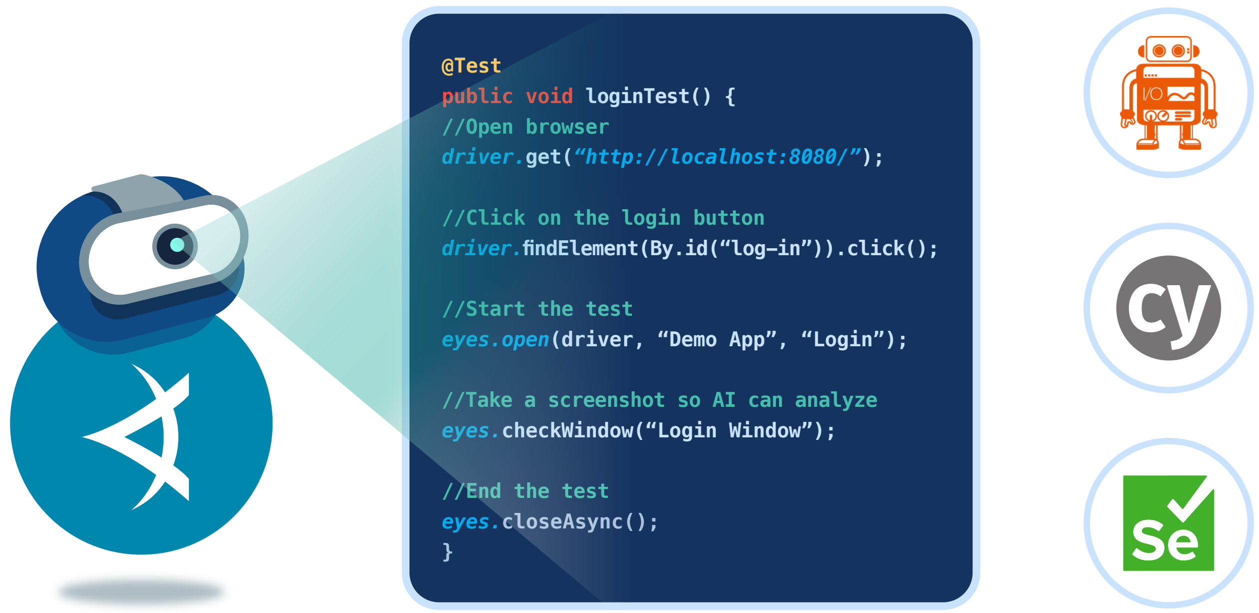
The Impact of Visual AI on Test Automation

Empirical Data from 288 Cypress.io, Selenium, and WebdriverIO Quality Engineers



```
@Test
public void loginTest() {
//Open browser
driver.get("http://localhost:8080/");

//Click on the login button
driver.findElement(By.id("log-in")).click();

//Start the test
eyes.open(driver, "Demo App", "Login");

//Take a screenshot so AI can analyze
eyes.checkWindow("Login Window");

//End the test
eyes.closeAsync();
}
```

applitools

AI-Powered Visual Testing & Monitoring

Quality Engineering Today

Why Read This Report?

In just 10 minutes, you will learn how you and your team can release higher quality apps faster and for less time and money than previously thought possible because of Visual AI. Sourced from 288 quality engineers who spent a **combined 1.5 years of quality engineering effort**, this study takes five of the most common UI/UX testing use cases and provides clear evidence of Visual AI's ability to augment, not rip and replace, code-based frameworks such as Selenium, WebdriverIO, and Cypress.io to help engineers expand test coverage and ship quality code at the speed of CI/CD.

Quality Management for Modern Applications is Challenging.

The vast majority of engineering teams are struggling with their quality engineering efforts. 68%* of us cite quality management as a key blocker to more agile releases and ultimately CI/CD, a top goal for 59%* of companies[1]. Efforts to have front-end developers own quality (e.g. shift left) are showing limited success, or failing outright in some cases. As a result, test automation "investment fatigue" is a concern among many technical leaders -- a frustration that investments in quality engineering are not delivering ROI as needed.

The Impact of Visual AI on Test Automation

What Is Visual AI?

Visual AI is a form of computer vision invented by Applitools in 2013 to help quality engineers test and monitor today's modern apps at the speed of CI/CD. Visual AI inspects every page, screen, viewport, and browser combination for both web and native mobile apps and reports back any regression it sees. Visual AI looks at applications the same way the human eye and brain do, but without tiring or making mistakes given it's 99.9999% level of accuracy.

Why Research the Impact of Visual AI on Test Automation?

According to Gartner[2], "Application leaders should embrace AI-augmented development now, or risk falling further behind digital leaders." AI promises to solve many modern technical problems, including testing and quality management problems, but it's hard to separate the truth from the reality in what really works. Many experiments using AI have failed in testing, or these AI approaches require opaque rip and replace investments that are not realistic for teams. Rather than making a promise and asking you to just trust us, we decided to prove it, objectively, using real world examples, in partnership with real testers at real companies dealing with test automation everyday.

Data Sourced From 1.5 Years of Quality Engineering Effort

Executing The Research. The Visual AI Rockstar Hackathon.

To execute the study, we built an application representative of 5 common, modern app functional testing use cases. In November 2019, a challenge was issued to testers all over the world to compete, and learn, by creating test suites for each of the 5 use cases using their preferred code-based approach, including Selenium, Cypress, or WebdriverIO, These same quality engineers then repeated the process for the exact same 5 use cases using Visual AI from Applitools after completing a 60 minute course on Test Automation University called Modern Functional Testing Using Visual AI[3]. Testers competed for 100 prizes worth $42,000 and were judged on their ability to provide 100% test coverage on all use cases, successfully run these tests, and most important catch all potential bugs using both approaches.

3,000
Participants

288
Testers Successfully Completed

100
Winners Were Chosen

We were blown away by the enthusiastic response from the testing community. Over 3,000 testers signed up to participate with 288 quality engineers ultimately submitting results after spending 11 hours each on average to complete the challenge. That's a total combined effort of 80 work weeks! Testers were geographically, firmographically, and technologically dispersed, providing the industry's largest, highest quality, and freely available data set for understanding the impact of Visual AI on test automation, and ultimately on the impact on quality management and release velocity for modern applications.

Tracy Mazelin
Paylocity
QA Engineer

"Completing the Applitools Hackathon was a keystone achievement in my career! I'm now 100% convinced that Visual AI testing is an essential tool for efficiently validating web and mobile software applications."

Viktar Silakou
Globant
Lead QA Automation Engineer

"This is the most interesting and useful event of the year in the field of testing automation. This allows you to take a look at test automation from a different point of view and gives an opportunity to radically improve your existing approaches."

Corina Zaharia
Mendix
Test Engineer

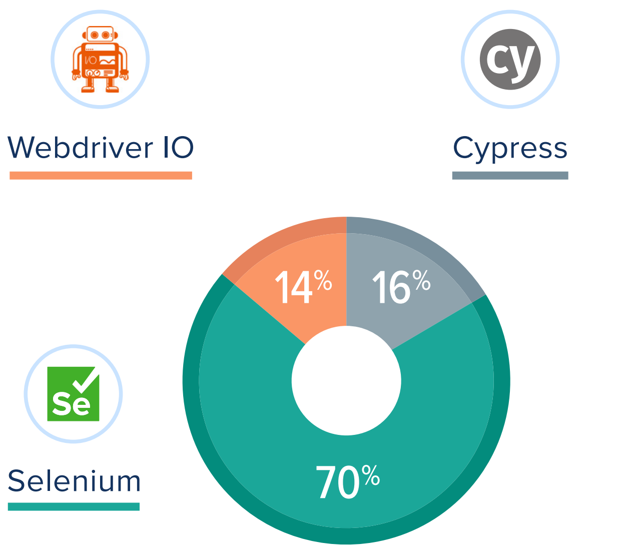
"This challenge propelled me to dig into alternative ways to traditional testing. While solving the challenge, I realized Applitools will save time on the proposed scenarios while still delivering the same value as other traditional frameworks. Congratulations for the initiative and the elegant manner chosen for making the rockstars understand how powerful and awesome Visual AI is."

Over 3,000 Participants From All Over the World.

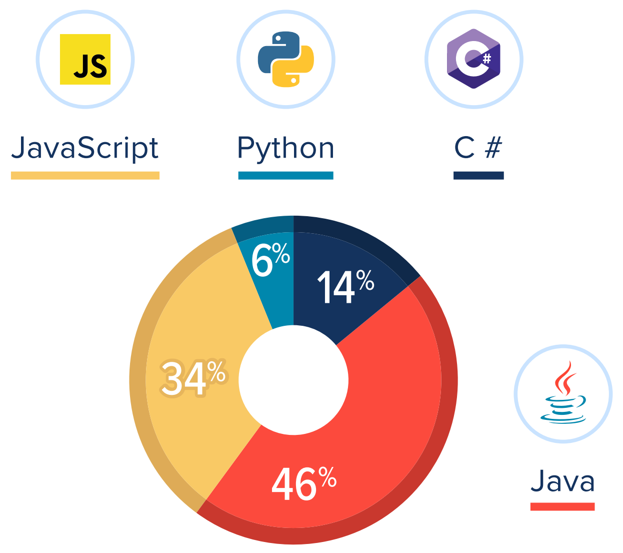
India	36%
United States	21%
United Kingdom	6%
Canada	3%
Australia	3%
Nigeria	2%
Brazil	2%
Germany	1%
Ukraine	1%
Romania	1%
Pakistan	1%
Netherlands	1%
Singapore	1%
South Africa	1%
Vietnam	1%
Indonesia	1%
Hungary	1%
Russia	1%
Greece	1%
Spain	1%
France	1%
Egypt	1%
Sri Lanka	1%
New Zealand	1%

Plus another 10% from 75 additional countries!

% OF SUBMISSIONS BY TEST FRAMEWORK



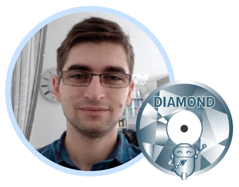
% OF SUBMISSIONS BY LANGUAGE



100 Visual AI Rockstar Hackathon Winners

Grand Prize Winners

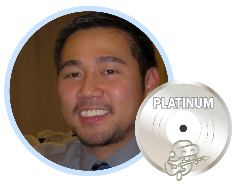
Corina Zaharia
Mendix



Ioan Cimpean
Congnizant
Softvision

Gavin Samuels
QualityWorks

Viktar Silakou
Globant



Hung Hau
Proofpoint

Tracy Mazelin
Paylocity
Corporation

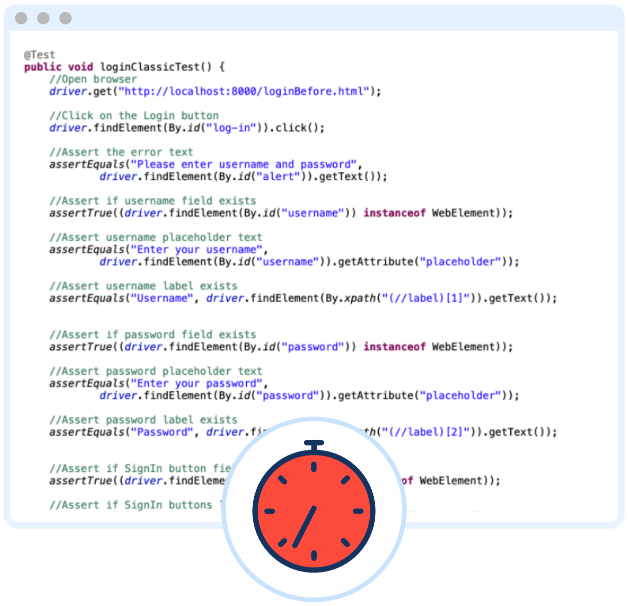
Oluseun Orebajo
FamsItSolutions

Arjan Blok
Mendix

Thai Pangsakulyanont
Taskworld

Adina Nicolae
Sparkware

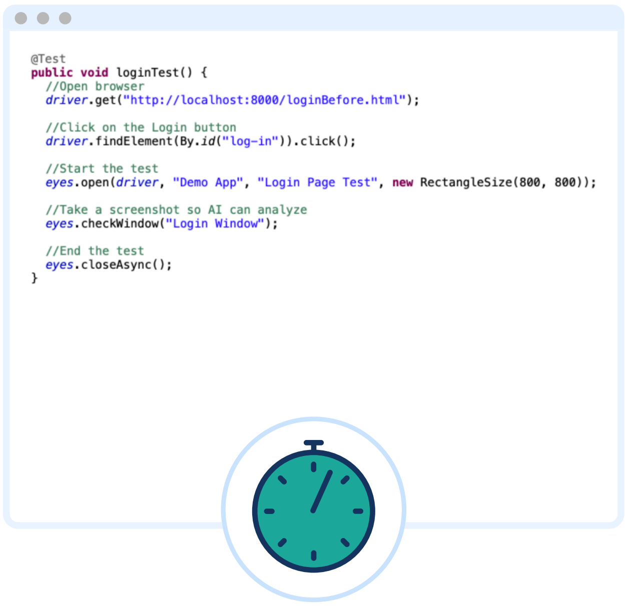
5.8x Faster Test Creation



```java
@Test
public void loginClassicTest() {
    //Open browser
    driver.get("http://localhost:8000/loginBefore.html");

    //Click on the Login button
    driver.findElement(By.id("log-in")).click();

    //Assert the error text
    assertEquals("Please enter username and password",
            driver.findElement(By.id("alert")).getText());

    //Assert if username field exists
    assertTrue((driver.findElement(By.id("username")) instanceof WebElement));

    //Assert username placeholder text
    assertEquals("Enter your username",
            driver.findElement(By.id("username")).getAttribute("placeholder"));

    //Assert username label exists
    assertEquals("Username", driver.findElement(By.xpath("(//label)[1]")).getText());


    //Assert if password field exists
    assertTrue((driver.findElement(By.id("password")) instanceof WebElement));

    //Assert password placeholder text
    assertEquals("Enter your password",
            driver.findElement(By.id("password")).getAttribute("placeholder"));

    //Assert password label exists
    assertEquals("Password", driver.fi              th("(//label)[2]")).getText());


    //Assert if SignIn button fie
    assertTrue((driver.findEleme                   of WebElement));

    //Assert if SignIn buttons
```

