Agile Methodologies for Software Development

Dr. Fritz Solms

May 4, 2014
Agile methodologies

"Reaction to bureaucracy of engineering methodologies" (Fowler).

- Started in 1990’s: Kent Beck, Ron Jeffries, Martin Fowler, ...

- Emphasis on
  - reaction to change,
  - agility,
  - operate in continuously changing requirements.

- More people than process centric.
  - Let people themselves decide how best they can contribute.
Traditional versus Agile Methods

**Waterfall**
- Aims to solidly understand requirements.
- Based on
  - Solid planning and risk management
  - Controlled engineering processes to build solution
- Assume
  - Problem well defined.
  - Predictable, repeatable, measurable engineering processes
  - *Process centric and systems engineering thinking*
  - Hierarchical management and control, responsibility assignment across teams with defined responsibilities
- *Customer:* only required for requirements and acceptance.

**Agile methods**
- Based on
  - Complex Adaptive Systems Theory
  - Adaption based on empirical information from short delivery feedbacks
  - Self-organizing teams
- *Customer:* Continuous collaboration & feedback
Agile methodologies more successful than waterfall

Origin of the agile manifesto

- Aimed to define common values and principles across agile methodologies.
Agile Values

- We value
  1. Individuals and interactions over processes and tools.
  2. Working software over comprehensive documentation.
  3. Customer collaboration over contract negotiation.
  4. Responding to change over following a plan.

- That is,
  while there is value in the items on the right,
  we value the items on the left more.
Agile Principles - 1

- **Highest priority:**
  - satisfaction via early and continuous delivery of valuable software.

- **Welcome changing requirements**
  - Provides customer competitive advantage.

- **Deliver working software frequently**
  - preferably every couple of weeks

- **Business people & developers work together throughout project.**

- **Build projects around motivated individuals.**
  - Provide environment, tools and trust them to get job done.

- **Most efficient & effective communication: face-to-face**

- **Working software = primary measure of progress.**
Agile Principles - 2

- Agile processes promote sustainable development.
  - Sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence & good design enhances agility.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- Best architectures, requirements, & designs emerge from self-organizing teams.
- At regular intervals:
  - team reflects on how to become more effective,
  - then tunes and adjusts its behavior accordingly.
Common Agile Concepts

- **User story**
  - A testable feature which provides some value to client.

- **PSI**
  - Potential shippable increment.

- **Velocity**
  - Productivity unit used by agile methodologies.
  - $velocity = \frac{\text{units of work}}{\text{unit time}}$
  - Units of work typically in story points (light-weight features).
Origins of Extreme Programming (XP)

- Early agile method resisting heavy processes
  - Kent Beck - 1996
- Designed for continuously changing requirements.
  - Business: new opportunities/new ideas on how to generate value.
  - Process effectively operating in context of continuously changing requirements can provide better business value.
- Business cycle has shortened in context of globalization & increased competition.
  - Require light-weight methodology with short delivery cycles.
XP values

1 Communication
   - Co-locate client & development team.
   - Simple, person-person communication channels
      - instead of extensive documentation.
   - Observable process:
      - Customer has real-time access to produced artifacts.

2 Simplicity
   - Anything which can be simplified by anyone, should be.
   - Continuous refactoring.

3 Feedback
   - Short feedback channels right through to client.
   - Short iterations.
   - Deliver regularly to production.

4 Courage
   - Do what you believe in.
   - Be open & honest about progress & problems.

5 Respect people
   - Trust client to understand their changing needs.
   - Trust developers and their competencies.
   - Let them select where best to contribute.
Process trade-offs

- 4 process variables
  - Scope
  - Cost
  - Time
  - Quality
- Changing one requires changing at least one other.
- XP:
  - Time & quality are sacred
    - always deliver quality on time.
  - Sacrifice scope & cost if necessary.
XP Activities

- Extreme programmers engage only in
  - **Listening**
    - on-site customer
    - face-to-face communication
  - **Designing**
    - lightweight, rapid
    - design sketches (typically not fed into bigger model).
  - **Coding**
    - Code is only true measure.
    - Code is documentation.
  - **Testing**
    - Develop tests first.
    - Formalization of requirement component.
    - Effectively contract.
    - Then develop implementation to pass test.
XP life cycle

