Domain VI Problem Detection and Resolution

The PMI-ACP Exam consists of 120 questions which can be categorised into seven domains. The Sixth domain: Domain VI Problem Detection and Resolution is the knowledge about "Continuously identifying problems, impediments, and risks; prioritizing and resolving in a timely manner; monitoring and communicating the problem resolution status; and implementing process improvements to prevent them from occurring again." (source: PMI-ACP Examination Content Outline).
Domain VI Problem Detection and Resolution accounts for 10% of all questions in the PMI-ACP Exam (i.e. ~12 questions among 120 PMI-ACP Exam questions)

According to the PMI-ACP Exam Content Outline, Domain VI Problem Detection and Resolution consists of 5 tasks:

1. **Encourage experimentation and communication** in order to discover problems or impediments that prevent maximal value delivery.
2. Identify and resolve issues and threats timely by **engaging the whole team**.
3. Issues should be **resolved by appropriate team member(s)**. In the case the issue cannot be resolved, the team should communicate with appropriate stakeholders to adjust project expectations and priorities.
4. Keep a **prioritized list of issues/threats/risks** to track assignment/ownership and the current status and to provide transparency.
5. Incorporate **resolution activities into task backlog** for future planning.

Below is a collection of the key knowledge addressed in Domain VI Problem Detection and Resolution and the 5 tasks related to the domain:

Risk / Threat Management:

- Risk is uncertainty that could affect the success/failure of the project. Risks become **problems or issues** once they actually occur.
- Risks can be threats or opportunities, negative project risks are considered as "**anti-value**".
- In order to maximize values, negative risks must be minimized while positive risks should be utilized. But once problems or issues arise, they must be resolved timely in order to reduce effects on value creation.
- Risk identification should involve the customer, project team and all relevant stakeholders
- **Five Core Risks** mentioned in the book "The Software Project Manager's Bridge to Agility"
  - **productivity variation** (difference between planned and actual performance)
  - **scope creep** (considerable additional requirements beyond initial agreement)
  - **specification breakdown** (lack of stakeholder consensus on requirements)
  - **intrinsic schedule flaw** (poor estimates of task durations)
  - **personnel loss** (the loss of human resources)
- Risks are assessed by **risk probability** (how likely it occurs) and **risk impact** (how severe the risk impact is):
  - **Risk Severity = Risk Probability x Risk Impact**
  - Risk probability can be a percentage value or a number on a relative scale (the higher the more likely)
  - Risk impact can be dollar value or a number on a relative scale (the higher the more costly to fix)
- As a general rule, "**riskier" features** (with high values) should be tested in **earlier sprints** to allow the project to "**fail fast**" as failing during the earlier phrase of the project is much less costly than failing during a later phrase
- Risk is high at the beginning of the project (both for traditional and Agile projects) but Agile projects have higher success rates as the very nature of Agile project management tends to reduce risks as changes are inherent to the projects
- Risk can be categorized into the following:
  - **Business –** related to business value

- **Technical –** about technology use and/or skill sets
- **Logistic –** related to schedule, funding, staffing, etc.
- **Others –** Political, Environmental, Societal, Technological, Legal or Economic (PESTLE)

- To tackle risks: **Identify Risks -> Assess Qualitatively and Quantitatively -> Plan Response -> Carry Out Responses Should Risks Arise -> Control and Review**
- **Risk Adjusted Backlog**
  - Prioritization criteria for backlog: **value, knowledge, uncertainty, risk**
  - Backlog can be re-prioritized by customers as needed to reduce risks while still realizing values
    - The customer can give each feature / risk response actions (non-functional requirements) on the backlog a value by, for features, assessing ROV and, for the case of risks, the costs involved (by multiplying the probability of the risk in %)
    - The backlog of features and risk response activities can then be prioritized based on the dollar values
  - Risk adjustment tries to identify and mitigate the risk at an early stage of development
  - 'Fail fast' allows the team to learn and adjust course
- **Risk Burn Down Graphs / Charts**
  - to show the risk exposure of the project
  - created by plotting the sum of the agreed risk exposure values **(impact x probability)** against iterations
  - to be updated regularly (per iteration) to reflect the change in risk exposure
  - general recommendation: top 10 risks are included
  - the risk burn down chart should have the total risks heading down while the project progresses
- **Risk-based Burn Up Chart**
  - tracks the targeted and actual product delivery progress
  - includes estimates of how likely the team is to achieve targeted value adjusted for risk by showing the optimistic, most likely and the worst-case scenario
- **Risk-based Spike**
  - **Spike:** a rapid time-boxed experiment (by the developers) to learn just enough about the "unknown" of a user story for estimation / establishing realistic expectations / as a proof of concept
  - the "unknown" can be: new technologies, new techniques can be a "proof of concept"
  - **spikes are carried out between sprints and before major epics / user stories**
  - products of a spike are usually intended to be thrown away types :
    - architectural spike: associated with an unknown area of the system, technology or application domain
    - non-architectural spike: others
  - **Risk based spike:** a spike to allow the team to eliminate or minimize some major risks
    - if the spike fails for every approach available, the project reaches a condition known as "fast failure", the cost of failure is much less than failing later

