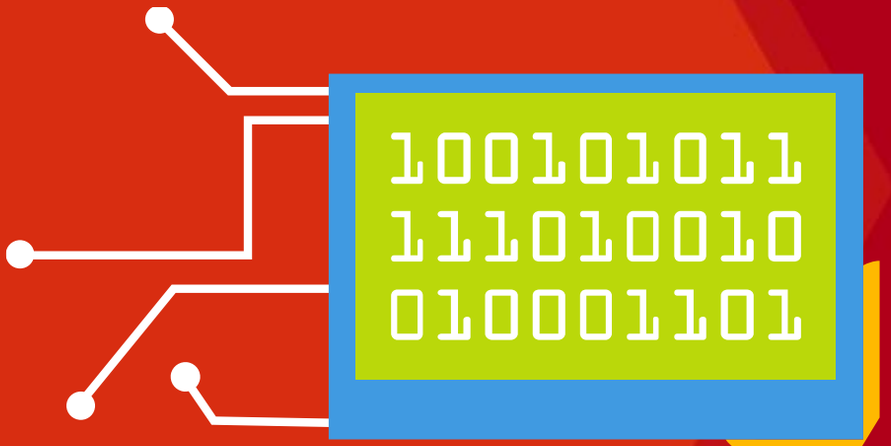
KONFERENCA

Microsoft

Agile Testing v2.0
Testing in the DevOps World

Ana Roje Ivančić, VS ALM MVP
Ognjen Bajić, VS ALM MVP

Ekobit



TEHNOLOGIJA

Speakers

Working with VS ALM tools since 2004.

WinDays 2005 preconf day on VSTS

Worked as Dev, PM, Test, RM, SM, PO…

VS ALM MVPs

Microsoft

Agenda

DevOps World and Quality

- Transform Engineering for Quality, Speed and Throughput
- Continuous Acceptance Testing

Continuous Delivery and Quality

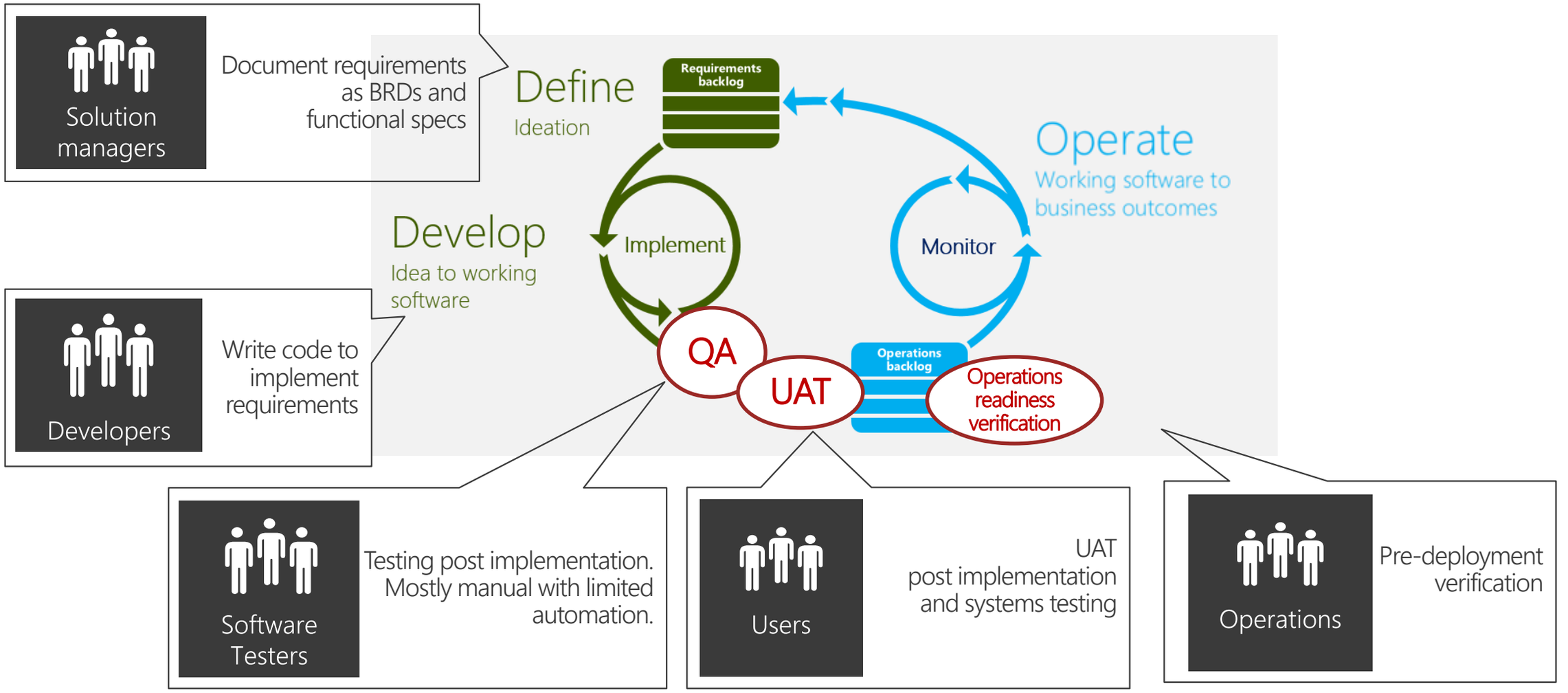
- Shift Left – Bring Quality related info in the Developer inner loop
- Shift Right – Testing in production

