Testing on the IBM i

Introduction to Wim Jongman

Wim Jongman,
CTO of Remain Software

Dutchy from Haarlem

4 kids, one wife

Nerd

REMAIN SOFTWARE

Remain Software specializes in delivering innovative and reliable solutions for application lifecycle management (ALM), change management, workflow, and DevOps. With 30 years of experience, we offer tools designed to streamline software development processes across various platforms, including IBM i, Windows, and Unix/Linux.

REMAIN SOFTWARE

TD/OMS, our flagship product, is a scalable and user-friendly ALM solution that supports software changes, development, deployment, and modernization projects. TD/OMS enables development teams to collaborate effectively and manage software components throughout different stages of development.

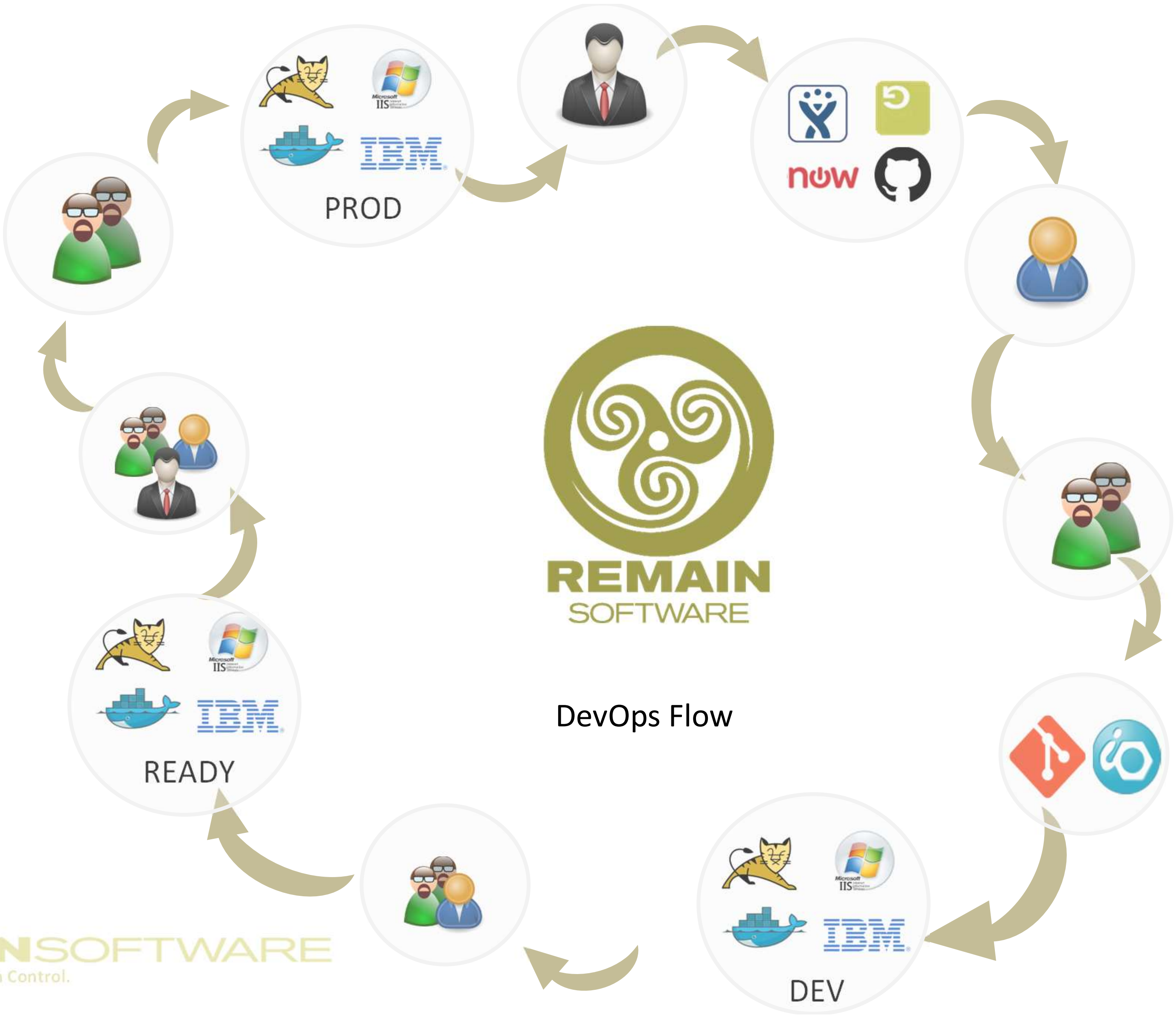
Happy DevOps teams use Remain Software

- IUnit – Open Source Unit Testing
- TD/OMS – Full Stack DevOps for IBM i
- Gravity - Ticketing and Workflow System
- OpenAPI Studio – REST Design and Generation
- OCTO – DevOps for IBM i Modernization
- XREF – Enterprise Source Cross Reference
- MiWorkplace – Lightweight RDi Replacement
- X-Analysis – Comprehensive Object Analytics
- ReplicTest – Full Testing Software for IBM i

REMAIN SOFTWARE

- https://remainsoftware.com
- https://miworkplace.com
- https://github.com/remainsoftware
- https://github.com/i-unit/iunit

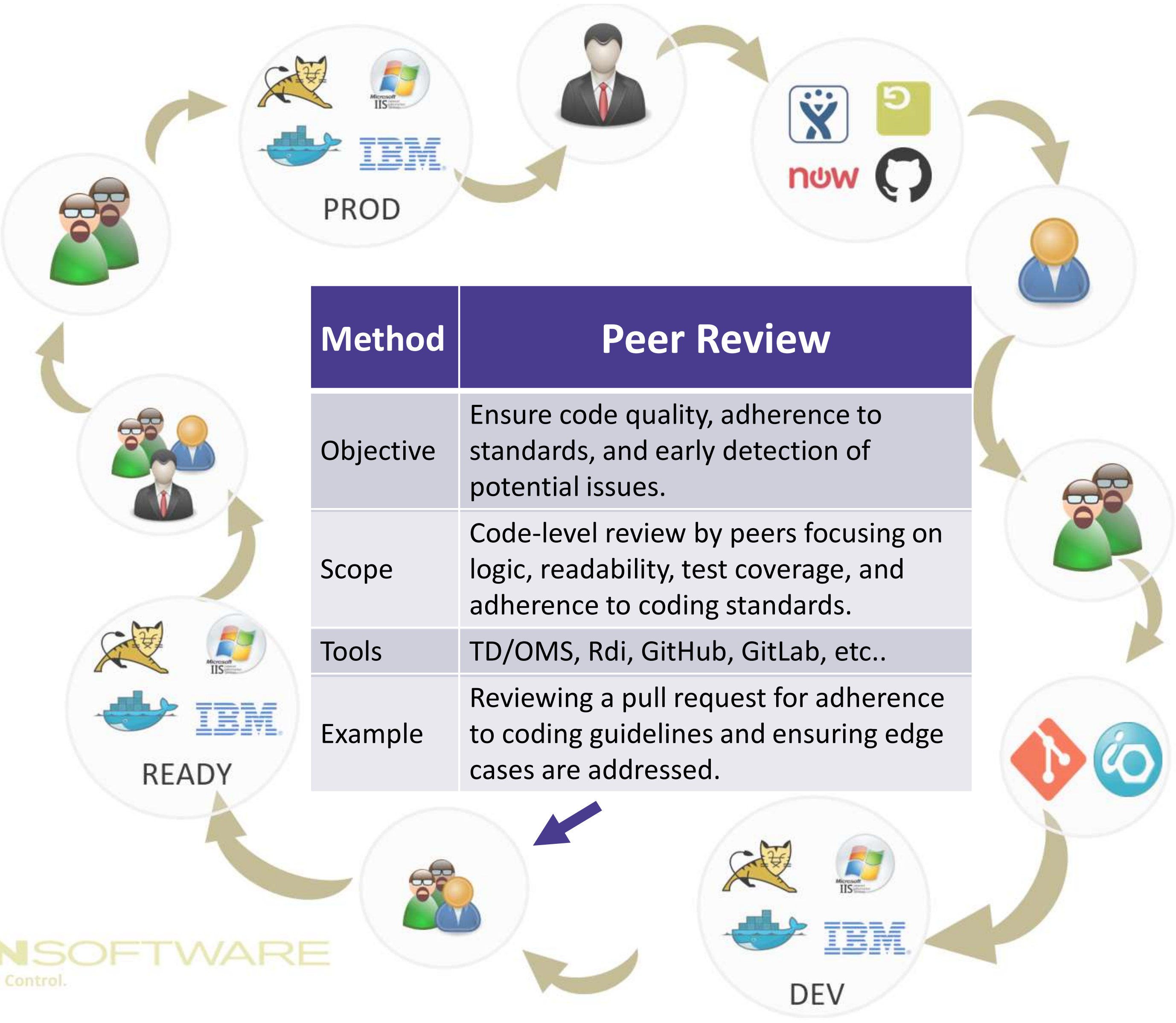
Testing in the Software Development Life Cycle