Problem Detection

- **Definition of Done (DoD)**
  - **Done** usually means the feature is 100% complete (including all the way from analysis, design, coding to user acceptance testing and delivery & documentation) and ready for production (shippable)
    - ✓ Done for a feature: feature/backlog item completed
    - ✓ Done for a sprint: work for a sprint completed
    - ✓ Done for a release: features shippable
  - The exact definition of done has been be agreed upon by the whole team (developer, product owner / customer, sponsor, etc.)
  - The definition of done includes acceptance criterion and acceptable risks
- **Frequent Validation and Verification**
  - early and frequent testing both within and outside the development team to reduce the cost of quality (change or failure)

- ✓ **validation:** (usually external) the assurance that a product, service, or system meets the needs of the customer
- ✓ **verification:** (usually internal of team) the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition

- Agile measure to ensure frequent validation and verification:
  - ✓ **testers** are included in the development team from the beginning taking part in user requirements collection
  - ✓ **unit tests** are created for continuous feedback for quality improvement and assurance
  - ✓ **automated testing tools** are used allowing quick and robust testing
  - ✓ **examples:** peer reviews, periodical code-reviews, refactoring, unit tests, automatic and manual testing
  - ✓ **feedback for various stages:** team (daily) -> product owner (during sprint) -> stakeholders (each sprint) -> customers (each release)

- **Variance and trend analysis**
  - variance is the measure of how far apart things are (how much the data vary from one another)
    - ✓ e.g. the distribution of data points, small variance indicates the data tend to be close to the mean (expected value)
  - **trend analysis** provides insights into the future which is more important for problem detection
    - ✓ though measurements are lagging, they will provide insights should trends be spotted
  - variance and trend analysis is important for controlling (problem detection) and continuous improvement, e.g the process to ensure quality
- **Control limits for Agile projects**
  - by plotting the time to delivery / velocity / escaped defects / etc. as a control chart
  - if some data fall outside the **upper / lower control limits**, a root cause analysis should be performed to rectify the issue
    - ✓ **common cause –** systematic issue, need to be dealt with through trend analysis
    - ✓ **special cause –** happens once only due to special reasons
  - another example is the WIP limit in Kanban boards
- **Escaped Defects**
  - Agile performance **(on quality)** can also be measure by the number of escaped defects (defects found by customers)
    - ✓ defects should be found and fixed during coding and testing
    - ✓ defects found early are much less expensive to fix than defects found late

Problem Resolution

- **The Five WHYs**
  - a systematic approach to analysing identifying the root cause of a problem / cause-and-effect for the problem or issue
  - perform by repeatedly asking the question "Why" for at least 5 times until the root cause has been identified
  - imaginary example: Looking for the root cause for failing the PMI-ACP Exam
    - ✓ 1. **Why** did I fail the PMI-ACP Exam?
    - ✓ - Because I got a lower mark than the passing mark
    - ✓ 2. **Why** did I get a lower mark?
    - ✓ - Because I was not sure about the answers to many questions.
    - ✓ 3. **Why** was I not sure about the answers to many questions?
    - ✓ - Because I could not remember some facts for the exam.
    - ✓ 4. **Why** couldn't I remember some facts for the exam?
    - ✓ - Because I was not familiar with the PMI-ACP Exam content.
    - ✓ 5. **Why** was I not familiar with the PMI-ACP Exam content?
    - ✓ - Because I did not spend enough time revising the PMI-ACP Exam notes.

- **Fishbone Diagram Analysis**
  - another tool for carrying out cause and effect analysis to help discover the root cause of a problem or the bottlenecks of processes
  - aka cause and effect diagrams/Ishikawa diagrams to use Fishbone diagram technique:
    - ✓ 1. write down the problem/issue as the "fish head" and draw a horizontal line as the "spine" of the fish
    - ✓ 2. think of major factors (at least four or above) involved in the problem/issue and draw line spinning off from the spine representing each factor
    - ✓ 3. identify possible causes and draw line spinning off the major factors
    - ✓ (your diagram will look like a fishbone now)
    - ✓ 4. analyse the fishbone diagram to single out the most possible causes to carry out further investigation

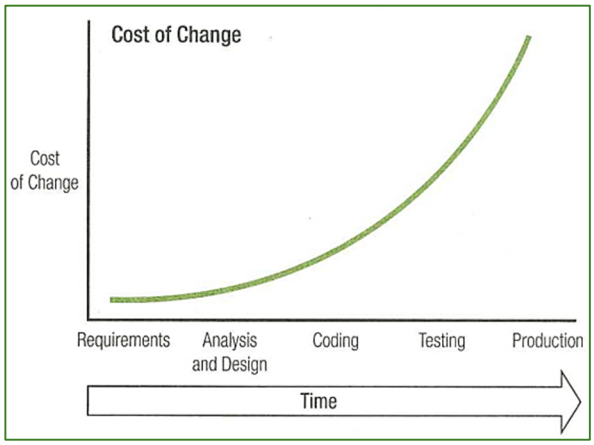
Understanding Problems:

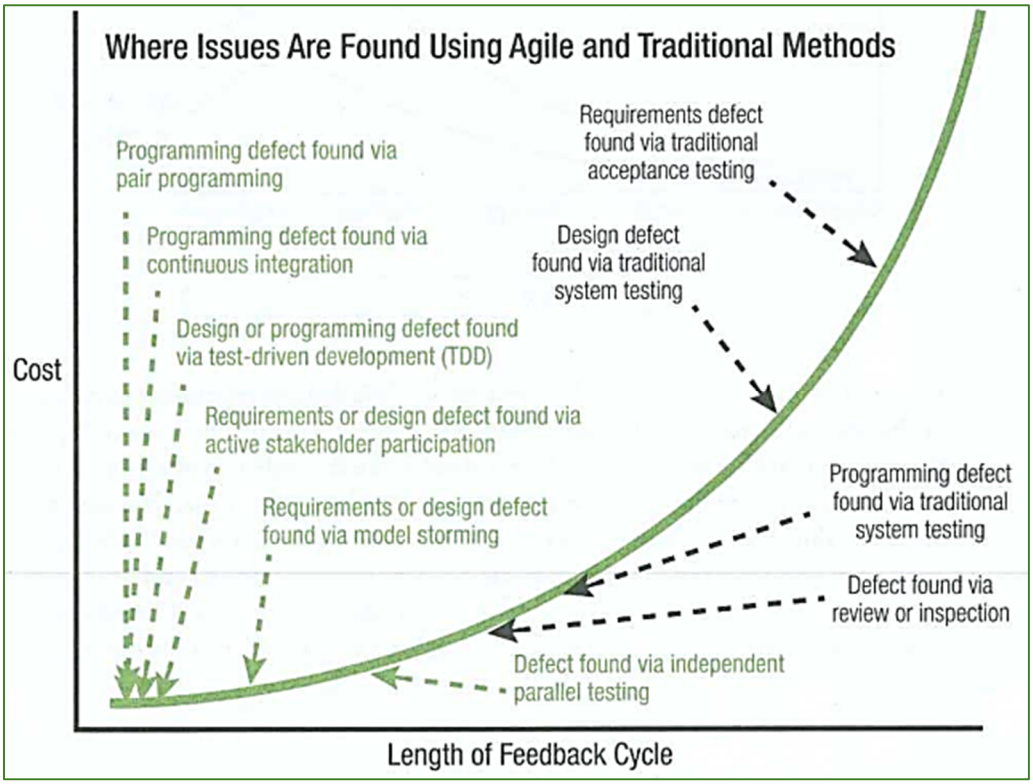
When a problem occurs on a project, it can be tempting to ignore the issue and just continue pushing forward, in the hope that it will somehow go away or resolve itself. But even a single problem can result in delay, waste, and rework that can bring our progress to a halt and even reverse it, completely derailing our project schedule. And the longer the problem is left unaddressed, the more its effects will be compounded.

The Cost of Change:

The curve of this graph shows that the longer a defect is left unaddressed, the more expensive it will be to fix. There are a number of reasons why the cost of fixing a problem increases over time, but here are two of the key factors:

- Over time, more work may have been built on top of the error or problem, so that more work will need to be done to fixit (and then have to be redone).
- The later we are in the development cycle, the more stakeholders will be impacted by the defect, making it that much more expensive to fix.





As we've seen, agile methods emphasize frequent verification and validation—both through active stakeholder participation (such as modelling, demonstrations, and reviews) and through software

development practices (such as pair programming, continuous integration, and test-driven development). These practices are all intended to find defects and problems as soon as possible, before the costs escalate too far up the cost of change curve.

Technical Debt:

Technical debt is the backlog of work that is caused by not doing regular cleanup, maintenance, and Standardization while the product is being built.
In software projects the solution to technical debt is refactoring. (“Red, Green, Refactor”). Refactoring is the process of taking time to simplify and standardize the code to make it easier to work on in the future. For example, if one of the developers comes up with a better way to validate user-entered text, then that approach should be applied everywhere, and the existing code will need to be updated to include it. We don’t want some parts of the system to use one way of validating text and other parts of the system to use another way. If multiple approaches are used, then future upgrades and changes will need to be made in several places. This will take longer, and since it is easy to miss one of the discrepancies, this can easily lead to errors.
All of this extra effort is technical debt. So, we need to do frequent refactoring to keep the code clean and reduce technical debt. There’s a saying that “Refactoring should be like daily hygiene, not spring cleaning.”

Failure Modes:

- **We make mistakes**
- **We prefer to fail conservatively**
- **We prefer to invent rather than research**
- **We are creatures of habit**
- **We are inconsistent**

Success Modes:

- **We are good at looking around.** This refers to our ability to observe, review, and notice when things are not right.
- **We are able to learn.** After seeing what’s wrong, we are able to find ways to fixit and expand our skills and knowledge along the way.
- **We are malleable.** Despite the common resistance to change, we do have the ability to change and accept new ideas and approaches.
- **We take pride in our work.** We are able to step outside of our job descriptions to repair or report an issue, because it is the right thing to do for the project.

Success Strategies:

- **Balance discipline with tolerance.**
- **Start with something concrete and tangible.**
- **Copy and alter.** (“blank page syndrome”)
- **Watch and listen.**
- **Support both concentration and communication.**
- **Match work assignments with the person.**
- **Retain the best talent.**
- **Use rewards that preserve joy.**
- **Combine rewards.**
- **Get feedback.**