DevOps World and Quality

Microsoft

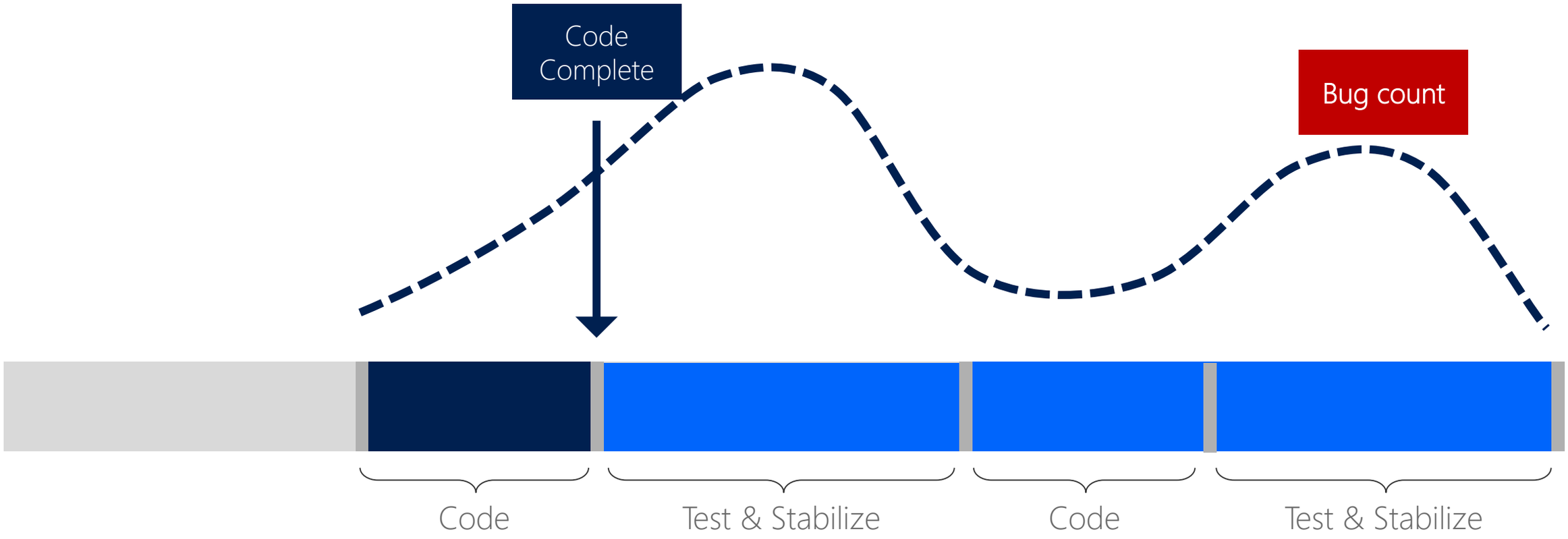
KONFERENCA

Conventional QA in the Modern App Lifecycle



Late engagement of stakeholder functions | **Late detection of unmet requirements** | **Increased cycle times** | **Increased costs**

Conventional QA and Quality



Transform Engineering for Quality, Speed and Throughput
Speed and Throughput

Multi-year cycles → Cloud cadence

- Deliver continuously or after every 2-3 week sprint

Automated Delivery Pipeline

- Ability to deliver on a push of a button - easily, quickly and frequently

Execute automated tests in all phases of the pipeline

- Unit tests during the Build
- Functional (acceptance) tests as part of the delivery in different environments
- Performance, load, usability and other tests

Transform Engineering for Quality, Speed and Throughput

Quality

Insist on Automated Tests

- Manual testing doesn't scale
- Can't have long-term control of quality with manual testing only

Acceptance tests

- Definition of Ready: Included in the requirements definition
- Definition of Done: Feature is Done when its automated acceptance tests are Done

Complement automated tests with manual testing

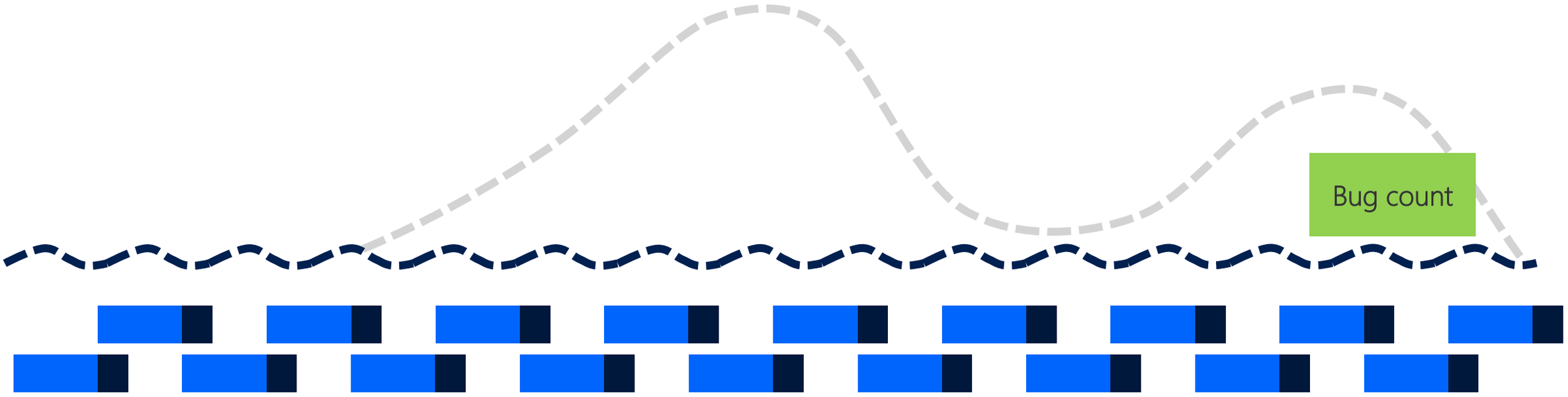
No more Dev and QA → Everyone is an Engineer

- Everybody writes production code and code for tests

Don't accept test failures → Insist on reliability and speed

- Treat test code and test infrastructure as production
- Only reliable tests and test infrastructure should survive

Quality After – With Transformed Engineering



Appropriate For Any Type Of Project/Product

Optimized for Web - Live site

Works perfectly for classic boxed products as well

- Automated „Check for update"

Necessary for sustainable support and future development

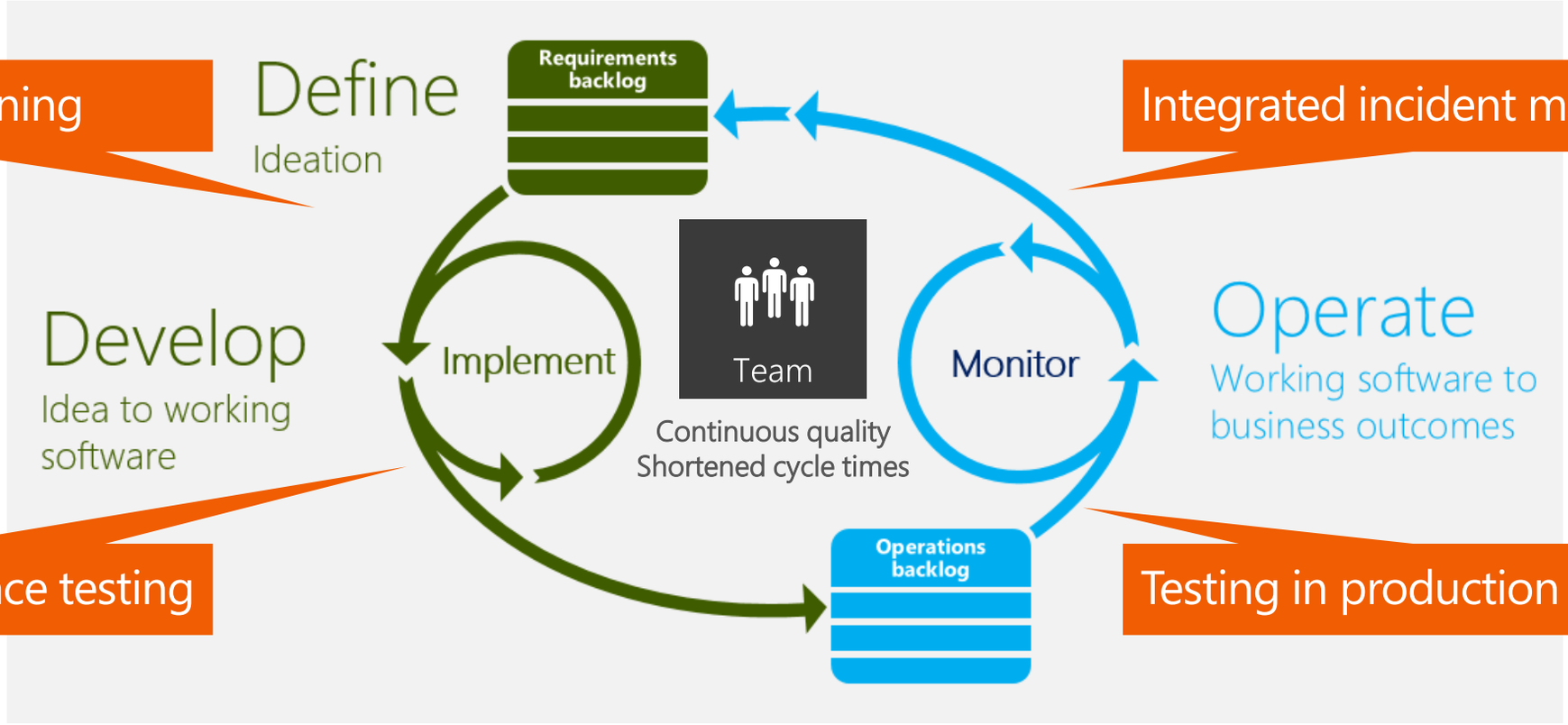
But we don't have any tests! Initial effort is too big!

