Orthogonal Defect Classification in Agile Development

Monika Jagia, IBM Software Group – India, monika.jagia@in.ibm.com
Seema Meena, IBM Software Group – India, seemeena@in.ibm.com

Information Management software

Orthogonal Defect Classification

- Orthogonal Defect Classification - A technology for software process measurement and analysis, providing a standard of unique, non-redundant attribute definitions for defect classification as well as analysis metrics and procedures for process evaluations
- Developed at IBM in the 1990s by Ram Chillarege.
- ODC is a way of classifying and analyzing defects and APARs to Uncover more defects during testing, Reduce the number of escapes to the field and Prevent defects from being injected in the code
- Methodology to characterize software defects and translate into process defects
- ODC shows teams how to ensure they are looking for the right defects at the right time.
- Benefits of ODC:
  - Describe the stability of the design and of the product as a whole (by ODC analysis we get information about source of defects (design, code, …))
  - Monitor test progress and effectiveness, in terms of how deeply they are able to get into the product, as opposed to how many tests have they run
  - Improved communication and decision making (“gut feel”)

Agile Development

- Process for detecting and removing design defects at the early stage .
- Agile Defined (IBM) - “Uses continuous stakeholder feedback to deliver high-quality, consumable code through user stories and a series of short, time-boxed iterations.”
- Characteristics of Agile:
  - Task-based Configuration Management
  - Building traceability automatically
  - Peer Reviews, Pair programming
  - Task agility
  - Regular builds, iterations & Releases
  - Scrum: Regular meetings to identify status and progress
  - Progressive and constant testing/verification/validation of the produced artifacts.
- Benefits of Agile :
  - Increased quality
  - Mapping to customer needs
  - Faster time to market
  - Responsiveness to change

Can ODC be used on agile projects?

- Absolutely! Agile teams still have to perform testing and have a way to measure quality. ODC concepts, attributes and tooling help in understanding the iteration, look at the data, and ask questions about it so you can improve.

How does ODC work with agile?

- Both milestone-driven and agile projects have a few things in common. They both need ways to measure quality, need to perform unit testing, functional/component testing, and system testing.
- In Agile, the ‘Activity within Phase’ chart may not provide value to the team since we’d expect to see all forms of testing occurring in each iteration (i.e.. it’s not a case of a defect escaping from one phase to another).
- It’s important to remember that each iteration will be different. The ODC data will reflect what happened in that iteration so the data could differ from the data of previous or future iterations.

How often should we assess the data in agile projects?

- It depends on the length of the iterations.
- Assessments at the mid-point of an iteration probably won’t yield ROI for projects with 2 week iterations.
- Shorter iterations will have fewer defects at the end of the iteration to assess. Research indicates you need about 30-35 defects before assessing the data and drawing conclusions. Teams will need to determine if there are sufficient defects at the end of the iteration to perform an assessment. If not, it may be best for the team to assess every couple of iterations.
- With longer iterations, consider doing mid-point and end of iteration assessments.
- In addition, there are pieces of data that should be watched weekly so that problems can be identified and attacked as soon as possible.
- jMYSTIQ provides the ability to automate the chart creation which reduces the overhead of creating charts weekly.

Can other agile practices affect the ODC data?

- Most definitely. Adoption of various agile practices (ex. Inspections, pair programming, etc) will affect defect injection. The more of these preventative practices the team adopts, the fewer defects will be in the product waiting to be detected.

How will the data differ for an agile project?

- Early discovery of defects including complex defects
- Higher percentage of simpler triggers (i.e. ‘coverage’ and ‘variation’) defects in ‘new’ function being dropped into an iteration as well as ‘missing’. In follow-on iterations those areas should stabilize such that missing decreases and more complex defects emerge.
- By the end of the project, when looking at the total set of defects, we’d expect to see all the triggers and a decline in the serious defects (basically no different from waterfall projects)

Is there anything extra we have to do?

- Teams need to understand the % hit to a component from iteration to iteration to help determine the anticipated defect trends for the current as well as future iterations (i.e. the longer the function is integrated into the system, the more stable it should be)

What’s the Minimum application of ODC?