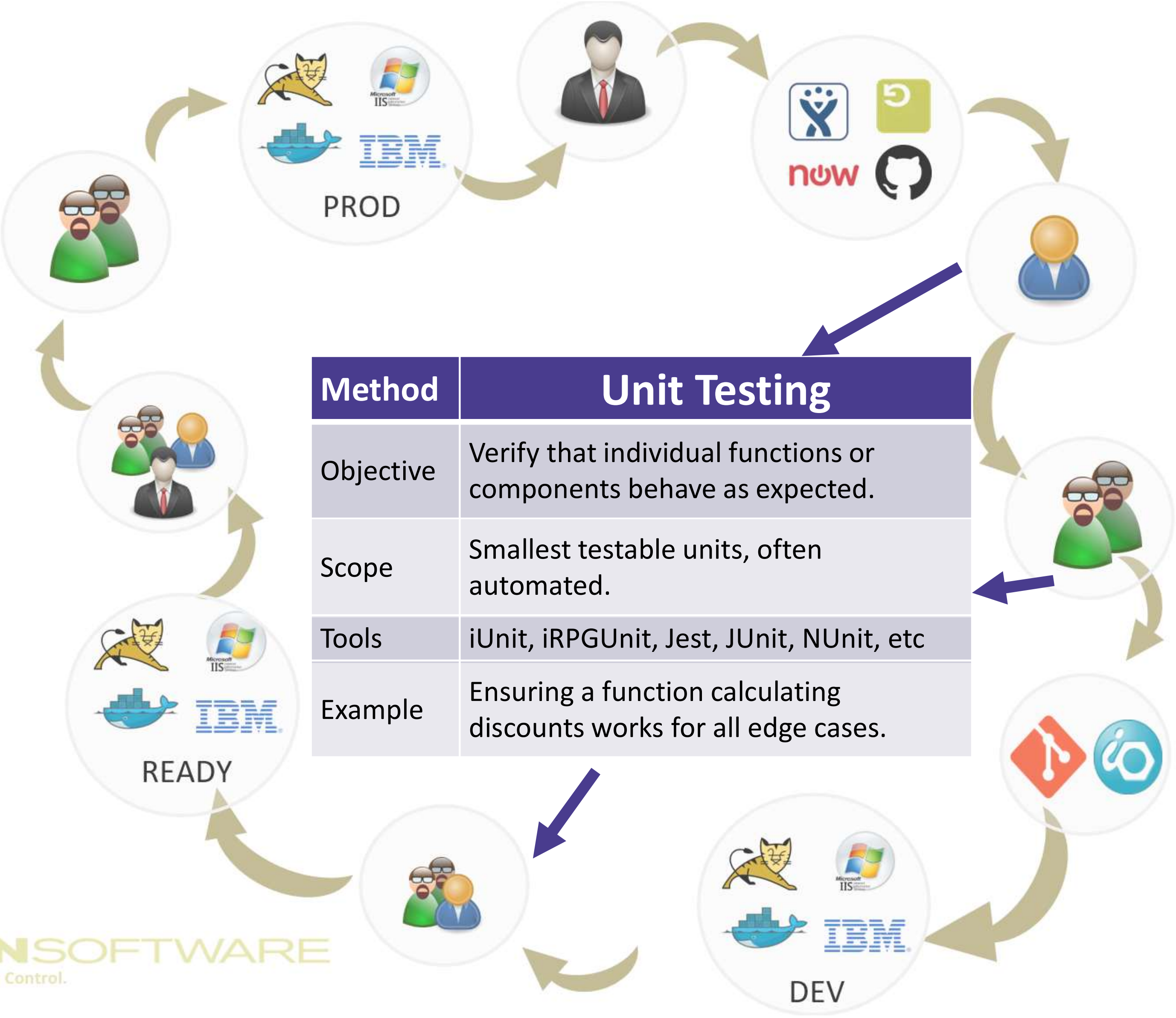
DevOps Flow

REMAINSOFTWARE

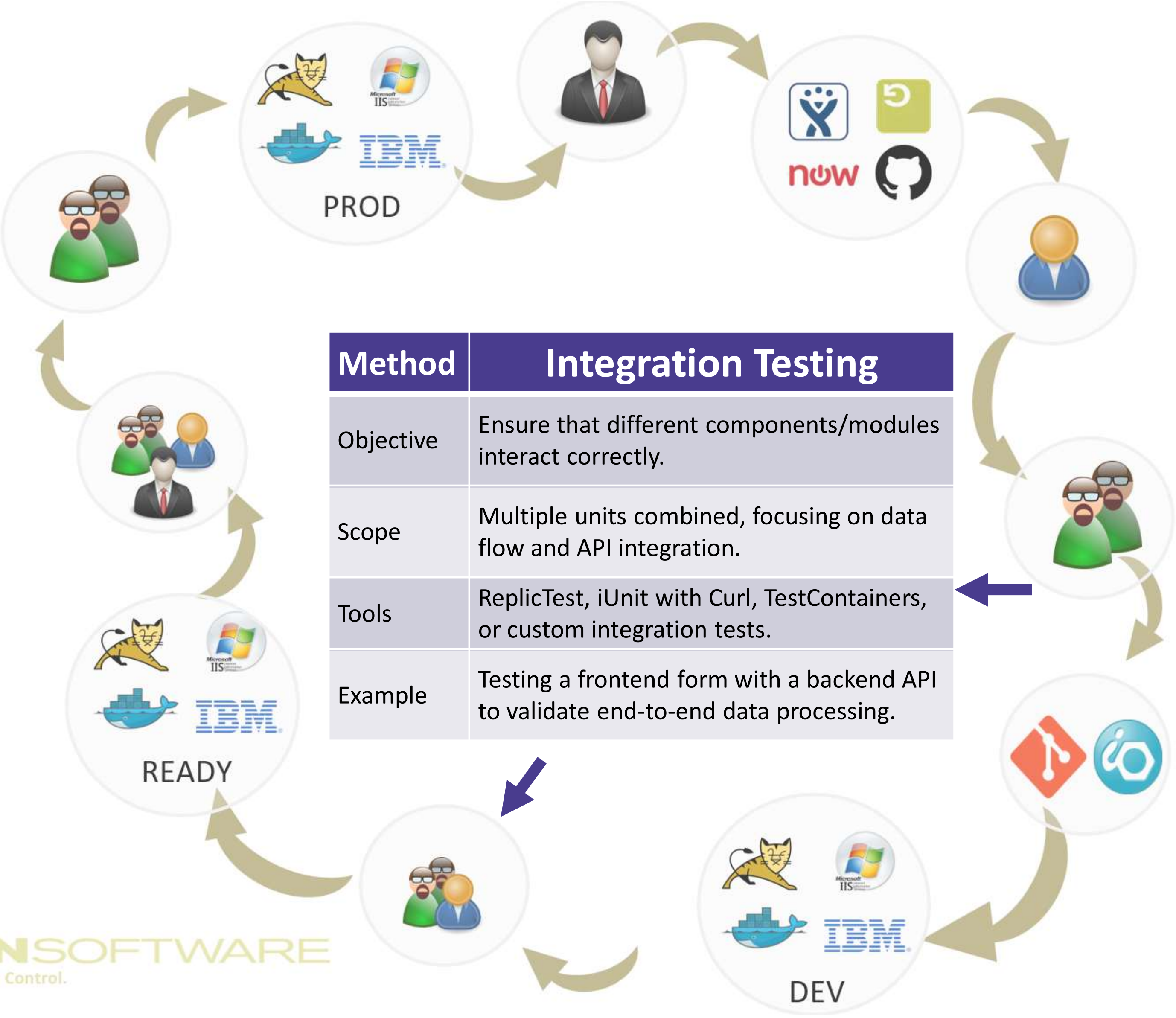
Embrace Change. *Remain* In Control.



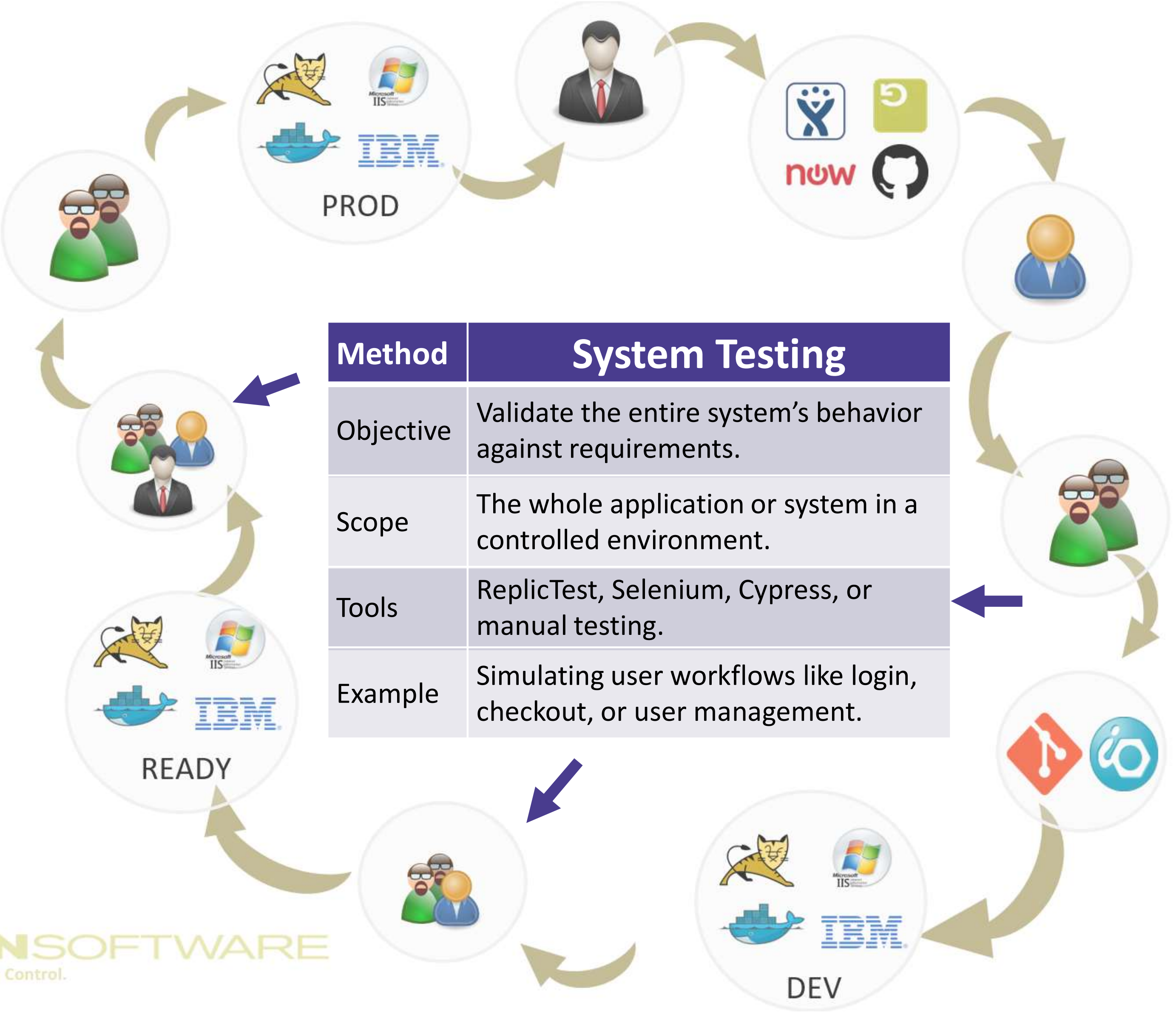
Method	Peer Review
Objective	Ensure code quality, adherence to standards, and early detection of potential issues.
Scope	Code-level review by peers focusing on logic, readability, test coverage, and adherence to coding standards.
Tools	TD/OMS, Rdi, GitHub, GitLab, etc..
Example	Reviewing a pull request for adherence to coding guidelines and ensuring edge cases are addressed.