7.0

Hours



```java
@Test
public void loginTest() {
  //Open browser
  driver.get("http://localhost:8000/loginBefore.html");

  //Click on the Login button
  driver.findElement(By.id("log-in")).click();

  //Start the test
  eyes.open(driver, "Demo App", "Login Page Test", new RectangleSize(800, 800));

  //Take a screenshot so AI can analyze
  eyes.checkWindow("Login Window");

  //End the test
  eyes.closeAsync();
}
```

1.2

Hours

What It Means to the Tester?

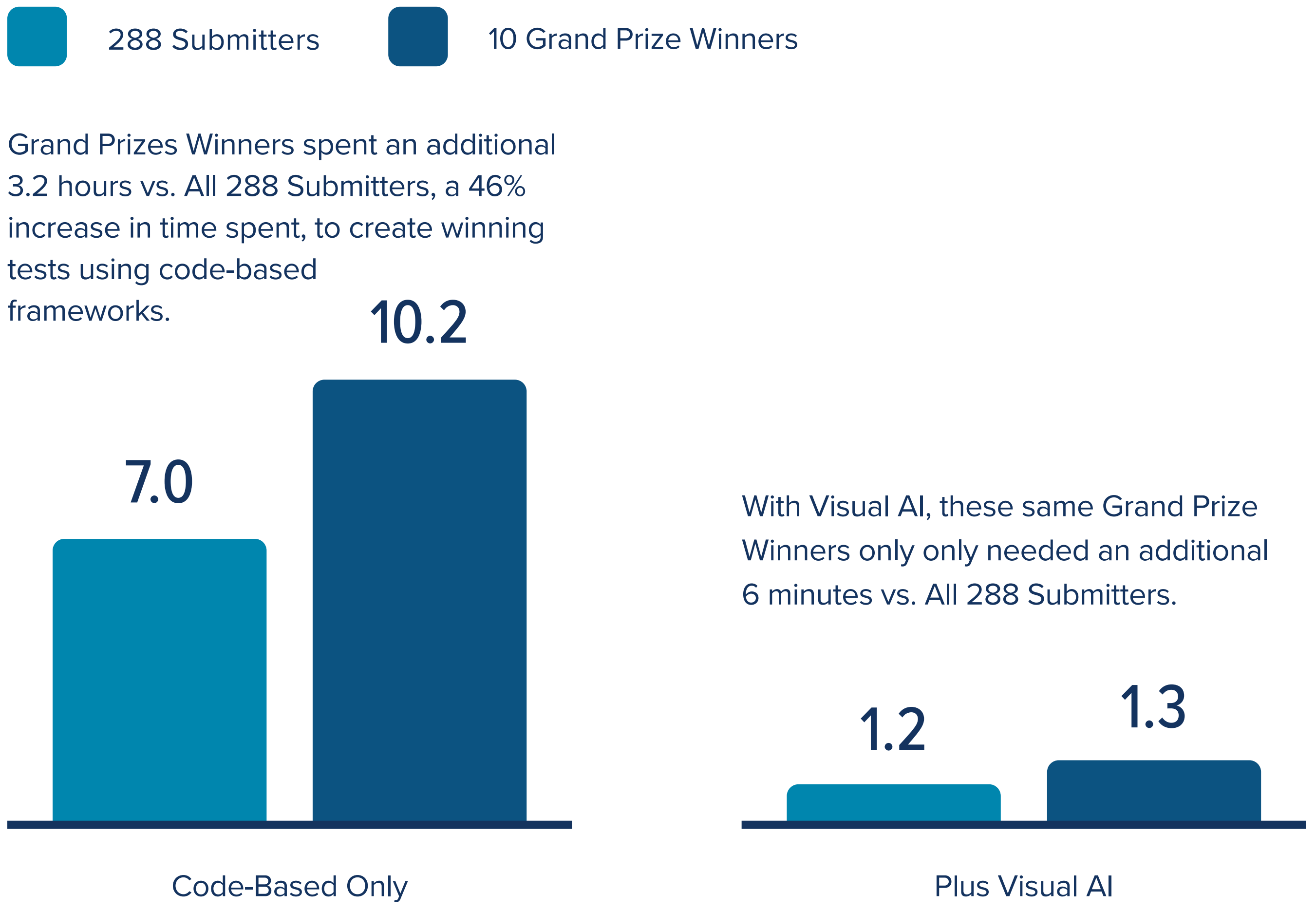
Testers are able to expand test coverage while simultaneously testing faster than ever before.

What It Means for the Team?

More coverage and faster testing helps teams release faster, a primary goal for almost all engineering teams.

Time to Write Tests in Hours

GRAND PRIZE WINNERS VS. ALL 288 SUBMITTERS



Grand Prizes Winners spent an additional 3.2 hours vs. All 288 Submitters, a 46% increase in time spent, to create winning tests using code-based frameworks.

With Visual AI, these same Grand Prize Winners only only needed an additional 6 minutes vs. All 288 Submitters.

Lauren Christianson
Quality Engineer
Delta Air Lines

"The Applitools hackathon was a great opportunity to try out Applitools Eyes, their Visual AI test automation tech. By the time I finished the challenge, I was really impressed by what it can do and how simple it is to use. Visual AI is definitely the future of test automation and Applitools is leading the way there!"

Time to Write Tests

TABLE 1

Time to Write Tests	Code-Based	Plus Visual AI	Visual AI Impact
All 288 Submitters	7.0 Hours	1.2 Hours	5.8X Faster
Top 100 Winners	7.1 Hours	1.1 Hours	6.4X Faster
Grand Prize Winners	10.2 Hours	1.3 Hours	7.8X Faster

Detailed Findings

- When using Visual AI, All 288 Submitters were able to author new tests in just 1.2 hours vs. 7.0 hours using code-based frameworks in isolation -- writing new tests was 5.8x faster using Visual AI!
- Focusing on the Grand Prize Winners emphasizes the time savings finding. This segment spent only 1.3 hours when using Visual AI versus 10.2 hours for code-based approaches in isolation – a 7.8X improvement in time to write tests.
- In order to increase test coverage to grand prize winning levels using code-based approaches, Grand Prize Winners had to spend 10.2 hours authoring tests, an additional 3.2 hours in comparison to All 288 Submitters. This was a significant 46% increase in the time spent.
- In contrast, the Visual AI creation time delta between All 288 Submitters (1.2 hours) and Grand Prize Winners (1.3 hours) was a mere 6 minute (or 8%) increase. This illustrates the simplicity of Visual AI and its ability to help less experienced testers perform at a higher level more quickly and easily.

In Conclusion

Grand Prize Winners had a time commitment increase of 46% in order to provide full coverage using code -based frameworks. In a real world setting, this fact either slows down releases to give quality engineers the time needed to do their work, or, more often, leads to quality degradation as test coverage drops in order to maintain better release velocity. By including Visual AI in their approach, quality engineers obtain the same amount of coverage 7.8x times faster and can use this found time to better manage quality.

5.9x More Test Code Efficient

351

ADJUSTED LINES OF TEST CODE

60

ADJUSTED LINES OF TEST CODE

What It Means to the Tester?

Testers can either achieve the same amount of test coverage using far less test code, or dramatically expand test coverage using the same amount of test code as they do today.

What It Means for the Team?

Test code is slow to write and slow to run. With less of it, teams can either expand coverage or run tests faster or some combination of both depending on their needs and goals.

What Do We Mean By Test Code Efficiency?

In addition to the standard metrics of tester efficiency, the study yielded a new measure called Test Code Efficiency. Similar to the concept of "code efficiency" that software developers look for in their work, test code efficiency measures both the number of lines of test code a quality engineer can write per hour and the efficiency each line of code provides. Visual AI, which captures images of an entire page with the a single open-ended line of code '**`eyes.check.window`**', is much more efficient because it is both easy to write and each individual line of code provides more coverage than code-based approaches that rely on closed-end assertions.

HOW WE CALCULATED TEST CODE EFFICIENCY

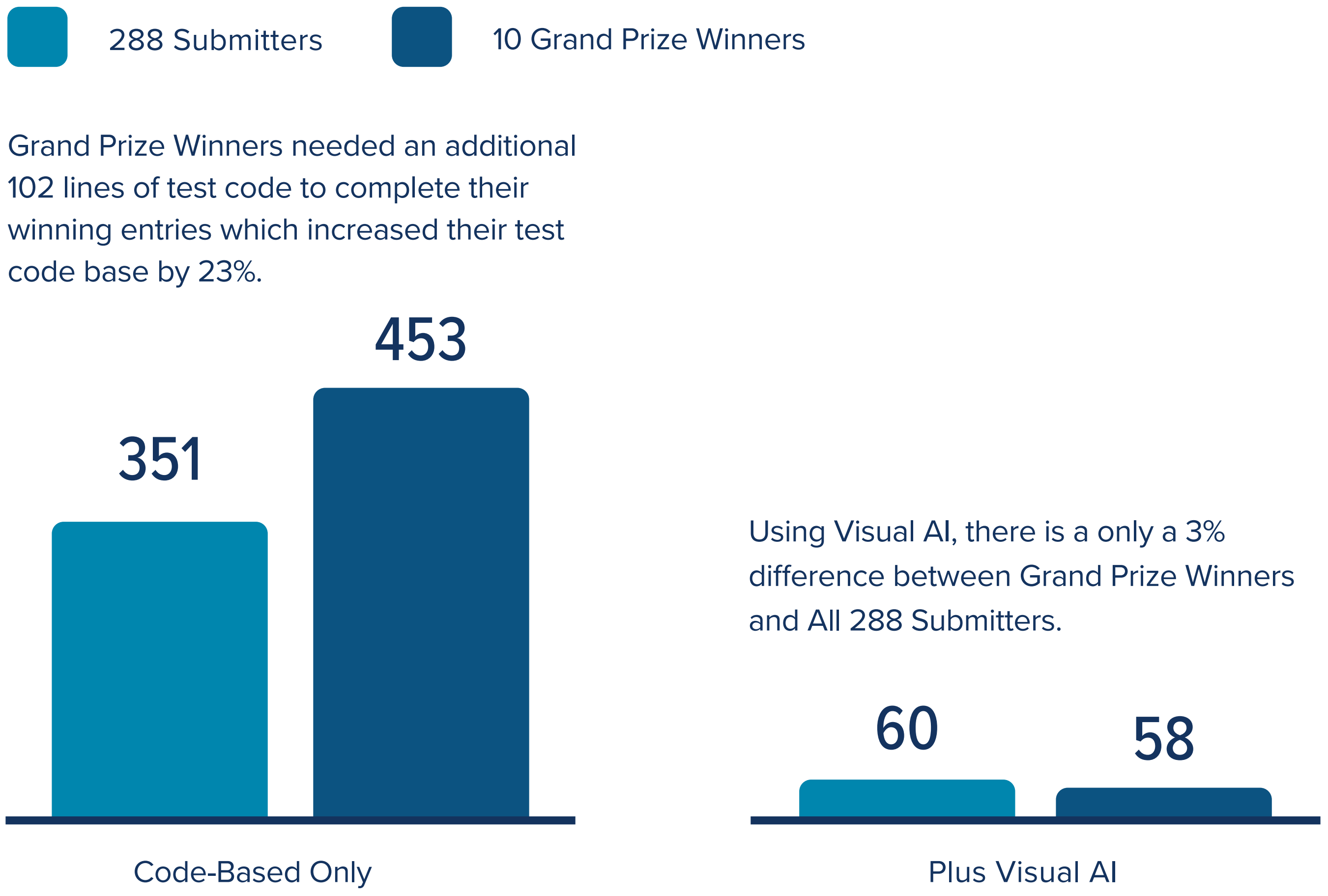
All 288 Submitters	A. Code-Based	B. Plus Visual AI	Explanation/Calculation
1. Raw Lines of Test Code	351 Lines	180 Lines	Average number of lines of test code
2. Time to Write Tests	7 Hours	1.2 Hours	Average time to write tests in hours
3. Lines of Test Code Per Hour	50 Lines Per Hour	150 Lines Per Hour	Row 1 divided by row 2 to adjust for the test creation time differences between code-based and Visual AI testing approaches
4. Visual AI Efficiency Factor	1.0 (Baseline)	3.0	Row 3, column B divided by Row 3, column A. This approaches treats code-based results in row 3 as the baseline for comparison.
5. Adjusted Lines of Test Code	**351 Lines of Code**	**60 Lines of Code**	Row 1 divided by row 4. This measures Test Code Efficiency by accounting for both the difference in raw code needed to provide coverage and the speed at which that code can be written.

