THE DZONE GUIDE TO

SOFTWARE DEVELOPMENT LIFECYCLE

QA AND TESTING VOLUME II

BROUGHT TO YOU IN PARTNERSHIP WITH

Perfecto

Dear Reader,

"We are uncovering better ways of developing software by doing it and helping others do it."

- Manifesto for Agile Software Development

In 2016, we are now 15 years since the initial drafting of the Agile Manifesto. And with that has come many iterations of how to best improve the Software Development Lifecycle. 15 years has seen the growth and evolution of technology beyond all of our expectations. So, now, over a decade later, how have agile practices impacted the lives of software developers since then?

According to our survey results, experience with Agile has been overwhelmingly positive. However, those who are newer to agile practices are slightly less positive than those who have more than 10 years of experience with it. Is this due to new practices? A honing in of practices that have been around for years? Or, an overall change of culture? It is evident that agile practices can help more than they harm the SDLC, but what processes work best depends on the team and its willingness to embrace it.

There may be various methodologies around the practice of Agile, but at the end of the day, it is a process centered around clear communication, process efficiency, and ensuring a successful end result. The Software Development Lifecycle is about connecting people with the right tools and processes to enable a smooth and efficient release.

In the *2016 Guide to the Software Development Lifecycle: QA and Testing*, we've explored the ownership of quality in the agile process, some ways to ease the pain points of performance testing, strategies centered around test-driven development, and reflecting on past experiences in order to enable future successes. We've surveyed over 500 software developers and interviewed executives to see their perspectives on SDLC, where it's headed, and current frustrations. We are eager for you to consume our findings and draw your own conclusions on Agile best practices.

The Agile process is all about sharing knowledge in order to improve the software development process. Here at DZone, we pride ourselves in our ability to take part in this knowledge-sharing. When we look back at the Agile Manifesto after another decade passes, it will be good to see how many more steps we've taken—setbacks and all—to improve the Software Development Lifecycle.

BY CAITLIN CANDELMO

DIRECTOR OF CONTENT & COMMUNITY AT DZONE

RESEARCH@DZONE.COM

TABLE OF CONTENTS

MARKETING
KELLET ATKINSON — DIRECTOR OF MARKETING
LAUREN CURATOLA — MARKETING SPECIALIST
KRISTEN PAGÀN — MARKETING SPECIALIST
NATALIE IANNELLO — MARKETING SPECIALIST

PRODUCTION
CHRIS SMITH — DIRECTOR OF PRODUCTION
ANDRE POWELL — SR. PRODUCTION COORDINATOR
G. RYAN SPAIN — PRODUCTION PUBLICATIONS EDITOR
ASHLEY SLATE — DESIGN DIRECTOR

EDITORIAL
CAITLIN CANDELMO — DIRECTOR OF CONTENT + COMMUNITY
MATT WERNER — CONTENT + COMMUNITY MANAGER
MICHAEL THARRINGTON — CONTENT + COMMUNITY MANAGER
NICOLE WOLFE — CONTENT COORDINATOR
MIKE GATES — CONTENT COORDINATOR
SARAH DAVIS — CONTENT COORDINATOR
TOM SMITH — RESEARCH ANALYST

BUSINESS
RICK ROSS — CEO
MATT SCHMIDT — PRESIDENT & CTO
JESSE DAVIS — EVP & COO
MATT O'BRIAN — DIRECTOR OF BUSINESS DEVELOPMENT
sales@dzone.com
ALEX CRAFTS — DIRECTOR OF MAJOR ACCOUNTS
JIM HOWARD — SR ACCOUNT EXECUTIVE
JIM DYER — ACCOUNT EXECUTIVE
ANDREW BARKER — ACCOUNT EXECUTIVE
CHRIS BRUMFIELD — ACCOUNT MANAGER
ANA JONES — ACCOUNT MANAGER

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

SPECIAL THANKS to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this report a great success.

Executive Summary

Fifteen years since the drafting of the Agile Manifesto, more and more philosophies and methodologies to improve the SDLC in the enterprise have emerged. While agile practices like sprints, Kanban boards, and Scrum meetings have caught on, there are still some disconnects between developers, leadership, and how Agile can improve the lives of each. To examine how Agile has impacted the working lives of the DZone audience, we surveyed 522 software professionals across the world, and with varying job roles and experience.

AGILE HAS AGED WELL WITH TIME

DATA Those who have practiced agile methodologies for less than 2 years are more likely to feel positive about their effects (61%) or very positive (20%). Those who have used agile methodologies for over 11 years are more likely to feel very positive about them (47%) rather than positive (43%).

IMPLICATIONS First, it can be inferred that the benefits of Agile are almost immediately apparent for those just getting started. 80% of those who've been working with Agile for less than 2 years feel positive feelings towards it. The second implication is that, unlike some enterprise or software trends, agile practices do not lose their usefulness the longer employees work in that system.

RECOMMENDATIONS Between specific methodologies we asked about, including Kanban, Scrum, and DevOps, Waterfall was the most disliked practice. 33% of users claimed to hate it, and 53% felt no strong positive or negative feelings about it. Based on the overwhelming positive feedback on the benefits of Agile, it's clear that at the very least, agile methodologies improve the way developers work and how they feel about that work. Enterprises that have not made the move to incorporate agile practices into the SDLC should do so.

WORK IN PROGRESS HAS AN IMPACT

DATA 36% of respondents who don't feel that Agile produces desired results for all stakeholders do not have a way to limit work in progress (WIP). Of all respondents who don't limit WIP (24%), 42% feel neutral towards Agile's ability to produce desired results. 24% of respondents do not have a way to limit work in progress (WIP). 54% limit WIP by working in iterations or sprints, and 31% use visualization techniques like burndown charts or Kanban boards to limit WIP.

IMPLICATIONS Since there is a relationship between those who don't limit WIP and those who have neutral or cynical thoughts on Agile, there seems to be an education gap between those who limit WIP and those who don't. Those who don't limit WIP may not know the benefits of Agile compared to how they're working now.

RECOMMENDATIONS Just reading about Agile may not be convincing enough a reason for organizations to change. Attending workshops, speaking with coaches or project managers, and researching case studies may also help. Context for stakeholders around WIP and bottlenecks may be missing as well. A good book to read is "The Goal," which popularized the "theory of constraints," which is built around removing bottlenecks and work in progress. While the context of the book examines the theory in a factory setting, it can be applied to software development. In fact, "The Phoenix Project" does just that.

IS AGILE HELPING EVERYONE?

DATA 62% of respondents believe Agile is producing desired results for all stakeholders, 17% say it is not, and 12% say it is, except for developers. At the same time, 65% of all users believe stakeholders either do not communicate or understand their goals clearly, compared to 29% who say they do. Of those who don't think Agile is delivering desired results, 90% feel that goals are not communicated well.

IMPLICATIONS It's clear that Agile needs a level of understanding that is shared between all business stakeholders. Those who are most cynical about Agile tend to feel that goals are not communicated well, but even then, many who do believe Agile produces the desired results do not feel like their goals are communicated clearly, but the benefits are clear to them and seem to outweigh that concern.

RECOMMENDATIONS Management should be very clear about 1) What the goals of the organization are; 2) Why these goals were set; and 3) What an employee's role is in meeting those goals. They should also consult with employees about company, team, and personal goals to make sure there's a clear understanding of what provides value for both employees and the business. If an employee feels a goal or metric is not indicative of the work they do, or if they feel that one of their goals will not provide value to the company, there should be an open discussion to determine whether that is the case or not. It's clear that those who don't believe that Agile is beneficial suffer from poor communication, and organizations should rectify this issue the best they can.

Key Research Findings

522 software professionals completed DZone's 2016 Software Development Lifecycle survey. Respondent demographics include:

- 44% of respondents identify as developers/ engineers, and 32% identify as development team leads.
- 70% of respondents have 10 years of experience or more as IT professionals. 44% of respondents have 15 years or more.
- 42% of respondents work at companies headquartered in Europe; 32% work in companies headquartered in the US.
- 22% of respondents work at companies with more than 10,000 employees; 26% work at companies between 500 and 10,000 employees.

WITH GREAT POWER...

Overall sentiment about agile development and methodologies among survey respondents is quite positive. 54% of respondents said they feel "positive" about agile, and 30% said they feel "very positive." Only 3% of respondents said they feel either "negative" or "very negative" about agile development or methodologies.

While most respondents feel positively about agile, experience seems to play a role into just how positive that feeling is. Respondents who say they have 0-2 years of experience with agile are more likely to feel "positive" about agile (61%) versus "very positive" (20%). As respondents' experience grows, this balance shifts towards "very positive;" respondents with 11+ years of experience with agile are more likely to feel "very positive" (47%) than "positive" (43%).

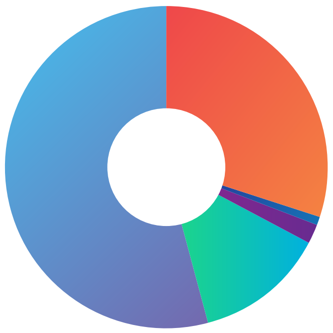
Other specific methodologies also received a fair amount of love. The majority of respondents who use Scrum, test-driven development, DevOps, Kanban, domain-driven design, and pair programming say they "use it and love it" over hating it or having no strong positive or negative feelings towards it. Waterfall methodology was the most hated, with 33% of waterfall users saying they "use it and hate it," though 53% of waterfall users do not feel strong positive or negative feelings about it.