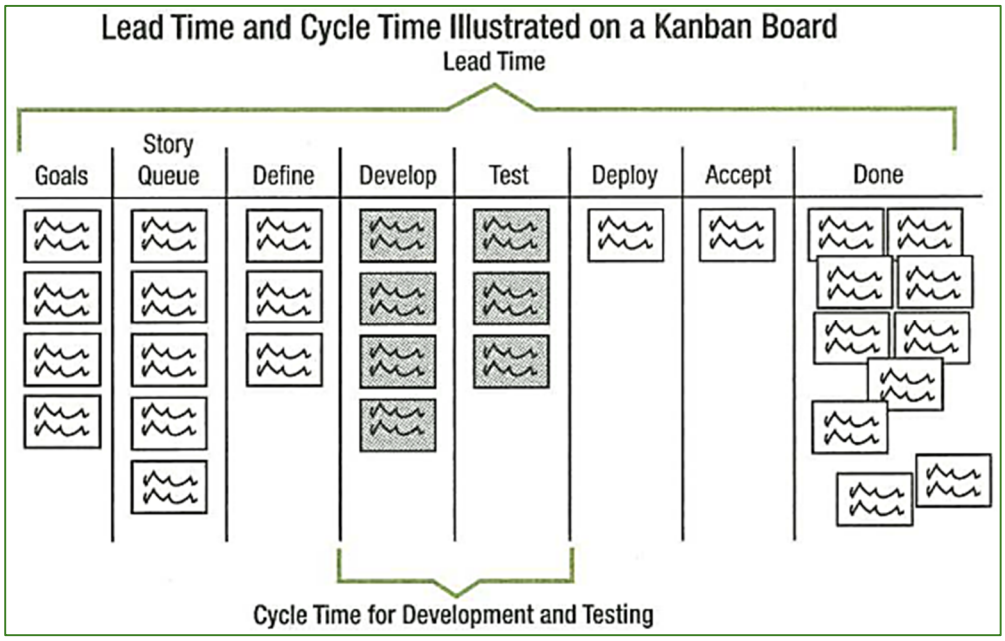
Detecting Problems

The daily stand-up meeting is also an important mechanism for identifying problems.

Lead Time and Cycle Time:

Lead time is a diagnostic tool that can be used to help identify and diagnose problems. This concept measures how long something takes to go through the entire process, for example, from design to shipping, or from requirements gathering through development to deployment.

Cycle time is a subset of lead time that measures how long something takes to go through part of the process, such as from product assembly to painting, or from coding to testing. For example, the cycle time for building a user story begins when the team starts working on it and ends when that item is finished, accepted, and begins delivering business value.



Here's a diagram that illustrates the difference between lead time and cycle time on a Kanban board:

Cycle Time, WIP, and Throughput:

In mathematical terms, cycle time is a function of WIP and throughput and can be calculated by using the following formula: **Cycle time = WIP / Throughput**

Throughput, the other variable in this equation, is the amount of work that can be processed through a system in a given amount of time—such as the amount of work the team can get done in one iteration.

Throughput and Productivity:

It's important to understand the difference between throughput and productivity since the exam questions may use both terms. We've said that (throughput is the average amount of work the team can get done in a time period (or their average completion time). Productivity, on the other hand, is the rate of efficiency at which the work is done—such as the amount of work done per team member.

For example, if a team's throughput goes up, it might be because their productivity (output per person) went up. But not necessarily. They might have gotten more work done because they added another person to the team. In that case, their individual productivity might have actually gone down in the process of getting the new team member oriented to the project and the team. Yet they still might have been able to get more work done since there were more hands on deck.

EXERCISE: CYCLE TIME

QUESTION: Here's a chance to test your understanding of cycle time, WIP, and throughput with an exercise. Imagine a bicycle factory that produces 25 bikes per day and typically works on 100 bikes at any given time. Calculate the average length of time it takes to make a bike.

ANSWER: The WIP, or the measure of work in progress, is the 100 bikes being worked on at any given time. Throughput is how many things go through the process per time interval; in this case, the throughput is 25 bikes per day. We are being asked to calculate how long it takes on average to make a bike, which is the cycle time (Ha!). Since **cycle time = WIP / throughput**, we cancalculate our cycle time as 100/25 = 4 days.

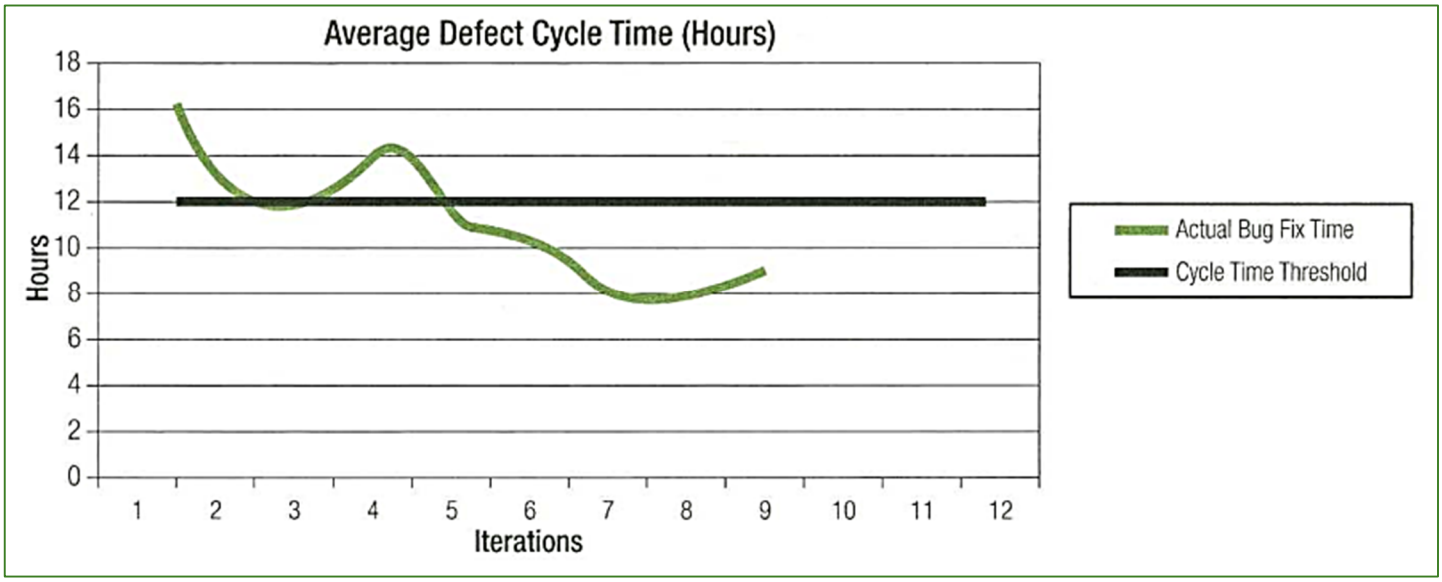
QUESTION: Improvements to the assembly process reduce the cycle time to 3 days and the WIP to 90 bikes in progress. What is the percentage of improvement in throughput?