Test Code Efficiency Based on Adjusted Lines of Test Code

GRAND PRIZE WINNERS VS. ALL 288 SUBMITTERS

Grand Prize Winners needed an additional 102 lines of test code to complete their winning entries which increased their test code base by 23%.

Using Visual AI, there is a only a 3% difference between Grand Prize Winners and All 288 Submitters.



Ryan LaPensee
Manager, Quality Engineering
ArborMetrix Inc.

"The Applitools Hackathon was truly eye opening to how much testing can be applied when using Visual AI assertions. As a Quality Engineer it gives me great joy to know how much test coverage Applitools can provide to an application."

Test Code Efficiency

TABLE 2

All 288 Submitters	**Code-Based**	**Plus Visual AI**	**Visual AI Impact**
Raw Lines of Test Code	351 Lines	180 Lines	49% Less Raw Test Code
Lines of Test Code Per Hour	50 Lines	150 Lines	3.0X More Productive
Adjusted Lines of Test Code	351 Lines of Code	60 Lines of Code	5.9X More Code Efficient

Top 100 Winners	**Code-Based**	**Plus Visual AI**	**Visual AI Impact**
Raw Lines of Test Code	373 Lines	165 Lines	56% Less Raw Code
Lines of Test Code Per Hour	52 Lines Per Hour	150 Lines Per Hour	2.9X More Productive
Adjusted Lines of Test Code	373 Lines of Code	55 Lines of Code	6.8X More Code Efficient

10 Grand Prize Winners	**Code Based**	**Plus Visual AI**	**Visual AI Impact**
Raw Lines of Test Code	453 Lines	145 Lines	68% Less Raw Code
Lines of Test Code Per Hour	45 Lines Per Hour	112 Lines Per Hour	2.5X More Coverage
Adjusted Lines of Test Code	453 Lines of Code	58 Lines of Code	7.8X More Code Efficient

Test Code Efficiency

Detailed Findings

- Data clearly indicates how much more code efficient testers can be when infusing Visual AI into their work. All 288 Submitters wrote 351 lines of raw code to obtain coverage in their code based framework and took 7 hours to do so vs. 180 lines of code using Visual AI and only 1.2 hours to do so.

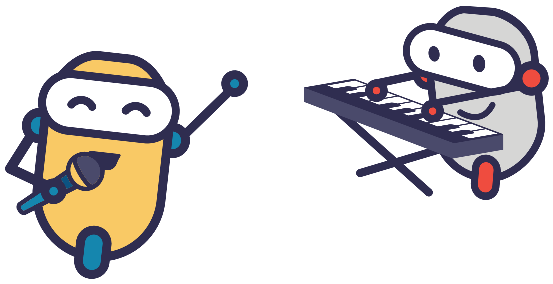
- To aid in the analysis, we developed a metric to help better explain the efficiency of Visual AI relative to code-based frameworks. Using this new metric of test code efficiency, when using Visual AI, quality engineers need only 60 lines of adjusted test code vs. 351 to provide the same amount of coverage in the same amount of time -- this leads to our finding that Visual AI is 5.9x more code efficient than code-based frameworks in isolation.

- En route to their winning submissions, Grand Prize Winners wrote 453 lines of raw code in order to provide coverage using code-based frameworks vs. 351 lines of raw code for All 288 Submitters. This 29% increase explains the additional 3.2 hours needed to author tests when using code-based approaches in isolation (see Table 1 above).

- In contrast, when using Visual AI, Grand Prize Winners wrote only 145 lines of raw vs. 180 for All 288 Submitters, a 19% decrease between the two groups. We suspect this decrease in raw code was driven by Grand Prize Winners better understanding of Visual AI and how to apply it most efficiently to obtain test coverage.

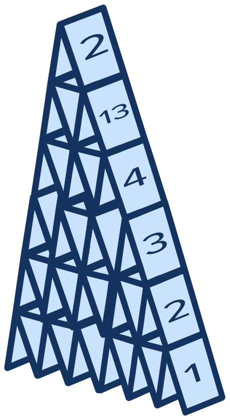
In Conclusion

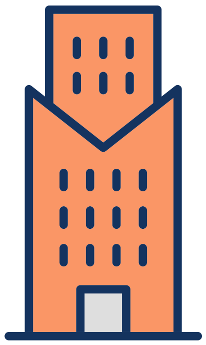
With just a few key tips on the optimal use of Visual AI, all testers can enjoy a 7.8x improvement in test code efficiency. This gives testers the time to both increase test coverage significantly, yet still complete testing orders of magnitude faster after adding Visual AI. This ability is vital to alleviating the testing bottleneck that remains a barrier to faster releases for most engineering teams.



3.8x Improvement In Test Code Stability

applitools





NUMBER OF LABELS	22	7
NUMBER OF LOCATORS	12	2
TOTAL LOCATORS & LABELS	34	9

What It Means to the Tester?

More stable code means tests break less often, allowing testers to expand coverage and spend more time managing quality.

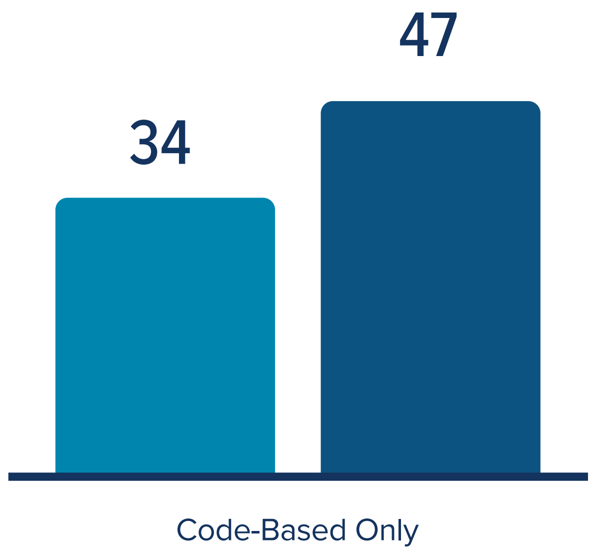
What It Means for the Team?

Teams with stable test code can test more or test faster or some combination of both depending on needs and goals.

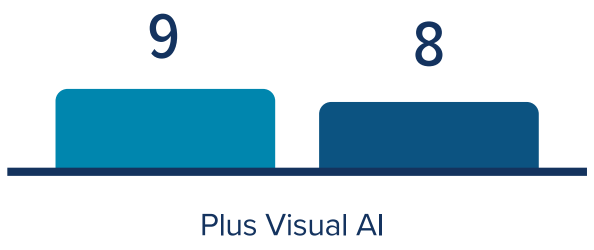
Test Code Stability Based on Number of Labels and Locators

GRAND PRIZE WINNERS VS. ALL 288 SUBMITTERS

Grand Prize Winners needed 13 additional locators and labels to provide winning test suites when using code-based frameworks in isolation.



When using Visual AI, the number of locators and labels is almost the same between All 288 Submitters and Grand Prize Winners.



Fernando Távora
Engineering Productivity Lead
Elementum SCM

"Applitools takes UI testing to the next level. It is not only about checking visual styles and colors, you can have entire functional checks covered too with much less code! "

Test Code Stability

TABLE 3

Number of DOM Locators	Code-Based	Plus Visual AI	Visual AI Impact
All 288 Submitters	22	7	3.1X More Stable
Top 100 Winners	26	6	4.3X More Stable
10 Grand Prize Winners	32	7	4.6X More Stable

Number of Text Labels	Code-Based	Plus Visual AI	Visual AI Impact
All 288 Submitters	12	2	6X More Stable
Top 100 Winners	14	1	14X More Stable
10 Grand Prize Winners	15	1	15X More Stable

Total # of Locators & Labels	Code-Based	Plus Visual AI	Visual AI Impact
All 288 Submitters	34	9	3.8X More Stable
Top 100 Winners	40	7	5.7X More Stable
10 Grand Prize Winners	47	8	5.9X More Stable

Test Code Stability

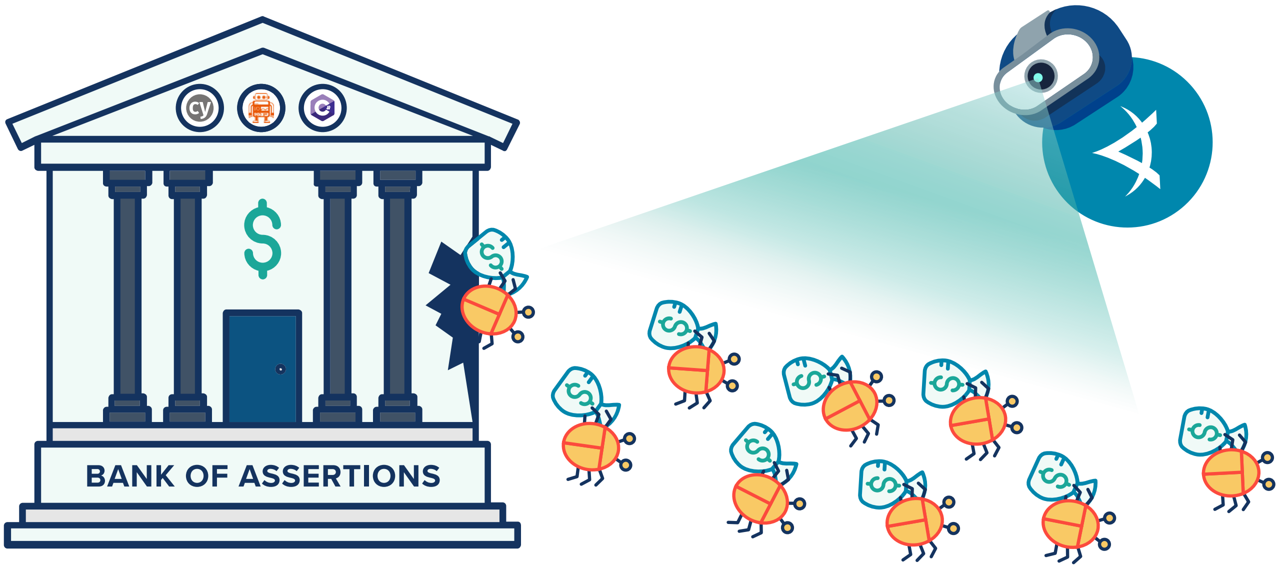
Detailed Findings

- Among All 288 Submitters, quality engineers needed 34 labels and locators on average using code-based frameworks vs. just 9 when using Visual AI -- this indicates 3.8x more test stability when using Visual AI.
- To win their Grand Prizes, this segment needed 47 locators and labels to provide coverage using code-based approaches vs. 34 for All 288 Submitters. This delta of 13 locators amounts to a 38% increase in potential maintenance issues with each new release candidate -- a steep price to pay for the additional coverage.
- In contrast, Grand Prize Winners had only 8 locators and labels after adding Visual AI to their test suite vs. 9 for All 288 Submitters. This is because Visual AI relies on open-ended images, not assertions, to manage quality for a fully rendered page or screen. These numbers are what led to the calculations of 3.8x and 5.9x improvement in test stability for All 288 Submitters and Grand Prize Winners respectively.
- Further, it is important to note the small 12% delta in locators and labels between Grand Prize Winners (8 total) and All 288 Submitters (9 total) after adding Visual AI. This is an indication of how much easier it is for testers to learn and apply Visual AI vs. the more complex code-based approaches.

In Conclusion

With just a few key tips on the optimal use of Visual AI, all testers can obtain a 5.9x improvement in overall test stability. This results in much faster testing cycles with fewer locators and labels at risk of throwing false positives when a new candidate is being tested. Bottom line - test maintenance is greatly reduced when Visual AI is leveraged!

45% More Effective at Catching Bugs Early



What It Means to the Tester?