- In agile process if the team decides not to classify defects, they can still derive value from ODC by taking the time to classify their test cases based on the ODC “Trigger” attribute.
- This is to ensure that:
  - The test cases are looking for all the different types of defects that could be “lurking” in the product.
  - The percentage of test cases aimed at the different triggers is appropriate

Value ODC Assessments Provide

Independent of the process, ODC will help teams assess three specific areas:

§ <u>**Defect Removal Effectiveness**</u> – This shows how well we're doing at uncovering the defects

§ <u>**Product Stability**</u> – This shows if the product is showing signs of increased or decreased stability

§ <u>**Defect Injection**</u> – This shows where the defects are getting injected so we can take action not to have the same mistakes happen again

ODC Metrics

§ **ODC Triggers** – Environment or conditions necessary to uncover the defect

- Coverage, Variation (parameters), Sequencing (order), Interaction

§ **ODC Defect Type** – What was fixed in the code

- Assignment/Initialization, Checking, Algorithm/Method, Timing/Serialization, Interface, Function/Class/Object

§ **ODC Qualifier** – Refers to defect types

- Incorrect, Missing, Extraneous

Background

- Product and Team Description
  - Teams working on different components
  - Integrating components with the main product
  - Teams comprising of 6 developers and 3 testers
  - Teams were new to agile methodologies
  - Each iteration of 3 weeks
- Agile Implementation Details
  - Agile Practices followed includes Paired programming, Test Driven Development, Continuous Integration.
  - Small Releases every 3-4 months
  - Daily Scrums
  - Mini Jam sessions

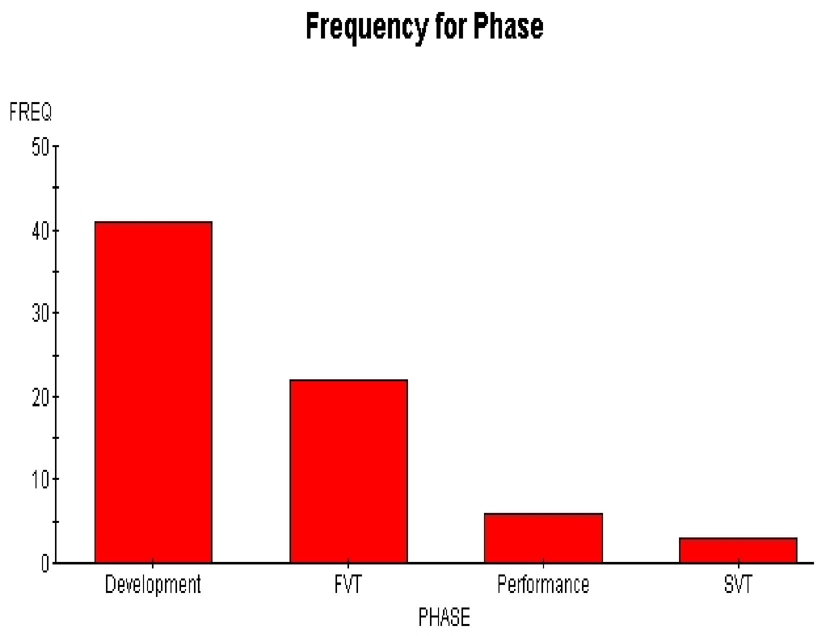
Defect Removal Effectiveness

- **Measure Results:**
  - **Defect rate**
    - Decrease through development (tdd, pp, and ci)
    - Increase through development (late arriving requirements)
  - **Triggers**
    - Elevated coverage and interaction early, variation and sequencing later (tdd and ci)
    - Elevated coverage as requirements arrive late
  - **Defect Type**
    - Low number assignment/initialization (pp)
    - High number Interface (ci)
    - High number Function/Class/Object (tdd, late requirements)