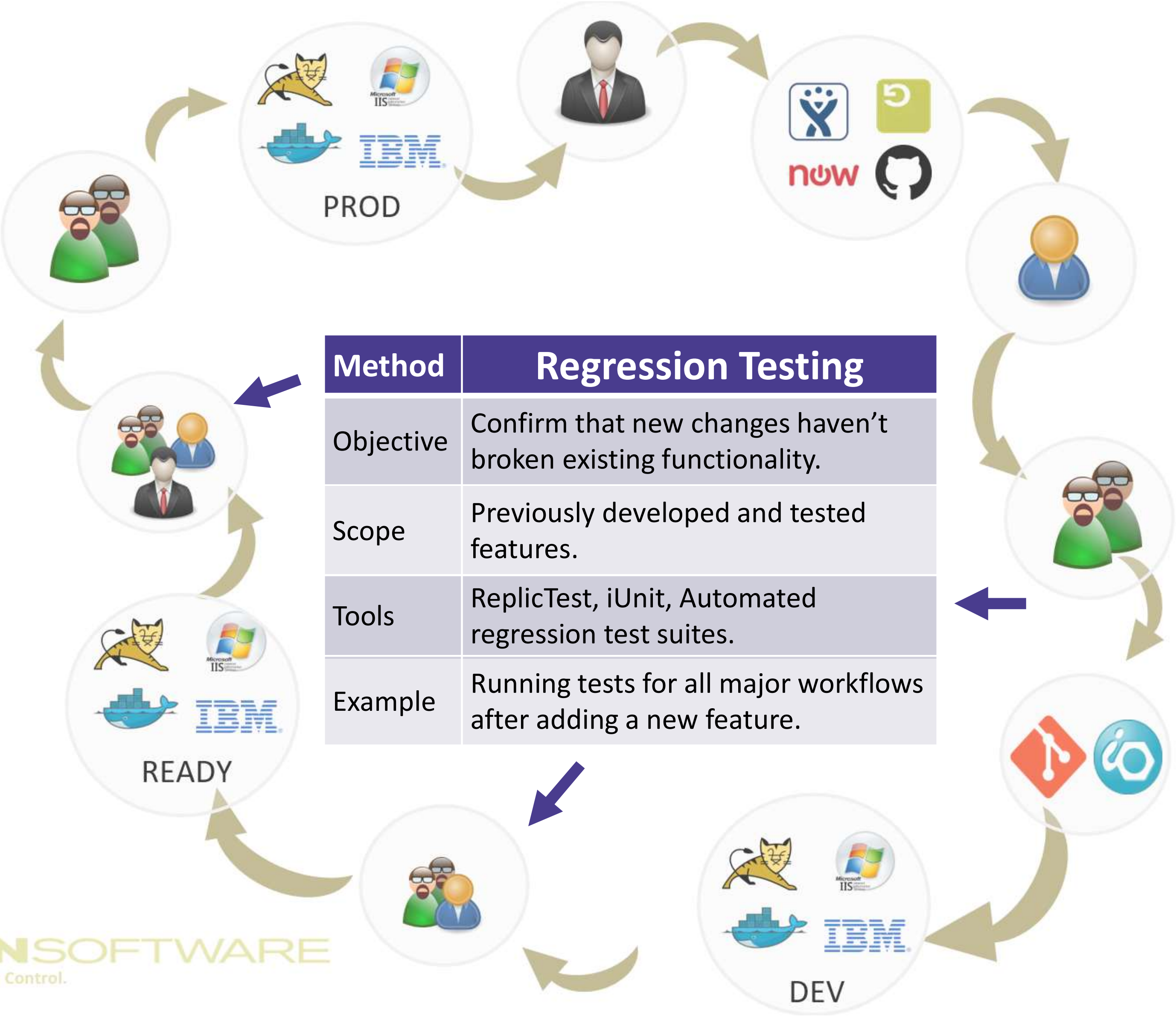
Method	Unit Testing
Objective	Verify that individual functions or components behave as expected.
Scope	Smallest testable units, often automated.
Tools	iUnit, iRPGUnit, Jest, JUnit, NUnit, etc
Example	Ensuring a function calculating discounts works for all edge cases.



Method	Integration Testing
Objective	Ensure that different components/modules interact correctly.
Scope	Multiple units combined, focusing on data flow and API integration.
Tools	ReplicTest, iUnit with Curl, TestContainers, or custom integration tests.
Example	Testing a frontend form with a backend API to validate end-to-end data processing.



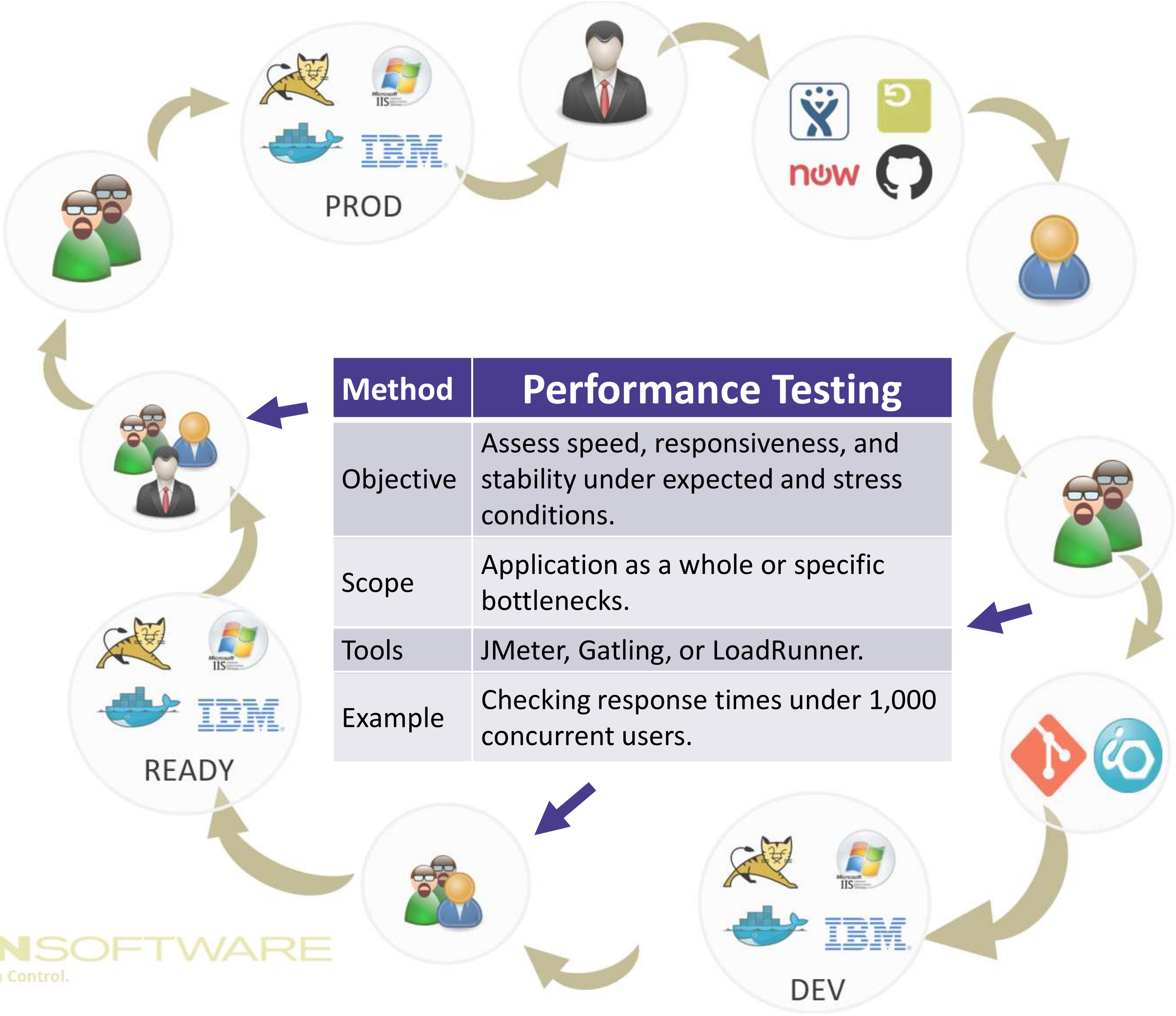
Method	System Testing
Objective	Validate the entire system’s behavior against requirements.
Scope	The whole application or system in a controlled environment.
Tools	ReplicTest, Selenium, Cypress, or manual testing.
Example	Simulating user workflows like login, checkout, or user management.



Method	Regression Testing
Objective	Confirm that new changes haven't broken existing functionality.
Scope	Previously developed and tested features.
Tools	ReplicTest, iUnit, Automated regression test suites.
Example	Running tests for all major workflows after adding a new feature.

REMAINSOFTWARE

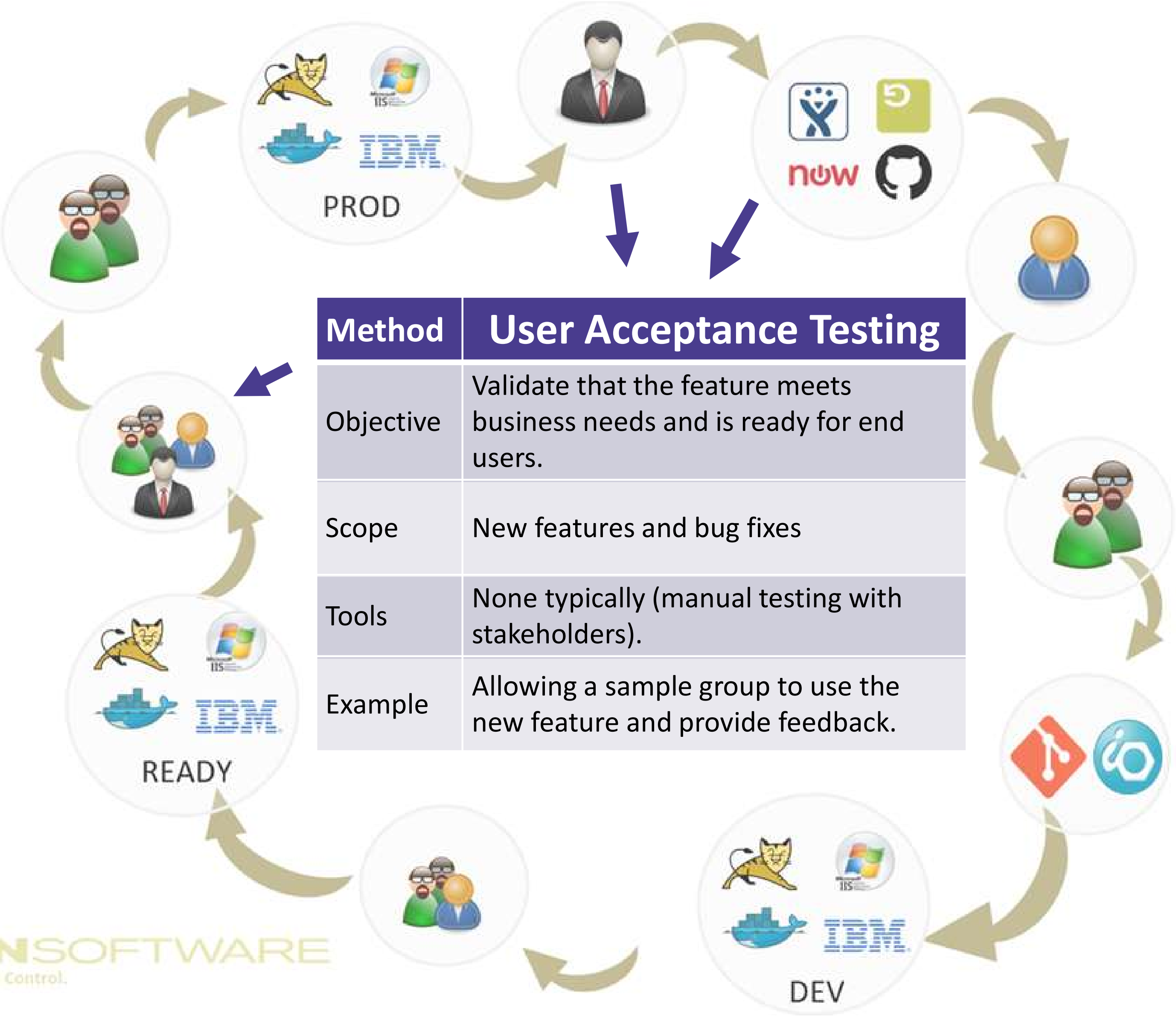
Embrace Change. Remain In Control.



