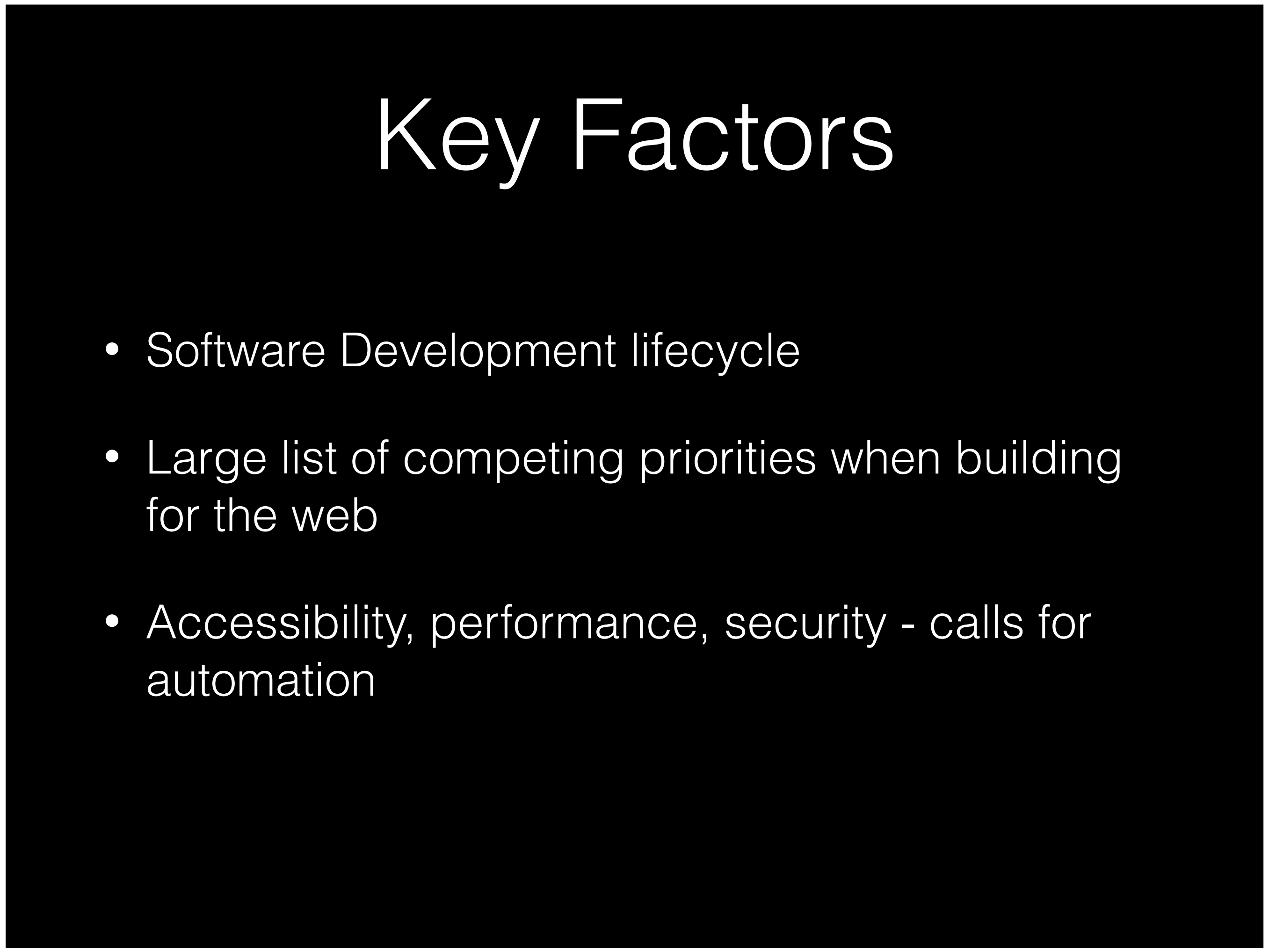
Shift-Left Accessibility

Sagar Barbhaya, Intuit Accessibility
CSUN 2018

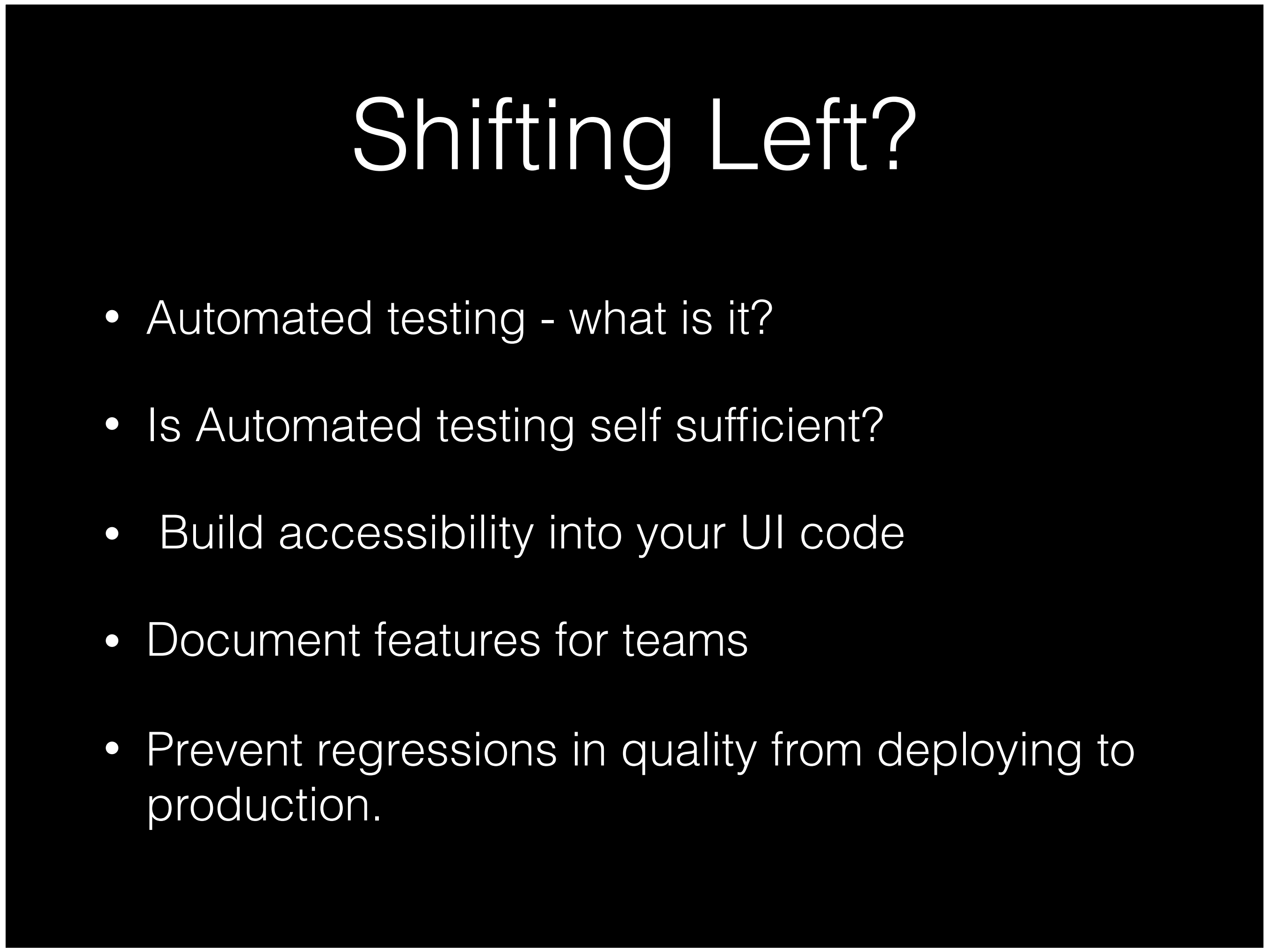
Twitter handle: @sagarbb160
Linkedin URL: https://www.linkedin.com/in/sagarbarbhaya/



Key Factors

- Software Development lifecycle
- Large list of competing priorities when building for the web
- Accessibility, performance, security - calls for automation

- If we look at SDLC for any application, there are a lot of things that need to be taken care of as a part of the development.
- Some of these include performance, security, accessibility.
- These should ideally be handled as a part of the dev cycle but generally are an after thought. This is where the problem starts.



Shifting Left?

- Automated testing - what is it?
- Is Automated testing self sufficient?
- Build accessibility into your UI code
- Document features for teams
- Prevent regressions in quality from deploying to production.