GETTING RESULTS

62% of respondents say that agile is producing the desired results for all stakeholders involved, while 17% believe this is not the case (12% believe agile is producing the desired results for all stakeholders except developers). Part of this may be impacted by how developers feel goals are communicated from business stakeholders. Overall, 65% of respondents believe business stakeholders do not understand or communicate these goals clearly, while 29% of respondents believe they do.

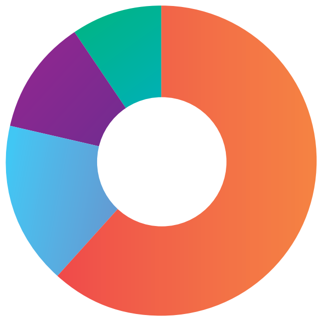
Looking at these responses together shows that 90% of the respondents who do not believe agile is producing

OVERALL, HOW DO YOU FEEL ABOUT AGILE DEVELOPMENT AND METHODOLOGIES?



%	
30	Very Positive
1	Very Negative
2	Negative
13	Neutral
54	Positive

IS AGILE PRODUCING THE DESIRED RESULTS FOR ALL STAKEHOLDERS INVOLVED (INCLUDING DEVELOPERS)?



%	
62	Yes
17	No
12	Probably for everyone EXCEPT developers
9	Doesn't Apply to Me

the desired results for all stakeholders also believe that business stakeholders do not clearly understand or communicate goals. And 79% of all respondents who think these goals are clearly communicated feel that agile produces the desired results for all stakeholders.

SETTING LIMITS

Limiting the amount of work in progress is one of the uses of agile or other development methodologies. Respondents mostly "commit to some limited work in an iteration" (54%) and/or "visualize progress using a burn-down chart or something similar" (31%) to limit work in progess. But 24% of respondents said they do not limit work in progress.

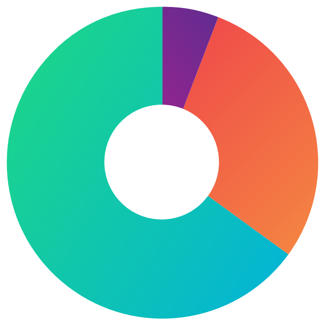
Not using some method to limit work in progress seems to have an impact on respondents' feelings towards agile methodologies and towards how effective they think it is. 42% of respondents who do not have a system to limit work in progress feel neutral towards agile, while only 21% of these respondents feel "positive" and 17% feel "very positive." Furthermore, 36% of respondents who don't think agile produces the desired results for all stakeholders also do not limit work in progress (well above the overall 24%), and only 16% of those who do think agile produces the desired results also do not limit work in progress.

MEASURING QUALITY

Feelings about the accuracy of quality metrics respondents are measured on were tepid at best. While 56% of respondents believed that these metrics are "somewhat accurate" and 30% believed they are "somewhat inaccurate," only 14% of respondents had feelings on either extreme (6% "very accurate" and 8% "very inaccurate").

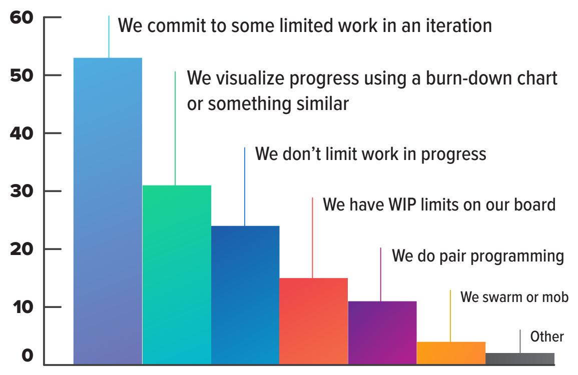
Unsurprisingly, these feelings affect how respondents see the time and energy spent collecting these metrics as an impact on their job satisfaction. Of the respondents who said this time and energy have a negative impact on their job satisfaction (15% of all responses), 47% felt these metrics are "somewhat inaccurate," and 18% felt these metrics are "very inaccurate." On the other hand, of the respondents who said this time and energy have a positive impact on job satisfaction, 68% believed these metrics to be "somewhat accurate," and 15% believed them to be "very accurate."

IN GENERAL, DO YOU FEEL THAT BUSINESS STAKEHOLDERS CLEARLY UNDERSTAND + COMMUNICATE THE GOALS FOR THE TECHNOLOGY SOLUTION?

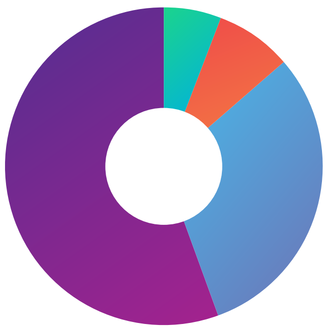


%	
6	Doesn't apply to me
29	Yes
65	No

HOW DO YOU LIMIT WORK IN PROGRESS?

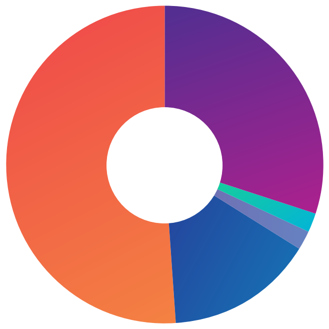


IN YOUR JUDGMENT, HOW ACCURATELY DO THE QUALITY METRICS YOU ARE MEASURED ON REFLECT THE QUALITY OF YOUR WORK?



%	
56	Somewhat accurately
31	Somewhat inaccurately
8	Very inaccurately
6	Very accurately

HOW DOES THE ENERGY SPENT COLLECTING + REVIEWING METRICS IMPACT YOUR JOB SATISFACTION?



%	
30	Positive impact
2	Very positive impact
2	Very negative impact
15	Negative Impact
51	Neutral

Who Owns Quality in Agile?

BY **DAN TOUSIGNANT**
PRESIDENT AT **CAPE PROJECT MANAGEMENT, INC.**

QUICK VIEW

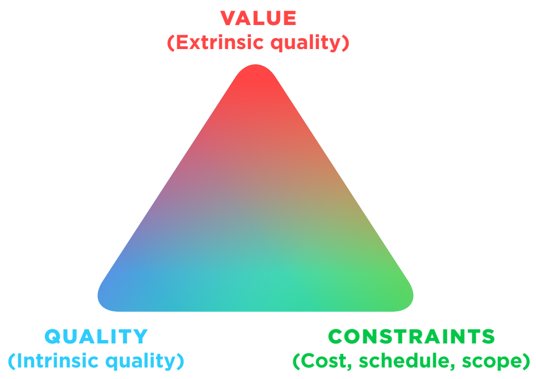
01 Extrinsic quality, e.g. customer satisfaction, is the most important measure of quality.

02 The Product Owner is ultimately responsible for product quality.

03 Scrum teams need to balance long-term performance and stability with short-term goals for delivery and value.

In Scrum, the expectation is that the entire Scrum Team owns quality, but what does that really mean? Isn't the Product Owner only worried about value? Doesn't the team own all the testing? Don't they really own quality? Like many other concepts in Agile, the answer to who owns quality is subtler than that, and as my first Agile mentor liked to say, "It depends."

THE AGILE TRIANGLE



Before we define ownership, let's define what we mean by quality in Agile. In terms of Agile software development, Jim Highsmith in *Agile Project Management: Creating Innovative Products* identifies quality as two points of the Agile Triangle: Intrinsic quality and Extrinsic quality.

This suggests that the definition of quality comes from two different sources, externally from the customer and internally from the development teams.

DEFINING QUALITY

Joseph Kelada, author of *Integrating Reengineering with Total Quality* defines these two aspects of quality:

Intrinsic Quality is all of the qualities that are built into the product: suitability, durability, reliability, uniformity, and maintainability. This type of quality can be measured quantitatively, such as test coverage, bugs per line of code, escaped defects, cohesion, etc.

Extrinsic Quality is the buyer's perception of quality and the value to the customer. This is a more qualitative measurement based upon sales, usage, and customer feedback.

When most people think of quality, they think of intrinsic quality, which is why we have team members who are Quality Assurance specialists. They aren't evaluating the customer's perception of the product, they are performing verification and validation (V&V): *Are we building the system right? Are we building the right system?* The goal of V&V is to test the product against the written business and technical requirements.

On Agile projects, we build products with the premise that every requirement ties back to a customer-facing vision and value proposition. If a requirement doesn't align with that vision, then it doesn't matter how reliable it is, for it is of no value. With each iteration and release, the goal is to provide the highest value features balanced against the cost to develop those features. This is the goal of each Sprint in Scrum.

Given this, I would suggest that Jim Highsmith's Agile triangle is visually inaccurate. It gives equal weight to intrinsic and extrinsic quality. The reality is that in order for organizations to compete, extrinsic quality, in most cases, is more important. Remember, the first principle in the Agile Manifesto, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

THE ROLE OF PRODUCT OWNER IN QUALITY

I recently encountered a simple example of how a Product Owner addressed intrinsic versus extrinsic quality while facilitating a Sprint planning meeting. A developer submitted a backlog item to refactor some previously written code that was causing a large increase in storage each time a report was run. This was an unexpected outcome of a customer-facing requirement. The assumption from a technical perspective was that the storage issue was egregious and the impact was large, but the effort to refactor the code was large as well. In a traditional software development environment it would be assumed that a defect like this would automatically be addressed in the next release. On this high-performing Scrum Team, the Product Owner questioned the value of addressing the defect in this release.