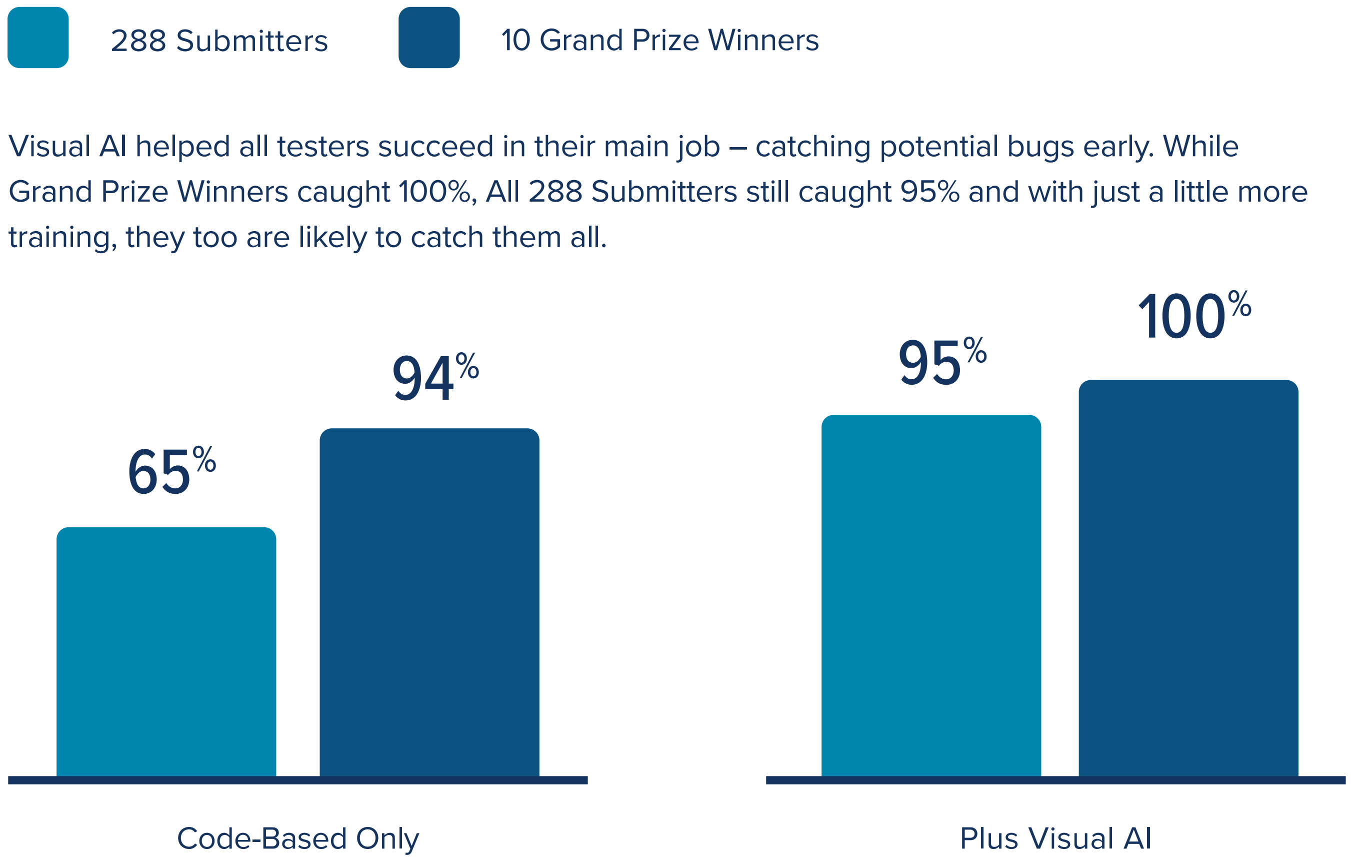
The primary goal of the tester is to catch bugs early. With faster test builds that are more stable and easily maintained, more bugs get caught before escaping into production.

What It Means for the Team?

Application quality improves, fewer late stage bugs escape, and teams release faster with more confidence.

% Of Potential Bugs Caught

GRAND PRIZE WINNERS VS. ALL 288 SUBMITTERS



Visual AI helped all testers succeed in their main job – catching potential bugs early. While Grand Prize Winners caught 100%, All 288 Submitters still caught 95% and with just a little more training, they too are likely to catch them all.

Dimpy Adhikary
Test Architect
Happiest Minds Technologies

“The Visual AI Hackathon is a unique way of learning as well as applying a new skill to solve UI automation challenges. Applitools can drastically reduce the number of lines of code, provide powerful functional validation, and many cool out of the box features to make UI automation tests less flaky.”

Percent of Potential Bugs Caught

TABLE 4

% Of Potential Bugs Caught	Code-Based	Plus Visual AI	Visual AI Impact
All 288 Submitters	65% (11)	94% (16)	45% More Effective
Top 100 Winners	76% (13)	94% (16)	24% More Effective
10 Grand Prize Winners	82% (14)	100% (17)	21% More Effective

Detailed Findings

- When using Visual AI, All 288 Submitters caught 11 out of 17 bugs using code-based approaches in isolation vs. 16 out of 17 with Visual AI -- this indicates Visual AI is 45% more effective at catching potential bugs pre-production.
- Using code-based frameworks in isolation, Grand Prize Winners were able to catch 14 out of the 17 potential bugs (82%) vs. only 11 (65%) for All 288 Submitters. However, this increased performance was costly both in terms of time (3.2 more hours as indicated by Table 1) and test code (102 additional lines of test code as indicated by table 2)
- Additionally, many Grand Prize Winners used a complex technique whereby they looked at the underlying JavaScript code on the app to inspect for changes as opposed to using test code assertions. While this was clever and fair in the context of the Hackathon competition, it is not typically available to testers in a real world setting. This reality renders some common test cases impossible using a code-based framework (see use cases 3 and 4 in the appendix).
- Finally and most importantly, using Visual AI, all 288 Submitters were able to catch 94% of bugs, while Grand Prize Winners caught all 20 bugs. This is a very small gap in performance indicating the ease of which Visual AI can be applied effectively to manage quality.

In Conclusion

Any tester now has access to incredibly accurate Visual AI technology that will increase early stage bug detection by 45% or more. This leads to better app quality, the ultimate goal of a quality engineer.

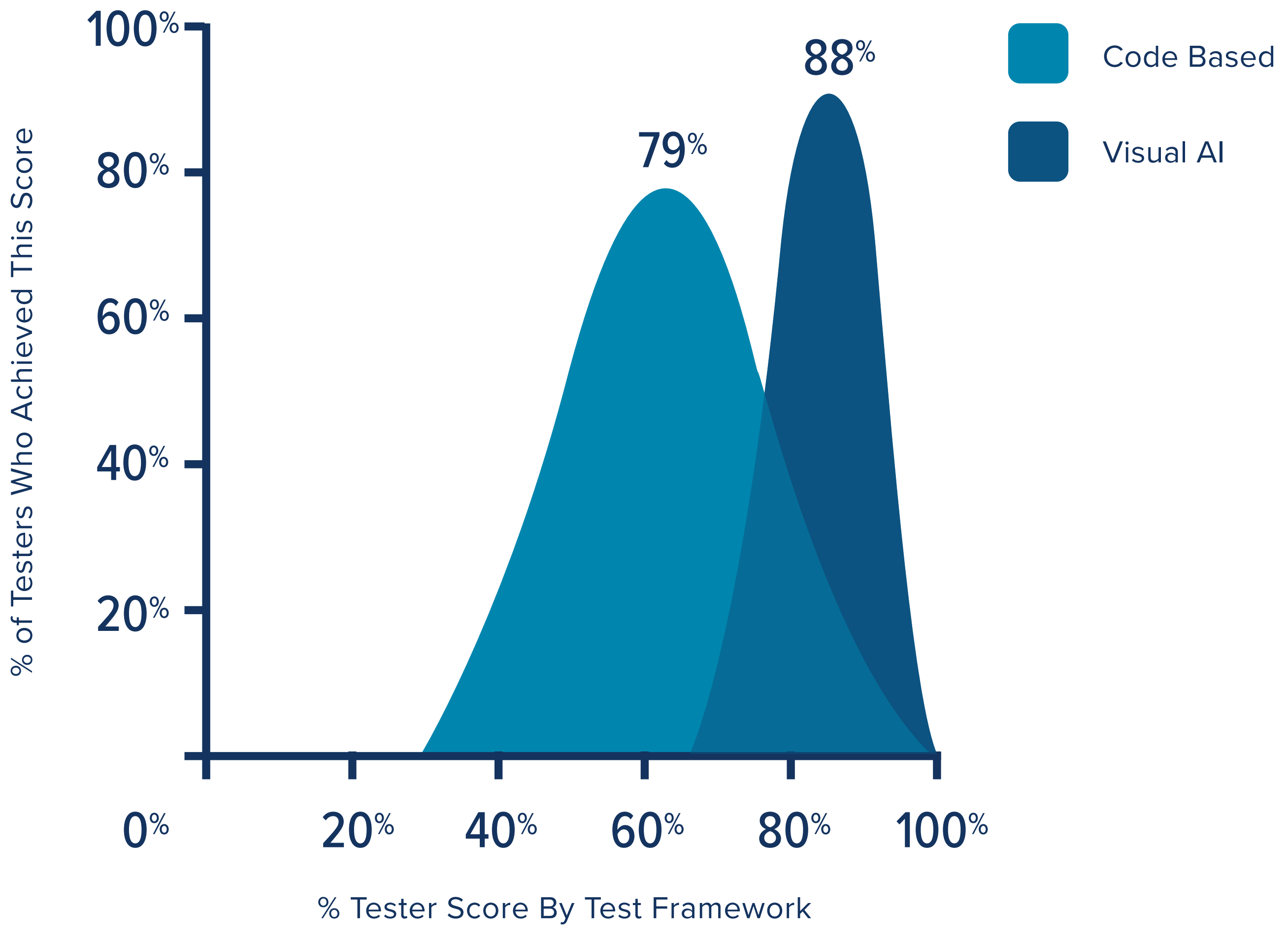
Coverage Ratings By Testing Approach

As part of the study, dozens of qualitative interviews were conducted with respondents asked to compare and contrast their experience with Applitools Visual AI in comparison to their preferred code-based test framework. This table summarizes this learning for readers.

Ratings By Test Approach	Code-Based[1]	With Visual AI[2]	Tester Pain Points Solved With Visual AI
Test Case 1: **UI Elements**	★★★	★★★★★★	• Reduces reliance on brittle locators and labels • Tests all UI elements on the page • Makes reporting and annotating much easier • Replaces multiple tests on a given page with a single test
Test Case 2: **Login Functionality**	★★★★	★★★★★★	• Covers traditional functional testing more easily • Adds pure visual test coverage (e.g. proper error messages?)
Test Case 3: **Table Sorting**	★★★★	★★★★★★	• Simplifies testing for this complex, if not impossible, test case • Easily add automated test coverage using Visual AI
Test Case 4: **Bar Charts**	★	★★★★★★	• Simplifies testing for this complex, if not impossible, test case • Easily provides test coverage since Visual AI does not require DOM access to work effectively
Test Case 5: **Dynamic Content**	★	★★★★★★	• Simplifies testing for this complex, if not impossible, test case • Easily provides test coverage since Visual AI can be set to "layout mode" to inspect layout integrity

THE IMPACT OF VISUAL AI

Easier to Learn Than Code-Based Frameworks



What It Means to the Tester?

Code based test frameworks are complex, especially when applied to today's modern apps. Visual AI provides an easy path to test automation for quality engineering professionals.