Automated testing is a great way to start weaving accessibility into your website, with the ultimate goal of shifting left more and more towards the UX and discovery process

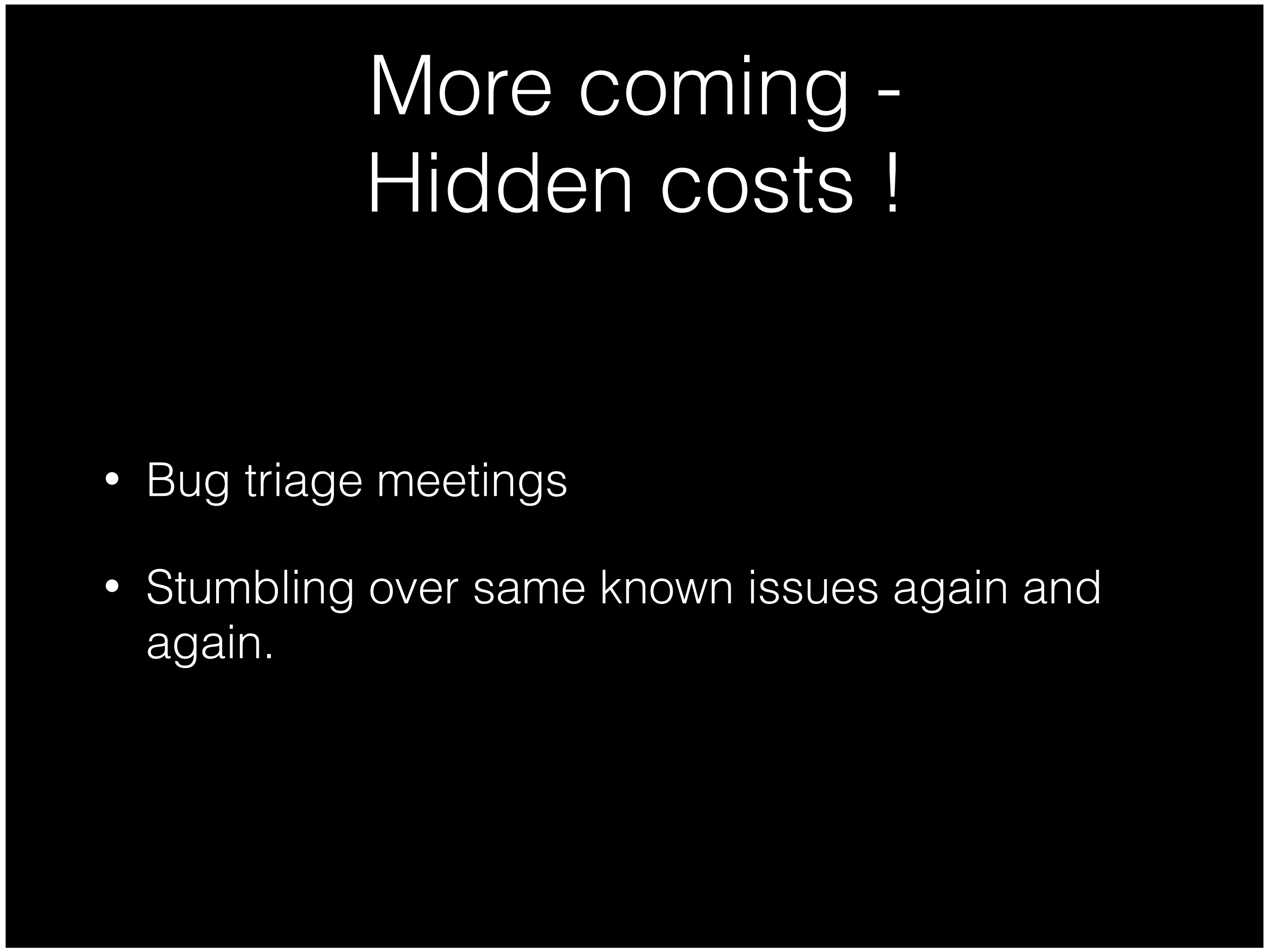
Its not fully self sufficient but it’s a valuable way to address easy wins and prevent basic fails. Free humans up for more complex tasks. determining whether an image is an image of text or whether an image contains important text information.

How Much Does It Actually Cost To Fix A Bug?

"At first, the cost of fixing a bug at the requirements stage is nominal, when everything is on the drawing board. But as the software moves along in its life cycle the cost of fixing a bug increases radically. We start at 1 times when we are at the initial development stage when a bug is no more than a change in notion. But at the design stage, the relative cost is 5 times what it was compared to the requirements stage, and then ten times what it was when it becomes code and on this goes until it the relative cost of a bug fix is 150 times what it was originally. Source "

I am going to be quoting some folks who have actually sat down and calculated the cost of fixing a bug later in the Software Development LifeCycle.