- Start small
- Cover key scenarios with automated functional tests and use them as smoke tests
- Integrate smoke tests in the release pipeline
- Automate acceptance tests for newly developed features
  - At least for the key scenarios

Embrace Acceptance Test Driven Development as long term strategy

Acceptance Test Driven Development

Quality enablement practices for continuous value delivery



Team integration | Early detection of unmet requirements | Shortened cycle times | Reduced costs

Delivery Pipeline And Testing Demo

Microsoft

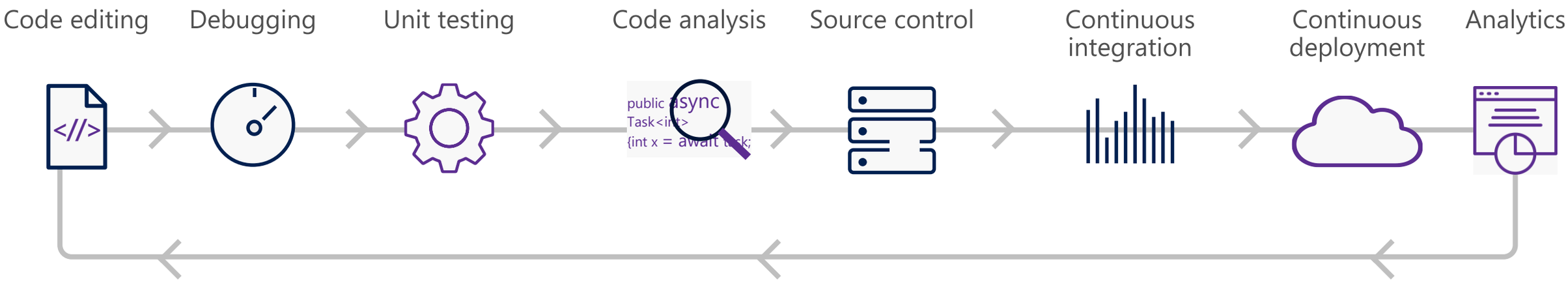
KONFERENCA

Continuous Delivery and Quality

Microsoft

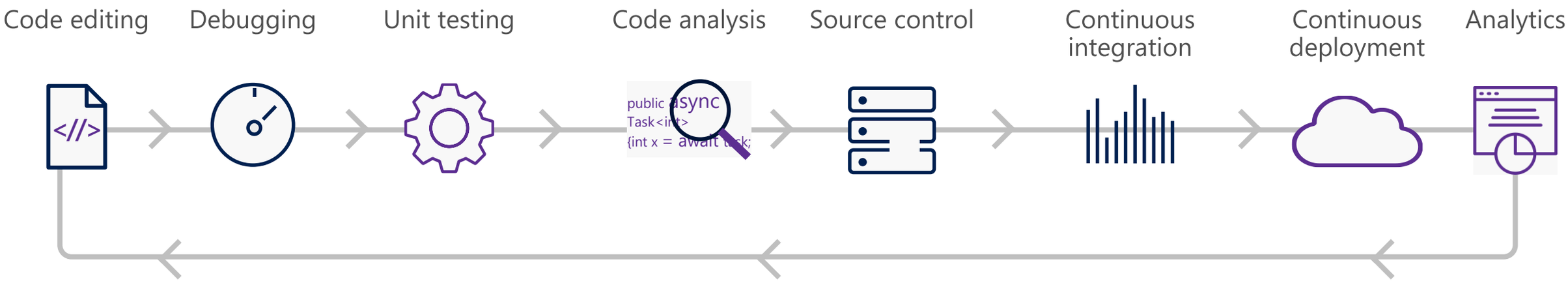
KONFERENCA

A Traditional Look At DevOps



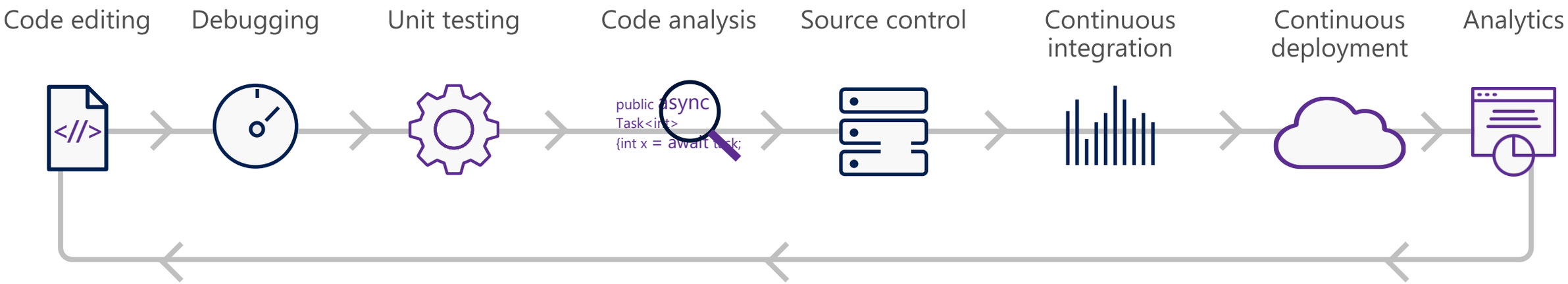
Development flows left to right and repeats

We Can Do **Better** Than Traditional DevOps!