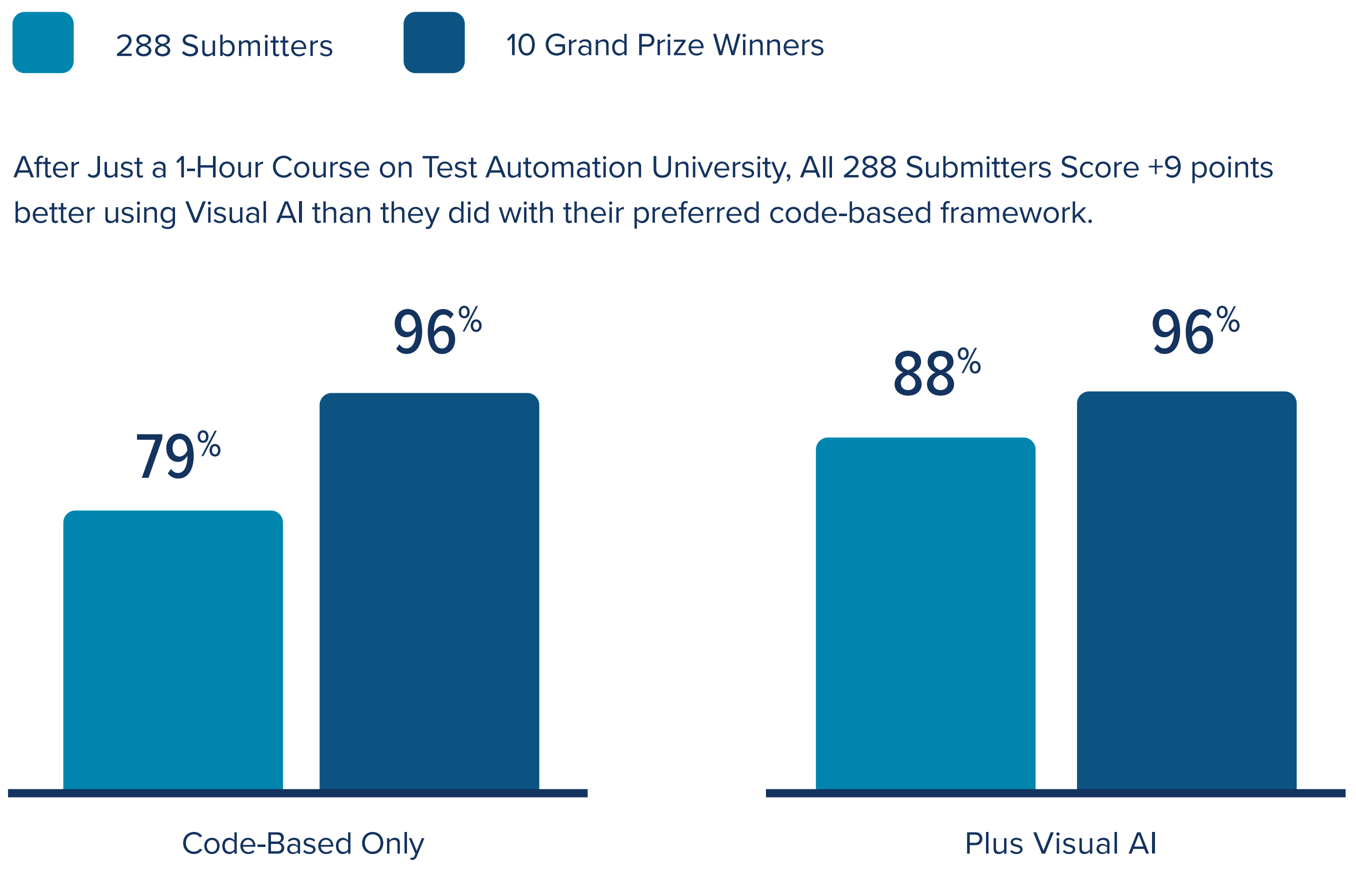
What It Means for the Team?

Upskilling the entire team is easy to do and allows you to deploy the most experienced quality engineers against your most difficult tasks.

Average Tester Score By Framework

GRAND PRIZE WINNERS VS. ALL 288 SUBMITTERS

After Just a 1-Hour Course on Test Automation University, All 288 Submitters Score +9 points better using Visual AI than they did with their preferred code-based framework.



Eric Davidson
Software Engineer
Chick-fil-A

"The Hackathon was an incredible opportunity to experience the power of visual testing. Applitools has created a tool that easily integrates with existing frameworks and quickly augments functional test suites. I look forward to seeing more of their innovative spirit as I advocate for them on the teams I support."

Tester Scores By Framework

TABLE 5

288 Submitters	Code-Based	Plus Visual AI	Visual AI Impact
Lowest Score	27%	66%	+39 Pts
Highest Score	100%	100%	No Difference
Average Score	79%	88%	+9 Pts

Top 100 Winners	Code Based	Plus Visual AI	Visual AI Impact
Lowest Score	69%	73%	+13 Pts
Highest Score	100%	100%	No Difference
Average Score	87%	93%	+6 Pts

Grand Prize Winners	Code Based	Plus Visual AI	Visual AI Impact
Lowest Score	79%	80%	1 Pt
Highest Score	100%	100%	No Difference
Average Score	96%	96%	No Difference

Tester Scores By Framework

Detailed Findings

- All 288 Submitters received an average scored an average of 79% when using code-based framework vs. 88% when using Visual AI. This +9 point score increase was achieved after only a 1-hour course at Test Automation University on “Functional Test Automation Through Visual AI”

- Scores amongst testers using code-based frameworks exclusively ranged from an average of 79% among All 288 Testers to 96% for the Grand Prize Winners, a spread of 17 points. As we mentioned previously, this increase in the success rate using code-based frameworks exclusively cost grand prize winners 3.2 hours of time (See Table 1) and 102 additional lines of code (See Table 2).

- The above results for code-based frameworks represents a large spread in scoring especially considering the length of time in market (over 20 years for Selenium) and level of available training.

- In contrast, average success rates among testers using Visual AI ranged from only 88% among All 288 Submitters to 96% for Grand Prize Winners, a tight spread of only 8 points.

- Additionally, the average success rate among All 288 Testers for Code Based frameworks was 79% vs. 88% when using Visual AI. **This is remarkable performance considering testers spent only an hour getting trained on Visual AI before applying it to their test suites.**

In Conclusion

Testers are able to perform in their function at a much higher level after only 60 minutes of training on how to apply Visual AI. This gives significantly more time for testers to increase coverage and manage quality as opposed to learning and writing test code.

Calculating the Value of Visual AI on Test Automation

ESTIMATE THE IMPACT FOR YOUR TEAM

Using the results of this study, it's now easy for you to estimate the impact of Visual on your quality engineering efforts. Simply follow the instructions below, or contact us at ImpactOfVisualAI@Applitools.com to get our help!

How many hours do you spend adding new tests per release?

To estimate the **number of hours you will save per release**, multiply the number above by .13

How many late stage bugs escape per month on average?

To estimate the **number of late stage bugs you will eliminate per month**, multiply the number above by .94

How many lines of functional test code does your team maintain currently?

To estimate the **number of lines of test code you will eliminate**, multiply the number above by .13

How many locators and labels combined does your team maintain in your functional test suite currently?

To estimate the **number of locator and labels you will eliminate**, multiply the number above by .17

The Impact of Visual AI on Test Automation

CALCULATION FOR A TYPICAL QUALITY ENGINEERING TEAM

	Input: Current Code-Based Approach	Output: Estimate After Adding Visual AI	Impact of Visual AI	
Hours Spent Creating New Tests	2,000	260	435	Test Creation Hours Saved
Number of Late Stage Bugs	208	10	198	Additional Bugs Caught Early Stage
Number of Lines of Test Code	50,000	6,500	10,875	Lines of Test Code Eliminated
Number of Locators & Labels	10,000	1,700	2,075	Number of Labels & Locators Eliminated

This is a conservative estimate considering only test creation. Were we to include test reporting, team collaboration, and test maintenance impacts, the impact is easily 3x or more higher than the estimates provided from this study.

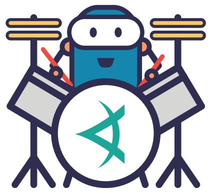
Annual Assumptions for the Typical 4-Person Quality Engineering Team

1. 60 minutes per test
2. 2,000 tests per team
3. 8 bugs per release with an average of 2 major releases per month. This estimates assumes CI, does not assume CD.
4. 25 lines of test code per test
5. 5 locators and labels in total per test

We encourage you to use the results of this study to estimate the impact for your team!

Are You Ready To Try It?

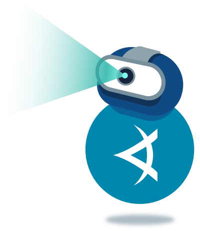
In Just A Few Hours, You Can Be Up and Running with Visual AI



A Virtual, Private Hackathon

Test your skills across your global team?

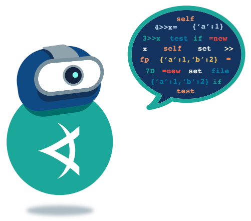
We will package up our Hackathon app, provide instructions, guide your team through the contest, judge results, and celebrate the winners!



Your Private Visual AI Upskill Webinar

Learn How to Use Visual AI From Angie Jones or Raja Rao?

For qualifying teams, we will secure a private webinar reviewing all the use cases, showing you how it works, and sharing more details about the results of this study.



We'll Run Your First 10 Visual AI Tests For Free

Ready To Start Now? Let's Get Going!

For qualifying teams, we will take a portion of your existing Code Based test suite and add Visual AI tests in just a few hours

Upskill For Free at Test Automation University

Upskill On Your Own and Join the TAU Community!

Modern Functional Test Automation Through Visual AI

Selenium Webdriver with Java

Introduction to Cypress

In Closing

3,168 hours of empirical data from 288 quality engineers. These engineers worked through five different testing use cases with traditional code based test frameworks and then again in combination with Applitools Visual AI technology. During this experience, these engineers learned what impact an emerging technology could have on their workflow and productivity in an effort to deliver high-quality apps faster.

Hyper efficient. Higher quality. Faster releases. What they discovered was Visual AI not only expanded their test coverage dramatically, but did so by replacing the time spent writing and maintaining test code with time spent managing quality. Further, these engineers realized the outcome of increased productivity and higher quality app delivery are achievable in an incredibly fast and stable environment that dev teams will appreciate.

Upgrade, don't rip and replace. The quality engineers also realized that the code based frameworks they are familiar with, including Selenium, Cypress.io, and WebdriverIO, are much easier to use effectively with Applitools Visual AI. Unlike code based framework, Visual AI does not require extensive training or large amount of time to provide effective test coverage. Teams struggling to initiate their test automation program, or hire test automation engineers, or upskill an existing team, will find it much easier to overcome all these issues using Applitools Visual AI.

Visual AI is the future of testing. Apps, websites and smart devices will continue to proliferate, in addition to the increasing number of screen sizes and page variations that customers demand. Soon, any attempt to manage the visual and functional quality of an app or website with the necessary quality assurance and test coverage is going to be impossible unless you have Visual AI technology.

Hero your customers and gain competitive advantage. As the expectations of faster software release cycles continue, testing teams and quality engineers must achieve the highest level of continuous quality for their business. They are the gatekeepers for quality in the digital transformation journey. Remember, Visual AI and test automation are only tools and not the end game. The end game is being customer-obsessed, and giving customers the highest quality digital experiences as quickly and efficiently as possible.

Appendix

Methodology

Build The Application - First, we built a simple application representing real world functional testing use cases. There were five use cases in all with a potential of 20 bugs. Use cases included testing 1. UI elements 2. login functionality 3. table sorting functionality, 4. data graphing functionality, and 5. dynamic content. All baselines and new candidates of the test cases were provided along with a description and testing goals.

Execute the Visual AI Rockstar Hackathon - In order to recruit testers and incent them to take 11 hours of their time to learn about Visual AI, we gamified the research and created a hackathon. This hackathon delivered 100 prizes worth US $42,000 including the two $5,000 Grand Prizes and eight $1,000 Platinum Prizes for the testers who provided the most test coverage, successfully ran their tests, and caught as many bugs as possible. Testers had from November 1st until November 30th, 2019 to submit their work. Winners were announced in late January 2020. You can learn who they are by visiting us here.

Recruit Representative Testers - Testers were recruited using a variety of tactics including social media, paid advertising, and direct marketing. Any tester, anywhere in the world could qualify. Our goal was to obtain at least 100 submissions, but in the end we received 288 submissions from among 3,024 participants. We ended up with a highly representative data set geographically, technographically, and demographically.

Represent Major Code Based Frameworks - Testers could qualify by successfully completing the challenge using Code Based frameworks of Cypress, WebdriverIO, or Selenium in languages of Java, Javascript, Python, C#, or Ruby. Next, using the appropriate Applitools Eye SDK, they would repeat the effort using Visual AI.

Judging Submissions - Highly experienced quality engineers judged every submission. 100 points were possible for each test framework for a total of 200 points in all. Points were awarded based on coverage, bugs caught, and test execution success against each of the five use cases. Judges also made qualitative observations about the test suites to add an additional layer of insight.

Aggregate and Analyze the Data - Data from each individual submission was logged, blinded to protect the identity of any individual, and then aggregated prior to analysis. Standard research quality control procedures were used to ensure that any result included was valid and met all the criteria for inclusion. Feedback was also obtained from All 288 Submitters and permission to use their name and a quote was obtained prior to publication.

Release the Findings - Findings were analyzed and released on April 7, 2020. Additional details by use case will be released throughout 2020. Academics or analysts who want to access more details should contact us at ImpactOfVisualAI@Applitools.com

Use Case 1: Test UI Elements

This use case showcases how testing UI elements, one of the most common tasks in automation testing, is made simple through Visual AI.