- Iterative with each iteration
  - Realizes a user story
    - Realizable within 2-3 weeks.
    - Understandable & testable by customer.
    - Adds some measurable business value.
    - Subset of full solution.
    - Commonly one or more use cases.
XP Phases

1. Exploration
   - Define user story.
   - Re-assess feasibility.

2. Planning
   - Identify engineering challenges.
   - Team brainstorms engineering tasks.
   - Developers select & sign-up for work units.

3. Iterations to 1’st release
   - Write/extend unit tests for features.
   - Update component to pass updated tests.

4. Productionizing
   - Integration testing.
   - User testing.
   - Release into production.

5. Death
   - Controlled process for removing system from production
12 XP practices - 1

- **Planning**
  - Team brainstorms engineering challenges.
  - Developers select & sign up for work units.

- **On-site customer**
  - Requirements = 80% of cost of errors (Standish reports)
  - On-site customer simplifies communication channel.

- **Simple design**
  - Break up into components if too complex.

- **Small releases**
  - Simple designs & tests, short feedback cycles.

- **Metaphor**
  - Find conceptual equivalents for difficult concepts.

- **Testing**
  - Tests defined by on-site customer.
  - Write tests first.
12 XP practices - 2

- **Refactoring**
  - Complexity is the enemy.
  - Only continuous refactoring will prevent drowning in complexity.

- **Pair programming**
  - Two developers per PC.
  - Oversight, knowledge distribution, risk reduction, more fun, higher productivity.

- **Collective ownership**
  - Anybody can improve anything.

- **Continuous integration**
  - To prevent high cost of unexpected integration problems.

- **40 hour week**
  - Longer hours ⇒ errors & demotivation.
  - Methodology should have prevented crisis from happening.

- **Coding standards**
  - Required for effective collective ownership.
Strengths of XP

- Focus on testing.
- Mutual learning – continuous learning.
- Very short feedback cycles and good project tracking.
- Flexible process able to address changing requirements in controlled way.
- Continuous delivery.
Criticism of XP

1. Where is high level architecture
   - Hard to address non-functional requirements.

2. Where is global scope & costing addressed?

3. Pair programming not always higher velocity.

4. Can be used to “bleed money off customers”

5. Short iterations can introduce high overheads - particularly for customer.
The Scrum methodology

- **Takeuchi & Nonaka - 1986**
  - Studied commercial software development projects.
  - Identified approaches which assist flexibility & speed.
  - **⇒ Scrum**
    - From rugby - scrum restarts process after incident.
    - Polymorphic, cross-functional teams.
    - Multiple, overlapping phases.

- **Process variables:**
  - Time, quality fixed.
  - Scope variable through backlog buffers.

- **Strong emphasis on continuous measurement.**
- **Explicitly manages collaboration/dependencies across teams.**
Overview of Scrum Process

Sprint

Product Owner

Scrum Master

Scrum Team

Daily Scrum (Standup) (15 mins.)

Analysis, Design, Build

Testing

Deployment

Potentially Shippable (working increment of Software)

Sprint Retrospective (1-3 hrs.)

Sprint Review (2-4 hrs.)

Product Backlog

Sprint Backlog

Initial Planning

Sprint Planning (1 day)

Copyright © 2011, William B. Heys
Scrum Values

- *Scrum* embraces general agile values
  - As in agile manifesto
Scrum-Specific Values

- **Focus**
  - Focus on specific functionality/problem and solve that only.
  - Scrum master removing distractions.

- **Openness**
  - Expose everything to team, management and client.
  - Open own contributions to be improved by others.
  - Through openness build *trust*.

- **Commitment**
  - Commit to professionalism, the team, to learning, openness, …
  - Commitment to challenges.
    - because honest failure is not punished.

- **Respect**
  - Einstein: *I speak to everyone in the same way, whether he is the garbage man or the president of the university*
  - Respect team members for their skills & insights.
  - Respect clients to know what they want.
  - Respect brings out the best in people.