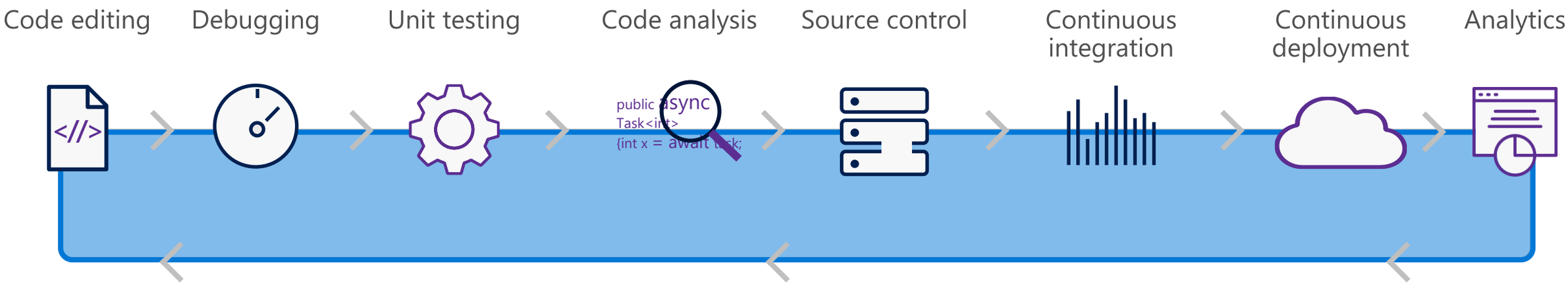
Tighten the cycle
Catch issues before check-in
Identify issues in production

“Shift Left”

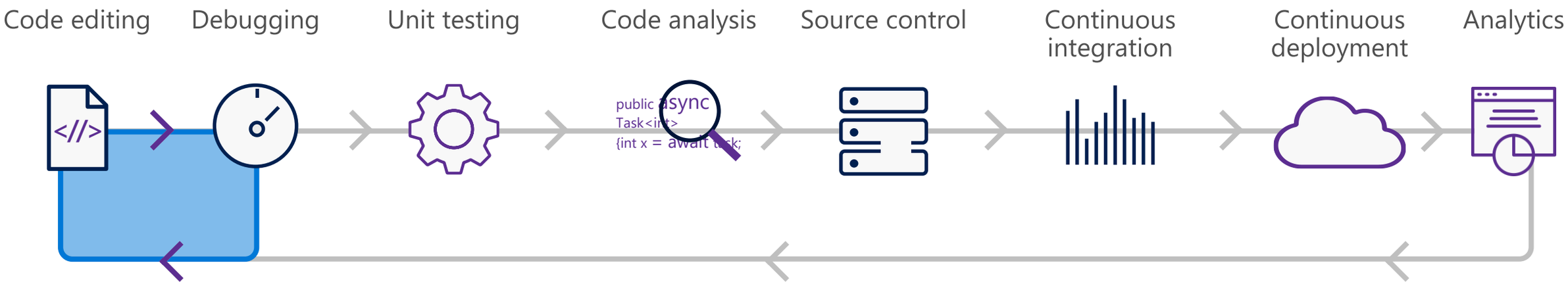


Continuous integration + real-time experiences = **shift left**

“Shift Left”

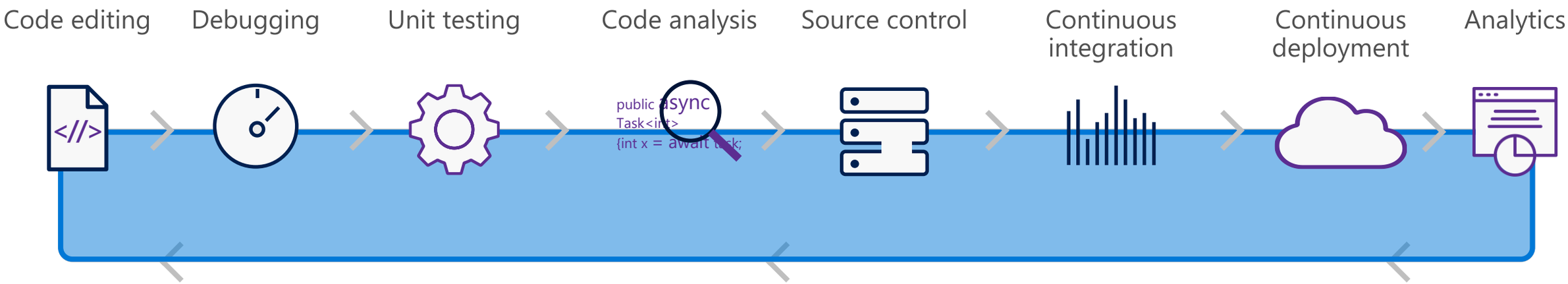

Code editing
Debugging
Unit testing
Code analysis
Source control
Continuous integration
Continuous deployment
Analytics
</>
public async
Task<int>
{int x = await task;

“Shift Left”

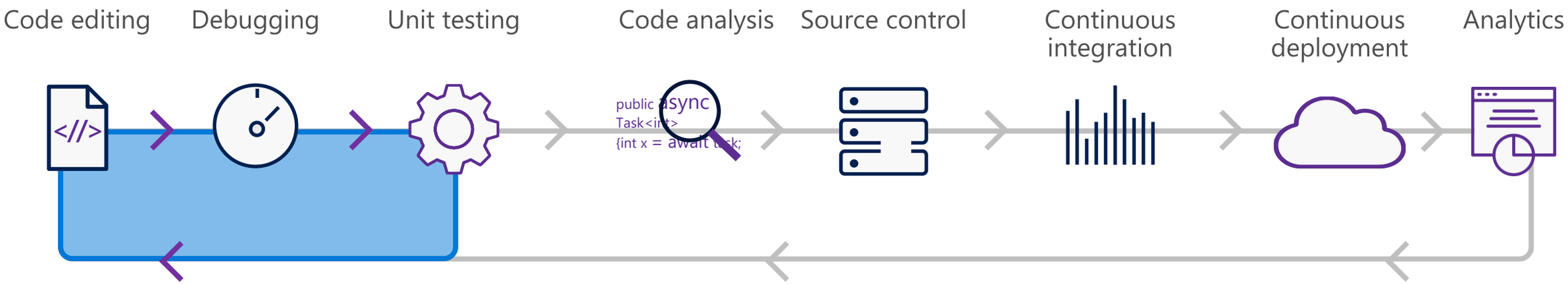


Edit and Continue – Pulled debugging into the edit/build cycle

“Shift Left”