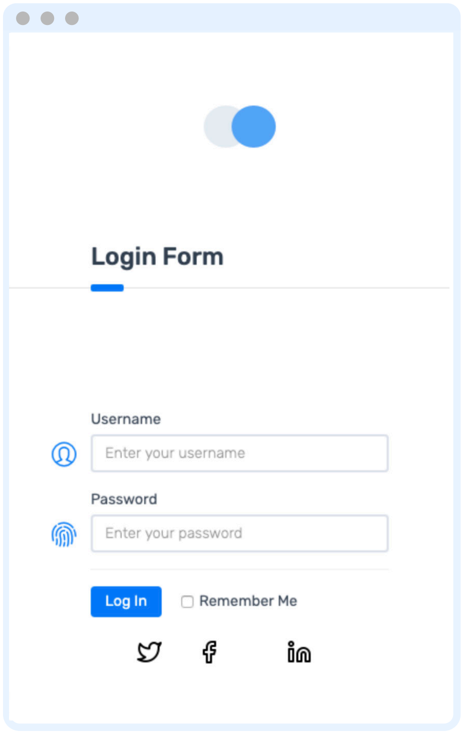
In traditional testing, people rely on flaky DOM locators, field labels, and text such as error messages to validate different aspects of UI elements. But the DOM locators, labels and messages can change at any point leading to a lot of unnecessary test failures and in turn lead to test maintenance. Secondly, and equally importantly, since people can't reasonably test the hundreds of UI elements on every page of a given app, they are forced to test a subset, again leading to a lot of production bugs due to lack of coverage.

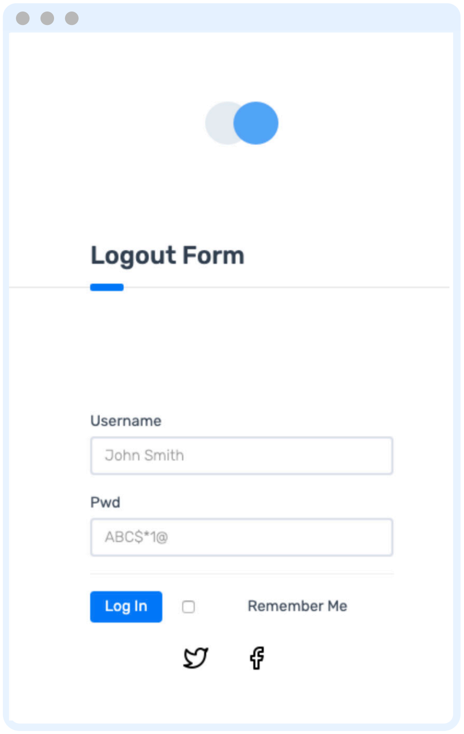
With Visual AI, you take a screenshot and validate the entire page. This limits the testers reliance on DOM locators, labels, and messages. Additionally, you can test all elements rather than having to pick and choose. Lastly, using the "Bug Region" feature of Applitools, you can annotate multiple bugs in the same single screenshot rather than writing multiple tests for multiple elements.

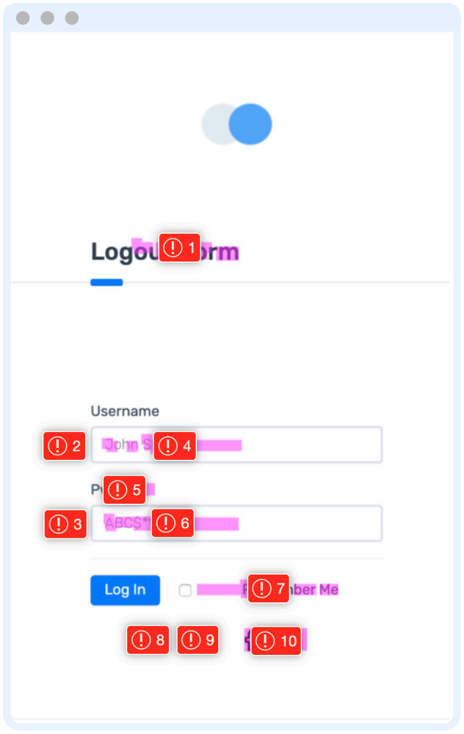
In the end using Visual AI, you'll write less code, get more coverage, and have far more reliable tests.

Use Case 1: Test UI Elements

FIND ALL MISSING ELEMENTS ON THIS PAGE AND REPORT THEM ACCURATELY







1. Incorrect Title Text - Visual Bug
2. Incorrect Username Response - Functional Bug
3. Incorrect Password Text - Visual Bug
4. Missing Password Icon - Visual Bug
5. Spacing Problem - Functional Bug
6. Missing Linked In Icon - Functional Bug
7. Spacing Problem - Functional Bug
8. User Name Response Incorrect
9. Broken Twitter Locator - False
10. Broken Facebook Locator - False Positive Positive

Use Case 2: Test Login Functionality

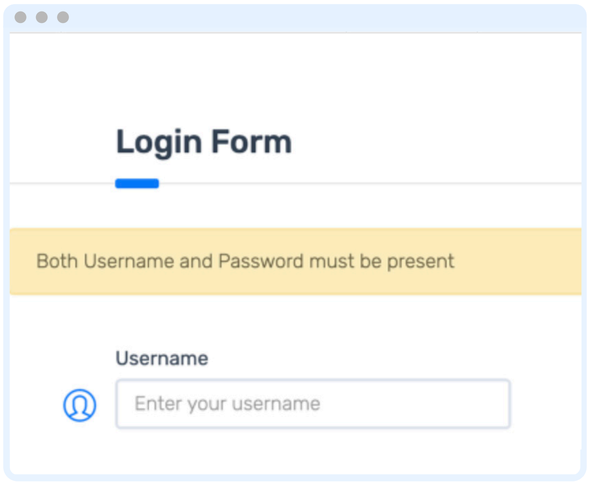
The purpose of this use case is to show how easy it is to perform true functional testing using Visual AI. As a bonus, testers also get visual testing providing increased coverage with no additional effort. To prove this, we asked people to test the functionality with different login data such as incorrect username, incorrect password, valid username, valid password, and so on. The idea was to test both failed and successful logins.

We are routinely challenged by people unfamiliar with Visual AI. **How can we do functional testing using Visual AI?** This doesn't make any sense? Actually, it does! After any UI functionality, the UI goes to a new state. Maybe you navigate to a new page, maybe you show a popup, maybe you show an error. Whatever the case may be, all we need to do is take a screenshot of that new state. If at any point in the future that functionality breaks, then the UI will be different. Our Visual AI will catch that difference easily, an indication the functionality is either broken or unexpectedly changed. This is functional testing using Visual AI.

Use Case 2: Test Login Functionality

TEST LOGIN FUNCTIONALITY ACROSS 4 COMBINATIONS OF INPUTS VIA DATA-DRIVEN TESTING

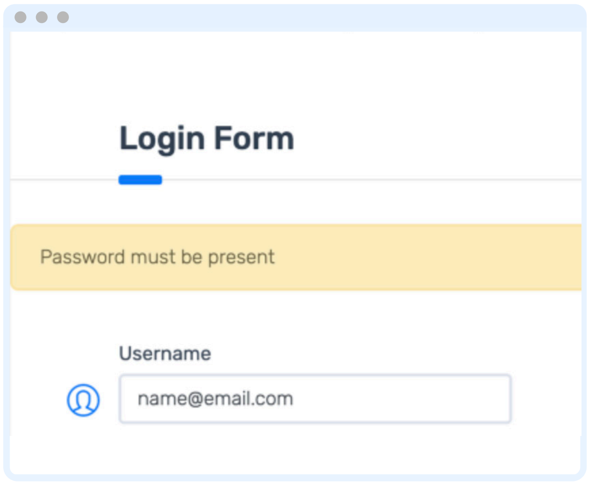
Baseline


Login Form

Both Username and Password must be present

Username

Enter your username

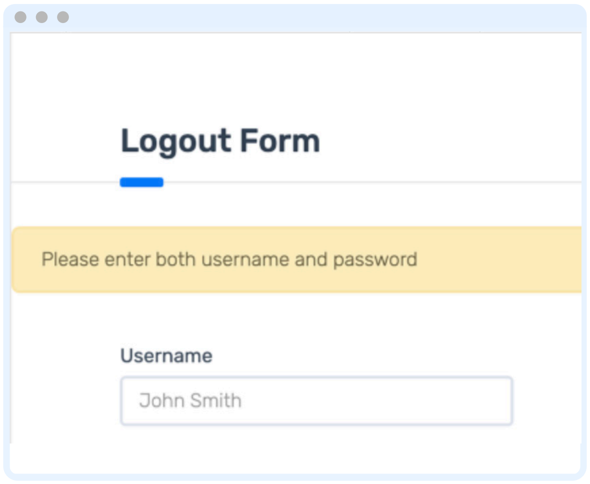

Login Form

Password must be present

Username

name@email.com

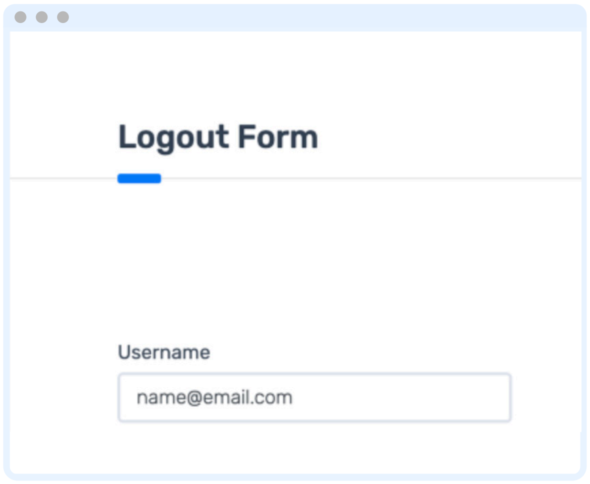
New Candidate


Logout Form

Please enter both username and password

Username

John Smith


Logout Form

Username

name@email.com

Test 2.1 Login without entering any username or password

Bug: Updated error message

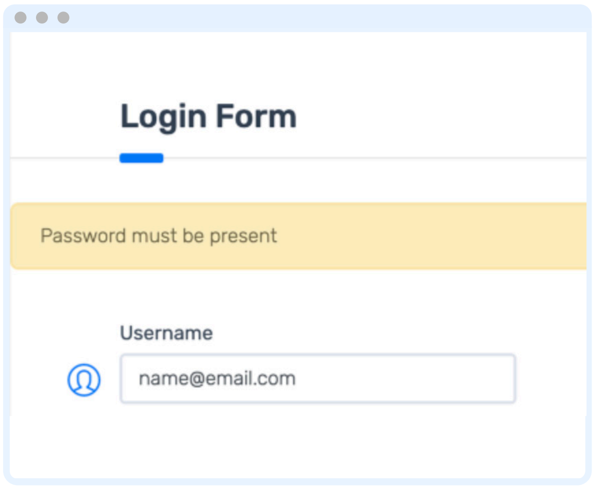
Test 2.2 Login by entering just username and no password

Bug: Missing Error message

Use Case 2: Test Login Functionality

TEST LOGIN FUNCTIONALITY ACROSS 4 COMBINATIONS OF INPUTS VIA DATA-DRIVEN TESTING

Baseline

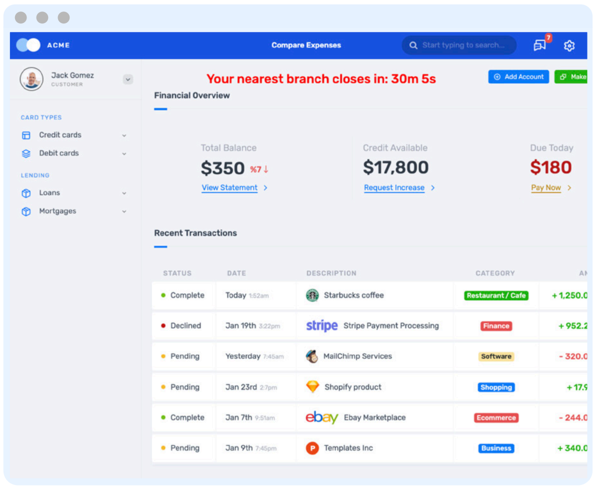


Login Form

Password must be present

Username

name@email.com



ACME · Compare Expenses · Start typing to search...