This involved an in-depth discussion of the issue and the options, and ultimately it came down to a quantifiable measurement. The technical debt of that storage issue could be measured in dollars and could easily be compared against the business value of other requirements. Once the issue was quantified, the Product Owner removed the requirement as a candidate for this release. The Product Owner had more "valuable" items slated for this release. Not all issues of extrinsic quality versus intrinsic quality can be that easily quantified, but in mature Agile teams, the correct dialogue will occur and the decision that is most often made will balance the short-term customer facing goals with the long-term viability of the product.

QUALITY OWNERSHIP

So, back to the original question, who owns quality? When speaking of Agile roles, it is easiest to use the Scrum roles. In the most recent annual VersionOne Agile survey, 75% of companies practicing Agile are using Scrum or at least the Scrum terminology. The roles of Product Owner, Scrum Master, and Development Team have become ubiquitous terms in our industry.

In the Scrum Guide, it states, "As Scrum Teams mature, it is expected that their definitions of 'Done' will expand to include more stringent criteria for higher quality." "Scrum Teams" includes all three Scrum roles. So this would presume that the entire Scrum Team owns the intrinsic quality. However, the Guide also states that the Product Owner is responsible for "Optimizing the value of the work the Development Team performs." So, ultimately, the Product Owner owns extrinsic quality, and as we just discussed, this may supersede intrinsic quality.

THE REALITY

Though organizations understand this theoretically, this is seldom put fully into practice. For the Product Owner to truly own quality, the following best practices would need to be in place:

- Quality Assurance specialists would functionally report to the business.
- The Product Owner would sign off on design documents.
- Performance and non-functional requirements would be approved by the Product Owner.
- All defects would be approved by the Product Owner before being added to the backlog.
- The scope of pre-production testing and readiness would be approved by the Product Owner.

Those changes and many more would need to occur, both culturally and organizationally, for an organization to truly align quality ownership with the Product Owner role.

At minimum, in order for the Product Owner to truly own value and quality, the Development Team needs to educate the Product Owner on the cost and effort of addressing intrinsic quality issues. The Product Owner needs to understand that well written code costs less to support and maintain. Technical debt eventually needs to be paid back and only an educated Product Owner can make the best decision as to when. This often creates conflicts within newly emerging Agile teams. In my experience, especially with large traditional organizations, this can shift the ownership of building the product away from the IT department and more to the business. This is intentional in Agile so that we ensure that delivering value to the customer is a primary goal.

Personally, I have seen too many well built, highly reliable, stable products sit on a shelf because ultimately they missed customer's expectations or were too late to market. Often a product that was not built half as well can dominate market share. There are exceptions to this of course, but for those companies at the forefront of their industries, they are adopting a Lean and Agile approach to software development that frontend-loads value. They then add stability and reliability once the product achieves traction and validates the ROI for both the initial investment and continuing investment.

It is a team effort to understand how each requirement, whether technical or functional, will drive value to the customer. This type of understanding requires close collaboration between the Product Owner and the development team. It means truly embracing the "one team" concept and allowing full transparency into the decisions about which requirements make it into each release. Ultimately, it will be the Product Owner's decision, but this can only occur in an environment that has truly embraced what it means to be Agile.

REFERENCES

Joseph Kelada, *Integrating Reengineering with Total Quality*

Jim Highsmith, *Agile Project Management: Creating Innovative Products*

10th Annual VersionOne Survey

Jeff Sutherland, Ken Schwaber 2016 Scrum Guide

DAN TOUSIGNANT has been managing software development projects for 20 years. He was first introduced to Agile via a Scrum Implementation in 2000 and has since adopted Agile as the primary method for managing software development projects. Dan holds a BS in Industrial Engineering from UMASS, Amherst and is a Professional Scrum Master, Certified Product Owner, PMI Agile Certified Practitioner and Certified Scrum Professional.

Facilitating Effective Agile Retrospectives

BY **BEN LINDERS**
INDEPENDENT CONSULTANT AT **BEN LINDERS CONSULTING**

QUICK VIEW

01 Agile retrospectives help teams to reflect, learn, and improve themselves.

02 Vary retrospective exercises and try different settings to prevent people from getting bored doing retrospectives.

03 Whoever facilitates the retrospective must have the proper skills to do it.

04 Teams should leave the retrospective with actions that they can and will do in the next iteration.

Agile teams use retrospectives to reflect, learn, and adopt their way of working. Let's explore how you can do effective agile retrospectives and what facilitators can do to get more value out of them.

An agile retrospective, or sprint retrospective as Scrum calls it, is a practice used by teams to reflect on their way of working, and to continuously become better in what they do. The whole team, including the product owner, attends the retrospective meeting. In the meeting they "inspect" how the iteration (sprint) has gone, and decide what and how they want to "adapt" their way of working to improve. Actions coming out of a retrospective are communicated and done in the next iteration, which makes retrospectives an effective way to do short-cycled improvement to deliver software faster with higher quality.

One technique that's often used in agile retrospectives is to ask questions to the Scrum team, and collect and cluster the answers to define improvements that the team will do in the next iteration. Asking questions is a technique that is easy to learn, but the effectiveness depends on the questions that you ask to the team. Questions that I often use and which have proven to help teams find things that they could improve upon are:

- What helps you to be successful as a team?
- How did you do this sprint?
- Where and when did it go wrong in this sprint?
- What do you expect, from who?
- Which tools or techniques proved to be useful? Which did not?
- Did we have any major bugs? How did we solve them?
- What is your biggest impediment?
- If you could change 1 thing, what would it be?
- What caused the problems that you had in this sprint?
- Where did things go wrong in coding/testing/etc?
- What's keeping you awake at night?
- Which things went smoothly in this sprint? Which didn't?

Here are some examples of the answers that came up using these questions:

- Several team members stated that they expect that the team starts and ends the stand-up on time and that all are present. Action decided was that someone who showed up late or was absent without notice had to buy the team a cake. Stand-up became shorter and more effective.
- Our biggest impediment is that we cannot test our software in a live environment. As a result, the team got access to the production environment to do integration tests.
- The team felt that things went smoothly because they focused on getting things done together and

helping each other. Since the company's reward system was still focused in individual behavior they started a discussion with their manager and HR on how to reward team behavior.

There's a risk of teams getting bored when retrospectives are always done in a similar way and the same actions come up. A solution to this is to introduce variation by using different retrospective exercises, and by making sure that actions get done (more on that below). Another thing that you can do is to chance the setting for the retrospective. Go outside, hop over to a nearby meeting center or training facility, visit a museum, or have a drink or dinner together. Basically try anything that moves team members out of the daily routine to help them take a fresh look at how they are doing and find new things to improve.

FACILITATING THE RETROSPECTIVE MEETING

The retrospective facilitator has to do everything needed to make retrospectives valuable for the team. This includes:

- Planning the meeting and assuring that team members can attend
- Selecting appropriate exercises and preparing them
- Running the retrospective meeting effectively
- Guarding the culture in the room, creating a safe environment where people speak up
- Setting the conditions for effective follow-up on the actions

The retrospective facilitator must have the proper skills to organize and lead the meeting:

- Excellent communication skills: Listen, ask questions, recap discussions
- Servant leadership: Process focused, independent, supportive to the team
- Maintaining an open culture: Encourage people, manage conflict, give feedback
- Goal-oriented: Planning and execution, able to adapt and improvise

Who facilitates the retrospective? In most teams it's the Scrum master who leads the retrospective, but it can also be another team member or a Scrum master from another team. Or it can be an agile coach, process responsible, quality manager, or even somebody from HR or a line manager. The position or role that the facilitator has is irrelevant; what matters is that the person can act as an independent facilitator and has the right skills.

My advice for a new team is to have an external facilitator for their first couple of retrospectives, instead of their Scrum master doing it. Such a team has just started on a journey to discover how to work in an agile way, and that means changing the way people work together. This change impacts everybody on the team, and will have a significant impact on how the Scrum master plays its role. Having to lead the retrospective, and reflect on the teams's and your own behavior and learn at the same time is hard. Therefore, my advice is to have the whole team, including the Scrum master, focus on the learning and get someone else to facilitate the meeting.

GETTING DOABLE ACTIONS, AND GETTING THEM DONE

Agile retrospectives are a great way to continuously improve the way of working. Getting doable actions out of a retrospective and getting retrospective actions done helps teams to learn and improve.

Actions can be technical, e.g. trying a new coding pattern or testing technique; process based, like doing code reviews or pairing; or in the way the team works together, like trying something new in the stand-up or changing the setup of the task board.

Real continuous improvement from retrospectives starts with making the results of the retrospective meeting actionable. Make sure that the team leaves the room with actions that they can and will do in the next iteration. Add those actions to the team status board and/or to the backlog, so that they are visible and will not be forgotten.

Here's an example from a team that I worked with. They did a retrospective, and found out that since there was a multi-disciplinary team, the skills and knowledge were spread out. In some cases there was only one person who was capable of doing certain work, which was undesirable to them.

They defined an action to "do more pairing." The action itself is still somewhat vague, but we agreed to use any opportunity for people to work together and develop new skills. This made the action doable.