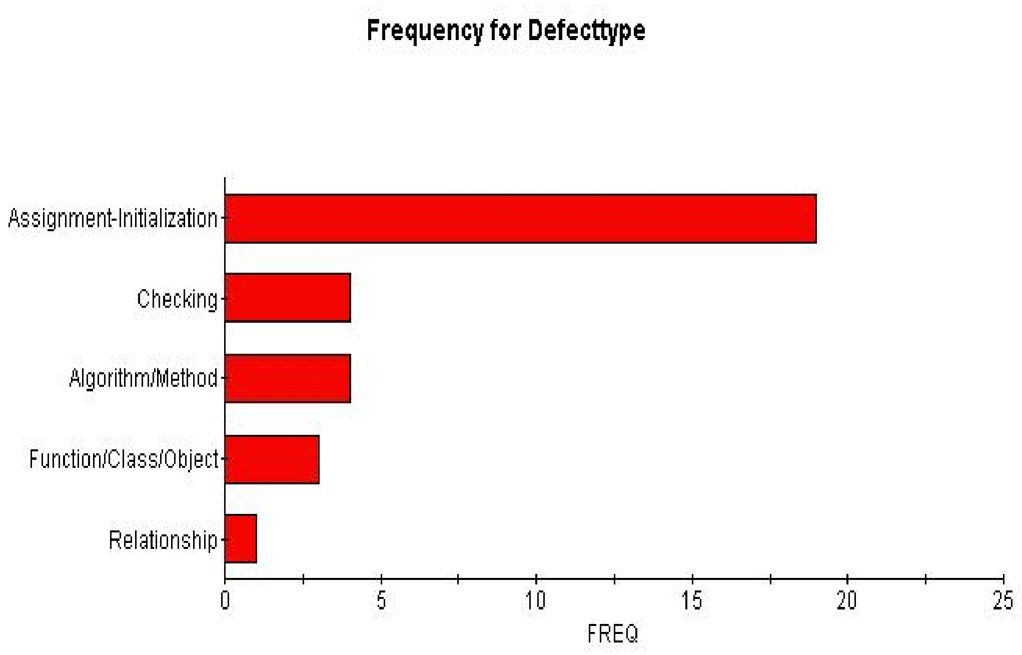
ci – continuous integration tdd – test driven development pp – paired programming

