Continuous Reliability:

An Agile Process to Deliver Higher Quality Applications

Rich Borucki

Agenda

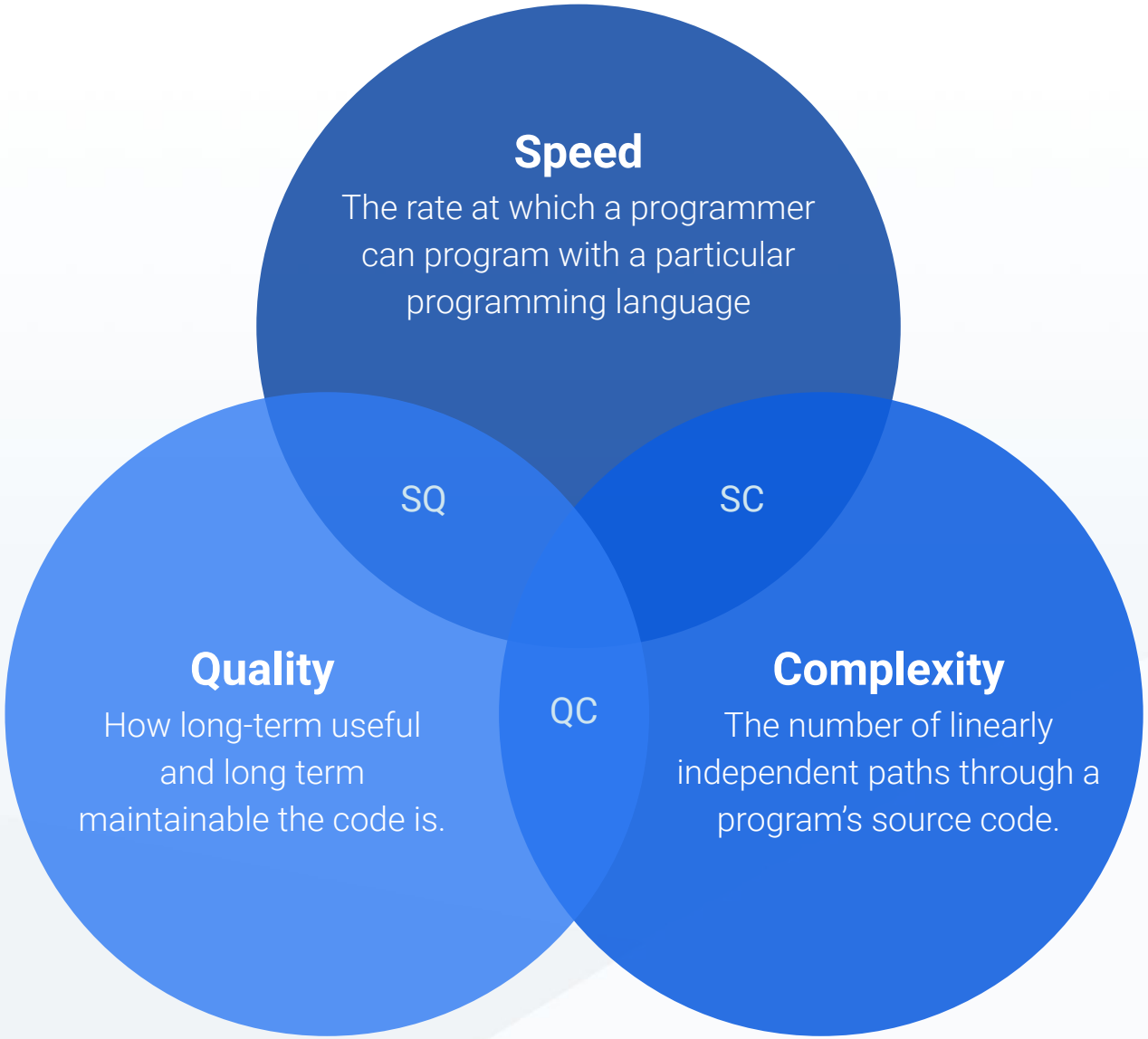
- What is Continuous Reliability?
- Current Challenges Delivering Quality Applications
- The Path to Continuous Reliability
- Q&A

What is Continuous Reliability?

Reliability is the **probability of a piece of software operating without failure while in a specified environment over a set duration of time.** In a perfect world, a reliable piece of software is completely defect free, does not create downtime and performs correctly in every scenario.

