Bing Agility

MODERN ENGINEERING PRINCIPLES FOR LARGE SCALE TEAMS AND SERVICES
Outline

1. A bit about Bing
2. Velocity... What does it mean?
3. What is tested?
4. Modern Engineering Principles
5. The inner and outer loop
6. Performance gating
A bit about Bing

WW > 300M users, 9B searches/month
US >100M users, 4B searches/month

1. BING IS GROWING

2. MORE WORK TO DO

Queries/UU (Dec 2014)

<table>
<thead>
<tr>
<th></th>
<th>Bing</th>
<th>Google</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>38.3</td>
<td>66.9</td>
</tr>
</tbody>
</table>

1. Bing on the road
Bing is designed to work beautifully across all your devices.

2. Bing Desktop
Get the beauty of the Bing homepage on your PC desktop every day.

3. DIFFERENTIATE

Bing makes predictions
Bing uses search, social, and other relevant data to make intelligent predictions about upcoming events, like sports games, reality TV shows, and more.
Velocity

Does not mean...

- Shipping untested code... (any bozo can do that)

Does mean...

- Shipping thoroughly tested code...
- Shipping with high quality
- Shipping fast!
What is tested?

- Browser
- Device
- Security
- Performance
- Localization
- Globalization
- Scenario Coverage
- Privacy
- Instrumentation
- Composition
Modern Engineering Principles

Current engineering landscape

Hundreds of engineers
- 2000 engineers, across all continents

Ship 4x/day
- Full build shipped to production, no live site issues!

Agile
- {design, dev, test} → ship (no P0 bugs) → repeat

One source tree
- Componentization, contracts, modularization

19.7% search market share (>30% share if Yahoo! is included)
Modern Engineering Principles

Test-Driven Evolution: 11 Principles

1. Automate every test, but don’t test everything
2. Run all tests for every single check-in
3. Tests are binary: either they all pass, or they all fail
4. No test selection. Run them all. Scale thru HW + SW + Quota
5. Retire/Change old definitions and concepts
6. Embrace the Open-Source!
7. Testing in Production (deploy to production, test in production)
8. Deployment gated by tests: if any test fails, rollback
9. Defensive coding techniques (code + test case for every check-in, small check-ins, code behind flights, etc.)
10. Be truly data driven
11. Live Site remains the King!
1. Automate every test, but don’t test everything

Make every test reliable:

- Use mock data to isolate the code
- Write Once, Run Against Multiple Contexts
- Have “contractual” tests running to validate FE → BE schema

Trust modern tools:

- UI automation is no longer fragile (Selenium)
- Cloud helps with elasticity for your tests (scaling out)

Have a browser matrix, stick with it and deal with the rest!
2. Run all tests for every single check-in

Integration of tests with Code Flow

- Takes one hour for the first review to come (idle time)
- Changes ➔ build ➔ deploy ➔ tests

20,000 tests <= 20min, code coverage ~65%

- Fast: mocked data
- Fast: Machines + Parallelism
- Fast: time quota system per feature team
3. Tests are binary: either they all pass, or they all fail

- No concept of priorities until the test fails
- All tests must pass, otherwise check-in’s blocked
4. No test selection. Run them all. Scale thru HW + SW + Quota

The problems with test selection:

• A complicated imperfect system b/w product and tests
• Makes the process non-deterministic
• Some tests will rarely run!

“Throw machines at the problem!”

• This is what most big software corporations do
• Combination of HW + SW + Quota system
5. Retire/Change old definitions and concepts – Simplify!

- Dev Documents and Test Plans ➔ One Page
- Test case priorities ➔ Until they fail, they are P0
- Test suites ➔ one
- Test pass ➔ done when the check-in goes thru
- Test environments ➔ production
- But what about destructive? ➔ production
- Code coverage ➔ just one data point
- Ship decision ➔ from managers to engineers, based on bugs
- Obsessed about bugs ➔ Obsessed about user impact
- Line b/w dev and test ➔ blurred
6. Embrace the Open-Source!

Don’t try to compete with them – join them

All our tools are now all based on open-source

• Selenium, WebPageTest, PhantomJS, JS libraries, and many others

The work involved:

• Streamline the approval process
• Plumbing & Stitching the tools to work on MS tech
7. Testing in Production (TiP)

The problems with test environments:
- Maintenance
- Not representative
- Infinite catch-up game

Use an “invisible” PROD environment
- Behind a non-rotate flight
- Behind a VIP that can’t be accessed from outside corpnet

Look at issue patterns in PROD
- Instrument every single aspect of the code
- Big data/machine learning/telemetry techniques

What about destructive tests?
- Do it in PROD! Failovers/Load/Switch off the power to a DC
- Better found by you than by someone else!
8. Deployment gated by tests: if any test fails, rollback

xPing: our version of Gomez/Keynote:
- Simple HTTP Gets
- xPing+: complex web-based scenarios using Selenium
- Runs continuously, alerts based on availability threshold
- E2E (no mocking)

Canary deployment:
- Deploy to one DC
- “Observe” the xPing tests
- All passed after N minutes? Push to the other DCs
- No? Rollback!
9. Defensive coding techniques

Code + functional test case for every check-in

Small, frequent check-ins

Defensive code – no assumptions!