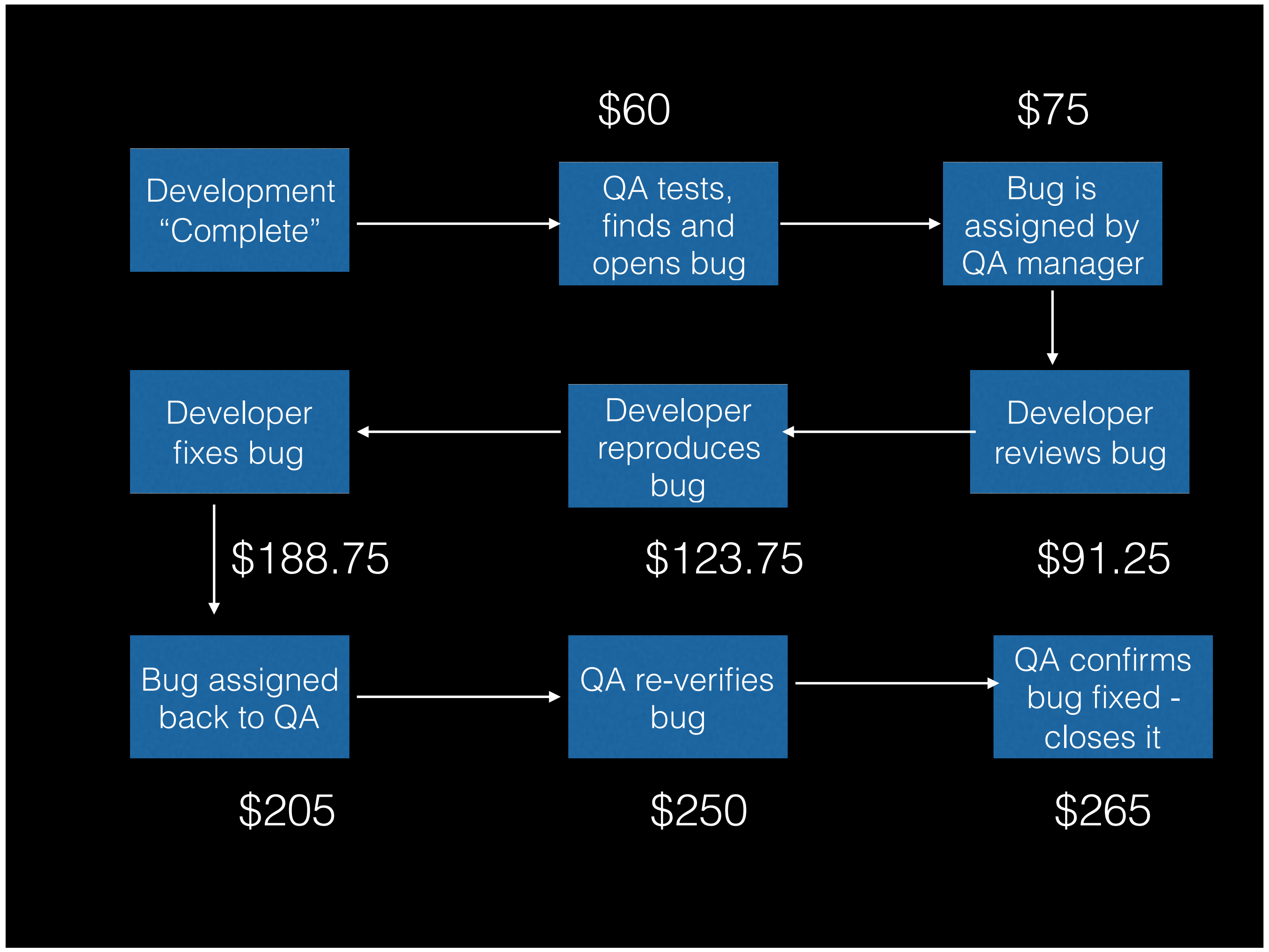
http://extreme-accessibility.karlgroves-sandbox.com/