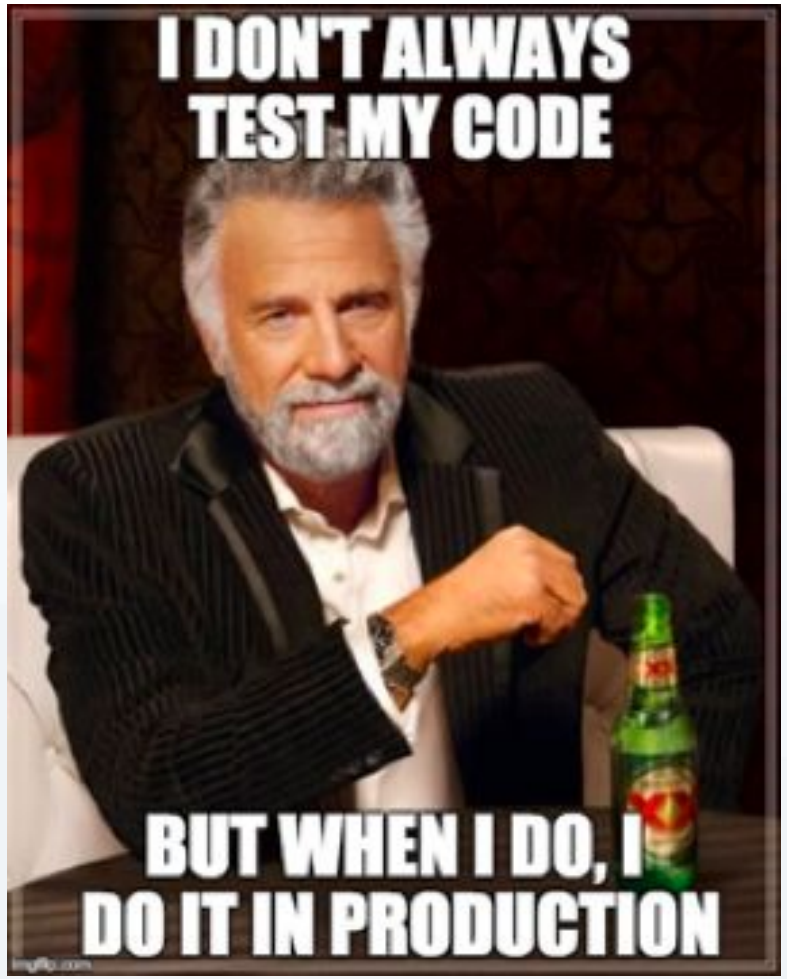
What is Continuous Reliability?

Continuous Reliability is the **notion of balancing speed, complexity and quality by taking a continuous, proactive approach to reliability** across the software delivery life cycle. It is achieved through the creation of data-driven quality gates and feedback loops.



Current Challenges

Applications Aren't Truly Tested Until Production



There is a lot of dev humor about this...

Reliability Challenges

Technical debt

3rd-party code

Finger-pointing

No way to prioritize

Unknown error number

Lack of context

Legacy Tools & Processes

Tools/Processes that put application quality at risk:

- Manual code reviews – peer reviews
- Manual testing – QA team writes and runs test cases
- Basic unit testing of methods – Devs write tests against methods
- Manual deployments
  - Wait for the QA region to become available
  - Manual build and deploy of applications

Questions to Ask Ourselves

How do we know when to promote code?

- “Head of QA gives the green light”
- “Query APM or Logs and see if there are less errors than last build”
- “Don’t have a choice because management wants features released”

How do we know if we have introduced new or resurfaced errors?

- “Don’t really know...all tests passed”
- “We do a lot of regex queries against the logs and we think we know”

How do we know release-over-release if our code is improving?

- “Less customer complaints”
- “Less Sev1 issues”
- “Don’t really know”