Jack Gomez
CUSTOMER

CARD TYPES
Credit cards
Debit cards
LENDING
Loans
Mortgages

Your nearest branch closes in: 30m 5s

Add Account · Make

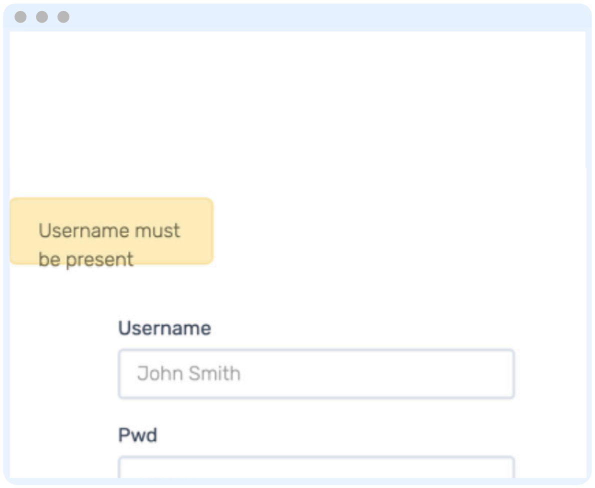
Financial Overview

Total Balance	Credit Available	Due Today
$350 %7↓	$17,800	$180
View Statement	Request Increase	Pay Now

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AM
● Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.0
● Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.2
● Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.0
● Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.9
● Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.0
● Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.0

New Candidate

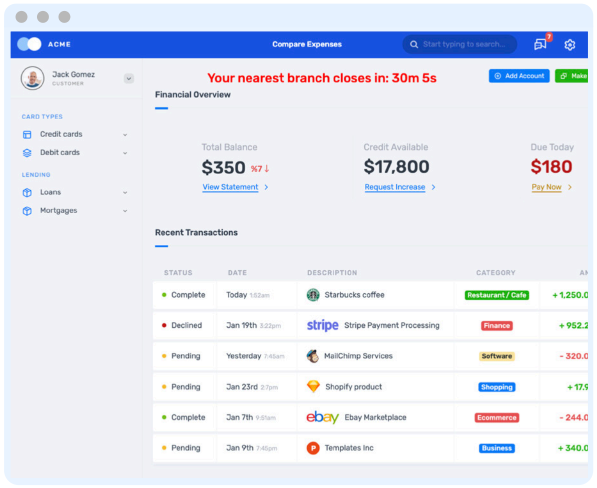


Username must be present

Username

John Smith

Pwd



ACME · Compare Expenses · Start typing to search...

Jack Gomez
CUSTOMER

CARD TYPES
Credit cards
Debit cards
LENDING
Loans
Mortgages

Your nearest branch closes in: 30m 5s

Add Account · Make

Financial Overview

Total Balance $350 %7↓ View Statement
Credit Available $17,800 Request Increase
Due Today $180 Pay Now

Recent Transactions

STATUS DATE DESCRIPTION CATEGORY AM
Complete Today 1:52am Starbucks coffee Restaurant / Cafe + 1,250.0
Declined Jan 19th 3:22pm stripe Stripe Payment Processing Finance + 952.2
Pending Yesterday 7:45am MailChimp Services Software - 320.0
Pending Jan 23rd 2:7pm Shopify product Shopping + 17.9
Complete Jan 7th 9:51am ebay Ebay Marketplace Ecommerce - 244.0
Pending Jan 9th 7:45pm Templates Inc Business + 340.0

Test 2.3 Login by entering just password

Bug: Error message not displayed correctly

Test 2.4 Login with valid username and password (success case)

Bug: None

Use Case 3: Test Table Sorting Functionality

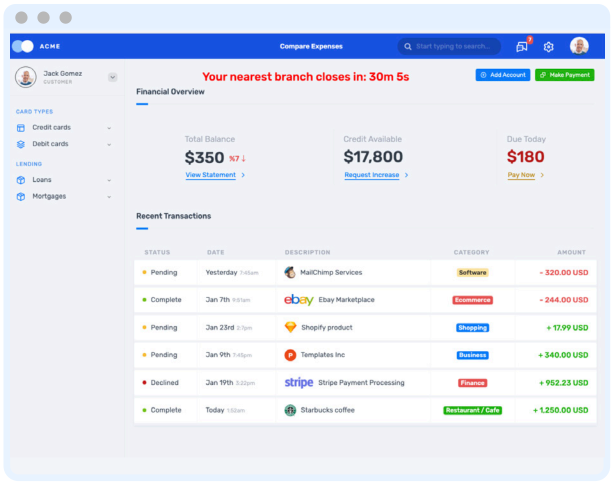
Tables are everywhere! In almost all modern applications (e.g. Yelp, Amazon, Banking, Retail, Software, Media, and more), the data is shown as a list or a table. More and more often, these tables have a sorting or filtering feature. Using traditional approaches, testing sorting functionality requires a lot of advanced programming knowledge and tons of test code because you need to recreate all that sorting functionality within the test code itself.

In this use case, we wanted to showcase how simple it is to test advanced sorting functionality with Visual AI. A tester can do so without any advanced coding knowledge while also using a lot less test code. Similar to our use case on login functionality, all you do is to take a screenshot after the sorting is done. If in the future the sorting result is different than what is expected, Visual AI will highlight it for you.

Use Case 3: Test Table Sorting Functionality

TEST THAT THE AMOUNT COLUMN SORTS IN ASCENDING ORDER

Baseline



Your nearest branch closes in: 30m 5s

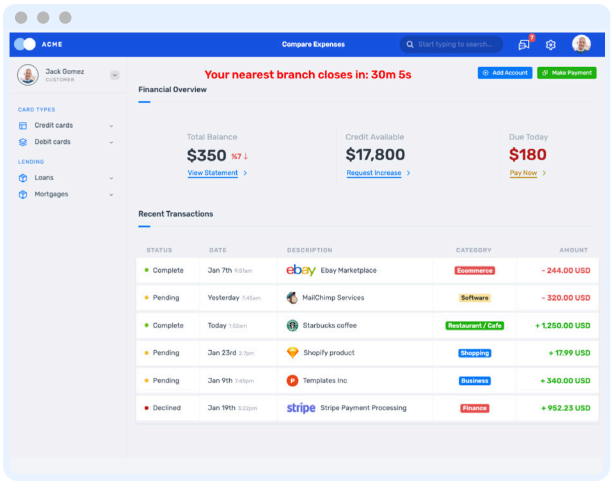
Financial Overview

Total Balance $350 %7 ↓ | Credit Available $17,800 | Due Today $180

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.00 USD
Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD
Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.00 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD

New Candidate



Your nearest branch closes in: 30m 5s

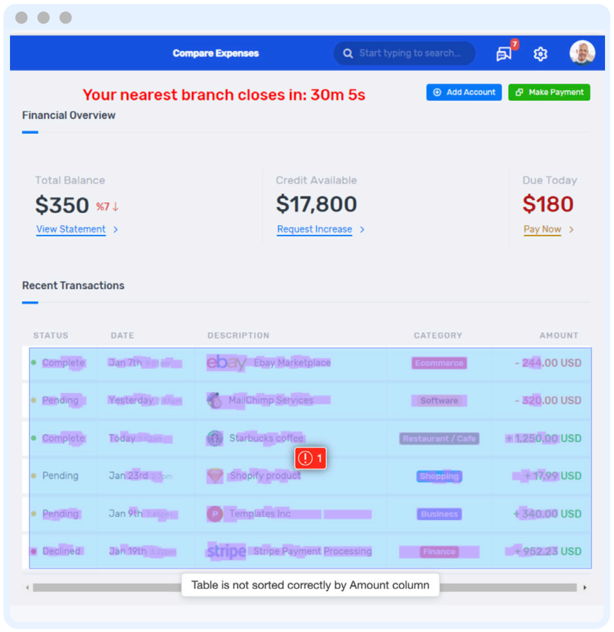
Financial Overview

Total Balance $350 %7 ↓ | Credit Available $17,800 | Due Today $180

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.00 USD
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD
Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.00 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD

Answer Key



Your nearest branch closes in: 30m 5s

Financial Overview

Total Balance $350 %7 ↓ | Credit Available $17,800 | Due Today $180

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Jan 7th	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Yesterday	MailChimp Services	Software	- 320.00 USD
Complete	Today	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD
Pending	Jan 23rd	Shopify product	Shopping	+ 17.99 USD
Pending	Jan 9th	Templates Inc	Business	+ 340.00 USD
Declined	Jan 19th	stripe Stripe Payment Processing	Finance	+ 952.23 USD

Table is not sorted correctly by Amount column

Bug: Table is not sorted correctly

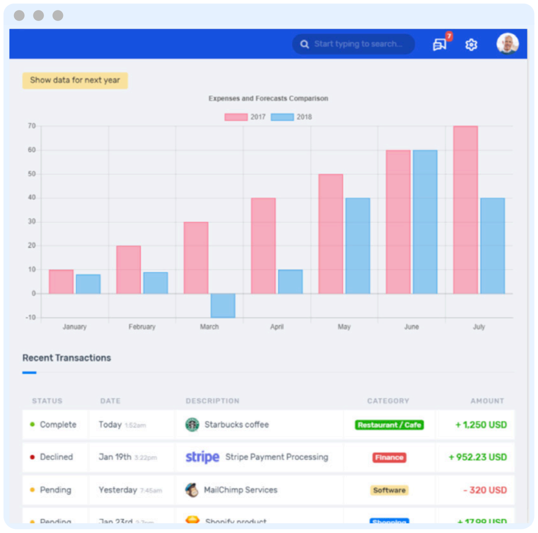
Use Case 4: Find Bugs in Barcharts

Similar to tables, bar charts and graphs are a vital component in modern apps. Since these components are usually built using technologies such as Canvas or SVG, they can't be tested using traditional testing tools because the DOM doesn't provide the access to do so. In response, teams generally resort to manual testing or no testing at all, leading to both slow testing cycles and production bugs.

Visual AI provides an elegant solution to this common testing problem. Since you are simply taking a screenshot, you can easily test artifacts like bar charts and graphs irrespective of what technology they are built with.

THE VISUAL AI

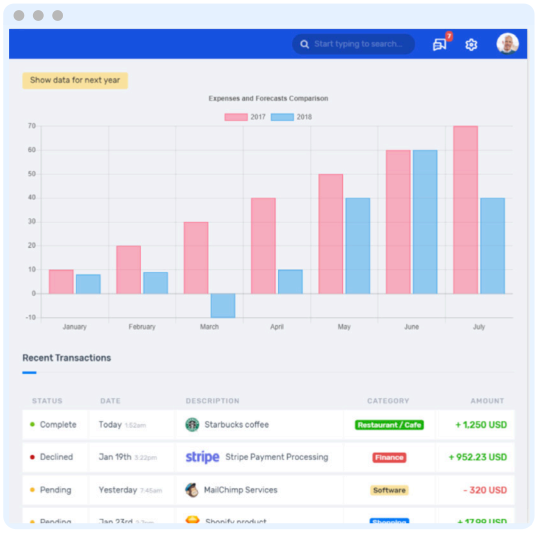
Baseline



Recent Transactions

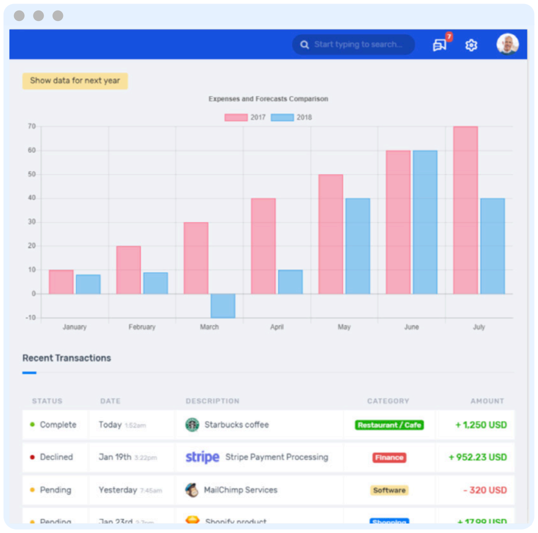
STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD

New Candidate



Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD



Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD

1. Bug: Barchart for January 2018 is different
2. Bug: Barchart for July 2018 is different

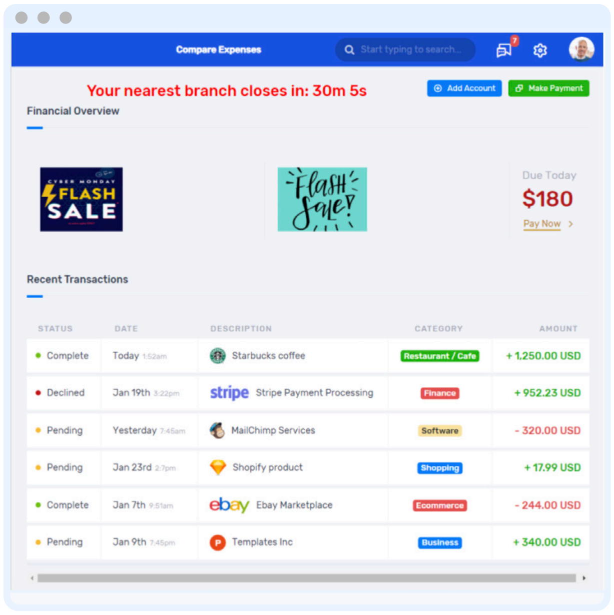
Use Case 5: Test Dynamic Content

Dynamic content is another common feature in modern applications. In fact, many applications including media, entertainment, and a wide variety of retail websites across dozens of commerce driven verticals A/B test dynamic content routinely. Creating automated tests for such applications is extremely difficult and time consuming, so typically teams end up testing them manually which is slow and prone to error.