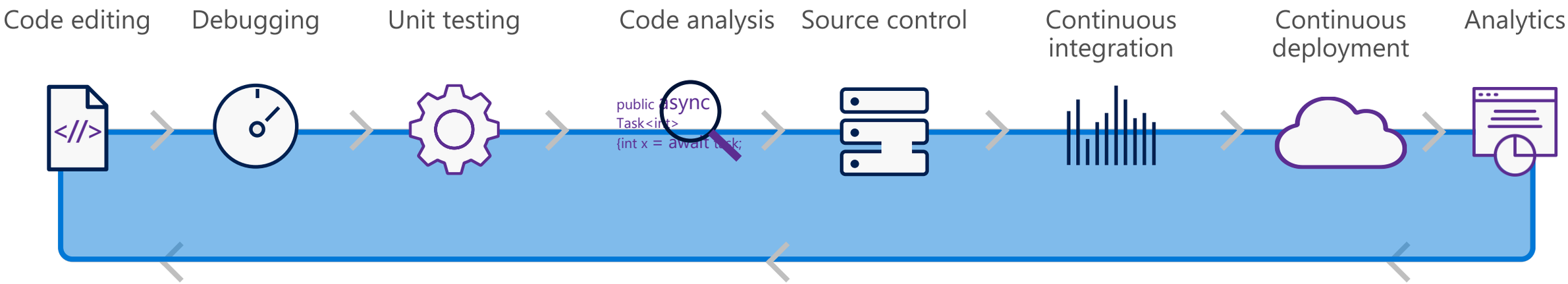

Code editing
Debugging
Unit testing
Code analysis
Source control
Continuous integration
Continuous deployment
Analytics
</>
public async
Task<int>
{int x = await task;

“Shift Left”

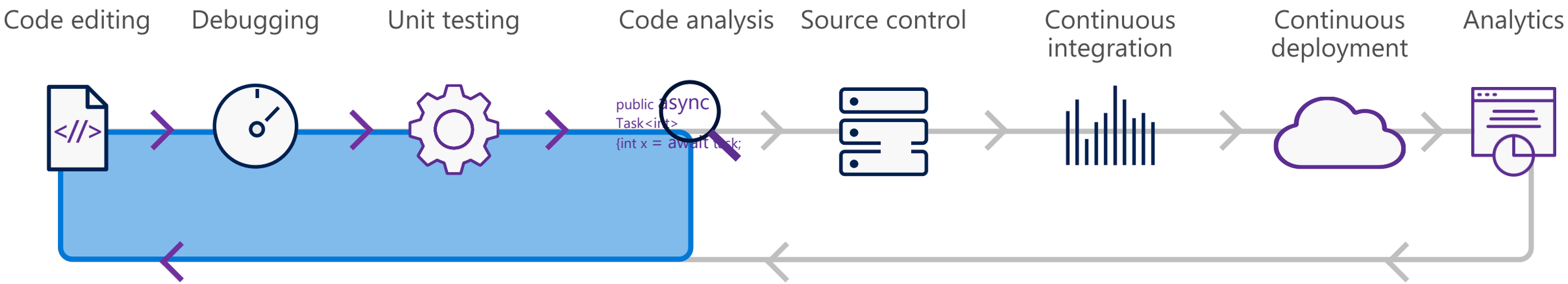


Live Unit Testing – Pulls quality further into the inner loop

“Shift Left”

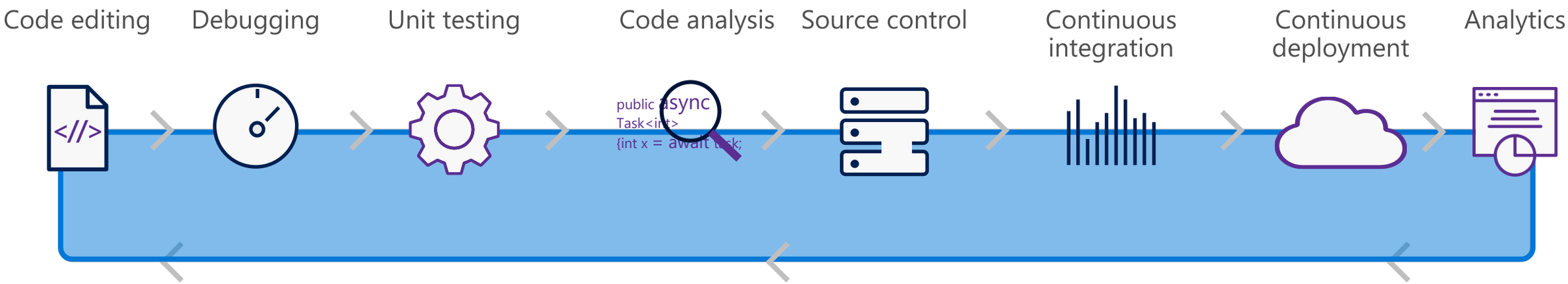

Code editing
Debugging
Unit testing
Code analysis
Source control
Continuous integration
Continuous deployment
Analytics
</>
public async
Task<int>
{int x = await task;

“Shift Left”

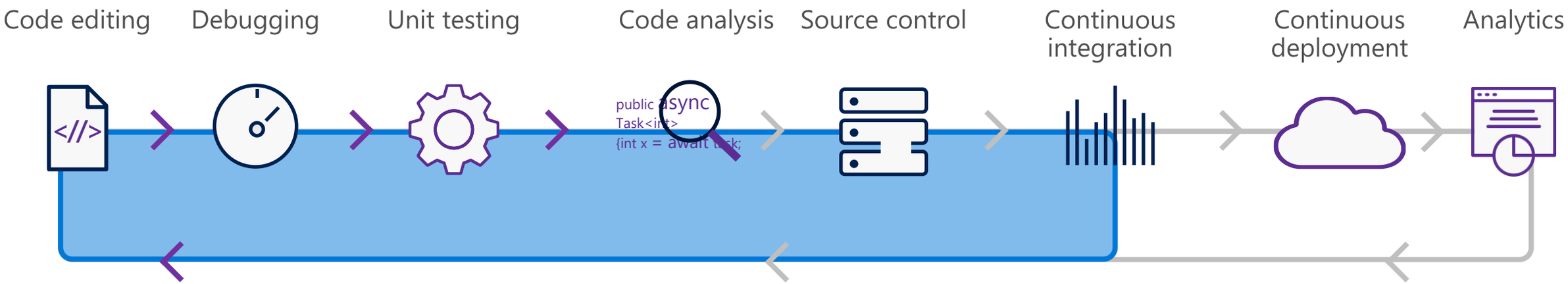


Live Code Analysis – Immediate feedback loop

“Shift Left”