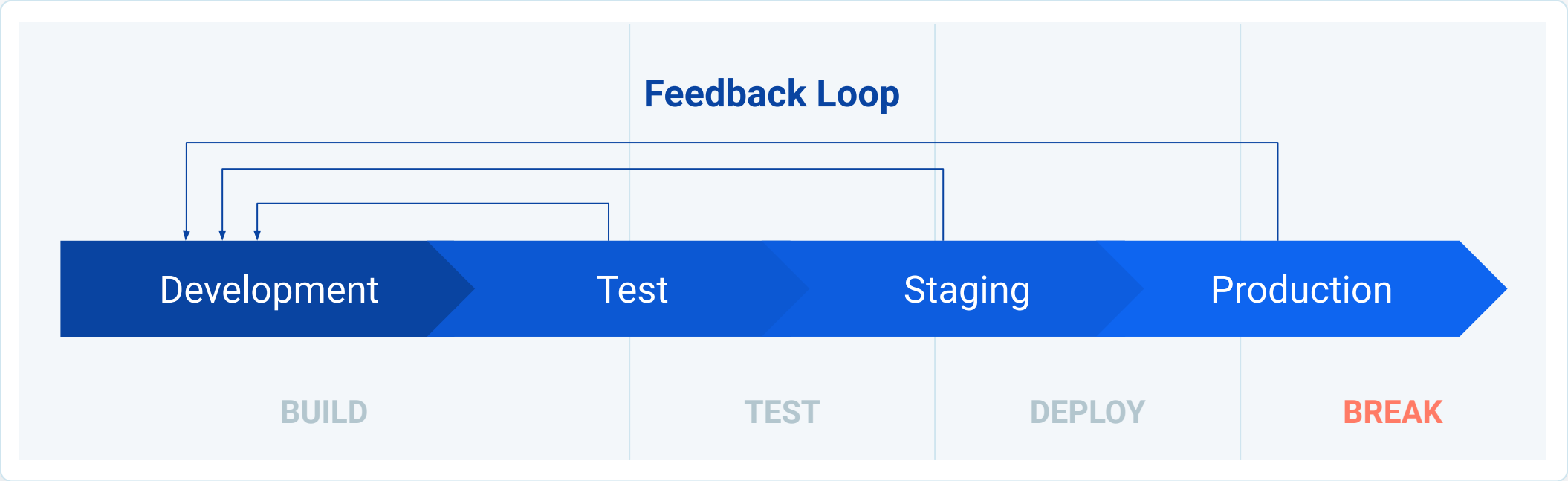
The Path to Continuous Reliability

OverOps

We Need to Build Smarter CI/CD Pipelines

Developer feedback, alerting, routing, context, metrics

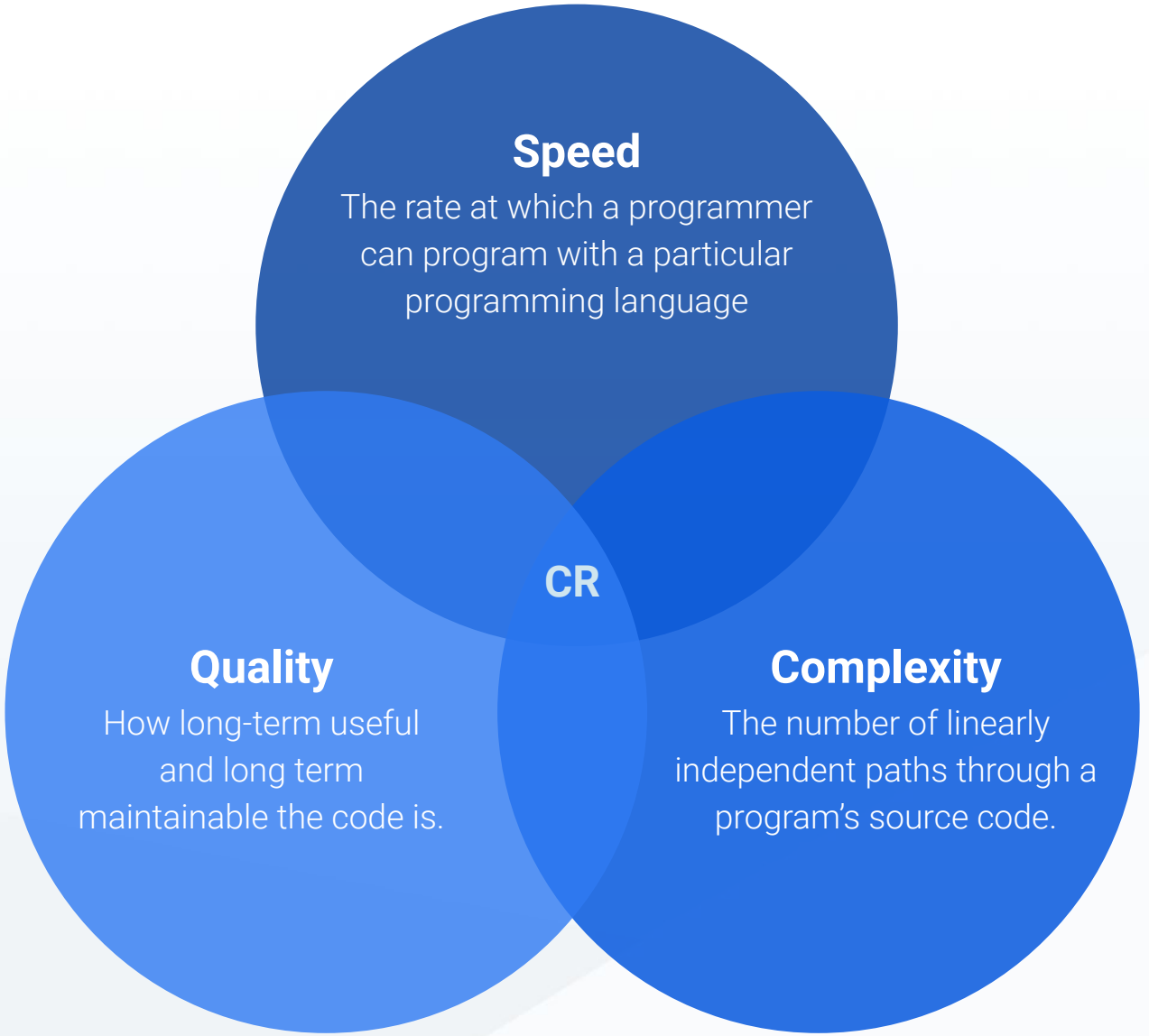
Automate as much as possible to eliminate human intervention