ANSWER: Once we've done this calculation, we still have to find the percentage of improvement over the

old throughput of 25 bikes per day. So, we subtract the old throughput from the new measurement (30-25 = 5), and then divide the difference by the old throughput (5/25 = 0.2, or 20%).
Therefore, increasing our throughput from 25 to 30 bikes per day is a 20 percent improvement.

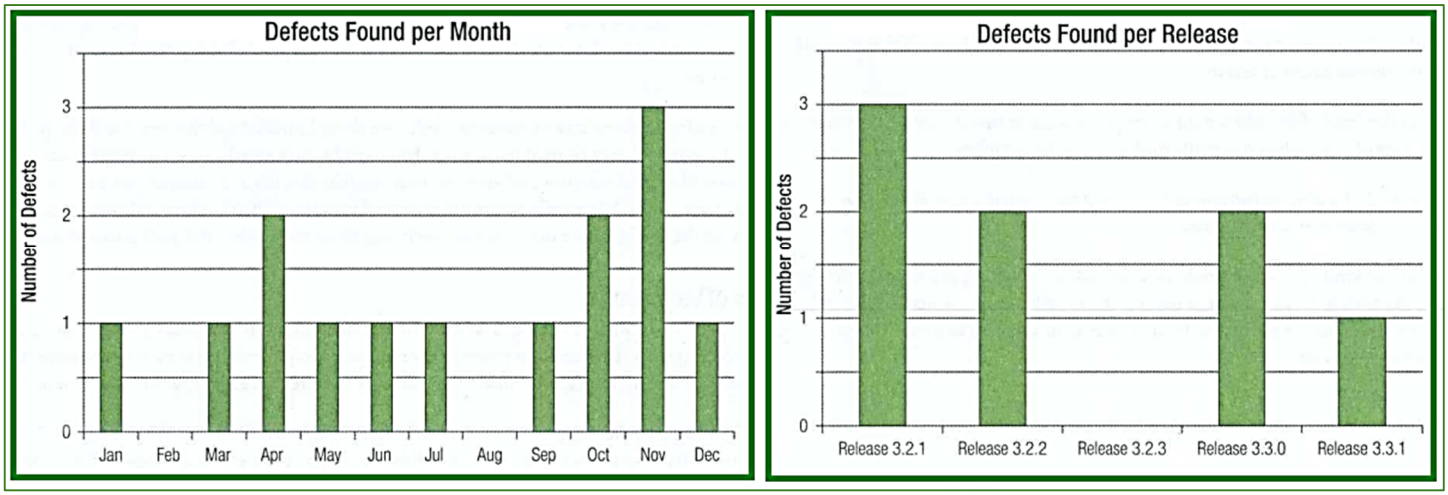
Defects:

"**Defect cycle time**" is the period between the time the defect was introduced and the time it was fixed. The length of the defect cycle time dictates how far up the cost of change graph the defect will go.
By tracking both their defect cycle time and their cycle time for creating new work, agile teams can Minimize both the potential for rework and the cost of any rework that is required.



Defect Rates:

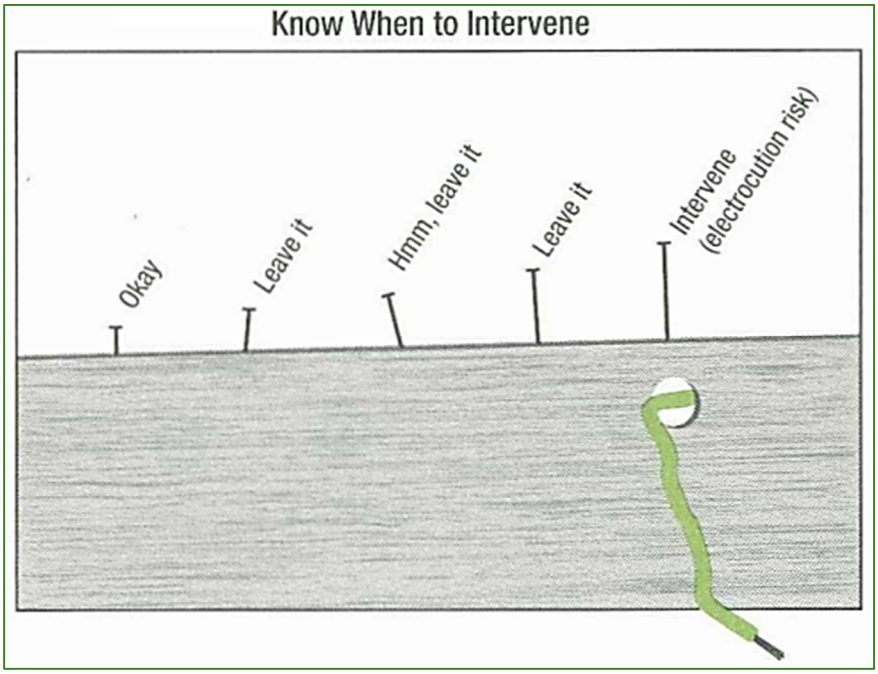
Unfortunately, no matter how hard we try to identify and prevent defects, there may be an occasional defect that makes it through all our tests and quality control processes and ends up in the final product, (in software development, these are called **escaped defects**.) Defects that are missed by testing are the most costly kinds of defects to fix, since they are on the top right end of the cost ofchange curve.