Method	Performance Testing
Objective	Assess speed, responsiveness, and stability under expected and stress conditions.
Scope	Application as a whole or specific bottlenecks.
Tools	JMeter, Gatling, or LoadRunner.
Example	Checking response times under 1,000 concurrent users.

REMAINSOFTWARE

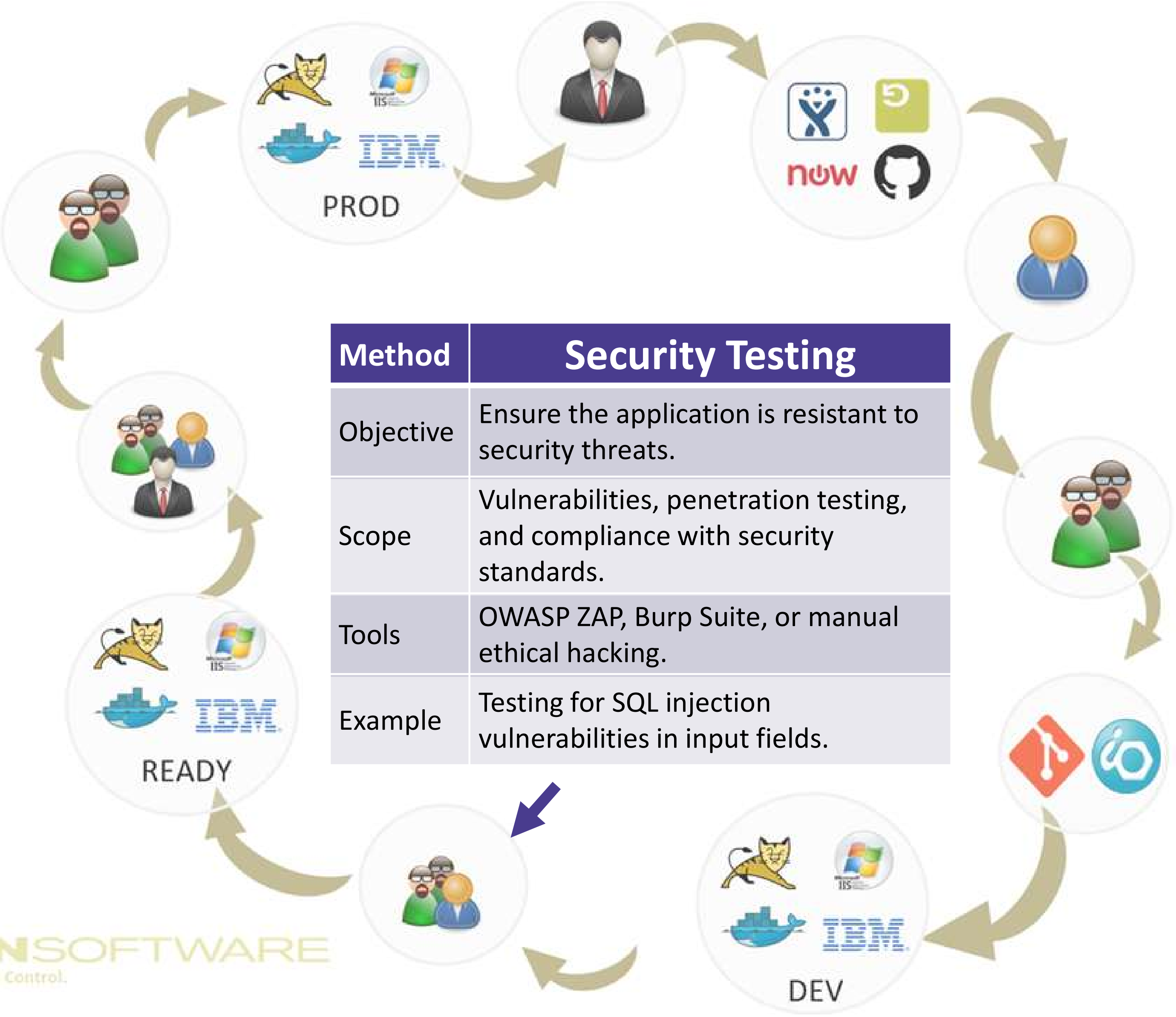
Embrace Change. Remain In Control.



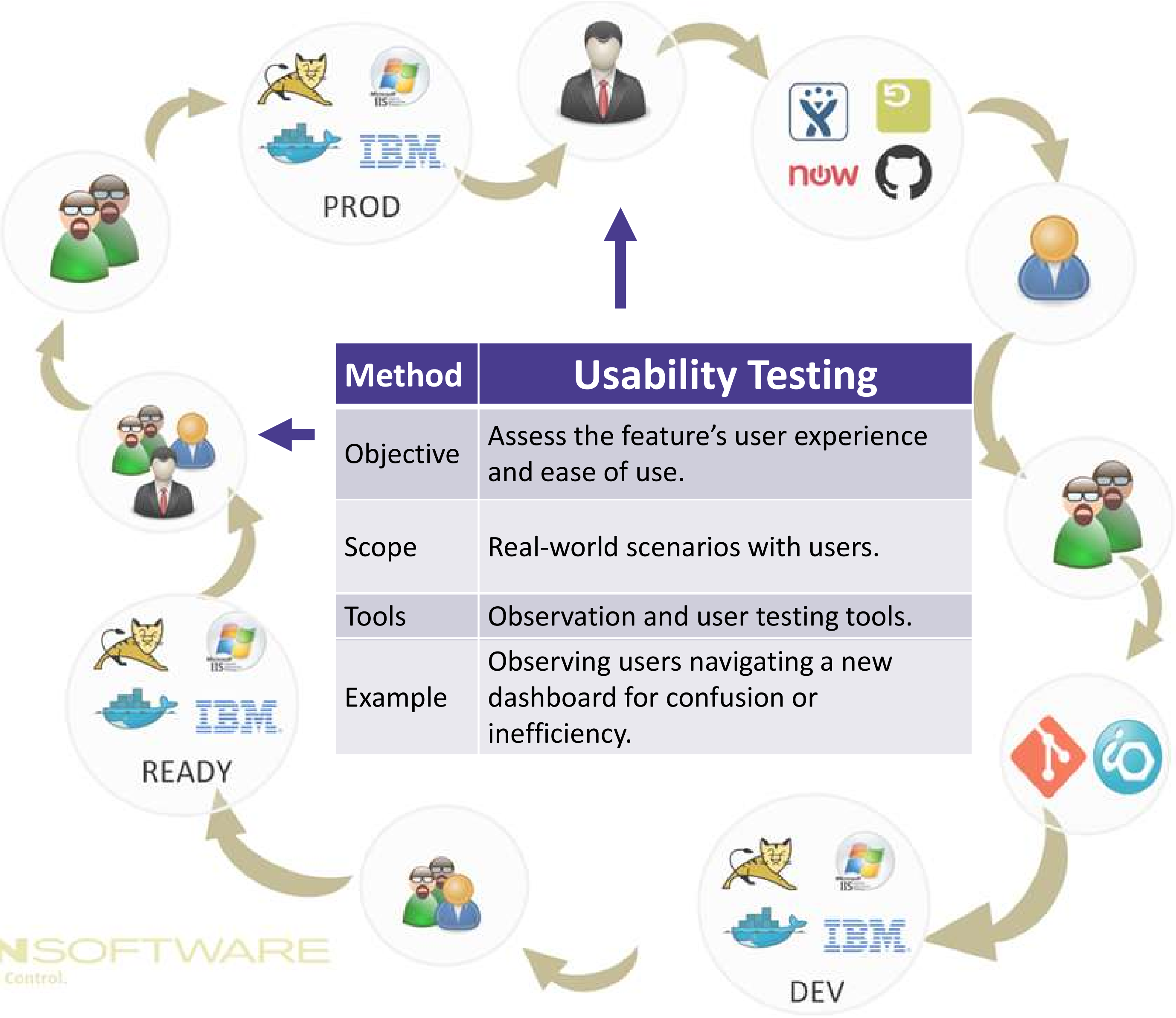
Method	User Acceptance Testing
Objective	Validate that the feature meets business needs and is ready for end users.
Scope	New features and bug fixes
Tools	None typically (manual testing with stakeholders).
Example	Allowing a sample group to use the new feature and provide feedback.

REMAINSOFTWARE

Embrace Change. Remain In Control.