CI/CD Workflow

Continuous Reliability (CR) Roadmap

- **Automate** as much as possible to eliminate human intervention
- **Know** your unknowns
- **Identify** errors sooner – shift left
- **Capture** more context to provide feedback to developers
- **Create** more/better code coverage
- **Collect** better data – metrics matter
- **Build** a culture of accountability



OverOps

Automate...Automate...Automate

- Static code analysis tools – automated approach
  - Goal is to identify weaknesses in code that might lead to vulnerabilities before testing
- Mix of JUnit, JMeter, Selenium, Load Runner, etc.
  - New tools emerging to identify gaps in code coverage
- Limited to no manual testing
- Automated builds and deployments (CI/CD)
  - QA environment auto build and teardown
- Dynamic code analysis tool – automated way to detect, deduplicate, alert and measure errors while code is being exercised
- Feedback loops as code moves through the pipeline
  - Quality gates identifying issues that stops a build from getting promoted

Know Your Unknowns

Total Visibility:
I know what I know

Manageability:
I know what I don't know

Challenge:
I don't know what I don't know

OverOps

Know Your Unknowns

- Critical exceptions – Nullpointers, IndexOutOfBounds, etc.
- Slowdowns – which processes are running slower
- Errors that are happening more frequently
- Unhandled and swallowed exceptions – don't show up in the logs
  - Little known fact – it takes 50-100 microseconds to generate the exception

```
STRING VALUE == NULL;
TRY  {
          IF (VALUE.EQUALSIGNORECASE ("HELLO") )
{
              // DO SOMETHING IF THEY MATCH
          }
} CATCH  {EXCEPTION E)  {
     // TODO: HANDLE EXCEPTION
     // I FORGOT TO HANDLE THIS....OOPS
}
```

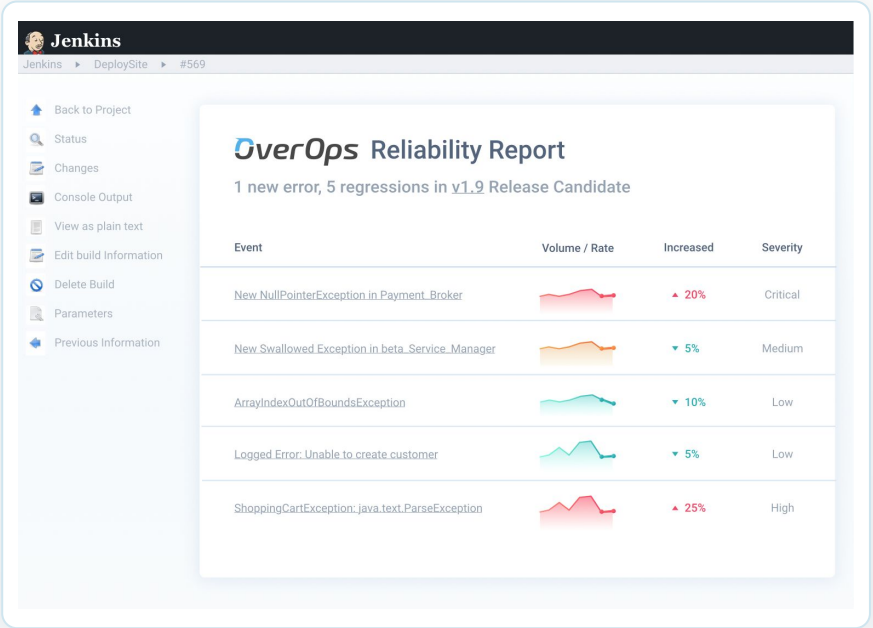
```
BOOLEAN PASS == TRUE;

TRY  {
     INTEGER.PARSEINT (VALUE);
} CATCH  {EXCEPTION E)  {
   PASS == FALSE;
}
```