Defect Removal Effectiveness

Defect Rate



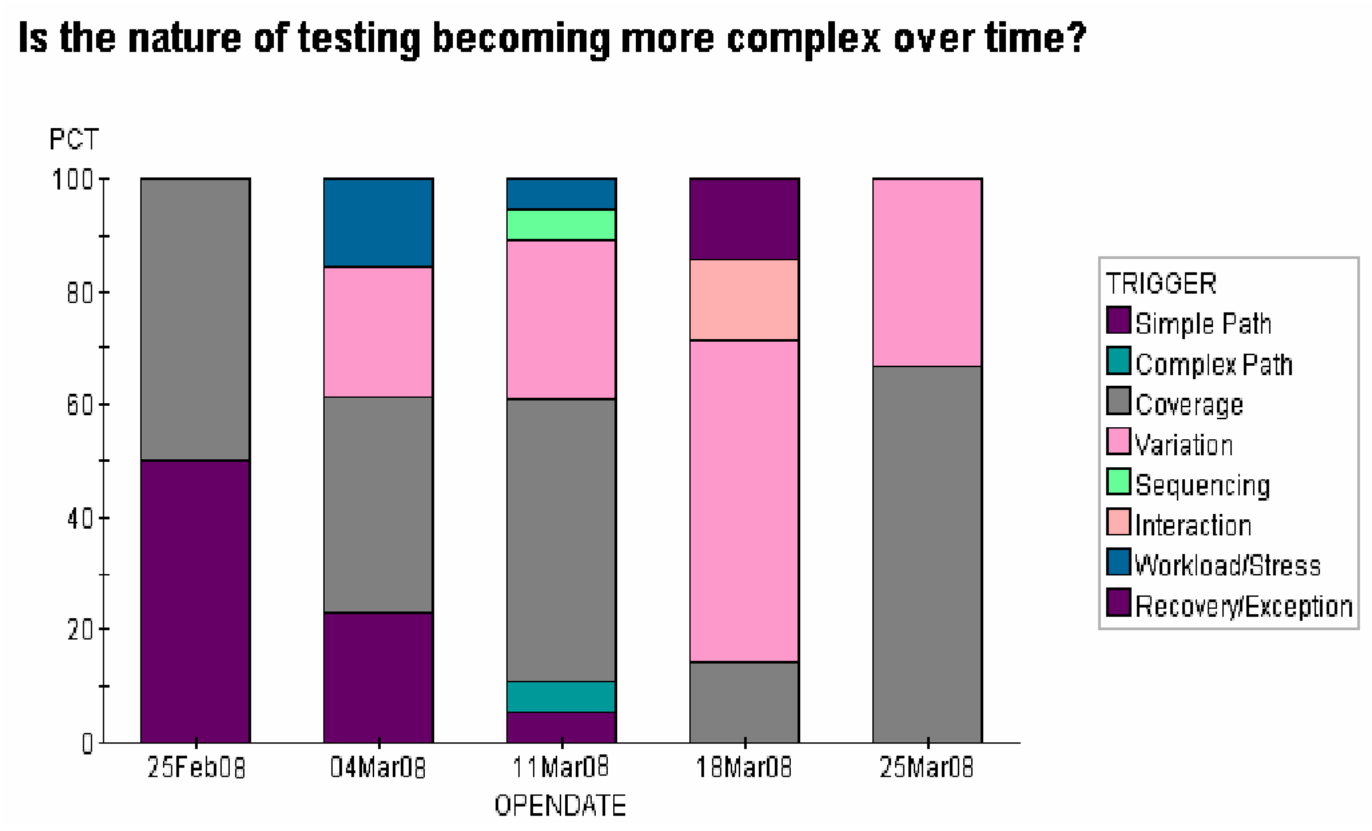
Defect Type



Found defects early, less expensive, decreased costs as compared to waterfall models

Too much assignment/init, No interface, no timing/serialization

Defect Removal Effectiveness - Trigger within Opendate



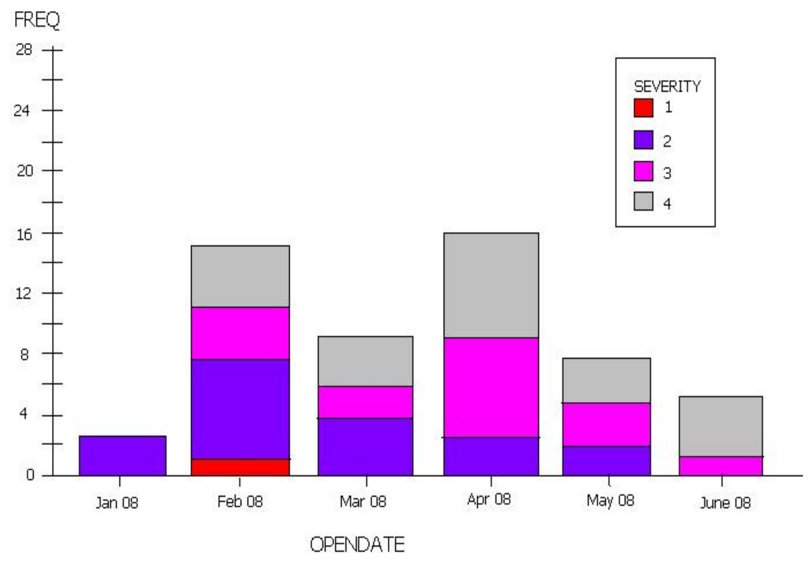
Too little variation, interaction, and no sequencing at all