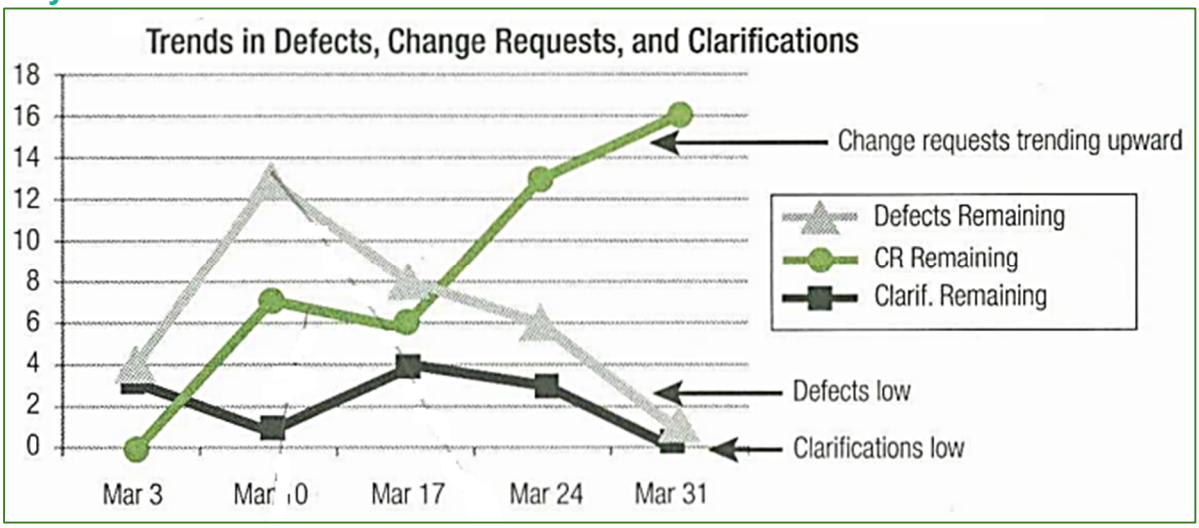
Variance Analysis:

Quality expert W. Edwards Deming classifies variance into common cause variation and special cause Variation. **Common cause variation** refers to the average day-to-day differences of doing work, and **Special cause variation** refers to the greater degrees of variance that are caused by special or new factors.

- **Mistake 1:** Reacting to an outcome as fit came from a special cause, when it actually came from common causes of variation.
- **Mistake 2:** Treating an outcome as if it came from common causes of variation, when it actually came from a special cause.

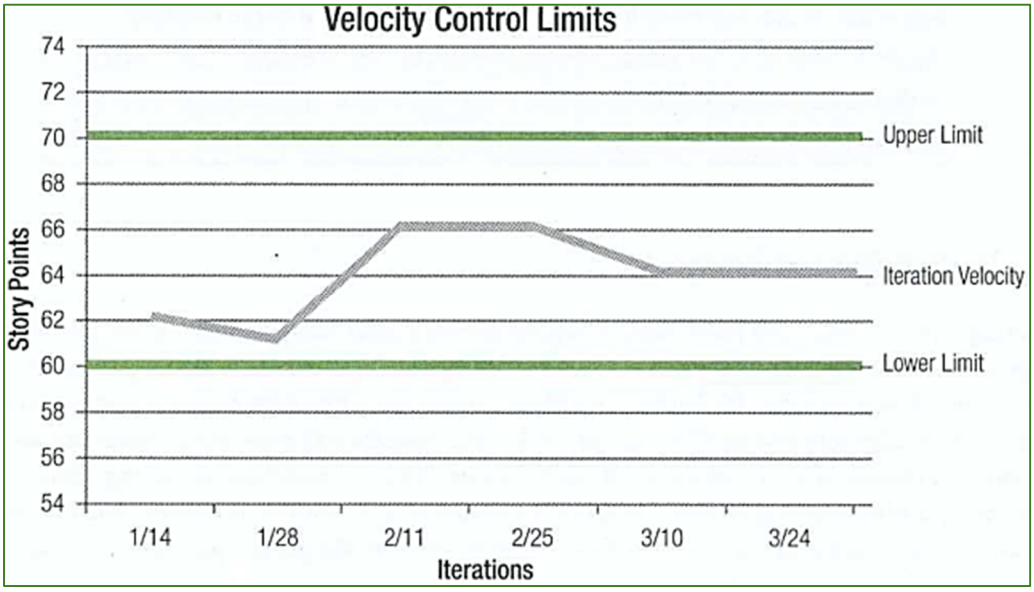


Trend Analysis:



Control Limits:

One way we can use control limits on an agile project is to monitor our velocity to gauge how likely it is that we will be able to complete the agreed-upon work by the release date. For example, if we have 600 points' worth of functionality left in the backlog and 10 months until deployment preparations begin, we should set our lower control limit at 60 points per month (600 / 10 = 60), since that is the minimum velocity required to meet this goal.