OverOps

Shift Left

- Test early and often – daily, hourly, at code check-in, etc.
- Implement quality gates to ensure code quality
  - Static and dynamic gates
  - Instant feedback to Dev team

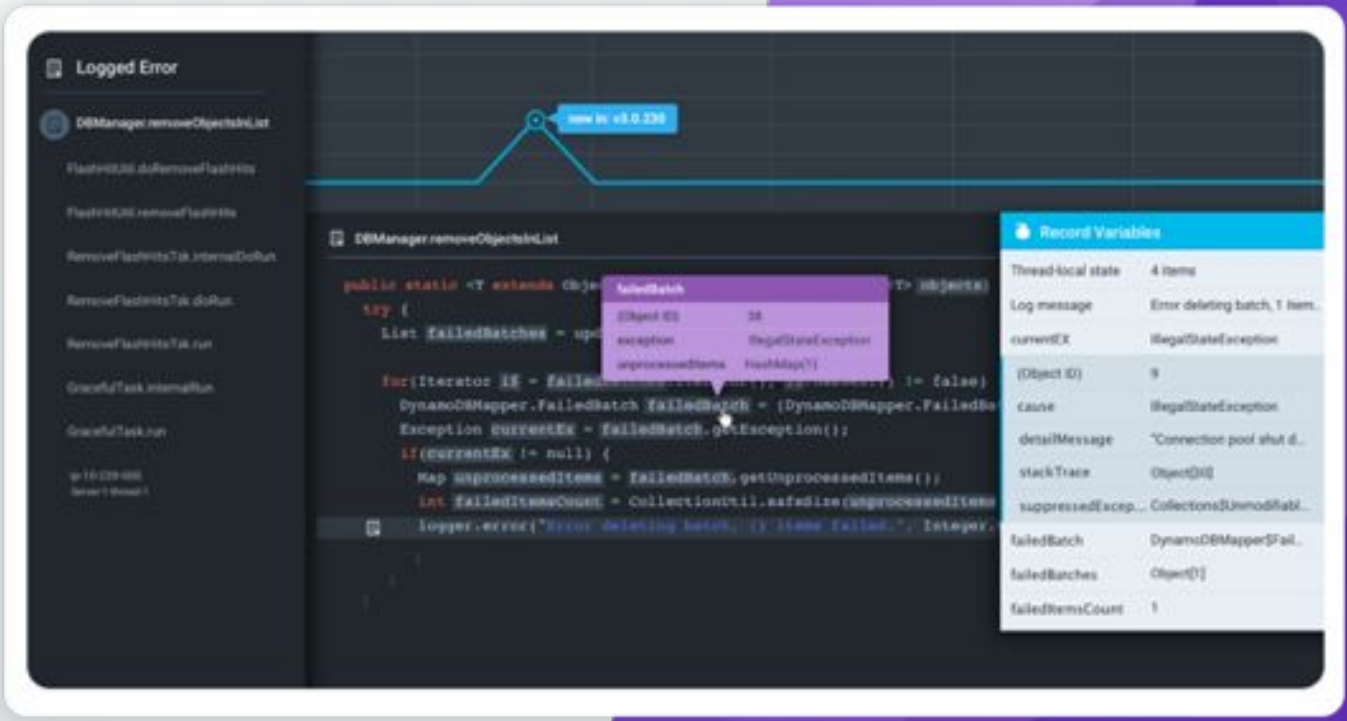


Shift Left = Early Detection + Cost Effectiveness + Time Saving + Release Smooth

Context Beyond the Stack Trace

Shameless OverOps plug

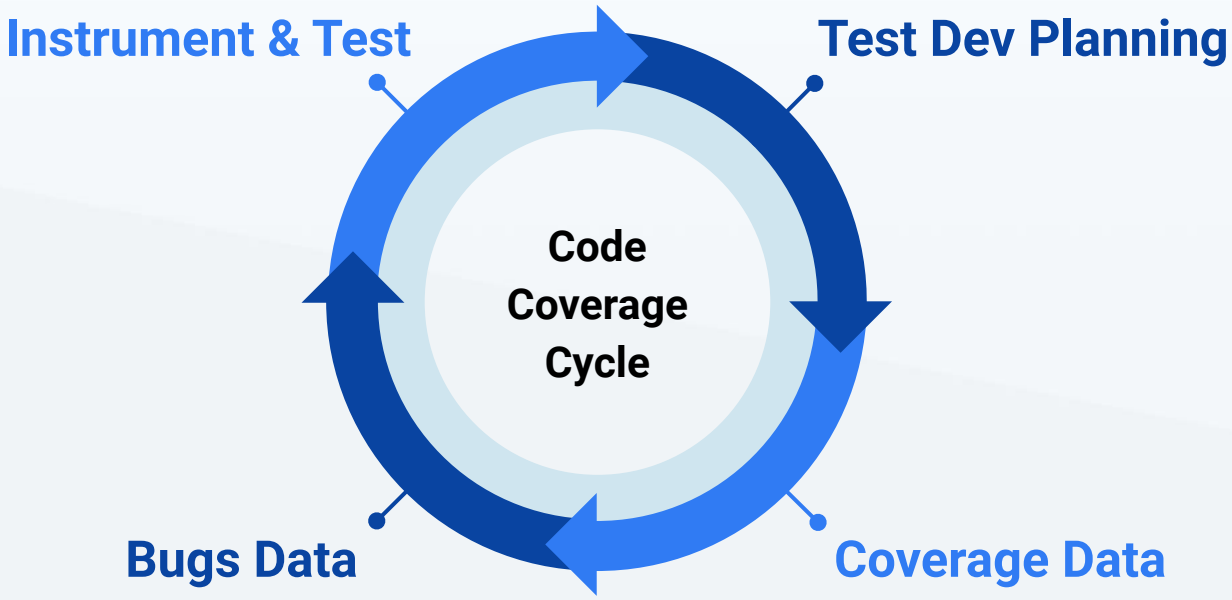