- **Courage**
  - Do what you believe in.
  - Courage to be open with problems and mistakes.
  - Scrum Master needs courage to stand up to management.
Scrum Roles

Core roles:

- **Product Owner**
  - Decides what work has to be done.
- **The Team**
  - Build the product.
- **The Scrum Master**
  - Servant leader assisting team to build product.

External Roles:

- **Management**
  - Make high-level decisions
- **Stakeholders**
  - Client, users, shareholders, ...
Product Owner

- Customer proxy
  - Liases between *product manager* & *scrum team*.
- Responsibilities:
  1. Communicate project vision to team.
  2. Feeding requirements into product backlog.
     - Detailed user stories can be developed by BA, but overseen by product owner.
     - Continuously answer questions around detailed requirements.
  3. Prioritization of backlog items.
  4. Definition of releases (PSIs: Potential Shippable Increments).
  5. Accept/reject implemented user stories.
  6. Monitor that team generates business value.
The Team

- **Responsibility:** *building & deliver product*
  - Detailed analysis
  - Software architecture,
  - Application design
  - Implementation,
  - Testing,
  - Technical documentation.

- **Team composition**
  - Small (3-9) & self-organizing.
  - Polymorphic (complementing skills).
Scrum Master

- Responsible for enabling team to effectively deliver product.

**Assist team**
- Coaches team in scrum.
- Isolates team from signals which would reduce velocity.
- Ensure team has resources required to do work.
- Support product owner.

**Process Management**
- Process customization.
- Ensures team adheres to process.
- Facilitate scrum meetings
  - Sprint planning meeting
  - Daily scrum
  - Sprint review
  - Sprint retrospective

**Interface to higher-level coordination**
- Represent team in *scrum of scrums*
- Communication channel to management.
# Product Owners & Product Managers

<table>
<thead>
<tr>
<th><strong>Product Manager</strong></th>
<th><strong>Product Owner</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer’s product champion</td>
<td>Customer proxy in scrum team</td>
</tr>
<tr>
<td>Vision &amp; ultimate requirements source</td>
<td>Manages product backlog</td>
</tr>
<tr>
<td>Responsible for return on investment</td>
<td>Responsible for team performance</td>
</tr>
<tr>
<td>Manages product release</td>
<td>Manages increment release</td>
</tr>
</tbody>
</table>
Project Management in Scrum

- **Traditional project management responsibilities re-distributed:**
  - Estimation → Team
    - Includes estimation of risk.
  - Task-assignment → individual team members.
    - at lower level to team members themselves.
  - Risk management and mitigation → Team
  - Scope & schedule trade-offs → Product Owner
  - Quality Control → Product Owner & Team

- **Still higher-level project management:**
  - Project resourcing across scrum teams.
  - Coordination and synchronization across teams
    - Scrum of scrums.
Scrum Concepts

- **Sprint**
  - Basic unit of development around a user story.

- **Spike**
  - A time-boxed period to research a concept or technology.

- **Tracer bullet**
  - A test development in current architecture & technology to proof something.

- **Sashimi**
  - A report that something is done
    - Usually complying to some prescribed format.

- **Pig and Chicken**
  - the person who is directly/somewhat responsible for a task.
    - from “*bacon is on the line*”
    - like *bacon & egg* (committed/involved)
Overview of Scrum Process

- Sprint Planning (1 day)
- Scrum Master
- Scrum Team
- Product Owner
- Product Backlog
- Initial Planning
- Sprint Backlog

- Daily Scrum (Standup) (15 mins.)
- Analysis, Design, Build
- Testing
- Deployment
- Potentially Shippable (working increment of) Software

Sprint

2 - 4 Weeks

- Sprint Review (2-4 hrs.)
- Sprint Retrospective (1-3 hrs.)
Features of Scrum Process

- **Iterative process**
  - Sprints & scrums.

- **Sprint:**
  - 1 iteration.
  - 7-30 days.
  - Delivers testable, deployable subset of final product.