During the iteration there was a task that was stuck. A piece of software needed to be tested, but the testers were busy testing other modules. A developer mentioned during the stand up that he wants to do it, but has little testing experience. The Scrum master asked the testers if one of them could pair with the developer to help him get started and learn how to do testing. He referred to the action on the task board, reminding the team that this was something that they decided to do in this iteration.

One of the testers stepped in, and the two people paired for just a couple of hours.

It turned out that this was more than enough for the developer to do the testing. With some additional questions he was able to finish the task. In next iterations he picked up more testing tasks and became better at testing. The team was able to do more testing this way, which helped them to get user stories finished.

In your actions you should also mention the "why:" the reason that you are doing the action, the problem you expect it to solve, the benefit that team will get from doing it, etc. This will motivate team members to do the action.

RETROSPECTIVE FACILITATION GOOD PRACTICES

To keep the retrospective effective, facilitators should focus on the following:

- Establishing an open and honest culture in the meeting
- Ensuring that all team members participate in the meeting
- Ensuring that the team establishes a shared understanding of how things went
- Helping the team to decide upon the vital few actions that they will take

Here's a couple of practices that I recommend for facilitating retrospectives and make them valuable for the participants (for more practices and tips, see 7 Retrospective Facilitation Good Practices).

Many retrospective facilitators use the prime directive to establish a culture where team members speak up and will be open and honest. It sets the assumption that team member did the best they could possibly do, and that the purpose of the retrospective is not to blame people.

Retrospective facilitators must be able to deal with negative issues. Help the team to focus on the issue and understand them and don't blame any team members for what has happened. This requires strong communication skills, being able to pay attention to verbal and non-verbal communication. For retrospectives with remote or distributed teams, facilitators should be able to make communication issues explicit and visible, keeping everyone involved and maintaining a good meeting culture.

A facilitator should serve the team in the meeting, and should not lead them based on their own opinion on what the team should do. It is important that a facilitator remains independent, which again can be challenging when the facilitator is also the Scrum master of the team.

RETROSPECTIVE EXERCISES

As a retrospective facilitator it's important to have a toolbox of retrospective exercises which you can use to design a retrospective. Before having a retrospective meeting, you want to prepare yourself by considering which exercises would be most suitable. That depends on the team, the situation at hand, and on what the team would like to work on.

Possible exercises that retrospective facilitators can use are a one-word retrospective, perfection game, exploring strengths with core qualities, and futurespectives. As an example, this is how you can do a one-word retrospectives where there are issues in a team that need to be discussed:

> Start by asking each team member to state how they feel about the past iteration in one word. Note these words on a flip chart or whiteboard. When everyone has spoken up, ask team members to describe why they feel this way. Stimulate a discussion where feelings are shared about the team that often don't reach the surface. Once team members recognize the problems, change the discussions towards actions to solve the team issues. Put these actions on the team board to make them visible for the next iteration.

My advice to retrospective facilitators is to learn many different retrospective exercises. The Retrospective Exercise Toolbox (benlinders.com/exercises) provides many exercises that facilitators can use to design valuable agile retrospectives. A great way to learn new retrospective exercises is by doing them. Practice an exercise, reflect how it went, learn, and improve yourself. You may need a specific exercise one day, so make sure that you are prepared.

BEN LINDERS is an Independent Consultant in Agile, Lean, Quality and Continuous Improvement, based in The Netherlands. Author of the books Getting the Value out of Agile Retrospectives What Drives Quality, and Continuous Improvement. As an adviser, coach and trainer he helps organizations by deploying effective software development and management practices. Ben shares his experience as a speaker, in a blog (Dutch and English) and on twitter @BenLinders.

How Collaborative Is Your Software Development Lifecycle?

By Eric Robertson, General Manager DevOps Product Line, CollabNet

Market needs shift rapidly today. You can improve development quality and speed in order to deliver better software faster. To do that, you have to increase cross-functional visibility, cooperation, and collaboration. This checklist will help you determine how collaborative your environment is.

	YES	NO	SOMEWHAT
1. Does your company still use the "throw it over the wall" approach at any stage?			
2. Is Agile training and culture being cultivated at the executive, manager, and developer levels? If not, assess training needs and start training.			
3. Are there key points in the development and delivery processes where breakdowns and delays occur? Examples of indicators and what to look for:			
1. Are competitors delivering innovation significantly faster than you?			
2. Have you ever developed functionality only to learn from sales that the market doesn't want it?			
3. Are marketing launches rushed and weak?			
4. Does sales have the tools and training to sell in advance of final delivery? Or are they playing catch-up?			
5. Do you have sufficient IT operations support all along the way?			
4. Who are the non-software personnel with unique processes who play a vital role in the market-readiness processes for a product? Typical examples are:			
1. Legal – IP, licenses			
2. Product Management – functionality, pricing, positioning			
3. Marketing – launches, go-to-market assets			
4. Sales – customer requirements, sales tools			
5. Services – training, deployment, other service			
5. Do those stakeholders have enough visibility to contribute earlier in the process? Examples of things to look for:			
1. Are product launches integrated?			
2. Does sales have customer input that should be considered earlier in the process?			
3. Does legal have enough lead time to process IP?			
6. Does the organization work cohesively to identify potential problems and solve them before development is too deep in the process?			
7. Are DevOps tools unified or being used in a silo?			
8. Do the DevOps tools you use facilitate cross-functional processes and collaboration?			
9. Do you use a platform to manage and connect the whole software delivery lifecycle, as well as to incorporate business teams and associated processes?			
10. Do you have a bird's-eye view of the entire project at all times?			

Overcoming Five Common Performance Testing Pain Points

BY **CARLO CADET**
DIRECTOR OF PRODUCT MARKETING AT PERFECTO MOBILE

QUICK VIEW

01 As the mobile and digital worlds continue to evolve, developers must focus on perfecting their digital presence and deliver superior customer experiences.

02 When it comes to their testing strategies, there are five pain points developers typically encounter, including slow manual testing, managing in-house testing labs, testing for various user conditions, optimizing apps throughout the lifecycle, and inconsistent testing.

03 It is no longer about mobile first, but instead about digital everywhere. As a result, organizations need to ensure that their digital experiences reflect the quality of their brand.

The stand-alone mobile strategy has become increasingly irrelevant. Why? The simple answer is that the mobile device has evolved into a single piece of a broader omni-channel strategy. Digital experiences are incorporating an infinite number of endpoints, sensors, and environments to engage users where they are via their mobile devices. "Digital" is also becoming the primary channel for reaching customers and building lasting loyal relationships.

As the importance of digital grows, so do the competitive pressures to deliver an experience that delights users. Users' attention spans are shortening and expectations are rising. Apps must perform flawlessly and deliver fresh content and UIs to remain compelling enough to keep users coming back.

However, this is all much easier said than done. Building great experiences in itself is a big undertaking, and ensuring that these experiences work the way they were designed to, across the digital ecosystem—including laptop browsers and mobile apps—requires consistently executed testing strategies.

As a result, testing and monitoring websites and apps are fundamental to successful strategies. Whether it's simply testing manually, building an in-house testing lab or leveraging cloud testing services, developers will experience multiple challenges perfecting their digital presence. Here are five pain points developers typically encounter within their testing strategies.

1. SLOW MANUAL TESTING

When new development teams embark on their testing regiment, the first pain point they often experience is lack of time. Developers will typically begin testing apps manually, which is slow and can create bottlenecks for the entire operation. Testers need to load apps onto multiple devices and test them in real-world situations. This might include a tour around town where testers evaluate app performances as they drive through tunnels or garages, enter elevators, or visit basements. The chance of human error with this approach can lead to overlooked bugs.

To overcome this pain point, many development teams turn to automation to alleviate inefficiency. In order to automate, teams often create their own testing labs by acquiring a variety of devices, as well as purchasing technologies to automate tests and simulate real-world situations while testing on real devices. As a result, this solution significantly improves testing efficiency, but also creates new problems.

2. MANAGING IN-HOUSE TESTING LABS

An in-house testing lab can be a difficult and expensive undertaking for many organizations. Up-front costs to buy 40 of the most popular devices can cost about $11,000, and new devices need to be constantly added in order to stay up-to-date. Connecting devices in the lab and installing apps on each device for every new build also requires considerable time. As development organizations expand, these labs also need to grow to handle increased demand, putting greater pressure on the lab.

Deciding on the most important devices and operating systems to include in a testing environment is also a point of stress when organizations build their own labs. Open Signal, a monitor of operator networks, identified 1,294 different Android device brands and 18,796 individual devices in the market between August 2014 and 2015. The enormous number of environments in which apps will be running will require testers to make difficult decisions about what assets to invest in.

Simulating all the potential user environments at scale is also a very challenging undertaking for a single testing team. Savvy development teams are outsourcing this functionality to experts that specialize in maintaining testing labs, and are able to spread these costs across a larger client base. While this strategy alleviates the headaches of managing an in-house testing lab, the many variables that exist in the real world will continue to cause problems for developers, which causes the third pain point – testing for user conditions.

3. TESTING FOR VARIOUS USER CONDITIONS

Testing apps to ensure they perform well on different device models, screen sizes, and operating system versions is a constant challenge for developers. As mobile and PC ecosystems merge, developers also must consider how their responsive experiences perform on laptop web browsers.

Functional testing can test basic app features on a variety of devices, but these approaches do not factor in the dynamic user environments that apps have to perform in. Understanding the effect that environmental factors have on app performance is key to optimal app performance.