- Dynamic code analysis
- Detects 100% of all errors
  - Even errors not in the logs
- Provides metrics
  - Error counts and rates
- Detects slowdowns
- Detects increasing errors



OverOps

Create More/Better Code Coverage

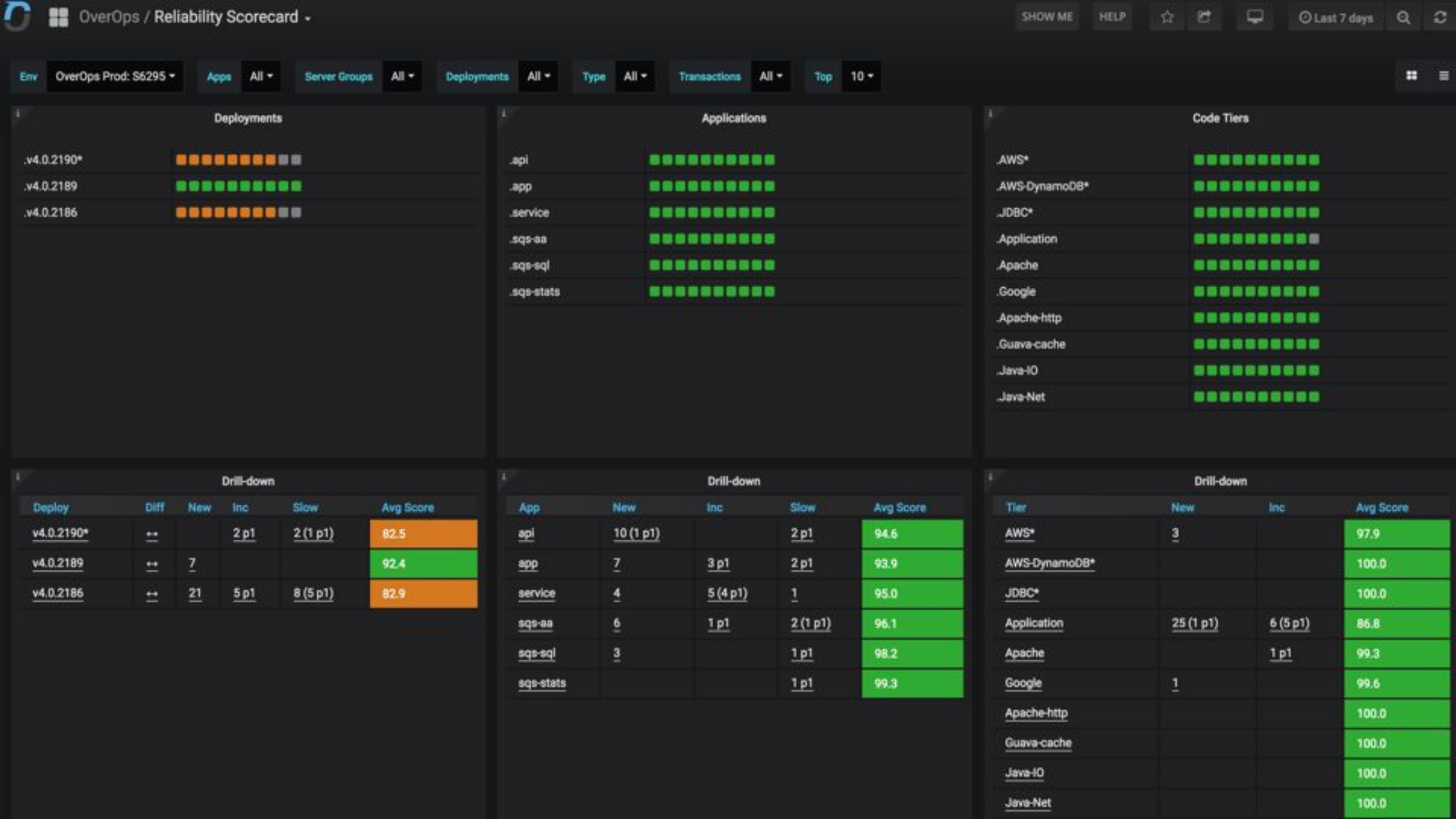
- Implement new tools to help detect code coverage gaps
- Leverage context from tools like OverOps to help create tests
  - Context of code path, data, environment, etc.