- **Scrum:**
  - Daily cycle.
  - Fine grained iteration.
  - Measurement on both, sprint & scrum levels.
  - Maintain product & sprint backlogs.
    - Outstanding work for product & iteration respectively.
Sprint Planning Meeting

- At start of sprint
  - \( \leq 8 \) hours
    - Typically: sprint-length in weeks \( \times 2 \) hours
- 1'\text{st} \text{ half: Product owner + Team}
  - Focus on requirements for sprint.
- 2'\text{nd} \text{ half: Team only}
  - Focus on work break-down, technical challenges and estimation.
Overview of Sprint Planning Meeting

- Team capacity
- Marketing backlog
- Business priorities
- Product Status
- Competition

Sprint planning meeting:

Sprint prioritization:
- Select sprint goal
- Analyze and evaluate marketing backlog

Sprint planning:
- Decide how to achieve sprint goal
- Create sprint backlog (tasks) from marketing backlog items (user stories / marketing tasks)
- Estimate sprint backlog in hours
Sprint Planning Meeting: Part 1

- **Participants**
  - Product owner + Team

- **Review product backlog**
  - Revisit prioritization of sprint backlog

- **Select scope of iteration**
  - Feed from product backlog onto sprint backlog
  - Prioritization of sprint backlog
  - Team has final say on scope of sprint
    - Product owner cannot force scope on team

- **Output**: Sprint goal
  - One-sentence description of outcome of sprint
  - Rough estimation to bound sprint scope
Sprint Planning Meeting: Part 2

- **Participants:**
  - Team only
  - Product owner not present in room, but available.

- **Plan work**
  - Decompose backlog items into work tasks.
  - Discuss ways to implement features.
  - Estimate work tasks (in hours).

- **Output:**
  - Detailed sprint backlog.
Daily Scrum Meeting

- Same time & location, < 15 minutes.
  - Stand-up meeting.
  - Facilitated by scrum master.

- 3 Questions discussed by each member:
  - Work I have done during last scrum (day).
  - Work I will be doing in this scrum.
  - Obstacles impeding my progress.
    - Team assists with exploring solutions to obstacles.
Stand-up Meeting
Sprint Review

- **Participants:**
  - Product owner
  - Team (including Scrum Master)
  - Any involved person is welcome to attend.
    - Stakeholders, CEO,
    - Sprint review = part of scrum
    - Anyone may ask questions

- **Duration:** Sprint length in weeks $\times$ 30 minutes

- **Team presents completed work to product owner**
  - Must be executable demo.
  - Present only completed work.
  - Product owner assesses against acceptance criteria.
    - Only fully completed work accepted.

- **Discussion of planned work not completed.**
  - Newly discovered work items added to backlog.
  - Product owner may discuss features required next.

- **Calculate project metrics**
  - Code coverage
  - Sprint velocity
General Guidelines for Sprint Review

- Must be informal meeting
  - No prepared slides.
  - Preparation $\approx$ half the duration of the meeting.
  - Focus: Demo of shippable product increment.
- Completed work
  - Implemented in production quality
  - Tested
  - Integrated
  - Documented
- Only production quality (shippable) code
  - No prototypes.
  - No short-cuts (e.g. hard-coded results/environmental aspects)
- Calculate velocity
  - against only fully completed work
Sprint Retrospective

- On last day of sprint
  - Duration: Sprint length in weeks × 45 minutes
- Participants
  - Only team (with scrum master)
  - Product owner not present.
    - Facilitates more frank discussion.
- Team members reflect on past sprint.
  - What worked well in sprint?
  - What could be improved?
- Continuous process improvement
  - Facilitated by scrum master.
  - Scrum master tries to identify impediments that impact team
    - Tries to find ways to remove them or reduce their impact.
Scaling Scrum
Scrum of Scrums

Meeting across multiple scrum teams.

- One representative of each scrum team.
- Briefed on individual scrum meetings:
  - Work done during last scrum (day).
  - Work to be done in this scrum.
  - Obstacles encountered.
- Plus discuss anything your team is doing which could affect other teams.
Overview of Scrum Artifacts

- Scrum is light on artifacts.
  - Covers only project management artifacts.
  - Focus on delivery.