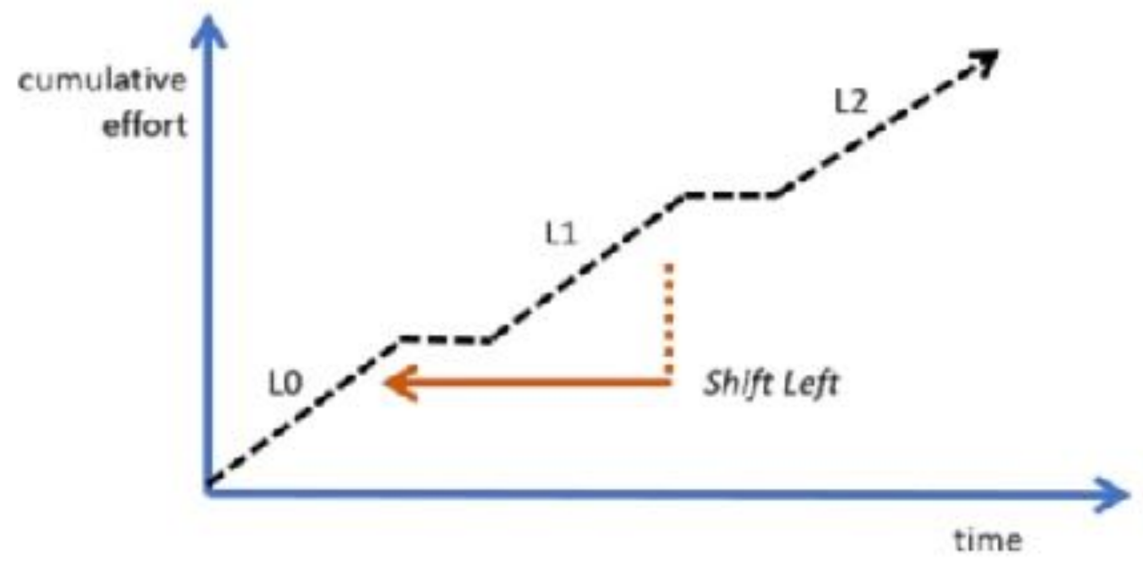
More coming - Hidden costs !

- Bug triage meetings
- Stumbling over same known issues again and again.

There is a lot of complication and layers when it comes to SDLC without shift left. We shall see in the next slide how a bad SDLC looks in terms of fixing bugs and you shall see the costs associated with each of these bad steps.


$60
$75
Development "Complete"
QA tests, finds and opens bug
Bug is assigned by QA manager
Developer reviews bug
$91.25
Developer reproduces bug
$123.75
Developer fixes bug
$188.75
Bug assigned back to QA
$205
QA re-verifies bug
$250
QA confirms bug fixed - closes it
$265

