

# Agile 2011 Conference

## Key Take Aways

August 2011



[www.synerzip.com](http://www.synerzip.com)

# Conference Overview

---

- “ August 8-12 in Salt Lake city
- “ 1604 participants, 43 countries, 268 sessions 180 speakers
- “ We (Vinayak and Hemant) attended 20 sessions each plus Exhibit booths of about 40 vendors of tools and training services
- “ This was the tenth anniversary of signing of agile manifesto. We had the opportunity to meet 15 of the 17 signatories.
- “ When they were asked if the agile manifesto in its original form was still relevant or it needed any amendments or addendum ? The unanimous answer was ~~No~~+. the manifesto is still relevant in its original form.

# Conference Organization

---

- “ New 1-day Executive Forum, with invited senior executives
- “ New stages and more opportunities to network in Open Jam
- “ New app to create and manage schedule on web as well as smartphone
- “ Big sponsors like Google or Microsoft were absent or kept out giving more relevant companies a bigger share of the lime light
- “ Lot of interaction and buzz on twitter

---

# Top 10 Take Aways

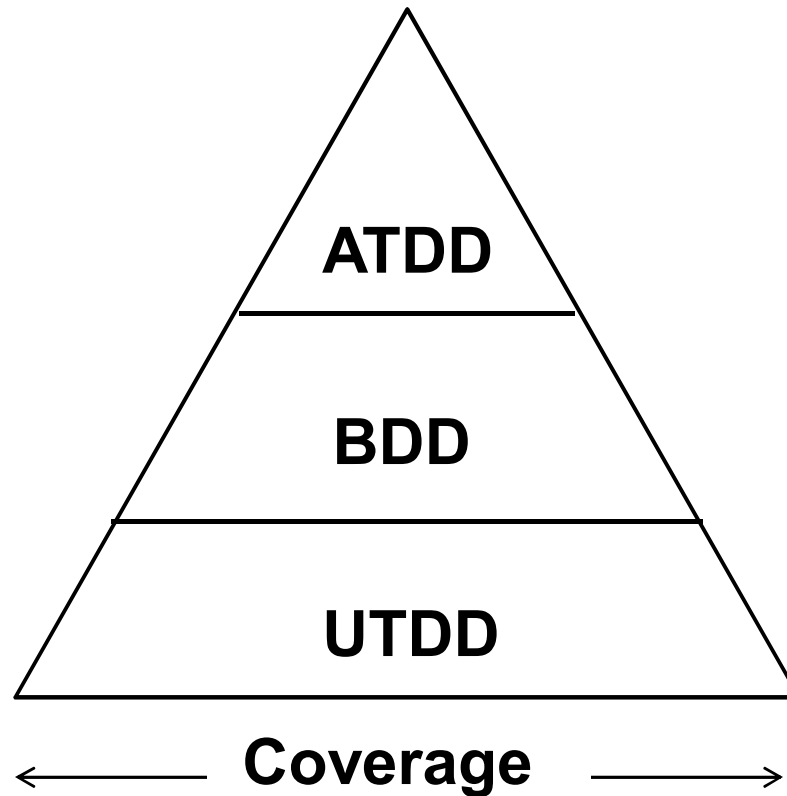
# 1. ATDD, BDD, & UTDD

---

- “ Widespread adoption of Acceptance TDD as a routine practice
- “ From Jenitta Andrea's session on ATDD
  - . %A TDD revolutionizes the way developers do software development+following the cycle of Card >Conversation>Confirmation
  - . %A TDD should not be an optional practice+
  - . Acceptance Tests = Trusted Specs, that stay in synch with code
- “ Typically written, in collaborative fashion, by %B Amigos+. Product Owner, Developer, and Tester (Christian Hassa)
- “ Tools: FITNESS, Cucumber/Gherkin, Raconteur, Concordian
- “ Of course, ATDD goes hand-in-hand with Unit TDD
- “ Unit TDD results in minimal code . %E functionality is an asset, Code is a liability+ (Kevlin Henney's Keynote)
- “ Evolving domain specific vocabulary common to all using Cucumber (BDD)
- “ Value of minimizing code transformation in TDD

# UTDD, BDD and ATDD

---



Unit tests due to their simplicity are low hanging fruits. We must strive to achieve maximum coverage using them.