Testing for every possible environmental condition may be unrealistic, but designing testing strategies that consider typical user conditions is a good way to avoid blind spots. For example, business travel apps should be tested on smartphones and in environments that simulate congested networks typical of airports.

Testing complex user environments prior to production is a great way to avoid blind spots, but technology and conditions are constantly changing. As a result, another key pain point includes optimizing apps throughout the lifecycle.

4. OPTIMIZING APPS THROUGHOUT THE LIFECYCLE

With code, platforms, and operating systems constantly changing, ending testing processes once code is out the door leaves apps vulnerable for failure in the future. DevOps teams that do not monitor apps post-production will only discover new bugs once reported by users, and will have to scramble to solve the issue based on limited historical data. At this point, it may be too late. Dimensional Research found that about 80 percent of users will only use a problematic app three times or fewer.

To solve this problem, developers are implementing strategies that regularly test apps already in-production to monitor their performance while the digital ecosystem changes. To minimize the time to resolve issues, developers have also implemented deep reporting and redundancy failure monitors that can quickly identify which devices and variables are creating the issues.

Regular testing throughout the app lifecycle is very important; however, testing also has to be consistent.

5. INCONSISTENT TESTING

Test scripts that are run in environments that are inconsistent can lead to developers chasing bugs that don't exist. Running different tests scripts can also lead to inaccurate results. Questions such as "did an app pass or fail due to changes in the code or differences in the lab?" can lead to a significant amount of wasted debugging time.

Organizations operating in-house testing labs are susceptible to a variability in power supply or differences caused by devices missing from the lab. Responsive experiences are incorporating both PC and mobile devices into a single experience, but often these development teams remain separate. This means different testing strategies and scripts. Responsive web experiences that fail on a desktop browser but not on mobile could be caused by a difference in the test script, making it more difficult to identify bugs.

By using cloud-based testing services, developers are able to maintain a consistent testing environment that have built-in redundancies. Leading cloud testing services are also offering capabilities that enable testing teams to test on both desktop and mobile web with one testing script.

LOOKING AHEAD

In the future, digital strategies need to engage customers with high-performing experience regardless of the location. It is no longer about mobile first, but instead about digital everywhere. As a result, organizations need to ensure that their digital experiences reflect the quality of their brand. Excellent digital engagement requires developers to avoid or overcome these five key pain points that reduce testing efficiency and quality overall.

CARLO CADET is Director of Product Marketing at Perfecto Mobile, the leading cloud-based mobile application testing and monitoring company. He is responsible for go-to-market strategy, product positioning, and messaging. Previously, he had similar responsibilities driving identity and access management, virtualization, and consumer identity protection initiatives at RSA Security. He earned an MBA from the MIT Sloan School of Management and a BSc in Computer Science from Tufts University.

just let me code

Spend less time fixing defects and more time building features

Perfecto

Accelerate web and mobile code validation with fast feedback using Perfecto's cloud platform

perfectomobile.com/justletmecode

Delivering New Features Means Less Rework Driven by Defects

While many teams aspire to increase velocity, too many are drowning in a backlog full of defects discovered late in the release cycle, or worse, in production. The result is that defects choke the effort to deliver new features.

Front-end development teams delivering responsive and mobile apps are meeting this challenge by a) increasing efforts to commit clean code and b) expanding automated test execution. Those winning the battle point to four critical enablers: 1) consensus on "Done"; 2) use of ATDD (Acceptance Test Driven Development); 3) easy & reliable access to common platforms in use; and 4) more recent adoption of UI Test Automation.

A common thread weaving between these four enablers is "test automation that consistently works." First and foremost, it behaves deterministically (zero flakiness) and executes fast (think instrumented UI Test frameworks). By embracing ATDD, teams build the right set of unit tests and functional tests that are executed both pre-commit and as part of each automated build. For front-end teams, expanding test automation requires implementing the right mix of functional, performance, navigation, user behavior, and, of course, real world rendering tests to minimize escaped defects that would be discovered later.

It's clear that winning the velocity race requires winning the race to embed quality into code development. Faster feedback is critical, but it all starts with committing cleaner code and the right mix of tests executed across the right platforms, conditions, and critical flows.

WRITTEN BY CARLO CADET
DIRECTOR, PRODUCT MARKETING, **PERFECTO MOBILE**

PARTNER SPOTLIGHT

Continuous Quality Lab By Perfecto

Perfecto

Perfecto mobilizes your brand by helping perfect the digital experiences that define it.

CATEGORY
Automated Testing Platform

NEW RELEASES
Every three weeks

OPEN SOURCE
No

CASE STUDY
Paychex is a leader in Payroll, HR, and Benefits Outsourcing solutions. Over a half million clients rely on the Paychex Flex app every day.

With an "automate everything" mindset, test automation has become ingrained within their culture and is reinforced as a defined element within their "Definition of Done." Using Perfecto's Continuous Quality Lab, they evolved from a manual testing approach to exercising hundreds of automated tests three or more times per day within their CI builds. Perfecto's Lab delivers the 24/7 reliability project teams require within their CI process.

Faster feedback is a critical enabler supporting the transition from quarterly to monthly releases out to production, as well as achieving continuous delivery.

STRENGTHS

- **Automation That Works:** The Forrester Wave™ Mobile Front-End Test Automation Tools LEADER
- **Real, Not Emulators:** Cloud architected quality platform—access to real devices and browsers
- **Test on Every Client:** Automated and manual testing—Desktop and Mobile Web, Hybrid and Native apps
- **Analytics You Care About:** Continuous quality visibility on status and trending

NOTABLE CUSTOMERS

- Lego
- Virgin Media
- ING
- Kaiser Permanente
- Marriott

BLOG blog.perfectomobile.com **TWITTER** @perfectomobile **WEBSITE** perfectomobile.com

Executive Insights on the Software Development Lifecycle

BY **TOM SMITH**
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01 The key to developing better software more quickly is to follow a DevOps/CI/CD methodology automating as many processes as possible.

02 The proliferation of architectures, frameworks, and languages has led to the necessity to adopt DevOps/CI/CD and automation.

03 The future for developers is DevOps. Start embracing operations and DevOps methodologies or be left behind.

To gather insights on the state of the software development lifecycle, we spoke with 19 executives at 16 different companies involved in software development for themselves and software development services for their clients. Specifically, we spoke to:

SAM REHMAN, CTO, Arxan

JOHN BASSO, CIO and Co-Founder, Amadeus Consulting

JOHN PURRIER, CTO, Automic

LASZLO SZALVAY, Director of Sales and Partnerships, cPrime

SCOTT ROSE, Senior Director of Product Management, and

MIKA ANDERSON, Product Manager, CollabNet

JEAN CAJIDE, VP of Corporate Development and

SAMER FALLOUH, VP of Engineering, Dialexa

ANDREAS GRABNER, Technology Strategist, and

BRETT HOFER, Global DevOps Practice Leader, Dynatrace

ANDERS WALLGREN, CTO, Electric Cloud

ALEXANDER POLYKOV, CTO, ERPScan

BARUCH SADOGURSKY, Developer Advocate, JFrog

ROB JUNCKER, VP of Engineering, LANDESK

MIKE STOWE, Developer Relations Manager, MuleSoft

ZEEV AVIDAN, VP of Product Management, OpenLegacy

JOAN WRABETZ, CTO, Quali

SUSHIL KUMAR, Chief Marketing Officer, Robin Systems

NIKHIL KAUL, Product Marketing Manager, SmartBear

KEY FINDINGS

01 The keys to developing software in a timely manner is following an Agile/DevOps methodology and all of the elements inherent in that methodology—communicating throughout the software development team, continuous delivery, continuous integration, fast iteration, and automation of all manual tasks to facilitate further acceleration of the process. Start by spending time on discovering what the client and end user wants and plan how to deliver it. Have a culture where people are excited about putting out high-quality software and learning what can be done to improve based on customer feedback.

02 The biggest changes in the development of software have been the proliferation of architectures, frameworks, and languages. However, the movement from Waterfall to Agile/DevOps has made it possible for apps to build to scale reliably through replication and automation, leading to greater quality and predictability in the software development lifecycle. Teams are collaborating on a regular basis and cycle times are in hours and minutes rather than weeks and days. Every step of the development process is faster due to the technology and less manual intervention. In addition, Agile-based platforms are designed for connectivity, which is critical given the different platforms and operating systems upon which applications need to work.

03 While more than 50 solutions were mentioned, the technical solutions most frequently mentioned were Java, Node.js, .NET, open source, and PHP. Several companies are building their own solutions on top of open-source platforms. All the respondents seemed to be using a range of technical solutions to ensure they'd have the right tools for the job based on their customers' needs and wants, as well as to optimize performance.

04 The real world problems companies are trying to solve involve helping build software that helps run companies. This may entail a new product or service to enable a client to succeed quickly, inexpensively, and flexibly. One provider enables companies to test their applications in a "real world" sandbox without going into production while another provides automation systems that enable companies to scale out into bigger containers and microservices. Several provide visibility and control over the software development pipeline and complete insight into the health of the application with method-level granularity. Many enterprise companies need to overcome technical debt and move towards more Agile methods, as well as efficiently managing demand to integrate a variety of systems.