- The artifacts:
  - The *Product backlog*
  - The *Sprint backlog*
  - The *increment*
  - *Burndown chart*
Product Backlog

- Ordered list of requirements/features
  - ordered on business value, date needed, risk, dependencies, …
    - Product owner ultimately responsible for ordering

- Feature/Requirement
  - written as “story”
  - Includes estimated effort to complete
    - Estimation done by development team
Sprint Backlog

- Subset of product backlog
  - Only those features to be delivered in current sprint.
  - Determined during sprint planning meeting
    - Typically choose as many features from top of product backlog to fill up sprint duration.

- Estimates refined during sprint planning meeting

- Stories broken down into smaller tasks
  - Each task should take between 4 & 16 hours.
  - Team members sign up for tasks.
Increment

- List of all features/requirements completed in sprint.
  - Each “completed feature” must meet completion criteria:
    - Completion criteria agreed upon with product owner.

- The outputs of the sprint must be
  - testable
  - installable
  - demonstratable
  - shippable
  - usable
Scrum Task Board: simple
Scrum Task Board
Scrum Task Board: sophisticated
Scrum Task Board: With annotations
Burn-Down Chart

Publically visible, continuously updated
Aim:

- To capture simple, meaningful metrics.

Metrics grouped into

1. Team metrics:
   - Internal assessment.
   - To assess team’s/sprint’s strengths and weaknesses.

2. Customer metrics:
   - Assess customer’s perception/assessment of success.
Scrum Team Metrics

Scrum Team Sprint Metrics

- Actual Stories Completed vs. Committed Stories
- Technical Debt Management
- Team Velocity
- Quality Delivered to Customer
- Team Enthusiasm
- Retrospective Process Improvement
- Scrum Team Communication
- Scrum Team’s Adherence to Scrum Rules & Engineering Practices
- Scrum Team’s Understanding of Sprint Scope and Goal

Sprint 1
Sprint 2
Sprint 3
Sprint 4
Sprint 5
Scrum Customer Metrics

Customer Satisfaction Survey-Sprint Review

- Did the sprint result in added value to the product?
- Overall satisfaction with the sprint
- Was each story acceptable?
- Do you feel anything new needs to be added to the release?
- Did you feel your opinions and comments were listened to?
- Was the new functionality as expected?
- Do you think you’ll be using the new functionality?
Encourages team work and transparency.
Joint effort on solving engineering challenges.
Very short feedback cycles and good project tracking.
Flexible process able to address changing requirements in controlled way.
Continuous delivery.
Low management overhead costs.
Criticism of scrum

1. Where is high level architecture, scope & costing addressed?

2. Features not completed fed into product backlog.
   - Avoid accountability.
   - Backlog may grow and affect high level costing in big way.

3. Team can avoid difficult features, leaving them in backlog until late.
   - From engineering perspective its good to tackle difficult features first.
   - Have bigger impact on overall architecture & design.

4. Works only well with co-location of clients and developers.
Agile estimation

Full requirements not known up-front.

Need

- Light-weight estimation for sprints.
- May also need rough estimate of
  - Project as a whole (including project risk).

Estimation commonly variations of *Wide-Band Delphi*

- Done by people who do the work.
- Includes consensus building and knowledge distribution.

If too many unknowns

- Use spikes.

Two variants:

1. **Planning Poker**
   - For rapid sprint estimation and requirements consensus building

2. **Probabilistic Wide Band Delphi**
   - For project as a whole.
Planning Poker

- Simple variation on *Wide-Band Delphi*
  - James Grenning (2002)
  - Popularized in Mike Cohn’s book *Agile Estimating and Planning*

- **Participants:**
  - Moderator
    - chairs session, but does not participate
  - Product owner
    - provides requirements in form of user stories
  - Team member = estimators.

- Estimates played by cards
  - Not spoken out loud.
    - Less likely to influence other team members.