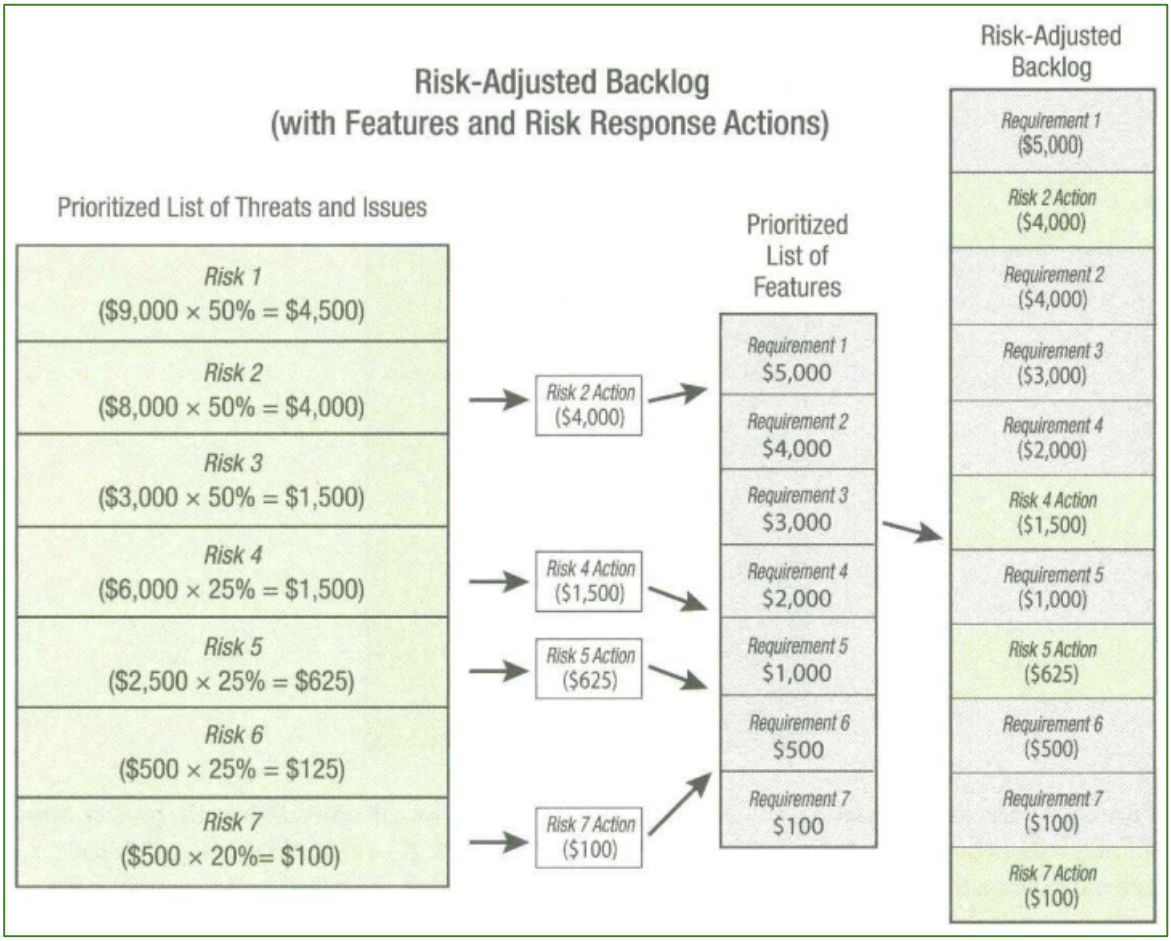
Managing Threats and Issues - Risk-Adjusted Backlog:

The backlog might start out as just a list of the business features involved in the project, divided into practice bundles of work—but once the risk response activities are added and prioritized (based on their anti-value),it can be referred to as a "risk-adjusted backlog". This single prioritized list is what allows agile teams to focus simultaneously on both value delivery and risk reduction activities.

Creating the Risk-Adjusted Backlog:

After the business representatives have (perhaps somewhat arbitrarily) attributed a dollar value to each of the product features, we can prioritize those features based on business value, as shown.
Next, we need some way to monetize the risk avoidance and risk mitigation activities. To get this figure, we can calculate the expected monetary value of each risk.