05 The most common issues affecting software development today are: 1) lack of planning; 2) time to market; 3) lack of awareness of security issues; and 4) cultural issues that hinder the adoption of a DevOps methodology. The standards for software development are out there, but they're not enforced. This causes trouble with integration and the speed with which systems and tools are moving leads to faster deprecation with tools not being supported. Minimizing time to market is the greatest benefit of Docker and automation, which can handle greater complexity and customization. Many companies are not aware of security issues and it's a challenge ensuring users are secure with everyone interacting with multiple applications and platforms. Lastly, fiefdoms and governance models that are already in place need to change from silos to making developers responsible for a stable environment and administrators used to the changes needed for continuous integration (CI) and continuous delivery (CD).

06 The biggest opportunities for improvement in the development of software are getting standards nailed down and enforced to help the software development lifecycle become smaller and more nimble, and to have a faster release by automating development, testing, and monitoring. Ultimately, enterprises will be able to develop software with the same ease and speed as start-ups. A large enterprise can interact across functional groups to increase velocity and quality, which in turn adds value of the product. DevOps, CI, and CD enable process optimization. Getting problems solved upstream mitigates time and effort waste downstream.

07 The biggest concerns around software development today are security, complexity/rate of change, and that culture change needs to implement DevOps. People don't realize the need to be concerned about security around applications. As IoT grows, it reaches deeper into our lives. The opportunity for mischief or errors is greater. We've lost the focus on the privacy of users' data. As tech is changing at a rapid pace, it's difficult to stay abreast of all the changes. Particularly, it's impossible for a client whose core business is not IT. We need to be concerned with how our clients digest all of these changes. Fiefdoms and silos need to be taken down to evolve to a DevOps culture. People in established enterprises are afraid of change and what it will mean for their jobs. They're not being objective about what is in the best interest of the organization to succeed moving forward.

08 Skills most frequently mentioned as needed by developers to create software that delivers value to customers include understanding customer needs and wants, staying up to date with current technologies, and taking an interest in operations since that's where the future lies. Developers need to understand what users are looking for and put the users at the center of what they're developing. They should build apps that are easy for users to get around. In the end, customers are often more concerned with the fact that it works and is usable. Get code in front of customers and get feedback early. You cannot iterate quality in. Be knowledgeable about scalability and large-cloud offerings. Stay on top of new developments, but shift left. Testing is moving to developers. While you would rather learn a new language than operations, very soon there will just be DevOps engineers and leftover developers. Value will be added, and significant salary earned by DevOps engineers.

09 As usual, respondents have a variety of different considerations with regards to software development today:

- The credibility of the sources developers are using, including open source, which is disrupting commercial tools.
- What gets priority—due dates or features?
- We need to keep an eye on the skillset gap of the people we are hiring and plan for the development of people in school to be better prepared for the jobs that we'll need talent for in the future.
- We need to talk about deployment and operation in addition to development to ensure end users are having a good experience with the application. How can we help improve the quality of apps at the beginning of development?
- Are developers moving into testing and development? Do they understand the need to do so? Do they view DevOps as an opportunity or a disruptor?
- What's the next thing to disrupt software development? Containers and microservices are changing the game today—what's on the horizon? Where do truly innovative companies like Apple come up with their ideas? We need to be able to trace the evolution of ideas.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

TDD Strategy in Real Life

QUICK VIEW

01 How do we use TDD when writing code in a complex system?

02 Which questions do we need to answer before writing the code?

03 How do we make decisions when constrained by existing design?

04 When are we really done?

BY **GIL ZILBERFELD**
AGILE COACH

TDD is always described as a design method for units, whether these are classes or modules. In the real world, however, we don't write just classes. We're writing complete flows that consist of classes, APIs, generated code, and data schemas. We still want to get the benefits of TDD, so how can we fit it into our work?

TDD comes from the world of unit testing. It's optimized for small pieces of code; small increments of functionality. BDD takes the test-first approach, adds functional and user semantics and tries to follow the same formula for the whole software.

TDD helps us build a class or a module. But in real life, we'll need to build complete scenarios. BDD helps in defining those flows, but doesn't help us design the software, like TDD does. Not to mention if we're constrained with existing systems.

There's an impedance mismatch. On the big picture side, we have stories (or even epics) that take a while to complete (especially if you don't write them from scratch, and you already have components you need to maintain, upgrade, and interact with).

On the other side of the scale, TDD allows you to design components, but it doesn't provide the feedback about all the flow. Also, it doesn't tell us if the *emergent design* is the "right" one. (PSA: There is not only "one" design, this is not Highlander.)

So what approach should we take? We want to use TDD for its benefits, but we have existing constraints we need to adhere to. If we're writing a complete new feature, we don't have any design at all.

Here's my approach. It requires some planning and thinking ahead, which never hurt anyone. Much.

Let's take it step-by-step.

STEP 1
IDENTIFY THE IMPORTANT STORIES
First, we need to identify the main stories (flows) we're going to write. If we have them in a form of BDD tests, great. If not, we need to at least have a list of the stories/ requirements that are important.

Why do this before designing a big solution that answers all the requirements? If we start before identifying the important stuff, we'll come up with an over-engineered architecture that has a lot of unnecessary code (*YAGNI*) and what it always bring to the party: bugs.

Like with TDD, our intention is to write the minimal code that works. However, in integration or system tests, we need to develop the minimal design that works.
That means we'll try to build incrementally, completing a story before moving to the next, and NOT completing a component for all requirements.

One more thing: it is very important we state the requirements or stories as a flow in the system. That means operations that the user or the system does, where data is transferred and operations are completed. If that is not the case, we'll find ourselves in this stage again after visiting step 2.

STEP 2
DEFINE THE ACCEPTANCE CRITERIA

This is extremely important, because if we just stick to "this flow should work" we're leaving it open ended. And with that, we'll be out of focus, building things we don't need. Like with TDD, we specify how we expect to execute and check the behavior we're interested in. And that requires that we know what the acceptance criteria is.

With BDD-style tests—which are really examples—it's easy. Although even with those, sometimes we leave the test in the Gherkin stage, and only later when we have something to connect do we do that.

Eventually, in a good agile process, the whole-team approach works best with backlog prioritization.

With plain stories and requirements, we run the risk of implementing something that is not needed. And, by specifying an acceptance criteria, we get the sense of the effort and complexity involved in the implementation. If we can't say how we're going to test the story, that's a sign we need to do more thinking before jumping to development. Those might lead us to re-prioritize stories in step 4.

STEP 3
IDENTIFY THE MAIN COMPONENTS THAT THE STORIES FLOW THROUGH

They can already be completed, half written, or non-existent. A story flows through the components, and it may either use them as-is, or it may need interface modification or extension, new coding, or a full rewrite.

While this is a design step—we're proposing a solution to how we're going to make the story work—it may not be the final one. We're doing just enough design up front. This helps to confirm the feasibility of a solution.

When we map the stories to components, we can see options for design. We can see how the different pieces of the puzzle fit, and if we have problems we need to attend to.

STEP 4 – RE-PRIORITIZE THE STORIES

"Wait, now the developers prioritize?"

I know, it's sounds a bit anti-Scrum, but hear me out.

Even after someone (product owner or similar) had already prioritized the stories in terms of value, there's still valid input that can come from the dev team.

Sometimes, there are dependencies between the stories. While story #1 is more valuable than #2, we won't release without the first 10 stories completed. That means we can play with the order, if there's a good explanation for it. In this case, if we can't easily test or develop story #1 without having #2 in place first, we'd probably want to develop #2 first.

Note, we're not questioning the value of the stories, rather how quickly we can complete them. If we deliver complete stories early, we get feedback early and can make decisions based on this feedback.

If, however, everything we do is suspect to change, and learning is what leads all the stories, then the higher value stories' delivery is more important, and we go with those.

Agile note: If you want to think about it as the team providing information about risk and effort and then the PO re-prioritizes the list—fine. Eventually, in a good agile process, the whole-team approach works best with backlog prioritization.

STEP 5
COMPONENT DECISIONS

Not going into the emergent discussion again, there is still room for some upfront design. Before we start coding we can make additional design decisions. For example, if the stories go through an existing component, one that is buggy and doesn't have any tests, we need to do something about it. If we just patch on code (TDD or not), there is still risk of breaking existing functionality. We can decide to wrap this component in tests, rewrite it, or build around it. This is not a design decision we'll make as part of the actual TDD cycle, but it has an impact on how **our** component interacts with the rest of the system.

Other decisions we might make are if we're not satisfied with the current design (are we ever?).

We can think about breaking down big modules or combining small modules, as part of the work we'll do. This is architectural refactoring—there are changes we plan to do in the architecture, without modifying existing functionality. We're doing this because we want to get a sense for where we're going. We want boundaries to guide us. Constraints are helpful, remember?

"Complete" means having some proof that the entire story works.

STEP 6
INTERFACE DECISIONS

We can also start identifying which connection points to use. While this is not a full interface definition between components (it's a bit early for that), we can decide on reusing existing interfaces, or adding or modifying them. Of course, if all the code is new, we can define whatever we want.

In TDD, we define the interface of our components through tests. If the interface depends on usage by existing consumers, this has an effect on our design.

We can agree on interface contracts if these can be "managed." Why the quote marks? When we discover we need changes to the interfaces (and we will), they should be easy to change—meaning, not too much work, or not introducing big risks. We don't want to define interfaces that take weeks to re-discuss, or re-test. If the interfaces are defined within a team, these are easily managed, so go ahead. If not, make sure not to close the definition too early.

STEP 7
WRITE THE CODE

Finally, we're getting to the coding bit.