- Discussion around estimates in order to
  - build common understanding,

- Team re-estimates
  - until sufficient consensus.
Planning Poker: Equipment

1. Feature list
   - Obtained from work-breakdown or user stories.

2. Physical or virtual cards (e.g. using Planning Poker Android App)
   - Fibonacci series
     - 0, 1, 2, 3, 5, 8, 13, 20, 40, 100
     - Alternatively T-shirt sizes: XS, XS, S, M, L, XL, XXL
     - Represent some units (e.g. days, some base-lining)
   - ? → “I need more information from the product owner.”
   - Coffee cup → “I need a break”
   - ∞ → user story is much too large & must be split up
   - Joker → “I do not want to estimate this user story.”

3. A timer (egg timer, timer app, …)
   - To limit length of discussions.

4. For distributed teams
   - Collaborative planning poker applications.
Planning Poker: The Process

1. **Product owner provides short overview of user story.**
   - Team members can ask questions

2. **Team members estimate**
   - Select card for estimate.
   - Simultaneously reveal estimate.
   - Largest & smallest identified.

3. **Discussions**
   - Smallest & largest explained by respective estimators.
   - Discussion around this.

4. **Re-estimate**
   - Team re-estimates
   - Until sufficient consensus achieved.

5. **Process repeated for other user stories assigned to sprint/iteration.**
Guidelines for Planning Poker

- Only people doing the work may vote.
  - Managers don’t vote.
- Limit discussions between re-estimates
  - use timer
- If tie on votes for consecutive cards
  - Choose larger vote and move on.
- If user story gets high estimates
  - break it up into smaller user stories/features.
- Consider using more abstract estimates
  - e.g. T-shirt sizes
  - baseline these to time units.
  - encourages estimation by analog.
- Keep it fun.
Benefits of Planning Poker

- Prevents anchoring.
  - Influences from people who are senior or have agendas.
- Spreads common understanding.
- Exposes complexities.
- Deepens understanding of requirements.
- Abstract units (e.g. T-shirt sizes) encourages estimation by analog.
Spikes

Definition

- A spike is a story or task aimed at answering a question or gathering information, rather than at producing shippable product.
  - A unit of work representing a *proof of concept*.
  - Often thrown away at end.
When is a Spike Needed?

- The user story estimates excessively uncertain or large.
  - No consensus could be obtained.
- There are significant technical concerns
  - Architectural concerns.
    - Non-functional requirements not met,
    - Scalability, reliability, performance, security, auditability, usability, integrability, modifiability, installability, portability, . . .
  - Design complexities.
    - System needs some re-factoring
  - Unknown technologies.
- Technical debt slows down progress.
- The team feels increasingly uncomfortable with the work done.
- Certain items are consistently pushed back onto the backlog.
Spike requirements

- Spikes must have a clear objective/outcome.
  - Don’t let the spike drift randomly into areas which interest the team.
- Spikes must be time-boxed.
  - They do not need to be estimated.
  - It’s about reserving some time for internal work.
    - Deepen understanding
    - Learning
    - Solve some technical problem.
- If burn-down chart i.t.o. story points
  - do not include spikes.
  - High-level estimation must keep room for spikes.
What is Wide-Band Delphi estimation?

- Consensus building framework for expert-based estimation.
  - Experts estimate independently and secretly.
  - Controlled information distribution across experts.
Why extend Wide-Band Delphi?

- Normal wide-band delphi does not yield probability distribution.
  - No assessment around uncertainties of estimates.
  - Little information about risk.
  - Uncertainty affects pricing
    - Need to prize not only work, but
    - also risk.
How is Wide-Band Delphi extended?

- Don’t ask each estimator for single estimate.
- Ask for min, expected, max.
  - Estimator would typically anyway consider these.
  - Will make estimator more comfortable.
  - Allows us to use Pert estimation to generate probability distribution for estimator.
  - ⇒ panel-wide probability distribution.
  - ⇒ estimate of cost/effort + estimate of risk.
The Probabilistic Wide-Band Delphi Method

Kickoff Meeting
- Discuss core responsibilities to be addressed, work breakdown, and core project risks.
- Discuss resources available for project with project management.
- Provide estimation check list to experts and provide opportunity to discuss it.