Code behind a flight – switchable on/off:

Example of simple explicit isolation in current production code
10. Be truly data driven

- Instrument every aspect of your code
- Build a pipeline to gather and analyze the data
- Flight → Fail 90% → Learn → Ship 10%
- Make informed decisions based on data

• Example:

Flowers at 1-800-FLOWERS - Same Day Delivery Available.
800flowers.com
🌟🌟🌟 (183920 reviews) - 37,000+ followers on Twitter
Same Day Delivery Available. 100% Satisfaction at 1-800-FLOWERS.

<table>
<thead>
<tr>
<th>Flowers</th>
<th>Anniversary Flowers</th>
<th>Rose Spectacular</th>
<th>Fresh Cuts</th>
<th>Best Selling Flowers</th>
<th>Birthday Flowers &amp; Gifts</th>
<th>Gift Baskets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quickly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delivery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warranty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviews</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social Media</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer Care</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Education</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Competitors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternatives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Guardrail Metrics:

<table>
<thead>
<tr>
<th></th>
<th>Treatment</th>
<th>Control</th>
<th>Delta [%]</th>
<th>Pval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Back 20</td>
<td>0.2295</td>
<td>0.2281</td>
<td>0.0014 [0.60%]</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Algo Pame Load Time(Overall PLT)</td>
<td>1212</td>
<td>1208</td>
<td>4.055 [0.34%]</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Revenue /UU</td>
<td>1.088</td>
<td>1.075</td>
<td>0.0130 [1.21%]</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Truncated Revenue / UU</td>
<td>0.8571</td>
<td>0.8504</td>
<td>0.0067 [0.79%]</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Distinct Queries / UU</td>
<td>14.67</td>
<td>14.67</td>
<td>–</td>
<td>1.001</td>
</tr>
<tr>
<td>Average Log Record Size (in KB)</td>
<td>111.4</td>
<td>111.1</td>
<td>0.2545 [0.23%]</td>
<td>&lt; 0.001</td>
</tr>
</tbody>
</table>
11. Live Site

Heavy monitoring in production:
- Organic Monitoring (counters and rules)
- Synthetic Simple Monitoring (xPing, 10K tests)
- Synthetic Advanced Monitoring (exploratory)

Availability:
- Based on real traffic (*Search Merged Logs*)
- Real-Time

DRI – Designated Responsible Individual

ITR – Incident Tracking Record
Challenges & Learnings

- Management must embrace it
- Put dedicated engineers on the problems
- Be date-driven (things won’t be perfect, but just do it!)
- This is a drastic change
  - Not everyone was happy... but don’t try to please everyone!
- Have challenging and insane goals
The Inner Dev Loop (on demand)

- Code Review Pipeline
  - BUILD
  - PASS
  - PASS

- Build pipeline
  - BUILD
  - PASS
  - Unit test

- Check in Wizard
  - Verify test results

- Source Code Repository

- IXP
  - BUILD
  - PASS
  - PASS
  - Mainline tests
Bing UX Functional Automation

Mocked functional automation
- Create and deploy mocked data
- Request it as a Backend response
Bing UX Functional Automation

<table>
<thead>
<tr>
<th>Vector</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP request/response</td>
<td>Mock</td>
</tr>
<tr>
<td>HTTP request/response</td>
<td>Live</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Mock</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Live</td>
</tr>
</tbody>
</table>

Test case

HTTP request

Browser

Selenium Grid

UX

BING BE

MOCK BE
Bing UX Functional Automation

<table>
<thead>
<tr>
<th>Vector</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP request/response</td>
<td>Mock</td>
</tr>
<tr>
<td>HTTP request/response</td>
<td>Live</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Mock</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Live</td>
</tr>
</tbody>
</table>

XML

HTTP request

Test case

Browser

Selenium Grid

UX

LIVE BE

MOCK BE
Bing UX Functional Automation

<table>
<thead>
<tr>
<th>Vector</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP request/response</td>
<td>Mock</td>
</tr>
<tr>
<td>HTTP request/response</td>
<td>Live</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Mock</td>
</tr>
<tr>
<td>Browser-driven</td>
<td>Live</td>
</tr>
</tbody>
</table>

HTTP request

Selenium Grid

Test case

Browser

UX

LIVE BE

MOCK BE
Code Reviews

- Parallel with build creation
- Parallel with test execution
- Can block check-in...
Checked-in code

- Has passed ALL tests
- WILL ship within hours
- OWNED by the feature teams
Continuous Delivery Loop (every day)

<table>
<thead>
<tr>
<th>Source Code Repository</th>
<th>Build pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>build</td>
</tr>
<tr>
<td></td>
<td>Unit test</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primary</th>
<th>Primary</th>
<th>Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canary tests</td>
<td>Canary tests</td>
<td>Canary tests</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Staging</th>
<th>TIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Canary tests</td>
</tr>
<tr>
<td></td>
<td>Mainline tests</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Canary PROD Env</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Canary tests</td>
</tr>
</tbody>
</table>
Performance Testing Strategy: Budgeting

Runs as a check-in test

Utilizes developer maintained budgets for resources

Below, identified an increase in page size due to a CSS change
Performance Testing Strategy: Time (Load Test)

Forks traffic from production (no PII, ~1M queries)
Results from initial requests cached & replayed
Runs for every check-in (2ms resolution)
Options: justify the increase, or offset it by optimizing other areas

![Graph showing performance over time](chart.png)
Questions?