Remember, we're doing it by the book. We'll write just enough code for the **story**. That's right—not a whole component at once. If you can use TDD at this point, great. Just stop when you get to the end of what the component needs to do for that story alone. If test-first won't work (for any reason) make sure you write tests when appropriate.

The **worst** we can do is to write the components "fully" for all the stories. What we'll end up with is "mostly-working complete" components.

STEP 8
INTEGRATE CONTINUOUSLY

I don't mean that in the CI-tool sense. Make sure you are always integrating the whole story to see how it works. You'll learn what works, and what needs to be changed. Then you adapt.

Since you and your team are focused on only one story, there's a lot less code to integrate. The integration is far less complex than integration of multiple, full-bodied components.

Obviously, we alternate between steps 7 and 8 until...

STEP 9
COMPLETE THE STORY AND MOVE TO THE NEXT

"Complete" means having some proof that it—the entire story—works. The proof should come in the form of an automated end-to-end test and a product review in order to get feedback. This feedback can (and will) change things, in terms of what comes next.

Then again, the waste is not going to be big if we built just enough for the first story.

And that's the important thing. Every code we have is a liability. It can contain bugs, it will require change, and we'd like to write as little as we can of it, just to make sure we're on the right track. TDD helps us with the design and with creating a risk mitigation (in the form of a regression test suite) that will help with the upcoming changes.

All the upfront thinking, considerations, identifying constraints and boundaries—they are all helpful in using TDD in the right context. Otherwise we'll be building the wrong thing right. We don't want that waste.

GIL ZILBERFELD has been in software since childhood, writing BASIC programs on his trusty Sinclair ZX81. With more than twenty years of developing commercial software, he has vast experience in software methodology and practices. Gil is an agile consultant, applying agile principles over the last decade. From automated testing to exploratory testing, design practices to team collaboration, scrum to Kanban, and lean startup methods—he's done it all. He is still learning from his successes and failures. Gil speaks frequently in international conferences about unit testing, TDD, agile practices and communication. He is the author of "Everyday Unit Testing", blogs at gilzilberfeld.com and in his spare time he shoots zombies, for fun.

Diving Deeper

INTO AGILE

TOP #AGILE TWITTER FEEDS

To follow right away

@benlinders	@jeffsutherland
@mikewcohn	@pragdave
@johannarothman	@gil_zilberfeld
@romanpichler	@krubinagile
@martinfowler	@jamesmarcusbach

TOP AGILE REFCARDZ

Git Patterns and Anti-Patterns

dzone.com/refcardz/git-patterns-and-anti-patterns

Suggests patterns and anti-patterns, including Hybrid SCM, Git champions, blessed repository, per-feature topic branches, and ALM integration.

Design Patterns

dzone.com/refcardz/design-patterns

Learn design patterns quickly with Jason McDonald's outstanding tutorial on the original 23 Gang of Four design patterns, including class diagrams, explanations, usage info, and real world examples.

Designing Quality Software

dzone.com/refcardz/designing-quality-software

Provides a comprehensive list of design rules, programming rules, testing rules, environment rules, and common sense rules to build better software.

TOP AGILE WEBSITES

The Agile Manifesto agilemanifesto.org

Scrum Guide scrumguides.org/scrum-guide.html

Agile Alliance agilealliance.org

AGILE ZONES

Learn more & engage your peers in our Agile-related topic portals

Agile

dzone.com/agile

In the software development world, Agile methodology has overthrown older styles of workflow in almost every sector. Although there are a wide variety of interpretations and specific techniques, the core principles of the Agile Manifesto can help any organization in any industry to improve their productivity and overall success. Agile Zone is your essential hub for Scrum, XP, Kanban, Lean Startup and more.

DevOps

dzone.com/devops

DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations/ system admins. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Performance

dzone.com/performance

Scalability and optimization are constant concerns for the developer and operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection, tool and technique comparisons, and tweaks to keep your code as efficient as possible.

RESOURCES

Extreme Programming Explained

amzn.to/2hbkkj7

The Ultimate Guide to the SDLC

ultimatesdlc.com

Test-Driven Development by Stephen Wells

bit.ly/2hbnsLP

Solutions Directory

This directory contains solutions for source control, test management, project management, code review, static code analysis, and issue management. It provides feature data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

1. SOURCE, PROJECT, AND TEST MANAGEMENT

COMPANY NAME	PRODUCT	CATEGORY	HOSTING	FREE TRIAL	WEBSITE
Acunote	Acunote	Project Management	SaaS	30 Day Free Trial	acunote.com
Aha!	Aha!	Project Management	SaaS	30 Day Free Trial	aha.io
Appvance	Appvance	Test Management	On-Premise or SaaS	n/a	appvance.com
Asana	Asana	Project Management	SaaS	Free for teams of up to 15 people.	asana.com
Assembla	Assembla	Project Management	SaaS	15 Day Free Trial	assembla.com
Atlassian	Bitbucket Server	Source Control, Code Review	On-Premise	30 Day Free Trial	atlassian.com/software/bitbucket/server
Atlassian	Confluence	Project Management	On-Premise or SaaS	Available Upon Request	atlassian.com/software/confluence
Atlassian	JIRA	Issue Tracking	On-Premise or SaaS	7 Day Free Trial	atlassian.com/software/jira
Basecamp	Basecamp	Project Management	SaaS	60 Day Free Trial	basecamp.com
Blossom	Blossom	Project Management	SaaS	14 Day Free Trial	blossom.co
CA Technologies	Application Quality and Testing Tools	Test Management	On-Premise	Available Upon Request	ca.com/us/products/develop-test.html
CA Technologies	CA Test Data Manager	Test Management	On-Premise	Available Upon Request	ca.com/us/products/ca-test-data-manager.html
CA Technologies	Endevor	Source Control	On-Premise	Available Upon Request	ca.com/us/products/ca-endevor-software-change-manager.html
CA Technologies	PPM	Project Management	On-Premise	Available Upon Request	ca.com/us/products/ca-project-portfolio-management.html
CA Technologies	Rally	Project Management, Test Management	On-Premise or SaaS	Free Community Edition	rallydev.com
Citrix	Podio	Project Management	SaaS	Available Upon Request	podio.com

COMPANY NAME	PRODUCT	CATEGORY	HOSTING	FREE TRIAL	WEBSITE
CollabNet	ScrumWorksPro	Project Management	On-Premise	30 Day Free Trial	collab.net/products/scrumworks
Collabnet	TeamForge	Source Control, Project Management	On-Premise or SaaS	30 Day Free Trial	collab.net/products/teamforge
DZone	AnswerHub	Project Management, Ideation, Knowledge Management, Collaboration	On-Premise or SaaS	Free trial is 15 days, with access to all features	dzonesoftware.com
Edgewall Software	Trac	Issue Tracking	On-Premise	Open Source	trac.edgewall.org
FitNesse	FitNesse	Test Management, Project Management	On-Premise	Open Source	fitnesse.org
Flow	Flow	Project Management	SaaS	15 Day Free Trial	getflow.com
Fog Creek	FogBugz	Issue Tracking, Project Management	On-Premise or SaaS	7 Day Free Trial	fogcreek.com/fogbugz
Fog Creek	Trello	Project Management	SaaS	Free, Gold Edition Available	trello.com
GitHub	GitHub	Source Control, Issue Tracking, Code Review	On-Premise or SaaS	45 Day Free Trial	github.com
HPE	ALM	Project Management	On-Premise or SaaS	On-Premise: 60 Day Free Trial, SaaS: 30 Day Free Trial	www8.hp.com/us/en/software-solutions/application-lifecycle-management.html
HPE	Quality Center Enterprise (QC)	Issue Tracking, Test Management	On-Premise or SaaS	Available Upon Request	www8.hp.com/us/en/software-solutions/quality-center-quality-management
IBM	IBM Rational	Source Control, Issue Tracking, Project Management	On-Premise	90 Day Free Trial	ibm.com/software/rational
JetBrains	IntelliJ IDEA	Source Control, Static Code Analysis	On-Premise	Free Community Edition; 30 Day Free Trial for Ulitmate version	jetbrains.com/idea
JetBrains	YouTrack	Project Management, Issue Tracker, Change Management	On-Premise or SaaS	30 Day Free Trial	jetbrains.com/youtrack
LeanKit	LeanKit	Project Management	SaaS	30 Day Free Trial	leankit.com
Micro Focus	Accurev	Source Control	On-Premise	30 Day Free Trial	microfocus.com/products/change-management/accurev
Micro Focus	Silk Portfolio	Issue Tracking, Project Management	On-Premise	45 Day Free Trial	microfocus.com/products/silk-portfolio
Microsoft	Team Foundation Server (TFS)	Project Management, Test Management, Source Control, Issue Tracking	On-Premise	Available Upon Request	visualstudio.com/en-us/products/tfs-overview-vs.aspx
Microsoft	Visual Studio	Source Control, Issue Tracking, Static Code Analysis	On-Premise or SaaS	Available Upon Request	visualstudio.com
Mozilla	Bugzilla	Issue Tracking	On-Premise	Open Source	bugzilla.org
Neotys	NeoLoad	Test Management	On-Premise	Free for up to 50 users	neotys.com/neoload
Oracle	Application Quality Management	Test Management	On-Premise	Free	oracle.com/technetwork/oem/app-quality-mgmt
Original Software	Application Quality Management (AWM) Solution	Test Management, Project Management	On-Premise	Available Upon Request	origsoft.com/products/qualify/