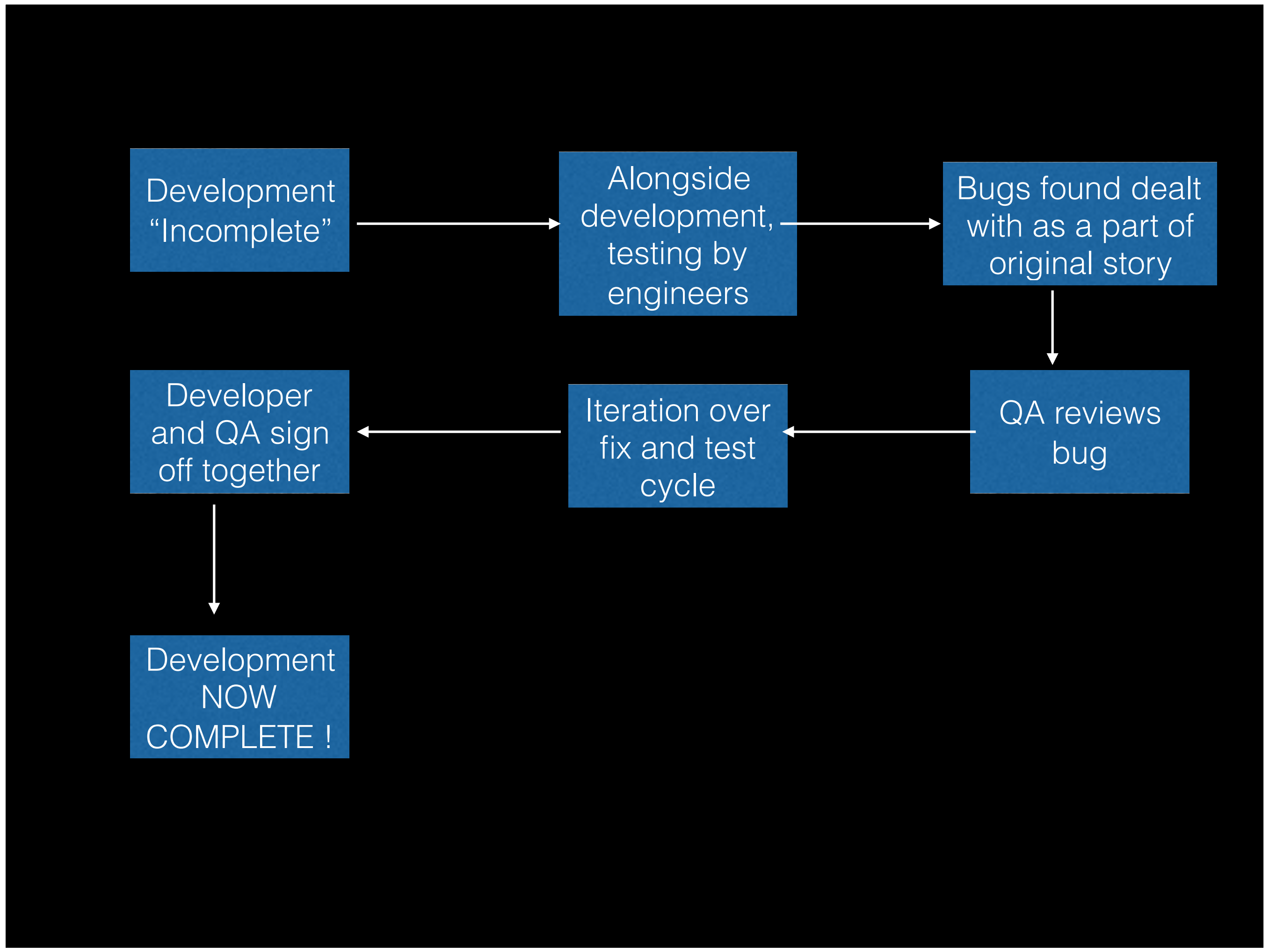
Graph Explaining Shift Left



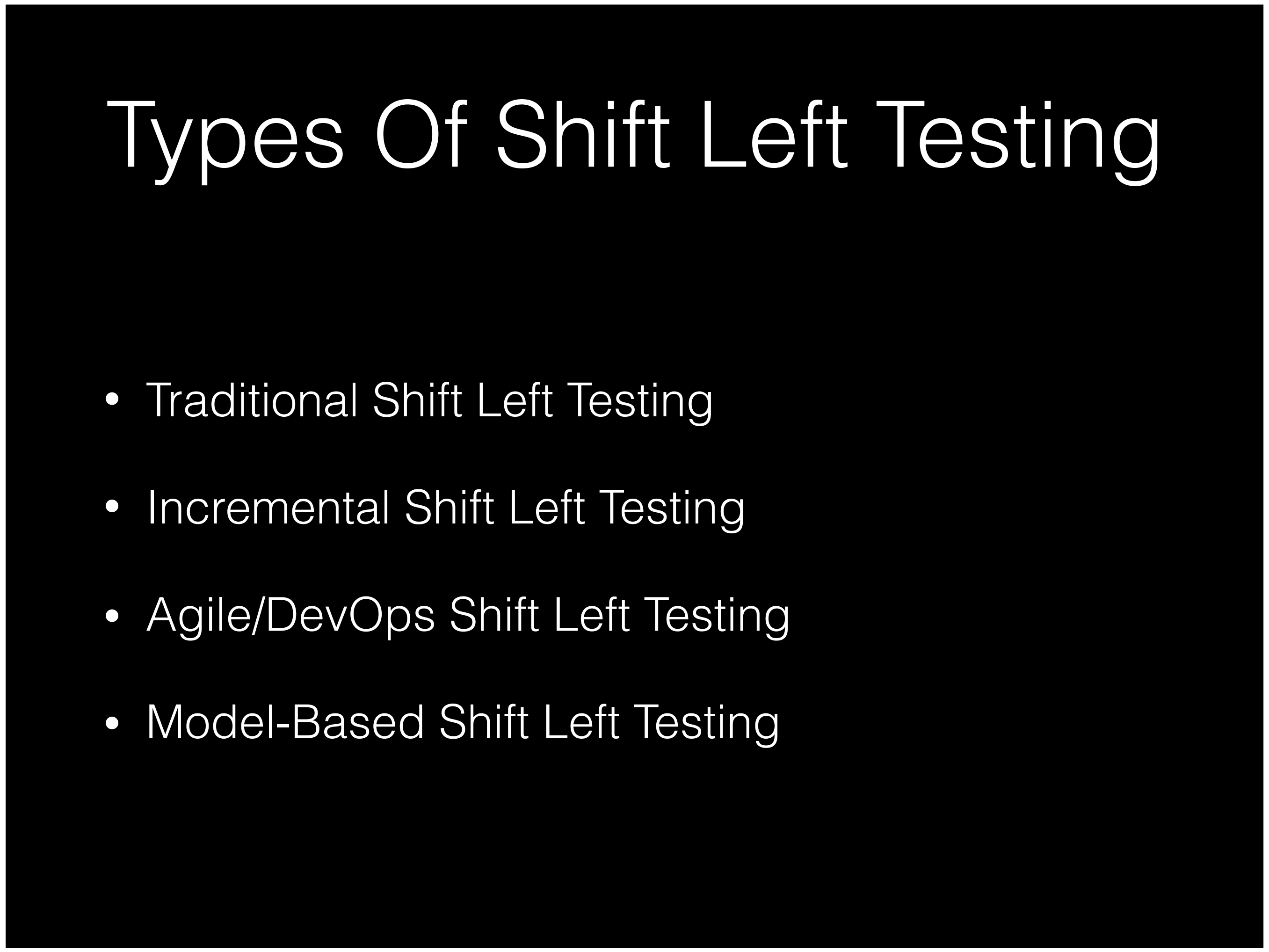
L0, L1, L2 are levels of activities that happen one after the other and show the amount of time and cost that they incur without shift left.