# TDD - Transformation

---

1. In more than one sessions Bob Martin and Venkat Subramaniam demonstrated that TDD simplifies and improves design, reduces code and improves coverage.
2. Venkat demonstrated how testing thread safety forces developers to make their code testable and as a result abandon `%synchronized+` code blocks or methods. He uses much simpler and elegant way of implementing the ReentrantLock interfaces `lock()` and `unlock()` methods
3. Bob Martin proposed `%Transformation priority premise+`
  1. Transformation vs re-factoring
  2. Logical baby steps transform the code from specific to generic
  3. Examples . nil to constant, constant to scalar, scalar to array, array to a conditional `%if+`, `%if+` to `%while+` etc.
  4. Premise and the demonstration showed how following this sequence is good hygiene . It will prevent from getting you into a situation where you would be able to make any progress unless you undo many steps.

# TDD - Refactoring

---

- “ Version control history reveals a lot about health of the project and potentially throws up refactoring opportunities. This topic was presented by Michael Feathers.
- “ Complexity of check ins grows in a zig-zag manner
- “ File churn grows exponentially with the number of files going up
- “ Version control systems like Git maintain a complexity index for every source code file that is checked in. You can derive important information like-
  - What is the frequency of commits across the day?
  - What is the frequency of commits across the day per day of the week?
  - What is the average complexity per method?
  - Do new methods appear more frequently when large methods reduce in complexity?
  - What is the average complexity of commits across the day?
  - How has complexity been increasing in the project over time?
  - What is the number of committers over time?
  - What is person X's commit frequency?
  - When do methods 'jump the shark'?
  - Do most classes reach a steady state or is most update periodic?

# 2. Technical Debt

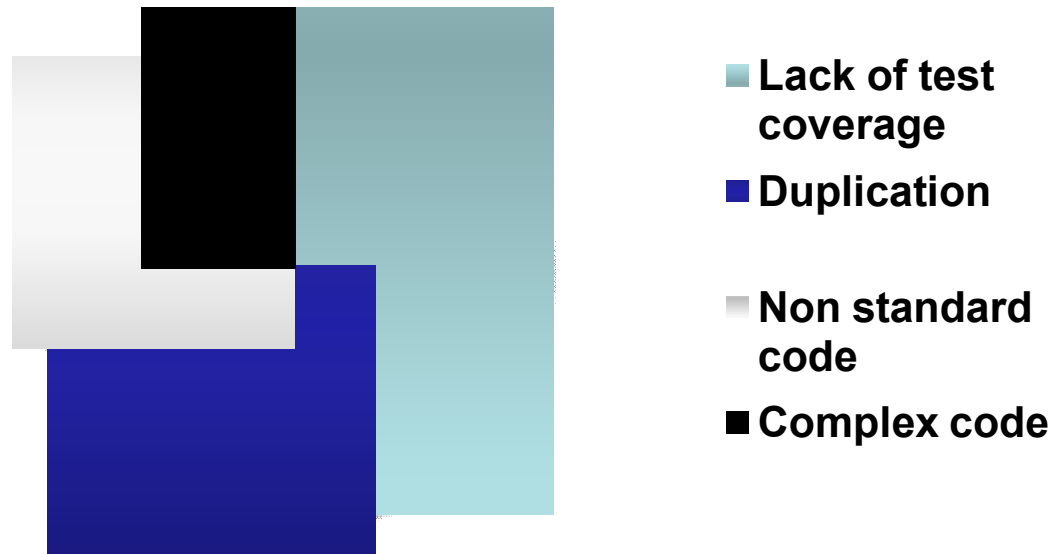
---

- “ The concept of Technical Debt is now widely understood and agreed to
- “ If not addressed, it compounds
  - . Makes it increasingly hard to add new features
  - . Results in high defect rate
- “ From Pradyumn Sharma's session
  - . Should be kept visible (explicitly) to stakeholders as part of product backlog
  - . ~~6+1~~ Rule: For every six iterations, have one refactoring iteration
- “ ~~Relentless~~ focus on managing technical debt+. Steve Green (Salesforce.com)
- “ Should be quantified in \$\$ = days of work x \$/day (Israel Gat)

# Technical Debt

---

“ Definition- Dollar amount worked out by multiplying the hourly rate of the development team multiplied by the hours it would take to fix the issues below