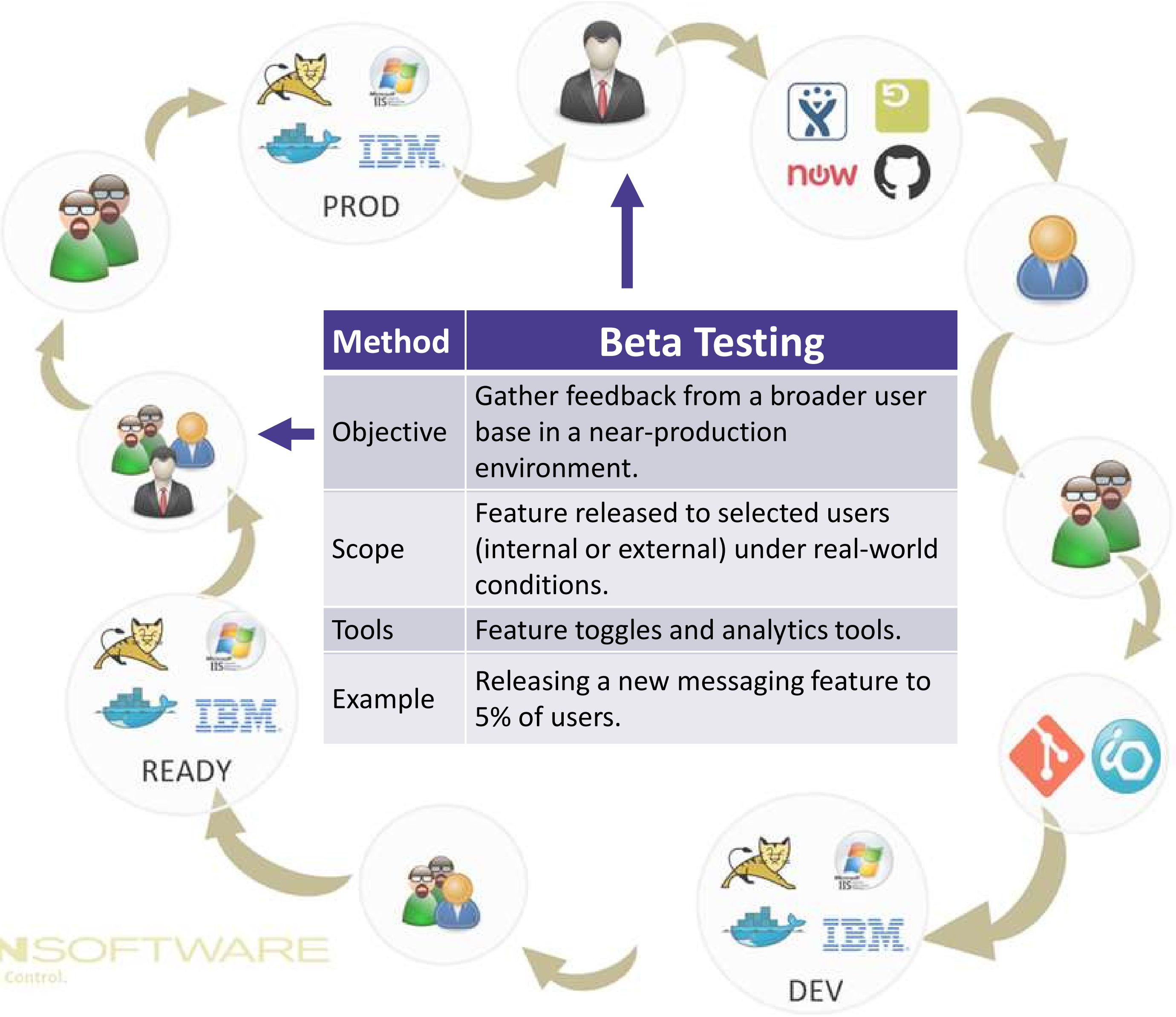


Method	Security Testing
Objective	Ensure the application is resistant to security threats.
Scope	Vulnerabilities, penetration testing, and compliance with security standards.
Tools	OWASP ZAP, Burp Suite, or manual ethical hacking.
Example	Testing for SQL injection vulnerabilities in input fields.



Method	Usability Testing
Objective	Assess the feature’s user experience and ease of use.
Scope	Real-world scenarios with users.
Tools	Observation and user testing tools.
Example	Observing users navigating a new dashboard for confusion or inefficiency.

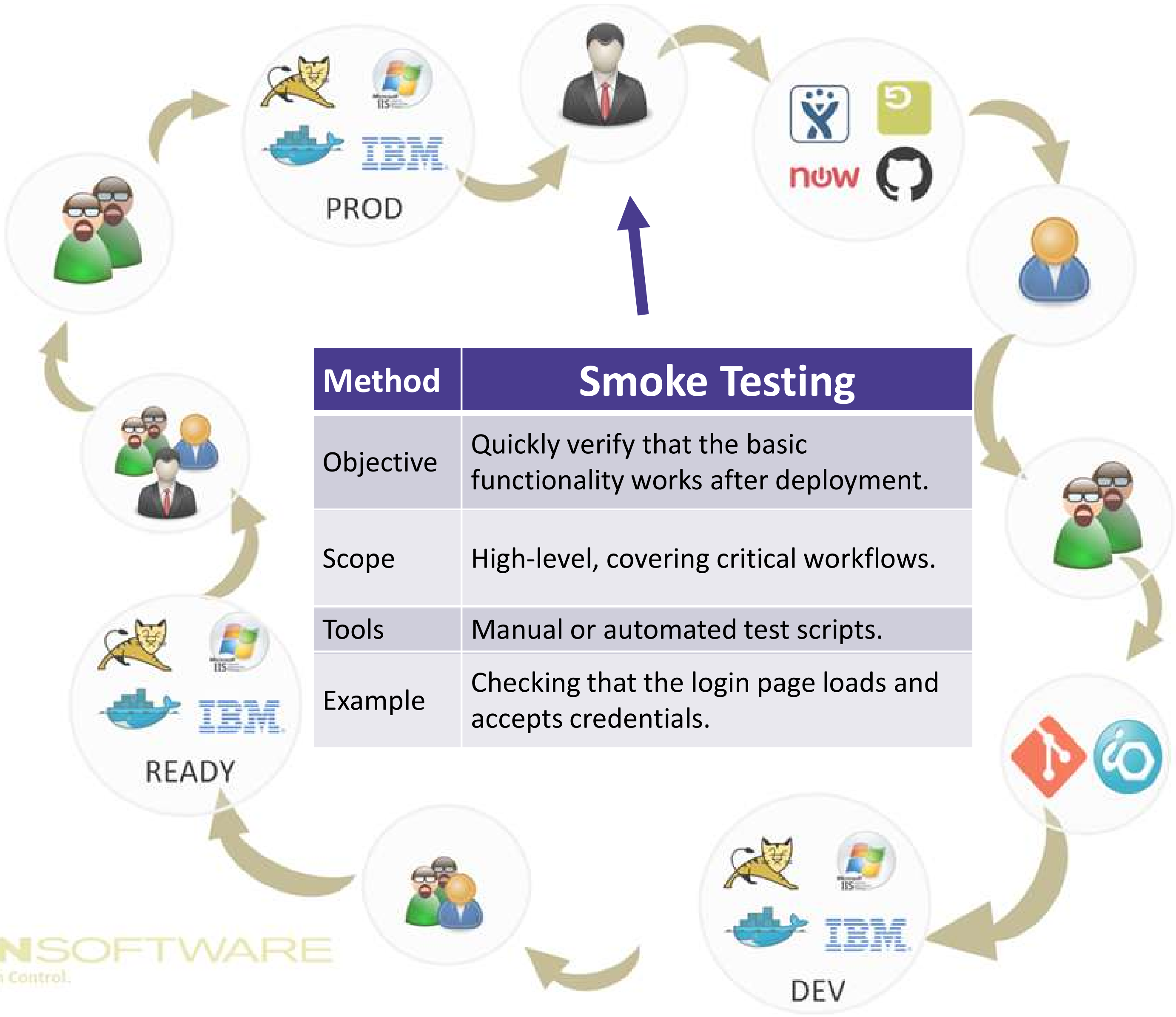
REMAINSOFTWARE
Embrace Change. Remain In Control.



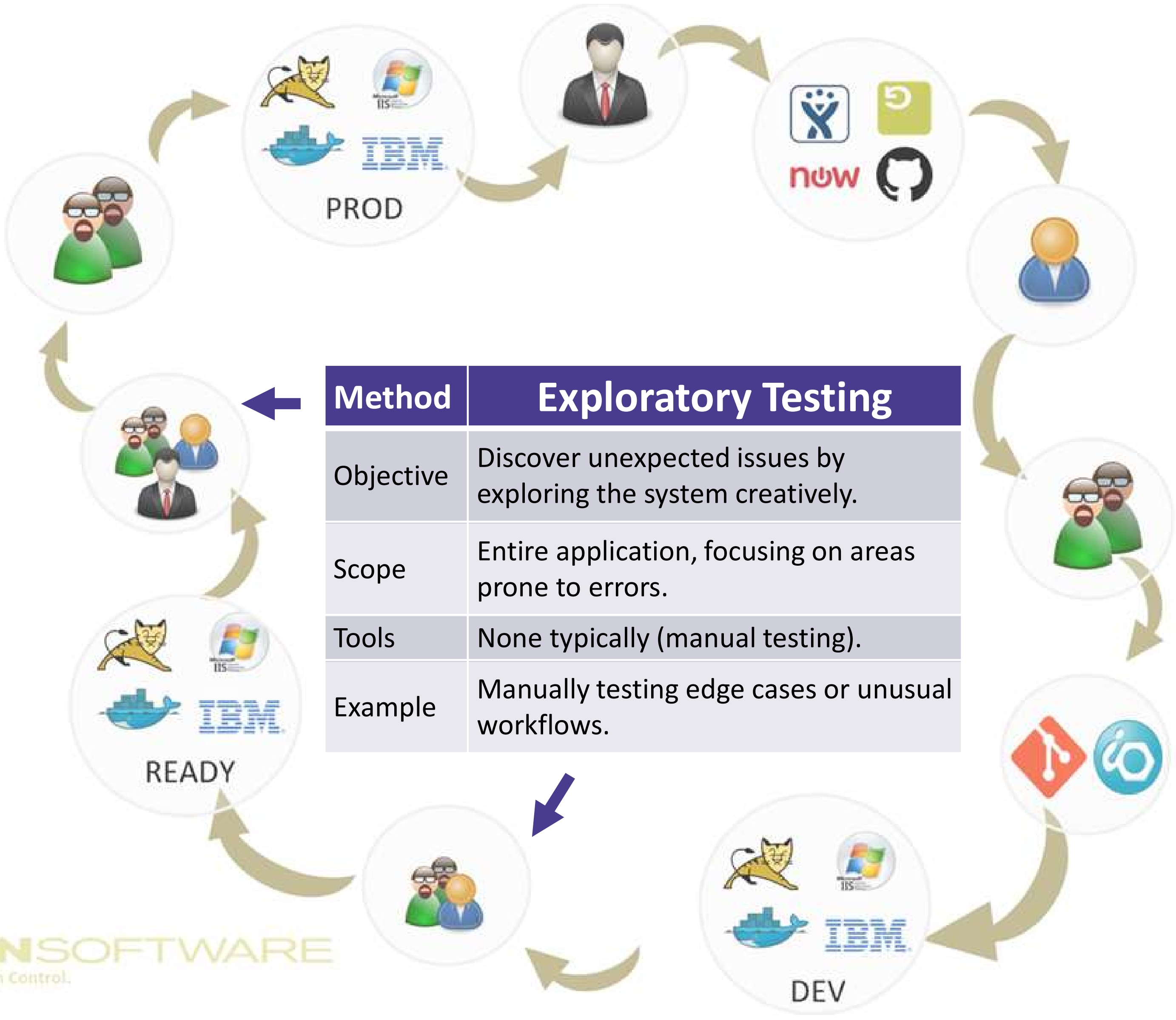
Method	Beta Testing
Objective	Gather feedback from a broader user base in a near-production environment.
Scope	Feature released to selected users (internal or external) under real-world conditions.
Tools	Feature toggles and analytics tools.
Example	Releasing a new messaging feature to 5% of users.