Product Stability

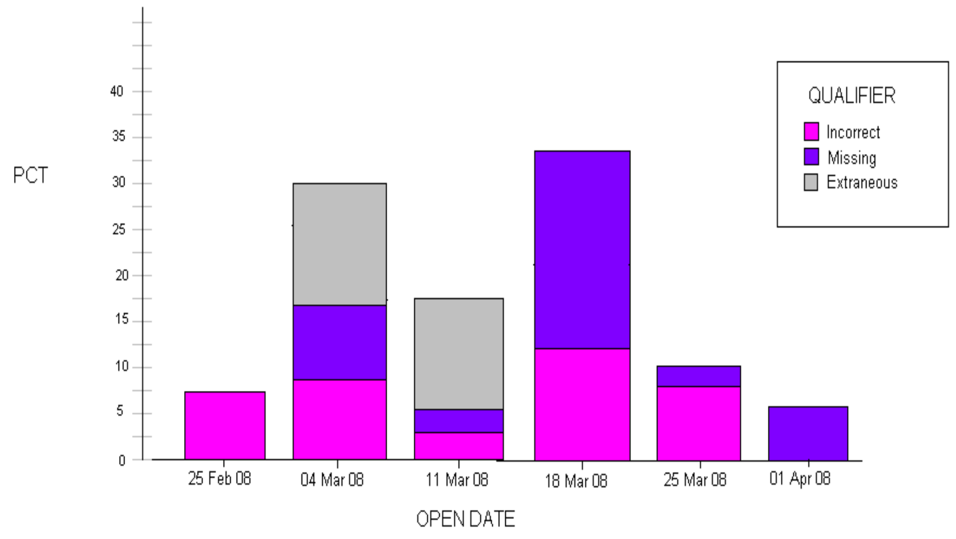
- **Measure Results:**
  - Severity: Percentage of high severity defects decreases
  - Qualifier: Percentage of missing is fairly small
- **Expectations:**
  - May decrease throughout development as requirements arrive late
  - Should increase through development with
    - Test driven development
    - Continuous integration
    - Simple design
    - Design improvement
- High stability is defined as Low # Sev 1's and 2's; Low missing code

Product Stability

Severity within Opendate



Qualifier within Opendate



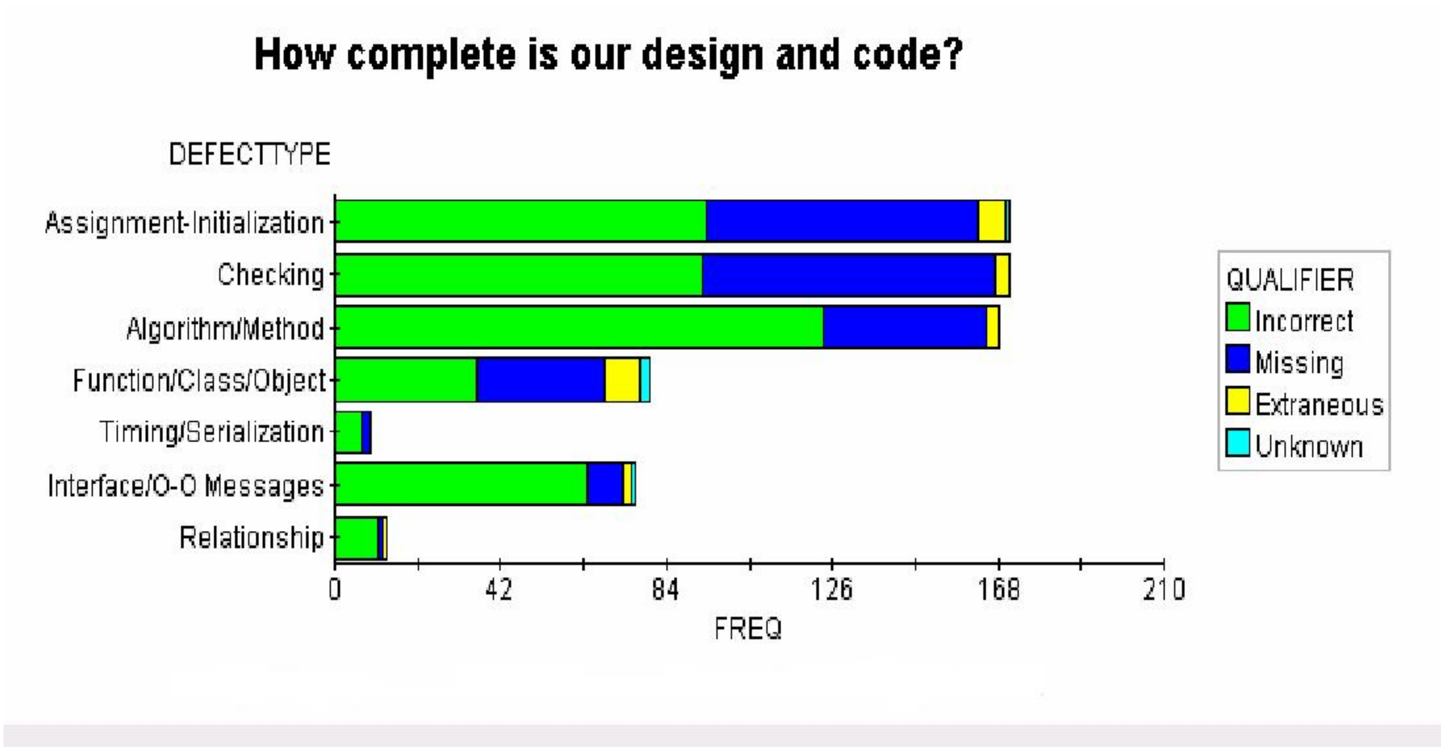
Percentage of severity 2 increasing through Feb. but there are no sev 1 or 2's in June