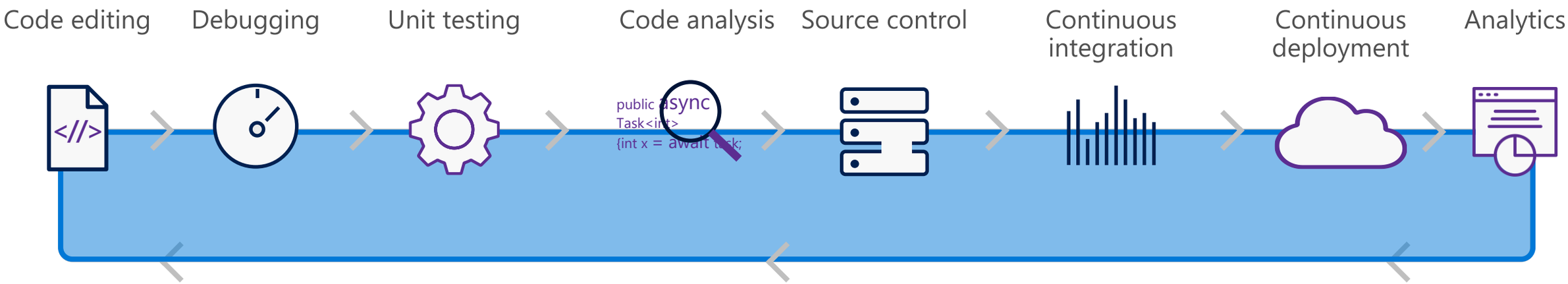

Code editing
Debugging
Unit testing
Code analysis
Source control
Continuous integration
Continuous deployment
Analytics
</>
public async
Task<int>
{int x = await task;

“Shift Left”

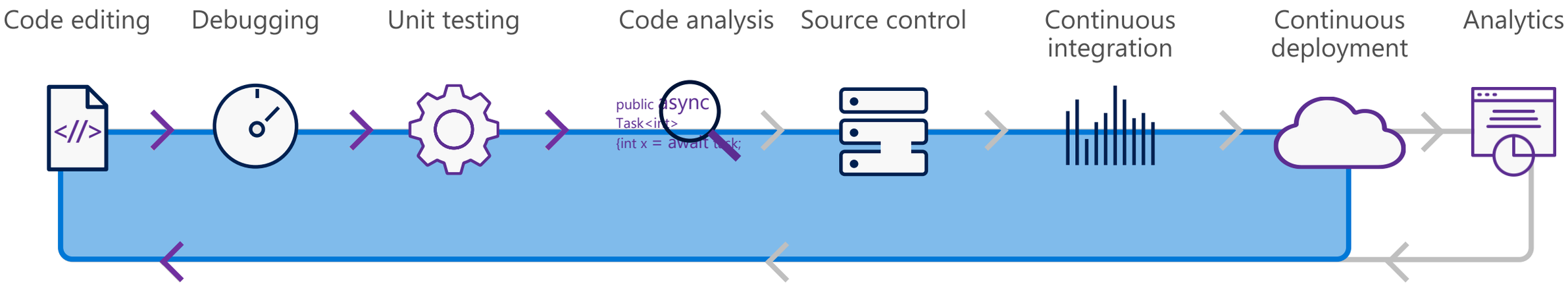


Unit Tests in automated Build – Immediate feedback loop

“Shift Left”


Code editing
Debugging
Unit testing
Code analysis
Source control
Continuous integration
Continuous deployment
Analytics
</>
public async
Task<int>
{int x = await task;

“Shift Left”



Acceptance Tests in delivery pipeline – Immediate feedback loop

Best Practice – Test Shift Left

Test early – more unit test like, less UI based tests

- Replace UI based functional tests with unit test based (functional and integration) tests
- Replace fragile UI tests with fast and robust (unit) tests
- Usually requires refactoring of the architecture

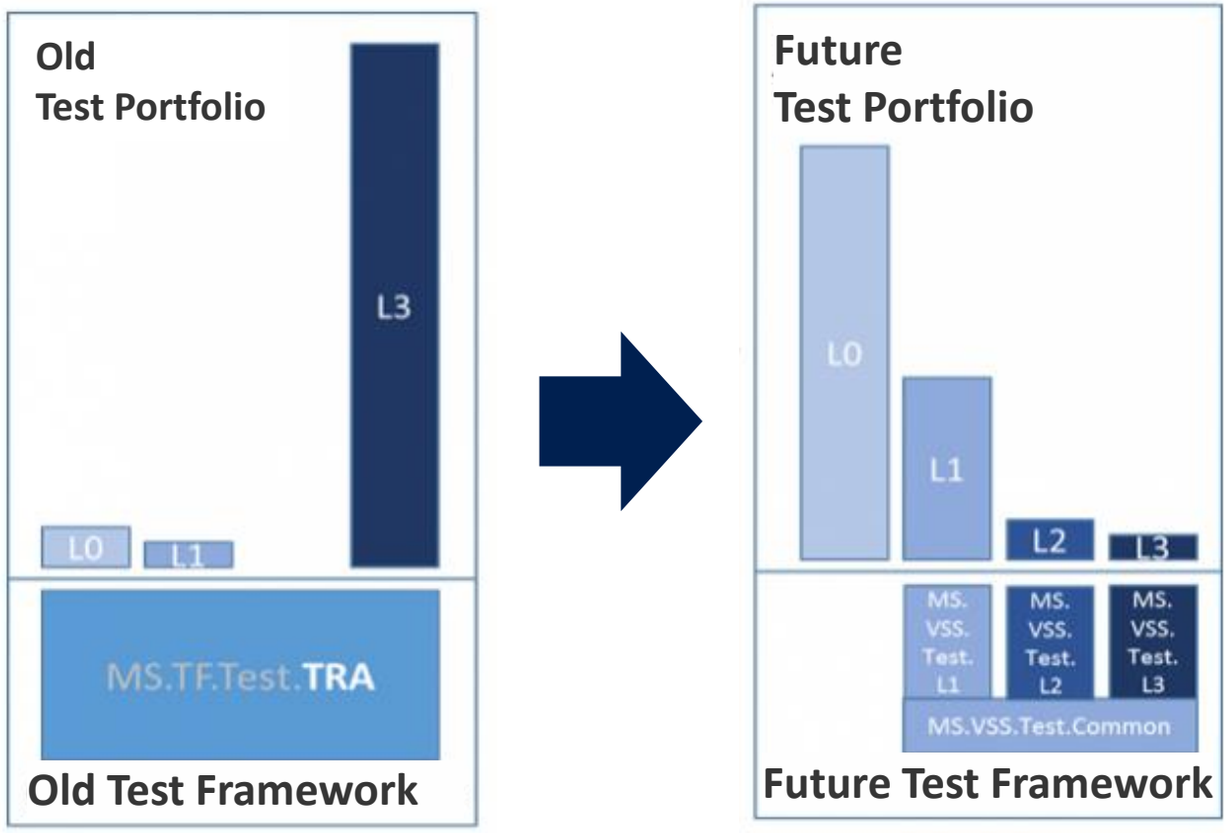
Tests become significantly faster and more reliable

Bugs are found earlier in the cycle

- Developers get quick feedback on their commit, which further reduces context switching