Delphi Estimation Meeting
- Each expert presents their estimate (minimum, expected and maximum) without justification.
- Experts process information and assess initial estimates.
- Panel discussion on this, expert with widest deviation from mean is asked to explain how they came to that assessment.
- Average estimate is calculated and communicated to panel.
- Significant differences between expert estimates?
- Estimates agree within acceptable error range typically around 10-20%.

Use Pert estimation to assess probability distributions for individual tasks and standard deviation of panel’s probability distribution for entire project estimate.

Capture minimum, expected & maximum estimate for individual tasks and mean and standard deviation of panel’s probability distribution for individual tasks and for project as a whole.
Pert estimation

- Have for estimator $i$
  - minimum value: $\epsilon_{m,i}$
  - expected value: $\epsilon_i$, and
  - maximum value: $\epsilon_{M,i}$

- Assume normal distributions: $P(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

- Use Pert estimator to calculate
  - mean for estimator $i$: $\mu_i = (\epsilon_{m,i} + 4\epsilon_i + \epsilon_{M,i})/6$
  - std deviation for estimator $i$: $\sigma_i = (\epsilon_{M,i} - \epsilon_{m,i})/6$

- Calculate panel-wide expected value across the $N$ estimators and communicate to team:
  $$\bar{\epsilon} = \frac{1}{N} \sum_{i} \epsilon_i$$

- Distribution of sum of normal distributions is also normal.

- Giving estimators different weights, we obtain:
  - mean of panel-wide probability distribution: $\mu = \sum_i w_i \mu_i$
  - std deviation for panel-wide distribution: $\sigma = \sqrt{\sum_i w_i^2 \sigma_i^2}$
What is Testing?

- *Processes which establish that system satisfies requirements specification.*
  - Many of these processes are usually automated.
  - Proof existence of quality issue (error).
  - Do not include error location as part of testing.
    - Testing team generally not in position to locate errors.
    - Would result in disincentive to find more errors.
    - Testing done against contracts, not specific implementations.
Faults versus Failures

- **Fault** = error which needs to be fixed.
  - Fault may cause failure.
  - May be handled to avoid failure.
  - May be handled in context of *fault-tolerant systems*.

- **Failure** = inability of system to meet requirements specification.
Purpose of Testing

- Proof to customer that system satisfies
  - requirements specification and
  - user expectations.
- Provide general assessment of system quality
  - which is preferably based on some defined measures.
- Make systems flexible.
- Allow multiple people to work on same code.
What is System Tested Against?

- Requirements specification
  - Formalization: Contract
    - Functional requirements.
    - Non-functional requirements.
Unit Testing

- Test functionality of unit.
  - Test unit assuming environment behaves according to contracts.
  - Test units across levels of granularity.
- Unit =
  - class or function/service,
  - component or service contract.
- Preferably test against contracts:
  - Any pluggable service provider tested against contract.
  - If all pre-conditions are met
    - service provided (no exception or error thrown)
    - all post-conditions hold true after service provided.
    - may have to formulate intermediate activities as observations of these as post-conditions.
  - **mock-out any lower level service providers or components.**
    - Wide variety of mocking frameworks available.
Integration Testing

- Test component in its environment.
  - DO not use mocking.
  - Test across levels of granularity.
Regression Testing

- After changes re-run tests
  - Unit tests
  - Integration tests
  - Possibly non-functional tests
Non-Functional Testing

- **Performance testing**
  - how long does it takes to complete a task?

- **Scalability testing**
  - How many tarequests can be processed per unit time?
  - How many concurrent users?
  - Database load testing

- **Reliability testing**
  - Impact of single point of failure.
  - Impact of a single process failure.

- **Security testing**
  - Can service/resource be accessed by non-authorized users?
  - Can information in transit be accessed
  - System security in failure mode?

- **Flexibility testing**
  - What can I change without having to change code?
  - How much code needs to be changed in case of typical requirements changes?
  - Integration flexibility, Portability, ...
**Test Data Generation**

- **Equivalence partitions (black box)**
  - Partition input/environment/output space into partitions for which behaviour is equivalent.
  - E.g. withdraw amount from account
    - 2 partitions (more or less than available funds)
    - Pick one test data from each partition.