Percentage of missing is fairly small compared to most waterfall projects

Defect Injection

- Defects can be injected during requirements, high level design, low level design, or coding
- Expectations:
  - Paired programming will help decrease injection of defects in coding
  - Accommodating late arriving requirements will show increase injection of defects during requirements and high level design

Defect Injection – Qualifier within Defect Type



A high frequency of incorrect code with Assignment type, Checking and Algorithm type indicates that the code is poorly written.

Conclusion

- Agile teams still have to perform testing and they still have to have a way to measure quality.
- The ODC concepts, attributes, and tooling are the same. Your project's assessments will differ. The key is to understand what’s going on in the iteration, look at the data, and ask questions about it.
- By understanding the different ODC attributes your team will make sure they are both effective and efficient in their testing and product development.
- ODC is able to evaluate implementation of certain agile methodologies and practices.
- Several agile practices appear to provide a clear win over waterfall particularly with finding defects early where it is less costly.
- Continue collection of metrics to monitor progress.

Conclusion (Contd)

- The team found defects early, maintained a fairly stable product, and mitigated risk due to accommodating late arriving requirements which resulted in their ability to deliver on time and within their budget.
- Effective defect removal need to improve test cases to include variations, interaction, sequencing scenarios.
- Persist with paired programming throughout entire project and get developers more fully engaged in coding, focusing on finding incorrect assignment/initialization defects.
- Practices contributing to this positive result were:
  - Test driven development
  - Continuous integration
  - Paired programming
  - Simple design

Challenges

- Define valid defect and work-in-progress development behavior
- Handle ODC assessments that may seem to be less Agile
- Integrating with product which has a waterfall process
- Maintaining stability and mitigating risk while accommodating late arriving requirements
- Collective ownership resulting in higher defect injection during coding
- Limiting the number of triggers and avoiding the activity field.
- Writing a bunch of code that wasn’t planned - because of defects that are discovered.
- Maintaining effectiveness of paired programming over long periods

References

- R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D. Moebus, B. Ray and M.Y. Wong, “Orthogonal defect Classification - A Concept for In-process Measurement,” IEEE Transactions on Software Engineering, pp. 943-956, November 1992.
- M. Butcher, H. Munro, and T. Kratschmer, "Improving software testing via ODC: Three case studies". IBM Systems Journal, vol. 41 No. 1 pp. 31-44, 2002
- Theresa Kratschmer, and Adrian Dick, “Evaluating The Effectiveness of an Agile Development Effort”, IBM Corporation, theresak@us.ibm.com
- ODC forum http://ibmforums.ibm.com/forums/forum.jspa?forumID=3240
- Center for Software Engineering http://www.research.ibm.com/softeng/ODC/ODC.HTM