REMAINSOFTWARE

Embrace Change. Remain In Control.



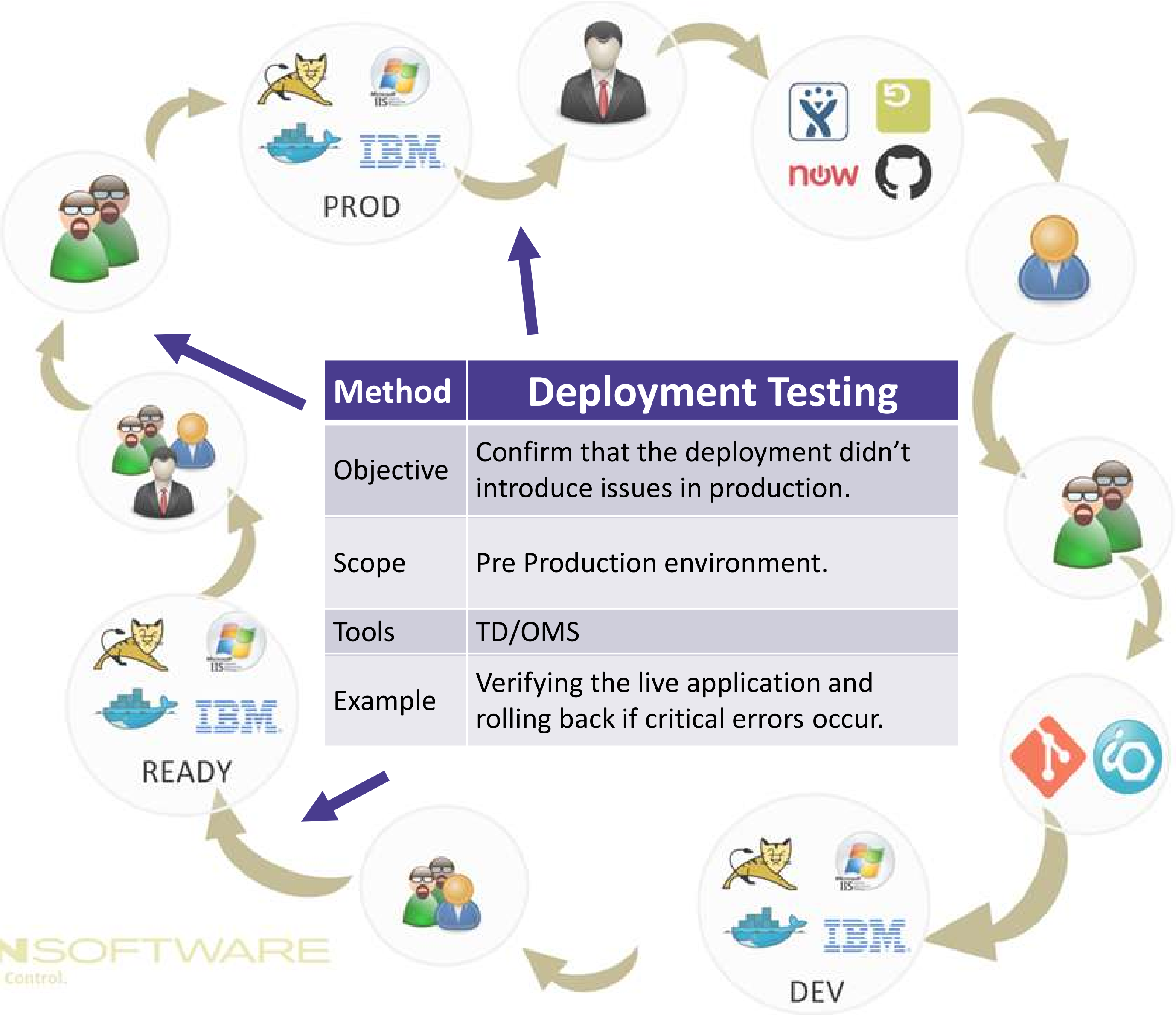
Method	Smoke Testing
Objective	Quickly verify that the basic functionality works after deployment.
Scope	High-level, covering critical workflows.
Tools	Manual or automated test scripts.
Example	Checking that the login page loads and accepts credentials.



Method	Exploratory Testing
Objective	Discover unexpected issues by exploring the system creatively.
Scope	Entire application, focusing on areas prone to errors.
Tools	None typically (manual testing).
Example	Manually testing edge cases or unusual workflows.

REMAINSOFTWARE

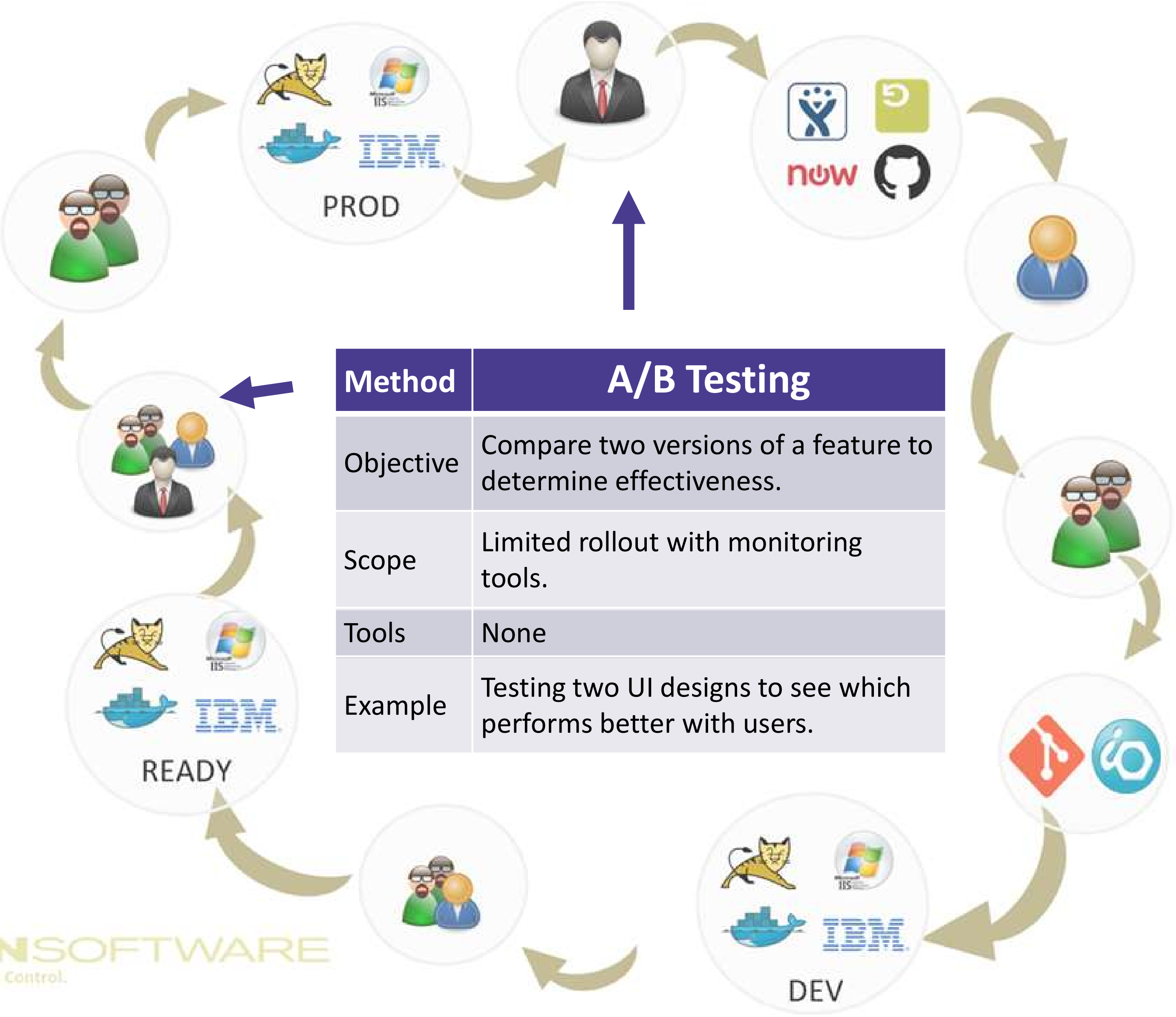
Embrace Change. Remain In Control.



Method	Deployment Testing
Objective	Confirm that the deployment didn't introduce issues in production.
Scope	Pre Production environment.
Tools	TD/OMS
Example	Verifying the live application and rolling back if critical errors occur.

REMAINSOFTWARE

Embrace Change. Remain In Control.



Method	A/B Testing
Objective	Compare two versions of a feature to determine effectiveness.
Scope	Limited rollout with monitoring tools.
Tools	None
Example	Testing two UI designs to see which performs better with users.

Method	Objective	Scope	Tools	Example
Peer Review	Ensure code quality, adherence to standards, and early detection of potential issues.	Code-level review by peers focusing on logic, readability, test coverage, and adherence to coding standards.	GitHub, GitLab, Bitbucket	Reviewing a pull request for adherence to coding guidelines and ensuring edge cases are addressed.
Unit Testing	Verify that individual functions or components behave as expected.	Smallest testable units, often automated.	Jest, JUnit, NUnit, etc.	Ensuring a function calculating discounts works for all edge cases.
Integration Testing	Ensure that different components/modules interact correctly.	Multiple units combined, focusing on data flow and API integration.	Postman, TestContainers, or custom integration tests.	Testing a frontend form with a backend API to validate end-to-end data processing.
System Testing	Validate the entire system's behavior against requirements.	The whole application or system in a controlled environment.	Selenium, Cypress, or manual testing.	Simulating user workflows like login, checkout, or user management.
Regression Testing	Confirm that new changes haven't broken existing functionality.	Previously developed and tested features.	Automated regression test suites.	Running tests for all major workflows after adding a new feature.
Performance Testing	Assess speed, responsiveness, and stability under expected and stress conditions.	Application as a whole or specific bottlenecks.	JMeter, Gatling, or LoadRunner.	Checking response times under 1,000 concurrent users.
User Acceptance Testing (UAT)	Validate that the feature meets business needs and is ready for end users.	Real-world scenarios with business stakeholders or selected users.	None typically (manual testing with stakeholders).	Allowing a sample group to use the new feature and provide feedback.
Security Testing	Ensure the application is resistant to security threats.	Vulnerabilities, penetration testing, and compliance with security standards.	OWASP ZAP, Burp Suite, or manual ethical hacking.	Testing for SQL injection vulnerabilities in input fields.
Usability Testing	Assess the feature's user experience and ease of use.	Real-world scenarios with user personas.	Observation and user testing tools.	Observing users navigating a new dashboard for confusion or inefficiency.
Beta Testing	Gather feedback from a broader user base in a near-production environment.	Feature released to selected users (internal or external) under real-world conditions.	Feature toggles and analytics tools.	Releasing a new messaging feature to 5% of users.
Smoke Testing	Quickly verify that the basic functionality works after deployment.	High-level, covering critical workflows.	Manual or automated test scripts.	Checking that the login page loads and accepts credentials.
Exploratory Testing	Discover unexpected issues by exploring the system creatively.	Entire application, focusing on areas prone to errors.	None typically (manual testing).	Manually testing edge cases or unusual workflows.