Visual AI really shines for these use cases. True computer vision technology like Applitools provides different Visual AI algorithms purpose-built for a variety of use cases that require "a different set of eyes". We then summon these various Visual AI "Modes" and apply them to specific "Regions" on the page in order to compare screenshots differently. In the case of dynamic content, Visual AI relies on a mode called "Layout". As the name suggest, it is to inspect the layout integrity of the page while ignoring the dynamic changes within that structure.

Use Case 5: Test Dynamic Content

MAKE SURE BOTH DYNAMIC ADS SHOW UP



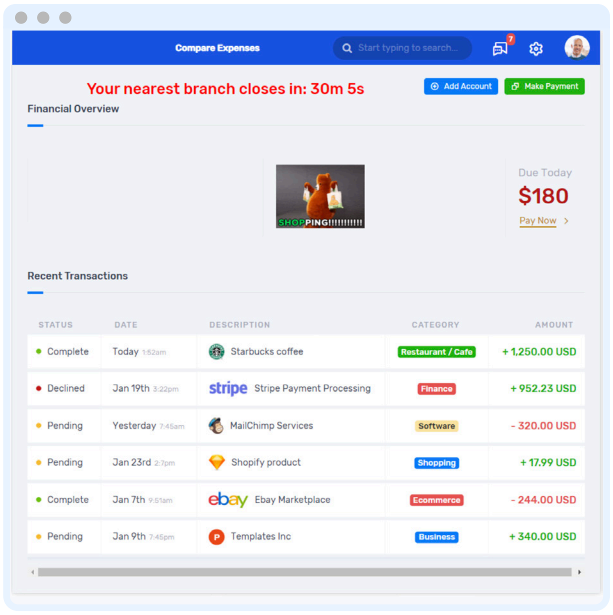
Your nearest branch closes in: 30m 5s

Financial Overview

Due Today
$180
Pay Now

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.00 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD
Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.00 USD



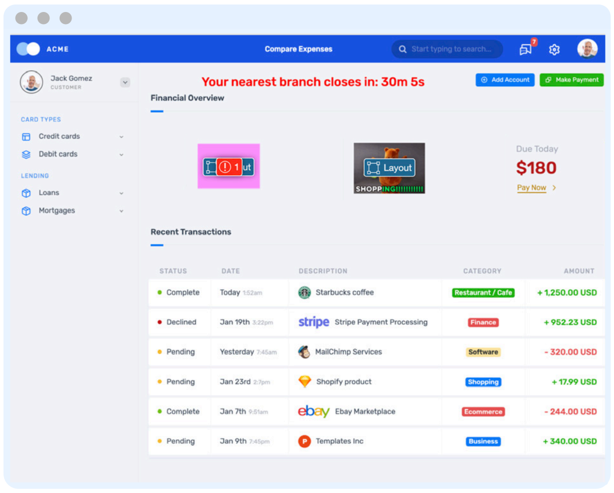
Your nearest branch closes in: 30m 5s

Financial Overview

Due Today
$180
Pay Now

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.00 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD
Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.00 USD



Your nearest branch closes in: 30m 5s

Financial Overview

Due Today
$180
Pay Now

Recent Transactions

STATUS	DATE	DESCRIPTION	CATEGORY	AMOUNT
Complete	Today 1:52am	Starbucks coffee	Restaurant / Cafe	+ 1,250.00 USD
Declined	Jan 19th 3:22pm	stripe Stripe Payment Processing	Finance	+ 952.23 USD
Pending	Yesterday 7:45am	MailChimp Services	Software	- 320.00 USD
Pending	Jan 23rd 2:7pm	Shopify product	Shopping	+ 17.99 USD
Complete	Jan 7th 9:51am	ebay Ebay Marketplace	Ecommerce	- 244.00 USD
Pending	Jan 9th 7:45pm	Templates Inc	Business	+ 340.00 USD

Bug: One of the dynamic ads is missing.

Solution: Mark both dynamic regions as "Layout regions" to these areas of the application using Visual AI. If you hit a bug, add a bug region on top of the missing ad to indicate the problem.

Measure Definitions

Time To Write Tests - The amount of time required to both author and successfully run new tests expressed in hours and minutes.

Number of Locators - Number of DOM locators used to provide test coverage in a test automation framework expressed as an absolute number.

Number of Labels - Number of field labels, or error messages used to provide test coverage in a test automation framework expressed as an absolute number.

Raw Lines of Test Code - Total lines of test code across all files written by the quality engineer for a given test suite expressed as an absolute number.

Lines of Test Code Per Hour - An average number of lines of test code written by the engineer in an hour expressed as an absolute number per hour.

Test Code Efficiency - A measure of the combination of both the number of lines of test code a quality engineer can write per hour and the amount of coverage each one of those lines provides expressed as an absolute number.

Number of Potential Bugs - A count of all features and UI artifacts that could cause either visual or functional bug if the candidate release was released without them fixing

Number of Bugs Caught - A count of all visual and functional bugs caught pre-production

Test Coverage - A description of the amount test coverage a quality team is providing an application expressed as a percentage.

Hackathon Submitter Score - The score the Hackathon participant scored on a scale of 1-100 points with a 100 being a perfect score. These scores are expressed as percentages.

Data Dimensions

RESEARCH SEGMENTS

Hackathon Applicants - Any quality engineer that signed up for the Visual AI Rockstar Hackathon. There were a total of 3,024 applicants.

All 288 Submitters - Any quality engineer that successfully completed the hackathon project. This is the full sample used as a foundation for the study and amounted to 3,168 hours, or 80 weeks, or 1.5 years of quality engineering data.

Top 100 Winners - The top 100 quality engineers who secured the highest point total for their ability to provide test coverage on all use cases and successfully catch potential bugs using both code-based and Visual AI approaches.

Grand Prize Winners - The top 10 quality engineers who scored the highest on the hackathon.

TECHNICAL DIMENSIONS:

Code Based - Using Code Based frameworks exclusively to build test suites and catch potential bugs.

Plus Visual AI - Adding Applitools Visual AI In combination with code based frameworks to build test suites and catch potential bugs.

Impact of Visual AI - The impact Applitools Visual Artificial Intelligence (AI) has on all measures in the study when used in combination with Code Based frameworks.

Glossary of Key Terms

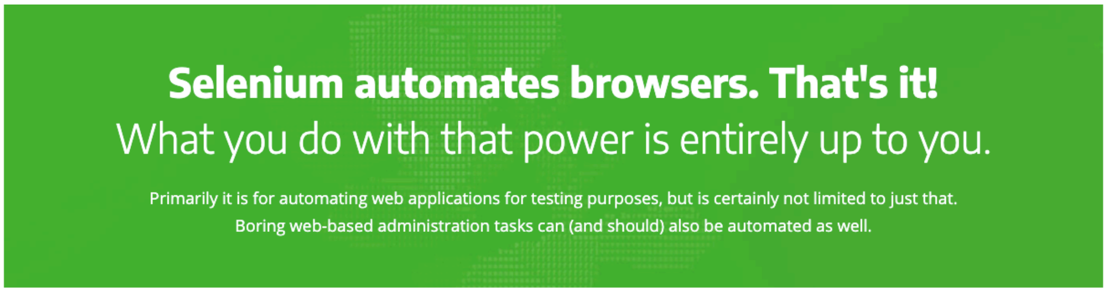
CI/CD - The continuous integration and deployment of feature code used by modern engineering teams.

Digital Transformation - The 21st century transition of brand and companies from the physical world to the digital world across all aspects of the business from marketing to sales to delivery to support.

Shift Left - Describes a quality management approach whereby feature developers assume responsibility for the quality of the features they develop often leveraging unit testing and other automated testing techniques applied prior to code check in.

Visual Testing - Testing a web or native mobile application by looking at the fully rendered pages and screens as they appear before customers. This was historically done manually or by using error prone pixel matching and DOM based tools, but more recently Applitools Visual AI has modernized and automated visual testing with 99.9999% accuracy and made it vital to Agile and CI/CD DevOps processes.

Browser Automation Tools - Browser automation tools connect to browsers and automate navigation, button clicks, data entry, and the return of this data from a DOM element within the browser. This leads to their main use of automating different types of common user paths and tasks through the browser. One major application of these browser automation tools is testing, but others include web scraping, taking screenshots, etc. Note that, despite the popular misconception, these tools are not purpose built test automation tools, but rather browser automation tools re-purposed for testing as one of their main use cases. The largest browser automation tool is Selenium which emerged in the late 90's as browser came on the scene.



Selenium automates browsers. That's it!

What you do with that power is entirely up to you.

Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that.
Boring web-based administration tasks can (and should) also be automated as well.

Functional Testing - A way to test if the application's functionality is working as intended for the user.

Glossary of Key Terms

Code Base Functional Testing - This is a common approach to automated functional testing whereby people do the following:

1. Use browser automation tools such as Selenium to perform navigation and data entry (form filling)
2. Write additional test code to grab data displayed in the browser using Selenium and/or
3. Use an assertion library such as JUnit.assert or Chai to validate that data, again using more test code

Visual AI Based Functional Testing - This is a modern approach to automated functional and visual testing whereby people replace test code with a screenshot of the end-state of the functionality, then compare that screenshot to the expected outcome every time they run the test. Since all functionality has an associated UI change, this will show a bug if the screenshot differs. To automate this type of testing, people do the following:

1. Use browser automation tools such as Selenium to perform navigation and data entry (form filling).
2. Replace test code with screenshots using the appropriate Applitools SDK, then let Applitools Visual AI manage the diffs.

Cypress (cypress.io) - A JavaScript based Test automation tool. You can find more info here

Webdriver.IO (webdriver.io) - A JavaScript based Test automation tool. You can find more info here

Selenium (www.selenium.dev) - A popular browser automation tool that's used heavily for functional testing. You can find more info here

Test Automation University (testAutomationu.com) - Free test automation courses with videos, transcripts, quizzes, credits, ranks badges, and certificates!

Visual AI Rockstar Hackathon - A global online hackathon that Applitools ran in November 2019.

Test Creation - Authoring or writing automation tests most often using a programming language (such as Java), an assertion library (JUnit.assert), a test framework(Junit) and a browser automation tool (Selenium).

Glossary of Key Terms

Test Maintenance - Updating previously created tests that are failed in the new run or fixing tests that are throwing false positives because of the changes in an application that's being tested.

Test Locators - These are DOM element locators that are used by the browser automation tool to locate the element within the DOM and then interact with that DOM element. These interactions, for example, could be clicking or returning values displayed in a DOM element.

Test Labels - These are actual texts that are displayed on the app. For example: Error messages, Field labels etc. In order to verify that text is actually displayed, test automation engineers need to copy that label and hard code the value as an expected result. Then, they compare this hard-coded text with the actual value in future test runs to verify quality of the new candidate.

Test Code Stability - Stability is a description of how often your tests fail or throw false-positives due to changes in Test Locators and Test Labels. If a test uses a lot of locators and labels, then it's considered less stable because these locators and labels can change in insignificant ways at any point leading to false-positives.

Test Coverage - A description of the amount test coverage a quality team is providing an application expressed as a percentage. 100% coverage is the desired goal. While that 100% goal not achievable or even necessary in practical terms, current coverage levels are generally considered inadequate, to slow to be achieved, or both, for modern apps.

Test Code Efficiency - Similar to the concept of "code efficiency" that software developers look for in their work, test code efficiency measures both the number of lines of test code a quality engineer can write per hour and the amount of coverage each one of those lines provides.

About Applitools

Applitools enables Visual AI powered test automation to help teams release high-quality web and mobile apps faster and more efficiently.

Applitools Visual AI modernizes important test automation use cases -- Functional Testing, Visual Testing, Web and Mobile UI/UX Testing, Cross Browser Testing, Responsive Web Design Testing, Cross Device Testing, PDF Testing, Accessibility Testing and Compliance Testing -- to transform the way organizations deliver innovation at the speed of CI/CD at a significantly lower Total Cost of Ownership (TCO).

Hundreds of companies from verticals such as Software, Banking, Insurance, Retail, Pharmaceuticals, and Publishing -- including 50 of the Fortune 100 -- use Applitools to deliver the best possible digital experiences to millions of customers on any device and browser, and across every screen size and operating system.

Applitools is headquartered in San Mateo, California, with an R&D center in Tel Aviv, Israel

Media Contact:

Jeremy Douglas
Catapult PR-IR
+1 303-581-7760, ext. 16
jdouglas@catapultpr-ir.com

LEARN MORE AT

applitools.com

Footnotes and Additional Resources

1. 2019 State of Automated Visual Testing: Continuous Quality in the Age of Digital Transformation
2. Innovation Insight Report
3. Jonathan Lipps Presentation
4. Modern Functional Test Automation Through Visual AI
5. NowTech Report
6. Critical Capabilities Report
7. Solution Paths Report
8. Hackathon Homepage