- **Boundary value analysis (black box)**
  - Find extreme/boundary points in input/environment space.
  - E.g. withdraw amount from account:
    - withdraw zero
    - withdraw exactly available funds

- **Path analysis (white box)**
  - Analyze logic to determine all paths.
  - For each path find input/environment data which results in path traversal.
Acceptance Testing

- users test whether
  - user happy to accept system, and
  - meets user requirements.
Configuration Testing

- Sometimes system released in different configurations.
- Each configuration needs to be tested.
Adapter Testing

- Test that adapter to external party functions correctly
- Includes
  - System adapters
    - Protocol marshalling/demarshalling
    - quality requirements (scalability, performance, reliability, security, ...)
  - Human adapters
    - GUIs and Web Front Ends
    - E.g. usability and efficiency
Agile Methodologies and Software Architecture

- One of the biggest criticisms of agile
  - Software architecture does not get sufficient attention.
  - Architecture not well handled within short iterations/sprints.

- Tic-tac-toe example from Agile proponent
  - We all know tic-tac-toe.
  - Easy to put architecture in place.
  - Start building.
  - Later find out that
    - System must be globally available to 20 million concurrent users.
    - Security requirements.
    - Reliability requirements, . . . .

- Suddenly architecture falls apart, but
  - We never had architectural requirements.
  - Only functional requirements.
What is Software Architecture?

Definition

Software architecture is the specification of the software infrastructure addressing non-functional requirements and hosting application components addressing functional requirements.

- Difference between architecture and application design
  - Architecture design addresses non-functional requirements.
    - many are cross-cutting concerns.
    - More in the technical domain.
  - Application design addresses functional requirements.
    - More in the problem domain.
Elements of software architecture

- Architectural components addressing infrastructural concerns.
- Infrastructural constraints in the form of architectural patterns or styles.
  - e.g. layering, microkernel, blackboard, pipes and filters, . . .
- Tactics or strategies through which non-functional requirements are concretely realized.
  - e.g. clustering, load-balancing, queueing, pre-processing, connection and thread pooling, interception, . . .
- Concepts and constraints for application components
  - e.g. stateless services, stateful components, functions (as in functional programming), entities, forms, . . .
Examples of architectural decisions

- Whether to use restful or SOAP-based web services for integration.
- How authentication and authorization is handled in application deployed in the architecture.
- Whether an in-memory cache is used and how it is handled.
- Whether to use a relational or a No-SQL database.
- The logging infrastructure.
- Whether dependency injection, run-time lookup, or compile-time binding is used.
- Whether to integrate via a message queue or an FT server.
- Whether to use a Java-EE application server or an Enterprise Services Bus.
Examples of Application Design

- How compound interest is calculated.
- How an order is processed.
- The design of the data structure of an invoice.
- The contents of a form through which data is captured.
How is Software ArchitectureHandled in Agile?

- Little differentiation between architecture and application design.
- Largely absorbed within Iterations and sprints.
  - Sprints/spikes during which non-functional requirements are addressed.
  - No up-front engineering phase.
- Not uncommon for agile projects to end up with
  - architectural deficiencies.
  - technical debt.
Adding an Architecture Phase

- Alternative hybrid approach
  - Elicit architectural requirements.
    - Architectural responsibilities
    - Quality requirements
    - Integration requirements
    - Architectural constraints
  - Introduce architecture engineering phase
    - Design software architecture
    - Select technologies and frameworks
  - Build applications addressing functional requirements using Agile Process
    - Iteratively delivering shippable product iterations.
Tools for Agile

- **Version control**
  - Concurrent version control over lock-modify-unlock
  - Distributed over centralized
  - e.g. git.

- **Building**
  - with integrated dependency management, testing, project documentation generation
  - e.g. Maven, nmakem rake

- **Testing**
  - junit, nunit, scalability testing tools, Jdots

- **Continuous integration**
  - Hudson, Apache Gump or Continuum,

- **Scrum tools**
  - Range of open source, online and proprietary tools supporting scrum.