Expected Monetary Value (EMV) = Risk Impact (in dollars) x Risk Probability (as a percentage)



Risk Severity:

To calculate risk severity, instead of using a risk's probability percentage and dollar impact, we instead rank its probability and impact on a simple scale—such as low (l),medium (2), and high (3). Then, we multiply those probability and impact rankings to calculate the risk's severity:

Risk Severity = Risk Probability x Risk Impact

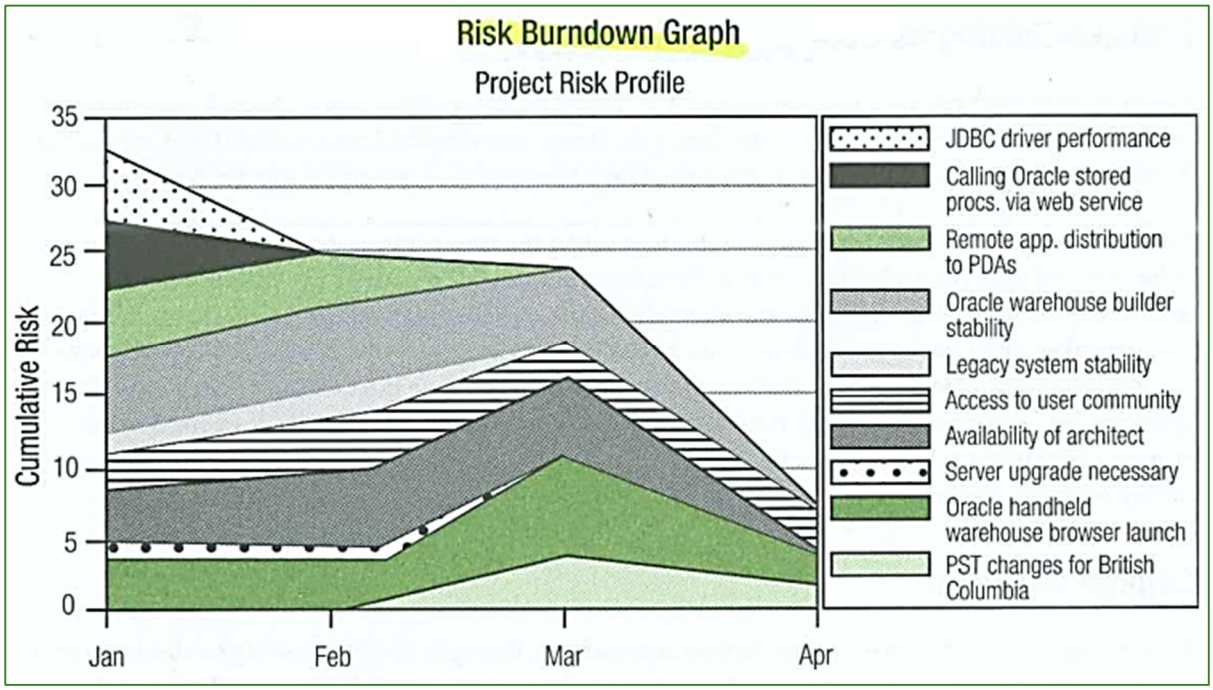
Now, this formula might look just like the one we used for EMV—however, its output will be different because the inputs are rankings. For example, let's say we decide to rank the probability and impact of our risks on a three-point scale, as low(1), medium (2), or high (3). Using this scale, a risk that has a high probability and a high impact will have a risk severity of 3 x 3 = 9. On the other hand, a risk that has a high probability and a low impact will have a risk severity of 3 x 1= 3.

Progress of Risks

		January			February			March			April		
ID	Short Risk Name	Imp.	Prob.	Sev.	Imp.	Prob.	Sev.	Imp.	Prob.	Sev.	Imp.	Prob.	Sev.
1	JDBC driver performance	3	2	6	3	0	0	3	0	0	3	0	0
2	Calling Oracle stored procs. via web service	2	2	4	2	0	0	2	0	0	2	0	0
3	Remote app. distribution to PDAs	3	2	6	3	1	3	3	0	0	3	0	0
4	Oracle warehouse builder stability	2	2	4	2	3	6	2	2	4	2	0	0
5	Legacy system stability	2	1	2	2	1	2	2	0	0	2	0	0
6	Access to user community	2	1	2	2	2	4	2	1	2	2	1	2
7	Availability of architect	2	2	4	2	3	6	2	2	4	2	0	0
8	Server upgrade necessary	1	2	2	1	1	1	1	0	0	1	0	0
9	Oracle handheld warehouse browser launch	3	1	3	3	1	3	3	3	9	3	1	3
10	PST changes for British Columbia	0	0	0	0	0	0	2	2	4	2	1	2
				33			25			23			7

Risk Burndown Graphs:

Risk burndown graphs are essentially stacked area graphs of cumulative project risk severity. The severity scores for each risk are plotted one on top of another to show the project s cumulative severity profile. When risks and their history of severity are displayed in this format, it is much easier to interpret the overall risk status and trends of the project.



Problem Solving as Continuous Improvement:

Agile methods consider the team's lessons learned to be too important to be reserved until the end of the project. Therefore, they include iterative opportunities for the team to capture those lessons, probe the strengths and weaknesses of their approach, and ensure that their mistakes won't be repeated. These team-based "problem solving" efforts are actually part of the continuous improvement process.

Engage the Team:

Agile methods engage the team in identifying, diagnosing, and solving threats and issues. Rather than leaving problem solving and correction to the project manager or external "fixers", this is very much a whole-team activity.

The Benefits of Team Engagement:

- **By asking the team for a solution, we inherit consensus for the proposal.**
- **Engaging the team accesses a broader knowledge base.**
- **Team solutions are practical.**
- **When consulted, people work hard to generate good ideas.**
- **Asking for help shows confidence, not weakness.**
- **Seeking others' ideas models desiredbehavior.**