By shifting left, these activities are moved to lower cost delivery channels, which optimizes costs and enables more expensive resources to focus on work that can't be performed by less skilled labor.

In addition to reducing costs, doing this properly has the benefits of reducing both the time and total effort associated with handling requests and dealing with issues—in addition to improving both customer satisfaction and service quality.


Development "Incomplete"
Alongside development, testing by engineers
Bugs found dealt with as a part of original story
QA reviews bug
Iteration over fix and test cycle
Developer and QA sign off together
Development NOW COMPLETE !

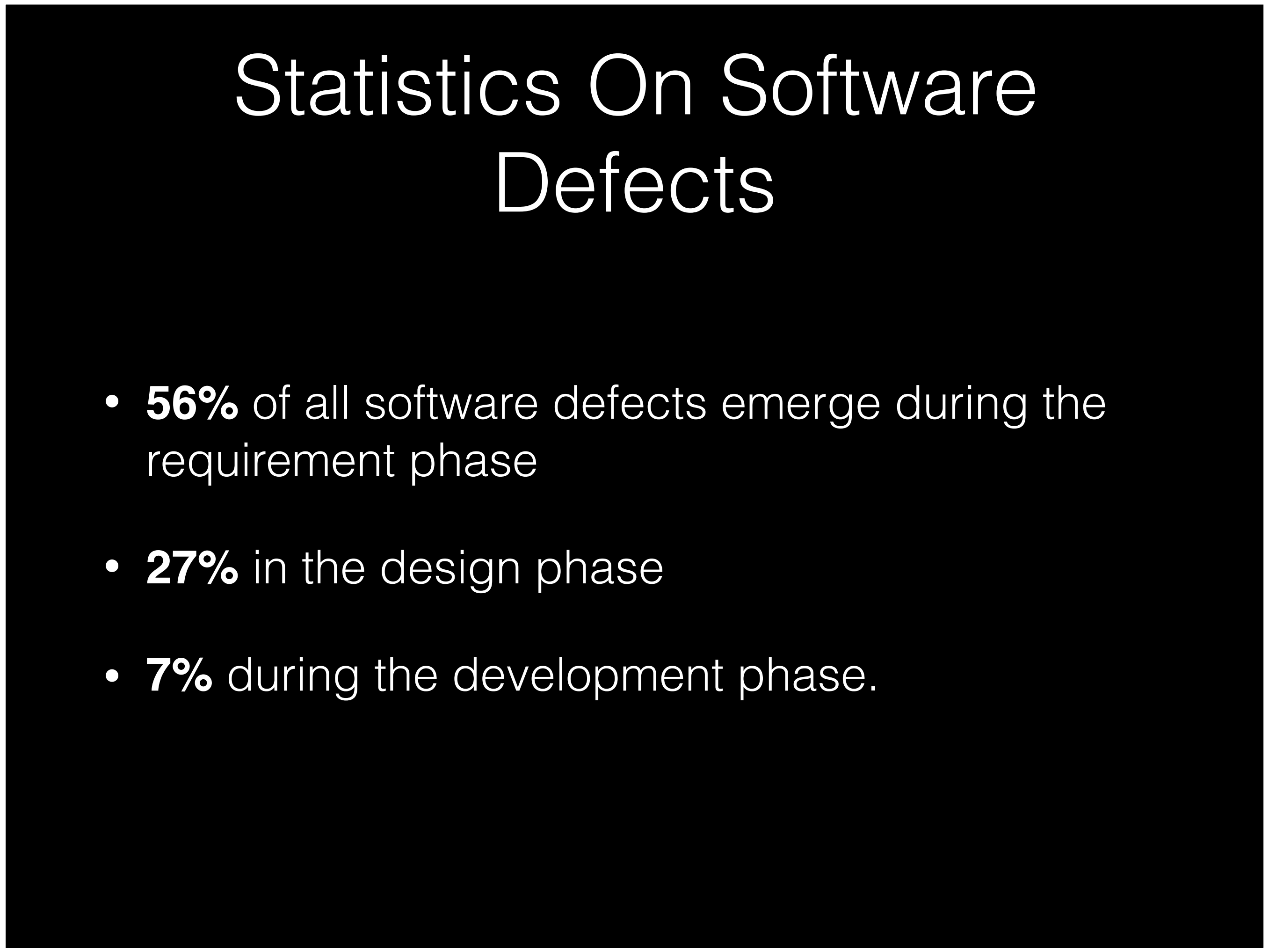
The right way to triage SDLC



Types Of Shift Left Testing

- Traditional Shift Left Testing
- Incremental Shift Left Testing
- Agile/DevOps Shift Left Testing
- Model-Based Shift Left Testing