Case Study - Changing the test portfolio balance

Shift Left in Microsoft's VSTS Team



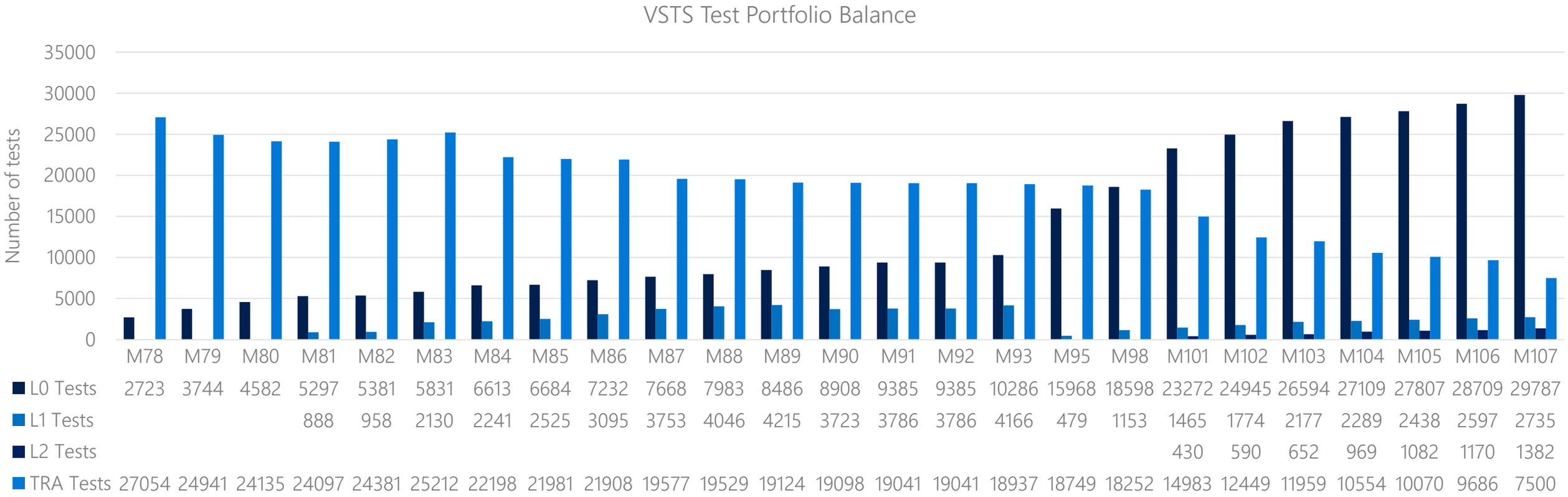
Principles

- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive

LO: Run against raw drop. Only access binaries. Run in the build. Must be fast and reliable.
L1: Run against raw drop. Can hit resources on a machine (eg. SQL test)
L2: Run against „special" deployed product.
L3: Run against production

Case Study - Changing the test portfolio balance

Shift Left in Microsoft's VSTS Team



	M78	M79	M80	M81	M82	M83	M84	M85	M86	M87	M88	M89	M90	M91	M92	M93	M95	M98	M101	M102	M103	M104	M105	M106	M107
L0 Tests	2723	3744	4582	5297	5381	5831	6613	6684	7232	7668	7983	8486	8908	9385	9385	10286	15968	18598	23272	24945	26594	27109	27807	28709	29787
L1 Tests				888	958	2130	2241	2525	3095	3753	4046	4215	3723	3786	3786	4166	479	1153	1465	1774	2177	2289	2438	2597	2735
L2 Tests																			430	590	652	969	1082	1170	1382
TRA Tests	27054	24941	24135	24097	24381	25212	22198	21981	21908	19577	19529	19124	19098	19041	19041	18937	18749	18252	14983	12449	11959	10554	10070	9686	7500

LO: Run against raw drop. Only access binaries. Run in the build. Must be fast and reliable.
L1: Run against raw drop. Can hit resources on a machine (e.g. SQL test)
L2: Run against „special" deployed product.
L3: Run against production

Microsoft

Demo
Live Unit Testing

Microsoft

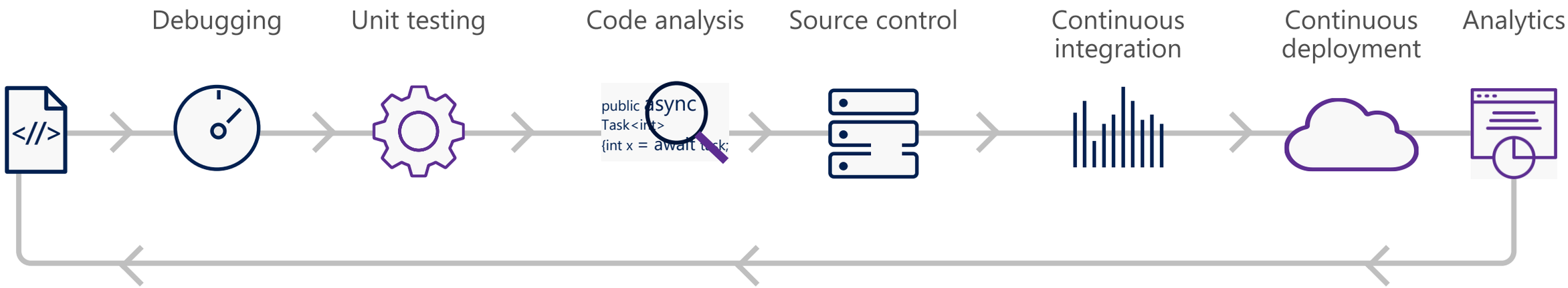
KONFERENCA

Shift Right –
Testing In Production

Microsoft

KONFERENCA

"Shift Left" vs. "Shift Right"



Continuous Integration + Real-time experiences = **Shift Left**

Continuous data mining + Near real-time data = **Shift Right**

Shift Right – Test in Production

Leverage end users as scalable manual testers

- Production is the best place to look for certain types of issues

Control exposure of features to users

- For web/cloud apps use multiple rings of production environments
- Use Feature flags

Monitor metrics and act on irregularities

- Proactively react, fix problems, redeploy

Rings of Production Environments

Attach different sets of users to different environments

Wait certain "*bake time*" in each environment before deploying into the next

A failure in an environment stops the deployment

Provides a way to contain regressions in new bits

Day/ Ring	1	2	3	4	5	6	7	8	9
0	Bin	Full	Full	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix
1	Hot Fix	Hot Fix	Hot Fix	Full	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix
2	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Full	Hot Fix	Full	Hot Fix
3	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Hot Fix	Full

Feature Flags

- Decouple deployment of features from their exposure to end-users
  - All done code is deployed, but feature flags control exposure
  - Granular control of user exposure to features down to an individual user
  - Combined best with multiple rings of production environments
  - Users can be added or removed with no redeployment
  - Turned off quickly
- Enable progressive experimentation & refinement
- Support early feedback
- Decouple engineering and marketing
  - Enables „dark launch" of features
- *Note! Avoid using feature flags for dark delivery of unfinished code*
  - *Prefer delivering more granular features in done state*