How many bugs reach production without testing?

Defect Density:

Industry averages suggest a defect density ranging from 1 to 5 bugs per KLOC.

KLOC = 1000 (1K) Lines Of Code.

Defect Detection Efficiency (DDE): Different testing methods have varying effectiveness in identifying defects. The DDE represents the percentage of total defects detected during a specific testing phase. While exact percentages can vary based on numerous factors, general industry observations provide the following approximate DDE values:

Wikipedia

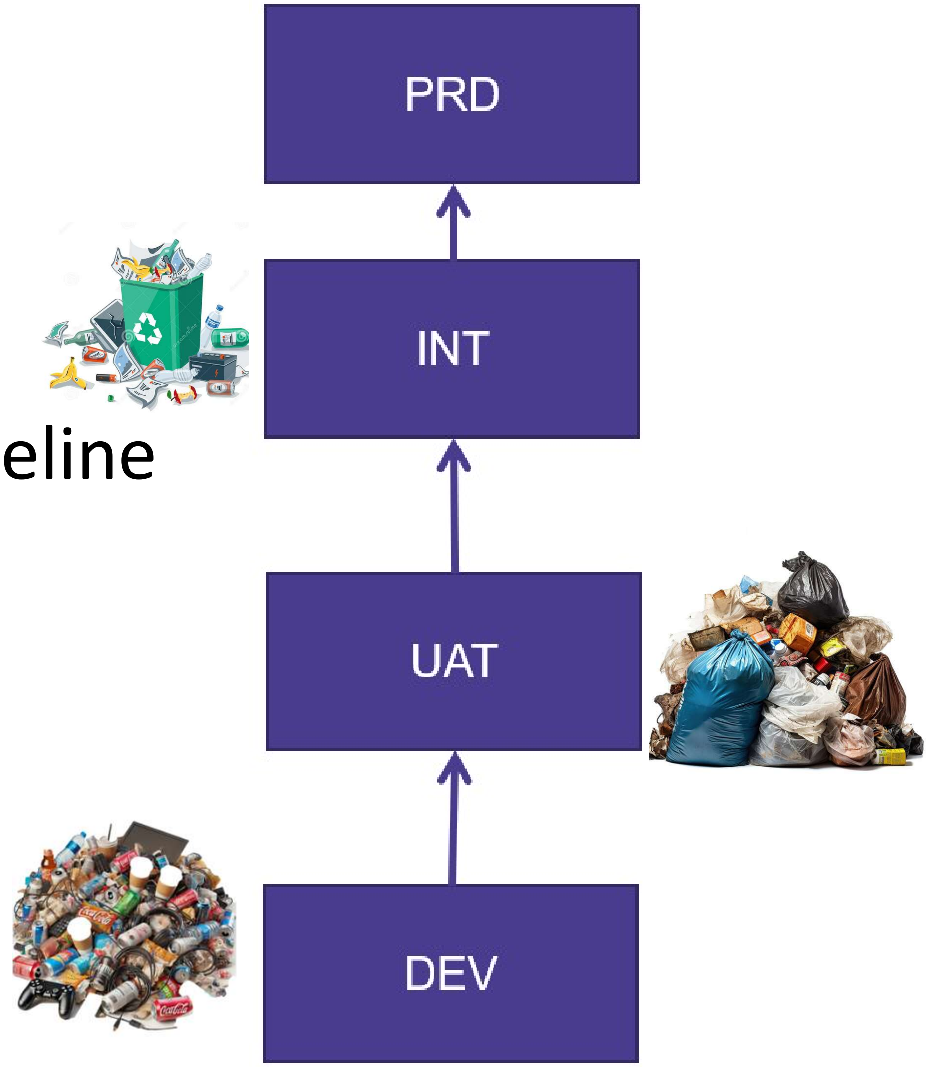
1. **Peer Review**: ~60%
2. **Unit Testing**: ~30%
3. **Integration Testing**: ~35%
4. **System Testing**: ~40%
5. **User Acceptance Testing (UAT)**: ~15%
6. **Beta Testing**: ~10%

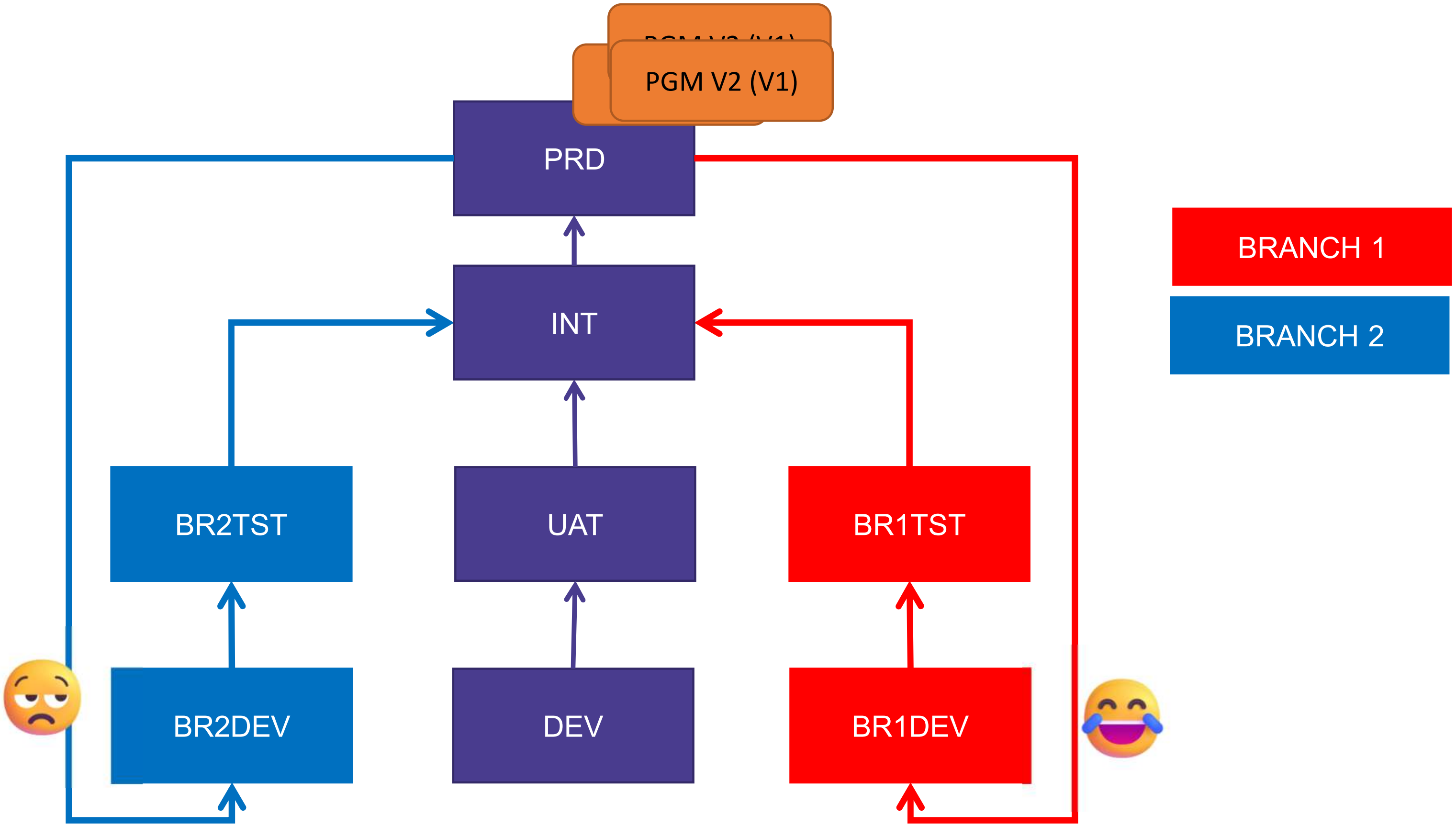
What are you testing?

The broken pipeline

The broken PROGRAMMER pipeline

The broken pipeline





PGM V2 (V1)

PGM V2 (V1)

PRD

INT

BR2TST

UAT

BR1TST

BR2DEV

DEV

BR1DEV

BRANCH 1

BRANCH 2

What **data** are you testing with?

$$T=f(P,C,D)$$

Where:

- T: Test Result
- P: Changed Program
- C: Test Case
- D: Input Data provided during the test

Failures can be attributed to:

1. **Change in the Program Logic (Regression):** $T = f(P_\Delta, C, D)$
   - P_Δ represents a change in the program logic or version.
   - A regression in the program may result in a failure.
2. **Change in the Data (New or Invalid Inputs):** $T = f(P, C, D_\Delta)$
   - D_Δ indicates that the input data has changed.
   - New or unexpected input data could lead to incorrect results.
3. **Change or Error in the Test Case Logic:** $T = f(P, C_\Delta, D)$
   - C_Δ reflects a change or error in the test case definition or expected outcome.
   - If the test case is incorrect, it might falsely indicate a failure.

- If your PROGRAM changes AND your DATA changes:
- Where is the issue?