OverOps

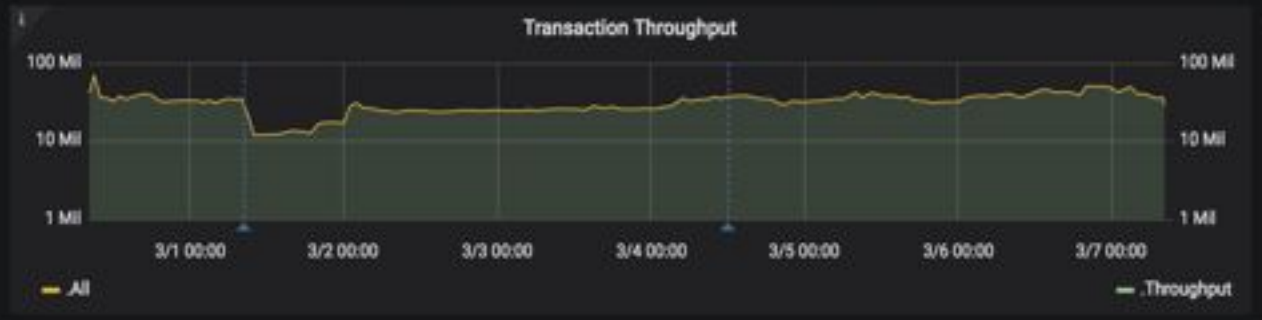
Metrics Matter

- Build a metrics hub
  - Error data from applications, databases, middleware, queues, etc.
  - System metrics, CPU, memory, GC, database connections, file handles, threads, etc.
- Correlate data to identify anomalies, build self-healing patterns, identify potential issues earlier
- Use machine learning to look at past issues to predict future issues
- Know release-over-release code quality
- Application health scorecard – weighted average of errors & slowdowns
- Know who your best developers are


OverOps / Reliability Scorecard
SHOW ME
HELP
Last 7 days
Env
OverOps Prod: S6295
Apps
All
Server Groups
All
Deployments
All
Type
All
Transactions
All
Top
10
Deployments
.v4.0.2190*
.v4.0.2189
.v4.0.2186
Applications
.api
.app
.service
.sqs-aa
.sqs-sql
.sqs-stats
Code Tiers
.AWS*
.AWS-DynamoDB*
.JDBC*
.Application
.Apache
.Google
.Apache-http
.Guava-cache
.Java-IO
.Java-Net
Drill-down
Deploy Diff New Inc Slow Avg Score
v4.0.2190* ↔ 2 p1 2 (1 p1) 82.5
v4.0.2189 ↔ 7 92.4
v4.0.2186 ↔ 21 5 p1 8 (5 p1) 82.9
Drill-down
App New Inc Slow Avg Score
api 10 (1 p1) 2 p1 94.6
app 7 3 p1 2 p1 93.9
service 4 5 (4 p1) 1 95.0
sqs-aa 6 1 p1 2 (1 p1) 96.1
sqs-sql 3 1 p1 98.2
sqs-stats 1 p1 99.3
Drill-down
Tier New Inc Avg Score
AWS* 3 97.9
AWS-DynamoDB* 100.0
JDBC* 100.0
Application 25 (1 p1) 6 (5 p1) 86.8
Apache 1 p1 99.3
Google 1 99.6
Apache-http 100.0
Guava-cache 100.0
Java-IO 100.0
Java-Net 100.0

Env OverOps Prod: S6295 | Apps All | Server Groups All | Deployments All | Type All | Transactions All

Reliability Score	Failed Transactions	Slow Transactions	New Errors	Error Increases	Unique Errors
83.6	0.01%	5	24	5	665



Culture of Accountability

It has become acceptable to have lots of errors in applications. When issues happen, we play the blame game, resulting in outages, Sev1 issues, customer incidents, etc.