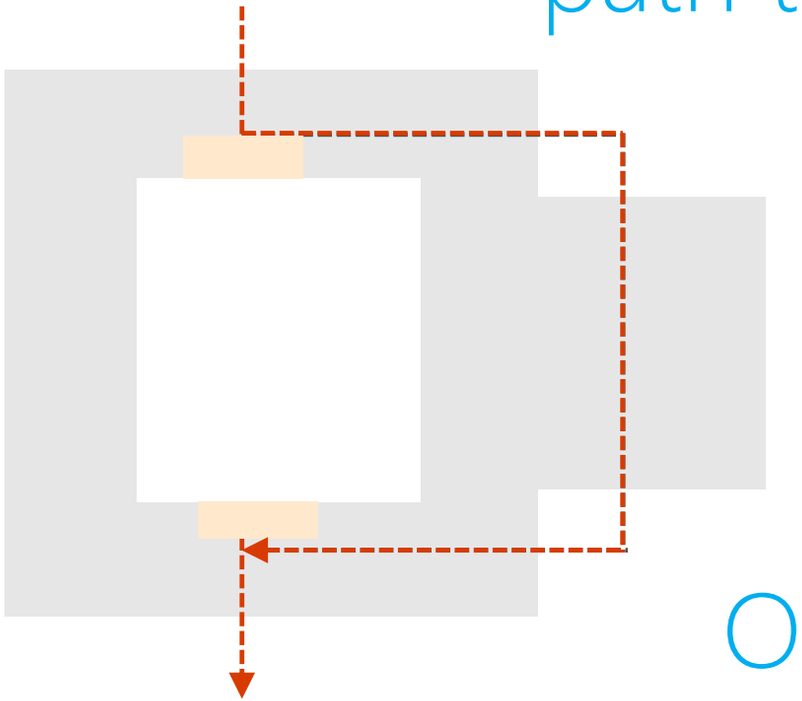
Feature Flags

Given some existing code path

ON

OFF

Introduce a new code path that is disabled…



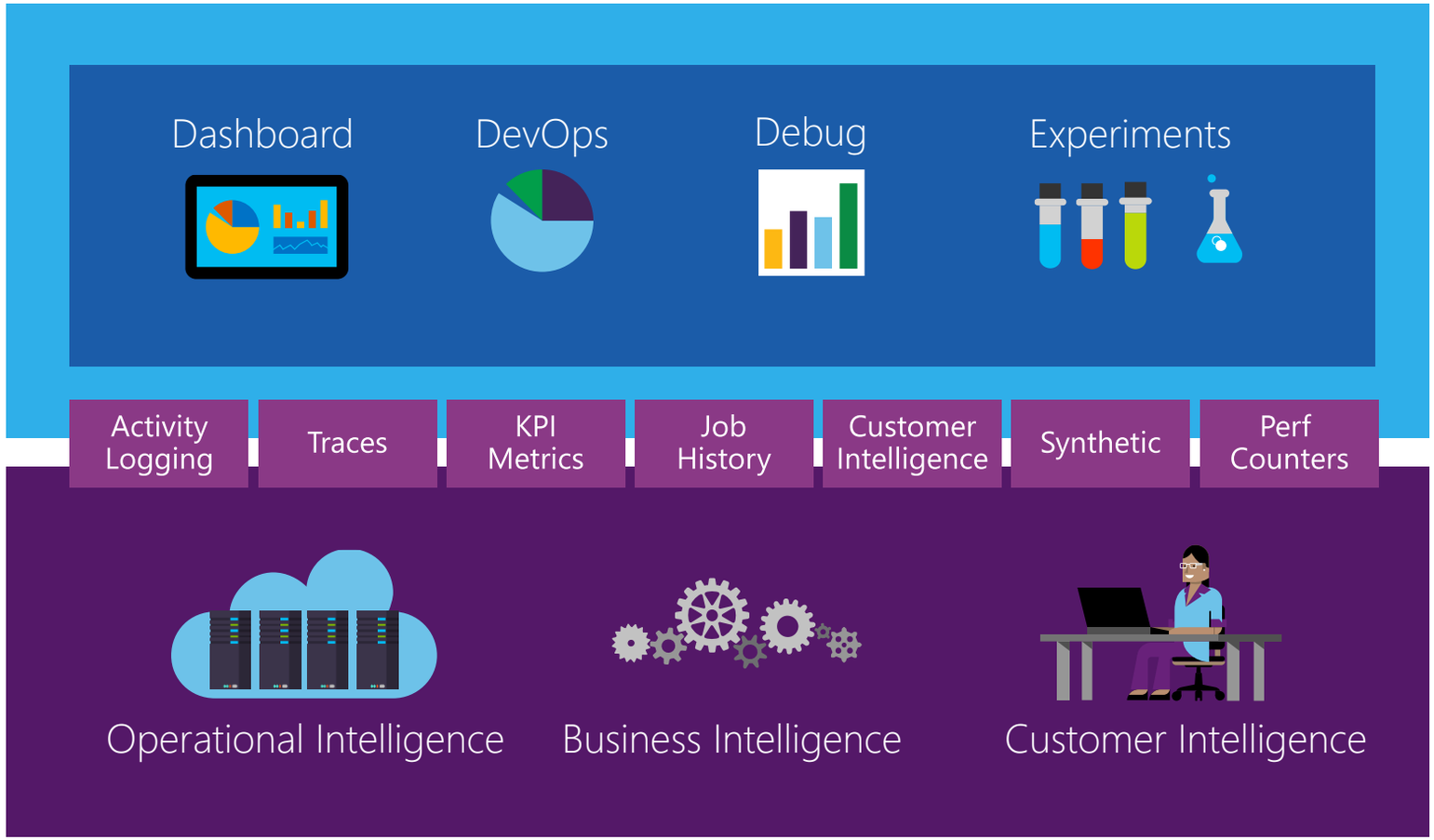
More code gets written…

When it's ready, the new code path is enabled

Once we're done, the flag is deleted

Measure everything – gather every possible metrics

Monitor for irregularities and react proactively



Feature Flags Demo

Microsoft

KONFERENCA

Agile Testing v2.0 – Testing in the DevOps World

Transform Engineering-Quality, Speed and Throughput

- Fast release cadence
- Automated Delivery Pipeline
- Automated Testing complemented with Manual testing
- Everybody writes production and test code
- Continuous Acceptance testing in every part of the delivery pipeline
- Replace UI functional tests with unit test based (functional and integration) tests

Shift Left – Test results available in dev tools in real time

- Continuous integration + real-time experiences

Shift Right – Testing in production

- Continuous data mining + near real-time data testing in production

Thank you!

Questions?

obajic @ ekobit.hr

aroje @ ekobit.hr

Microsoft

KONFERENCA