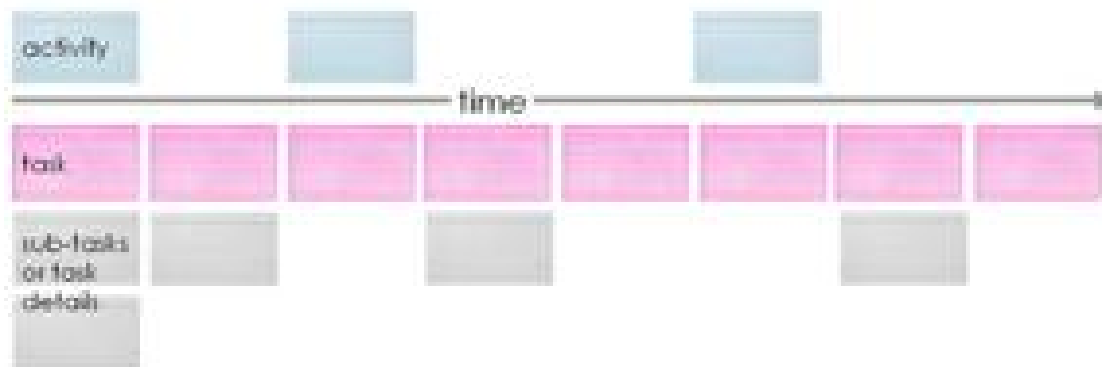


“ Technical debt like any debt accrues interest. If you decide to postpone the hardening iteration by a couple of iterations ; it would take more than one hardening iteration to fix the debt.

# 3. Requirements Elaboration

---

- “ Tools to capture and communicate user stories in the bigger picture
- “ David Hussman’s collaborative chartering technique helps teams to understand why they are building what they are building by defining the project charter in a template.
- “ Ellen Gottesdiener and Mary Gorman’s structured conversations approach helps to elicit requirements
  - “ Involve all partners (customer, technology business
  - “ Explore to expand and evaluate to narrow down options
  - “ Conversations structured around product functionality, users, actions, controls, data, quality and non functional features.
- “ Jeff Patton showed how story maps can be used to organize the backlog in a logical sequence. It also helps as a tool to break a release into iterations.



# 4. Lean/Kanban

---

- “ Kanban is now a mainstream Agile technique
- “ From Ron Jeffries & Chet Hendrickson’s session
  - . General agreement that Kanban’s single-piece flow is a much better way to manage work than iterations
  - . Kanban is better because it looks at the overall process, not just development
- “ %Scrum leaves out global optimization, which Lean/Kanban does well+(Alan Shalloway)
- “ Kanban’s (small) WIP limit brings focus to the bottleneck, encouraging pairing and team collaboration to address that bottleneck (David Bland, [www.kanban101.com](http://www.kanban101.com))
- “ Agile 2.0 = Kanban (Industry Analysts’s session)

# 5. Value Focus

---

- “ A lot of discussion on need to bring attention back to **value delivered** by Agile teams
- “ Lean/Kanban is better at keeping focus on Value, Agile/Scrum seem to get caught up in %~~of~~ delivering features faster+and often loses sight of value
- “ From Patrick Phillips session
  - . Value is defined by the %~~of~~ paying customer+
  - . Ask if the customer will pay for next iteration . STOP, if value of next iteration is less than cost
  - . Add %~~of~~ Value Points+to each story, use for prioritization (along with Story Points)

# 6. Continuous Delivery

---

- “ Continuous integration, deployment and delivery!
- “ Continuous delivery using mobile, cloud and social networking will take agile to the next level (Israel Gat)
- “ Ruby as a language was seen to be better suited for continuous integration, deployment and delivery (Julian Simpson)
- “ Should require at most one manual step to take build into production (Jeff Nielsen's 5 Numbers)

# 7. QA Automation

---

- “ Widely adopted, standard practice
- “ Overall theme of %taking pride in quality+
- “ Your customer's experience with your product is your real %brand+not your logo! (David Dalka)
- “ From Steve Green's session (Salesforce.com)
  - . Big executive commitment to QA automation
  - . Automated tests stay current with iteration release
  - . OK for teams to sign-up for fewer features/stories, but not compromise on quality
  - . %Stop the (code) line, if 1% or more of automated tests are not passing+!
- “ QA automation improves developer productivity by 20%
- “ Usually a separate QA automation infrastructure team
- “ %The most powerful tool in software+. Richard Sheridan