What to do?

Change the status quo

- Bug bounties, bonuses for applications that meet SLAs
- Error burn down time for each sprint
- Set goals for code quality in your team

Hold everyone accountable

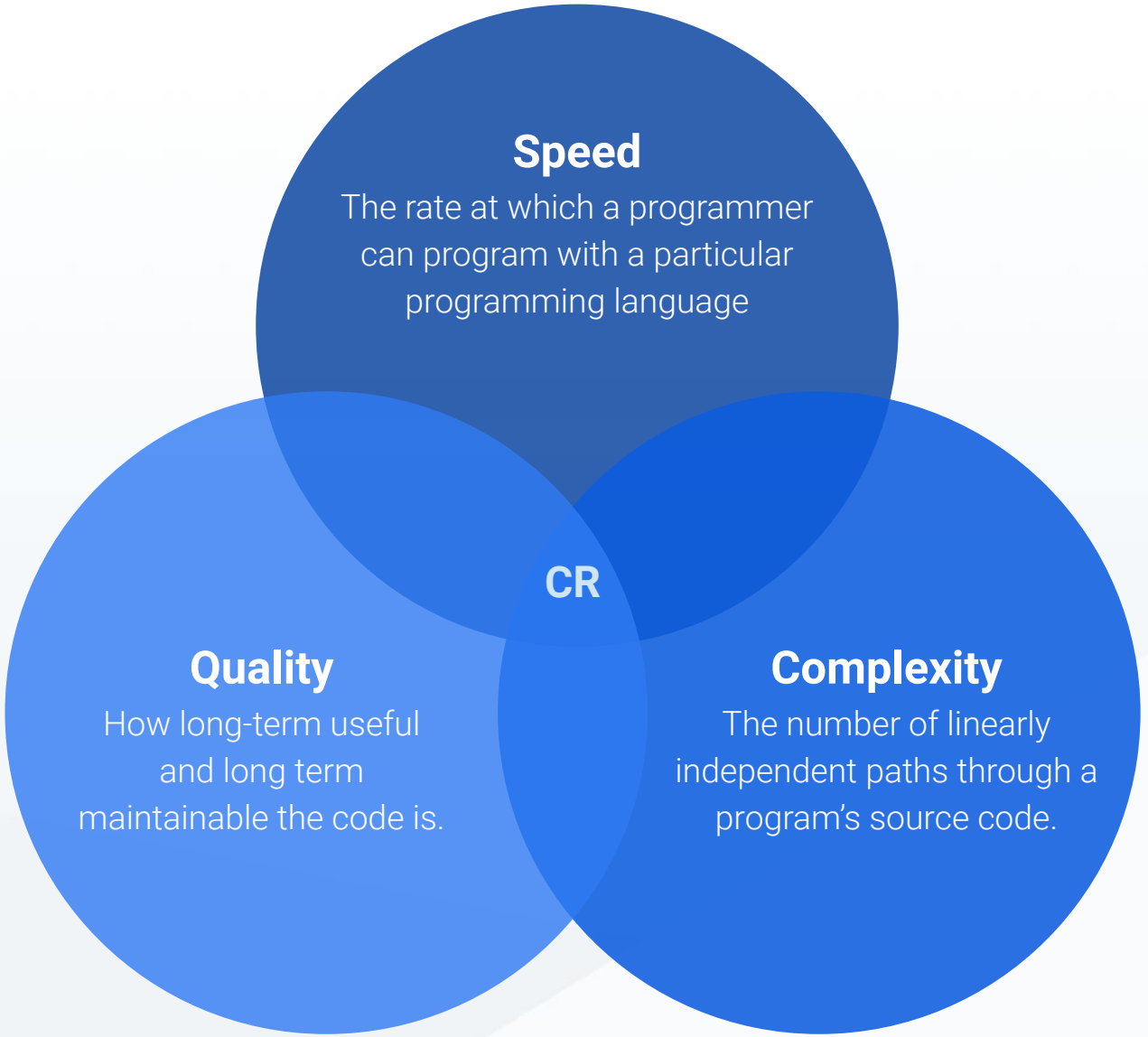
- Correlate all errors to commits and build a leaderboard
- Require the dev who wrote the code to fix it

Leverage tools like OverOps that detect 100% of errors and work to fix them all

- Don’t allow code to be promoted that has errors

Continuous Reliability (CR) Roadmap

- **Automate** as much as possible to eliminate human intervention
- **Know** your unknowns
- **Identify** errors sooner – shift left
- **Capture** more context to provide feedback to developers
- **Create** more/better code coverage
- **Collect** better data – metrics matter
- **Build** a culture of accountability



OverOps

Questions?