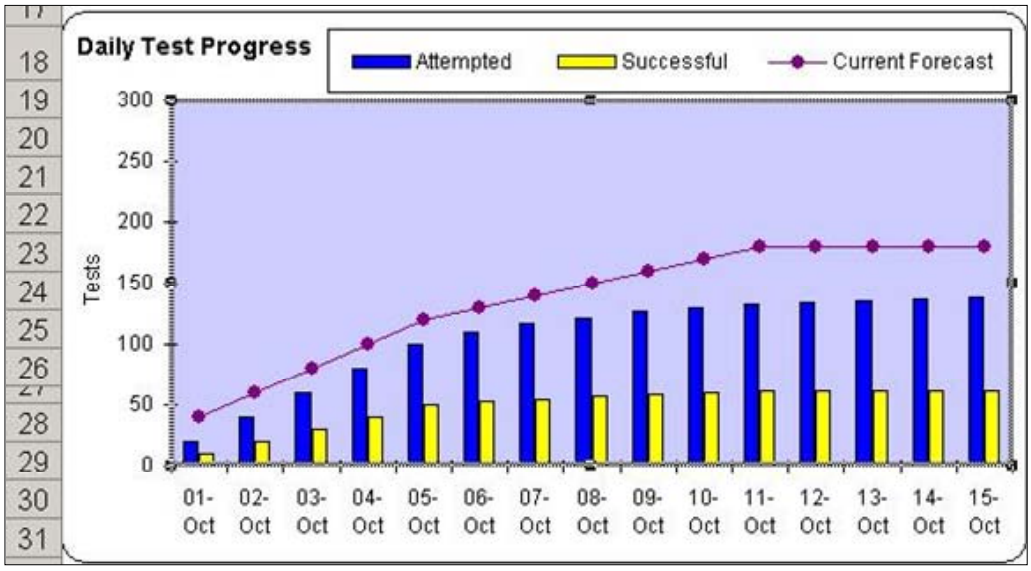
- Ambiguous chain of commands e.g.:
  - You were put in charge of User Acceptance Testing but have no access to the project owner or a customer PM.
  - You were brought in as a test expert to help a customer with User Acceptance Testing but were paid by the software vendor.
- You are not getting answers for your questions; escalated issues are not being resolved.
- Testers are pressured to “certify” a release for production.
- Acceptance testing starts with an obviously unfinished and unusually buggy product.

Death March – testing

- Try avoiding adversarial relationships with developers – you are members of the same team and depend on each other’s help.
- Provide honest feedback to developers.
  They are working on defect fixes – they need this information.
- Do not forget to properly document all defects.
  Proper record keeping is your best friend under these circumstances.

Death March – status reporting

- Tracking the delivered functionality:



Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 45

Death March – providing options

- Possible options when a deadline can’t be met:
  - Breaking a big release into several smaller releases and delivering a bare minimum release by a deadline.
  - Field trial – delivering a release only to a subset of all customers.
  - Pilot – moving an application into the production environment for further testing.

Managing “Testing Cycles” efficiently, © 2006 Yury Makedonov 46

Death March – summary

- Quit? After all you only live once.
- What to do:
  - Provide honest feedback to developers
    They are working on defect fixes – they need this information.
  - Provide honest feedback to management.
    This information may help them make some important desisions.
  - Provide management with additional options.
- What not to do:
  - Do not provide an overly optimistic status report.

Conclusion

- To successfully manage Testing Cycles you have to understand their nature and use the corresponding techniques.

Q & A

Questions?

Contact Information

Yury Makedonov

Principal Consultant
IVM-S

(416) 481-8685
yury@ivm-s.com

http://www.softwaretestconsulting.com

Appendix – further reading

- The Winston Royce paper, "Managing the Development of Large Software Systems“
- The Edward Yourdon book, “Death March”
- Google, Wikipedia, etc.