COMPANY NAME	PRODUCT	CATEGORY	HOSTING	FREE TRIAL	WEBSITE
Parasoft	Development Testing Platform	Test Management	On-Premise	Available Upon Request	parasoft.com/solution/development-testing
Perfecto Mobile	Perfecto Mobile	Test Automation	On-Premise or SaaS	2 Free hours	perfectomobile.com/solutions/perfecto-test-automation
Pivotal	Pivotal Tracker	Project Management	SaaS	30 Day Free Trial	pivotaltracker.com
ProductPlan	ProductPlan	Project Management	SaaS	30 Day Free Trial	productplan.com
QASymphony	qTest Manager	Test Management	On-Premise or SaaS	14 Day Free Trial	qasymphony.com/software-testing-tools/qtest-manager/test-case-management/
Redmine	Redmine	Project Management	On-Premise	Open Source	redmine.org
SauceLabs	SauceLabs	Test Management	SaaS	14 Day Free Trial	saucelabs.com
Shore Labs	Kanban Tool	Project Management	On-Premise or SaaS	14 Day Free Trial	kanbantool.com
SmartBear	SoapUI	Test Management	On-Premise	Available Upon Request	soapui.org
SmartBear	TestComplete Suite	Test Management	On-Premise	30 Day Free Trial	smartbear.com/product/testcomplete
Soasta	TouchTest	Test Management	On-Premise or SaaS	30 Day Free Trial	soasta.com/mobile-testing
SolutionsIQ	Agile Transformation Solution	Project Management, Agile Transformation, Professional Services	n/a	Available Upon Request	solutionsiq.com
Sprintly	Sprint.ly	Project Management	SaaS	30 Day Free Trial	sprint.ly
TargetProcess	TargetProcess	Project Management	On-Premise or SaaS	Free, Standard; On-Site packages, 30 Day Free Trial	targetprocess.com
Tasktop	Tasktop	Code Review, Project Management	On-Premise	Available Upon Request	tasktop.com
Telerik / Progress	Test Studio	Test Management	On-Premise	Available Upon Request	telerik.com/teststudio
TestPlant	EggPlant	Test Management	On-Premise or SaaS	Available Upon Request	testplant.com/eggplant/testing-tools/
ThoughtWorks	Gauge	Test Management	On-Premise	Open Source	getgauge.io
ThoughtWorks	Mingle	Project Management	SaaS	30 Day Free Trial	thoughtworks.com/mingle
Tricentis	Tricentis Tosca Testsuite	Test Management	On-Premise	14 Day Free Trial	tricentis.com/tricentis-tosca-testsuite
VersionOne	VersionOne Lifecycle	Project Management	On-Premise or SaaS	30 Day Free Trial	versionone.com/product/lifecycle/overview
WhiteHat Security	Sentinel	DAST, SAST, Mobile Application Security Testing	SaaS	30 Day Free Trial	whitehatsec.com/products
Xamarin	Test Cloud	Test Management	SaaS	30 Day Free Trial	xamarin.com/test-cloud
XebiaLabs	XL TestView	Test Management	On-Premise	Available Upon Request	xebialabs.com/products/xl-testview
Zephyr	Zephyr	Test Management	On-Premise or SaaS	Free Community Edition	getzephyr.com

2. STATIC CODE ANALYSIS

COMPANY NAME	PRODUCT	CATEGORY	LANGUAGE SUPPORT	FREE TRIAL	WEBSITE
Atlassian	Clover	Code Coverage	Java, Groovy	30 Day Free Trial	atlassian.com/software/clover
bitHound	Bithound.io	Code Review	JavaScript	90 Day Free Trial	bithound.io
Black Duck	Black Duck	Open Source Auditing	All Major Languages	14 Day Free Trial	blackducksoftware.com
Checkstyle	Checkstyle	Static Code Analysis	Java	Open Source	checkstyle.sourceforge.net
Code Climate	Code Climate	Static Code Analysis	PHP, Ruby, JavaScript, Python	14 Day Free Trial	codeclimate.com
CodeNarc	CodeNarc	Static Code Analysis	Groovy	Open Source	codenarc.sourceforge.net
CQSE	ConQAT	Static Code Analysis	Java, C#, C++, Javascript, ABAP, Ada, and many other languages	Free	cqse.eu
DevExpress	CodeRush for Visual Studio	Static Code Analysis	C#, VB10, ASP, .NET, HTML, JavaScript, XAML, C++	30 Day Free Trial	devexpress.com/products/coderush
Facebook	Infer	Static Code Analysis	Objective-C, Java, or C	Open Source	fbinfer.com
Findbugs	Findbugs	Static Code Analysis	Java	Open Source	findbugs.sourceforge.net
Flexera Software	Palamida	Open Source Auditing	All Major Languages	Open Source	palamida.com/products
Gitcolony	Gitcolony	Code Review	All Major Languages	30 Day Free Trial	gitcolony.com
Hello2Morrow	Sonograph	Static Code Analysis	Java, C#, C/C++	Free for non-commercial use	hello2morrow.com/products/sonargraph
JetBrains	Upsource	Code Review	JavaScript	Free 10 User Plan	jetbrains.com/upsource
NCover	Code Central	Code Coverage	All Major Languages	21 Day Free Trial	ncover.com/products/code-central
Parasoft	Parasoft	Static Code Analysis	C, C++, Java, .NET(C#, VB.NET, etc.), JSP, JavaScript, XML, and other languages	Available upon request	parasoft.com
RogueWave	Klocwork	Static Code Analysis	C, C++, C#, and Java	Available upon request	klocwork.com/products-services/klocwork
RogueWave	Open Logic Enterprise	Open Source Auditing	All Major Languages	Free Edition	roguewave.com/products-services/open-source-support
SonarSource	SonarQube	Static Code Analysis	All Major Languages	Open Source	sonarqube.org
Squale	Squale	Static Code Analysis	Java, C/C++, .NET, PHP,, Cobol, and others	Open Source	squale.org
Synopsys	Coverity SAVE	Static Code Analysis	C, C++, C#, and Java source code	30 Day Free Trial	coverity.com/products/coverity-save

DZone

GLOSSARY

ACCEPTANCE CRITERIA Pass/fail conditions that a piece of software or a sprint must meet in order to be considered finished.

AGILE A set of software development principles focused on adapting to changing requirements and continuously improving products and processes.

AGILE COACH A long-term consultant focused on transitioning an organization to practice agile development.

BACKLOG A collection of feature requests and user stories that have not been completed and are not scheduled for any upcoming iterations.

BEHAVIOR-DRIVEN DEVELOPMENT A requirement born out of test-driven development, where tests of any unit of software should be specified in terms of the desired behavior for the business case of the unit.

CONTINUOUS DELIVERY The automation and optimization of deploying software to production as soon as changes have been made.

CONTINUOUS INTEGRATION A development practice that requires developers to submit code to a shared repository to be tested before being deployed to production.

DEPENDENCY A piece of software that is relied on by another piece of software in order to function as intended.

EMERGENT DESIGN The practice of letting software design be determined by pieces of that software as it's built, rather than designing the application at the start of the SDLC.

FLOW Proactive action to remove barriers for team members so work is completed in a timely manner with as little difficulty as possible.

FUNCTIONAL TESTING Comparing the observed behavior of a software component to the intended behavior based on documentation and user stories.

LEAN APPROACH The philosophy of creating a product with as little waste as possible, inspired by Toyota's manufacturing process.

OPEN SOURCE A classification for software whose source code is available to modify or use at no cost.

PRODUCT OWNER A team member who is the key stakeholder of a project, responsible for prioritizing user stories and backlogged issues.

REFACTORING The process of changing the structure of an application without changing the external behavior.

REGRESSION TESTING Ensuring that software that was previously developed and passed tests still functions as intended after changes have been made to other parts of the application.

RELEASE CANDIDATE Any version of a piece of software that could be released as a final product.

REPLICATION Storing the same data on several different devices to improve fault tolerance and stability and create backups.

REQUIREMENTS What a piece of software needs to do, as defined by the business and by user stories.

RESPONSIVE USER EXPERIENCE A web design principle that allows web apps to be viewed in different ways depending on the size of the device being used to view them.

RETROSPECTIVES Meetings at the end of sprints or iterations, where the team discusses the iteration and how to improve processes going forward.

SCRUM An agile methodology focused on several small teams independently working in short sprints or iterations. Scrum also involves daily planning meetings and regular retrospectives on how to improve sprints in the future.

SCRUM MASTER A team member who manages communication between teams practicing Scrum and organizes regular planning meetings and retrospectives.

SPRINT A regular, repeatable cycle of time to work on particular pieces of software, usually one to two weeks. Also called iterations.

TECHNICAL DEBT The future development work that will have to be done when a piece of code, usually a "quick fix," is implemented without proper checks or testing, or without being communicated to the team.

TEST-DRIVEN DEVELOPMENT A development strategy in which tests that are meant to fail are written and new code is only added to a project if it passes those tests.

UNIT TESTING The practice of testing the functionality of the smallest usable parts of an application, such as a class in object-oriented programming.

USER STORIES A description of how an end user uses a piece of software to do a specific task.

WATERFALL A development process that defines all possible requirements, design, architecture, and deadlines for a piece of software before development starts.

YAGNI Abbreviation for "You Aren't Gonna Need It," referring to code that does not add functionality to pass tests or meet requirements.