Bottom line:

- Don’t change your test data

Unit Testing

```
pgm parm(&num1 &operator &num2 &result)

dcl &num1 *dec (15 5)
dcl &operator *char (1)
dcl &num2 *dec (15 5)
dcl &result *dec (15 5)

if (&operator *eq '+') then(do)
  chgvar &result (&num1 + &num2)
  return
enddo

if (&operator *eq 'x') then(do)
  chgvar &result (&num1 + &num2)
  return
enddo

endpgm
```

Unit Testing rules

```
pgm

dcl &num1 *dec (15 5)
dcl &operator *char (1)
dcl &num2 *dec (15 5)
dcl &result *dec (15 5)

chgvar &num1 (5)
chgvar &num2 (10)
call calculator parm(&num1 '+' &num2 &result)

if (&result *ne 15) then(do)
   SNDPGMMSG MSGID(CPF9898) +
           MSGF(QCPFMSG) +
           MSGTYPE(*ESCAPE) +
           MSGDTA('Calc addition: Expected 15
but got ' *cat %char(&result))
enddo

endpgm
```

```
pgm

dcl &num1 *dec (15 5)
dcl &operator *char (1)
dcl &num2 *dec (15 5)
dcl &result *dec (15 5)

chgvar &num1 (5)
chgvar &num2 (10)
call calculator parm(&num1 'x' &num2 &result)

if (&result *ne 50) then(do)
   SNDPGMMSG MSGID(CPF9898) +
           MSGF(QCPFMSG) +
           MSGTYPE(*ESCAPE) +
           MSGDTA('Calc mulitply: Expected 50
but got ' *cat %char(&result))
enddo

endpgm
```

Unit Testing

Where do your business rules live?

Unit Testing

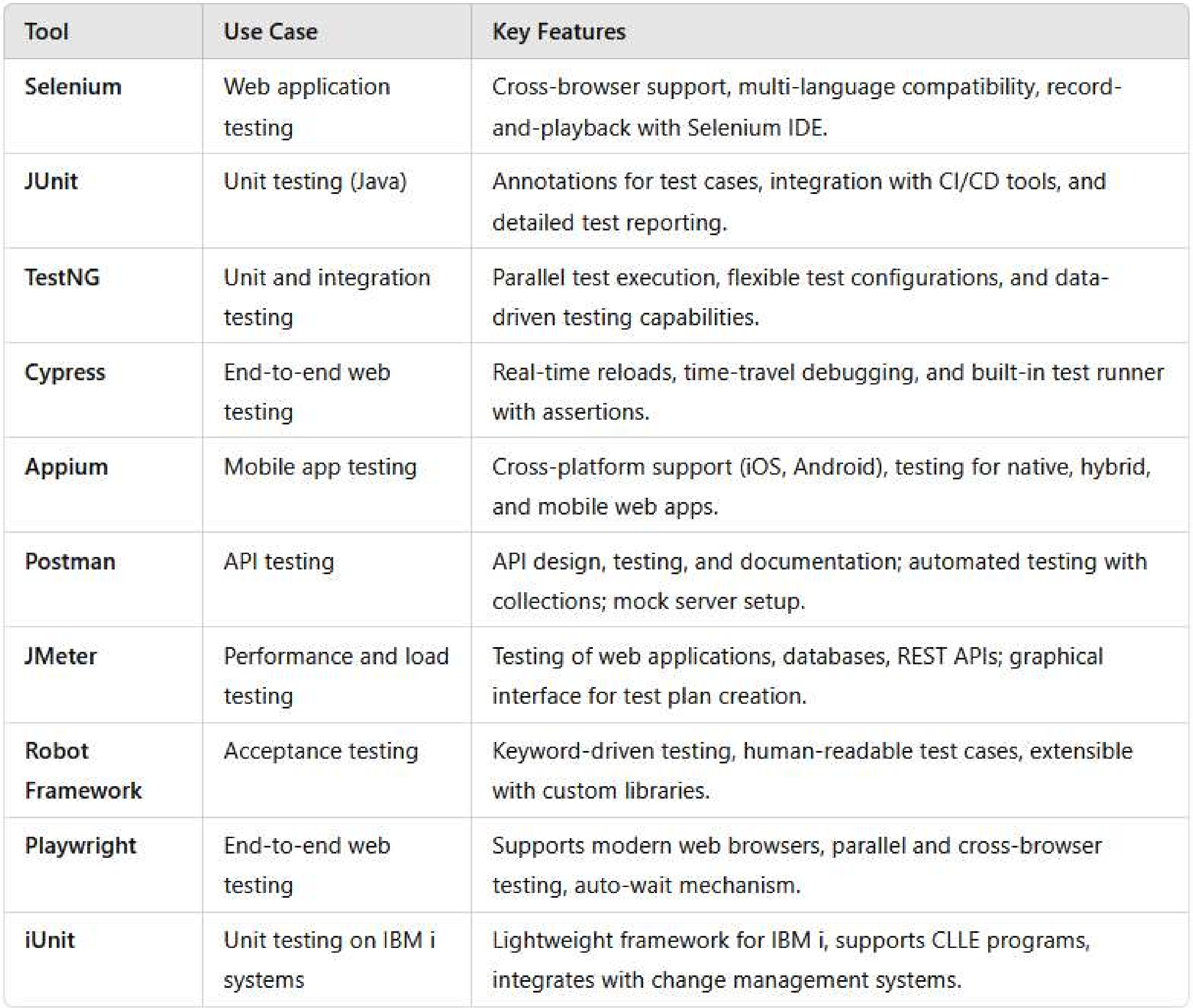
- Code according to MVC principles
  - Split Business Rules from UI
- Use Service Programs
- Code with modernization in mind

Peer Review

Integration Test

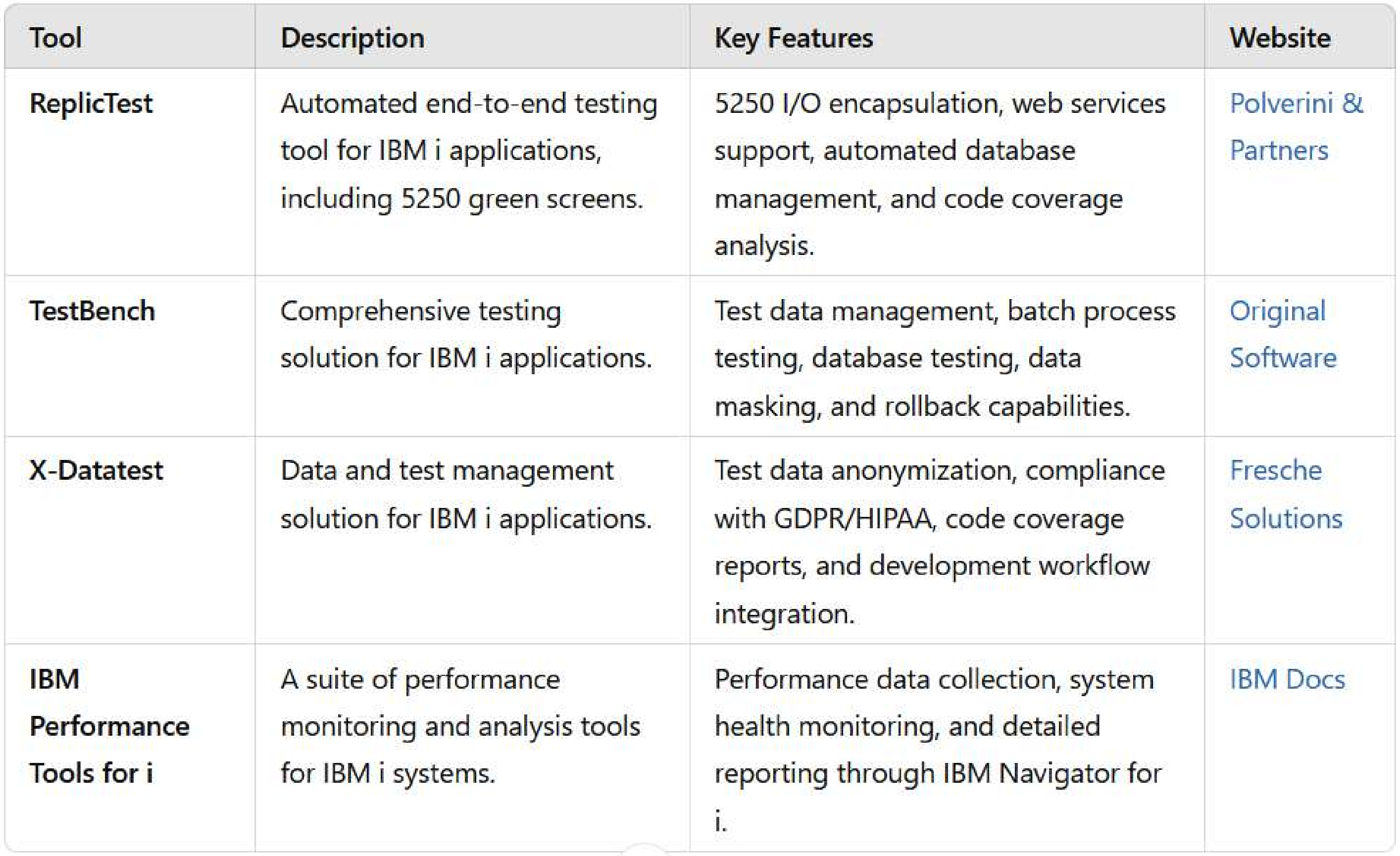
JEST, Junit, Etc..

Open Source Tools



Tool	Use Case	Key Features
Selenium	Web application testing	Cross-browser support, multi-language compatibility, record-and-playback with Selenium IDE.
JUnit	Unit testing (Java)	Annotations for test cases, integration with CI/CD tools, and detailed test reporting.
TestNG	Unit and integration testing	Parallel test execution, flexible test configurations, and data-driven testing capabilities.
Cypress	End-to-end web testing	Real-time reloads, time-travel debugging, and built-in test runner with assertions.
Appium	Mobile app testing	Cross-platform support (iOS, Android), testing for native, hybrid, and mobile web apps.
Postman	API testing	API design, testing, and documentation; automated testing with collections; mock server setup.
JMeter	Performance and load testing	Testing of web applications, databases, REST APIs; graphical interface for test plan creation.
Robot Framework	Acceptance testing	Keyword-driven testing, human-readable test cases, extensible with custom libraries.
Playwright	End-to-end web testing	Supports modern web browsers, parallel and cross-browser testing, auto-wait mechanism.
iUnit	Unit testing on IBM i systems	Lightweight framework for IBM i, supports CLLE programs, integrates with change management systems.

Commercial Tools



Tool	Description	Key Features	Website
ReplicTest	Automated end-to-end testing tool for IBM i applications, including 5250 green screens.	5250 I/O encapsulation, web services support, automated database management, and code coverage analysis.	Polverini & Partners
TestBench	Comprehensive testing solution for IBM i applications.	Test data management, batch process testing, database testing, data masking, and rollback capabilities.	Original Software
X-Datatest	Data and test management solution for IBM i applications.	Test data anonymization, compliance with GDPR/HIPAA, code coverage reports, and development workflow integration.	Fresche Solutions
IBM Performance Tools for i	A suite of performance monitoring and analysis tools for IBM i systems.	Performance data collection, system health monitoring, and detailed reporting through IBM Navigator for i.	IBM Docs