# 8. Pair Programming

---

- “ Widely adopted, and often a standard routine practice now
- “ Goes hand-in-hand with TDD
- “ TDD/Pairing helps get over our confirmation bias - %bugs are inherent in our brains+(Laurent Bossavit's session)
- “ From Richard Sheridan's session (Menlo Innovations)
  - . Periodic (every week) rotation of pairs
    - “ %Every team member needs to touch every story+
    - “ More collaboration and all around better team-work
  - . Removes dependence on any specific team member
  - . Allows breaking Brooks's Law (The Mythical Man-month)
  - . Results in better code, better quality, and ultimately better productivity
  - . %100% pair programming is the most powerful managerial tool ever+
- “ From Jeff Nielsen's session
  - . In a 5 person team, at least 3 should be able to explain details on any code section
  - . %90% of team members paired within last 2 days+

# 9. Sustainable Pace

---

- “ Teams work at sustainable pace of 40hr/week, no weekend, no missed vacation
- “ In fact, build some slack time for creativity, encourage team to think outside the box
- “ No tracking or hours, no timesheets!

# 10. Agile Games Incubator

---

- “ Games improve retention of principles learnt, they can be played in a group / team environment and they bring playfulness to learning or even collaborative decision-making required for estimation or prioritization.
- “ Don McGreal and Michael McCullough showed how if one does not find a game suitable to drive home your point/ principle; you can devise one yourself by systematically following a well defined process.
- “ Step 1- Define the problem around improving process / practice or value.
- “ Step 2- Identify the objectives that you would like to drive home
- “ Step 3- Decide on one of Emotional, Impressional or Physical type for the game you are about to devise.
- “ Step 4- Brainstorm and invent the game by being inspired, demonstrate courage while choosing options and %Keep it simple+
- “ Step 5- Debrief and learn to course correct and improve

# Other Salient Observations

---

1. Agile is not just a technique . it's a movement.
2. Many presentations across disciplines had useful concepts from organizational psychology , marketing and advertising.
3. Communities would be able to deliver better value to solve many problems. Our own ideas in open jam found resonance. Israel Gat also agreed that future of agile in the broader form would require a change in the present corporate structure
4. Using Personas is now a standard, widely adopted practice

# Other Salient Observations

---

5. Many presentations across disciplines had useful concepts from organizational psychology , marketing and advertising.
6. Continuous delivery using mobile, cloud and social networking will take agile to the next level (Israel Gat)
7. Ruby as a language was seen to be better suited for continuous integration, deployment and delivery (Julian Simpson)

# Typical Teams

---

- “ 1 Product Owner for 1 or 2 Agile teams
- “ 1 Scrum Master for 2 or 3 teams
- “ Dev-QA
  - . 4 Dev to 2 QA
  - . 3 Dev to 2 QA
  - . 4 Dev to 1 QA
- “ No separate %Architect+, team owns the architecture (one of the team members may act like an %Architect owner+to facilitate architecture decisions)
- “ QA Automation infrastructure
  - . Usually a separate team
  - . Sometimes developer jump in to accelerate progress
- “ One Product Manager over several Product Owners
  - . Product Manager is external (customer, market) facing, while Product Owner is inward (dev/QA team facing)
  - . Product Manager does global prioritization on features

# 11. Functional Org Structure

---

- “ Cross-functional Agile teams are prevalent in otherwise functional org structures
- “ Matrix organization
  - . Functional reporting/mgmt structure, e.g. Dev, QA, Technical Writers, etc.
  - . Cross functional Agile team by product/capability

# 12. Enterprise Agile

---

- “ Agile is now applied well outside of development organization
  - . Marketing, Tech Ops, IT, etc.
  - . Great case-study by [Salesforce.com](https://www.salesforce.com)
- “ Large scale Agile teams, 500+ member
  - . Majority are geographically distributed teams
  - . Enterprise Scaling Model by Scott Ambler of IBM

# Contact Information

---

” Hemant Elhence *(Dallas based)*

- . [hemant@synerzip.com](mailto:hemant@synerzip.com)
- . Cell Phone: 214.762.4873

” [www.synerzip.com](http://www.synerzip.com)

” HQ and US office in Dallas, TX

- . 14228 Midway Rd, #130, Dallas, TX 75244
- . Office Tel: 469.322.0349
- . Office Fax: 469.322.0490

” Development center in Pune, India.