https://en.wikipedia.org/wiki/Shift_left_testing

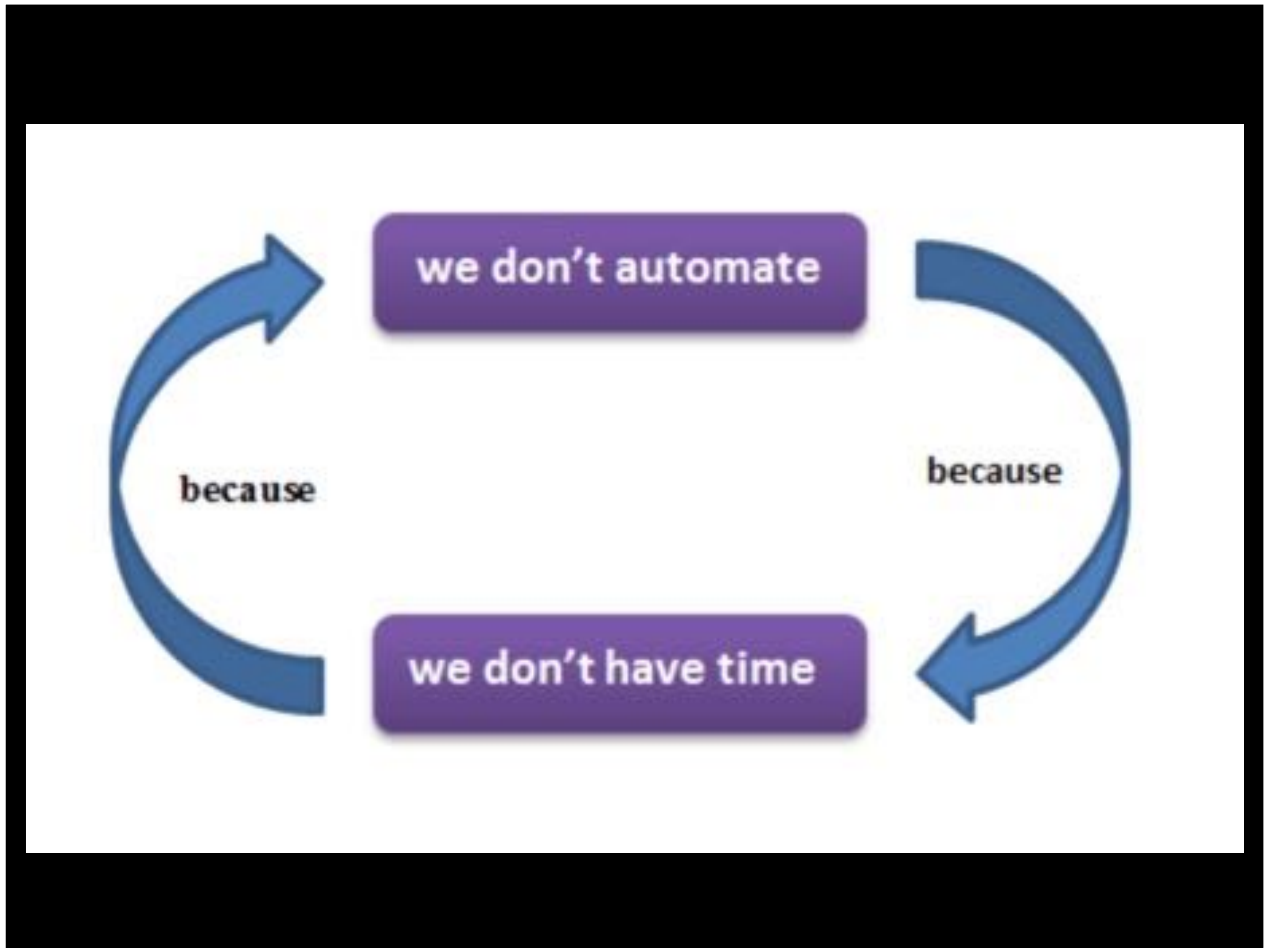


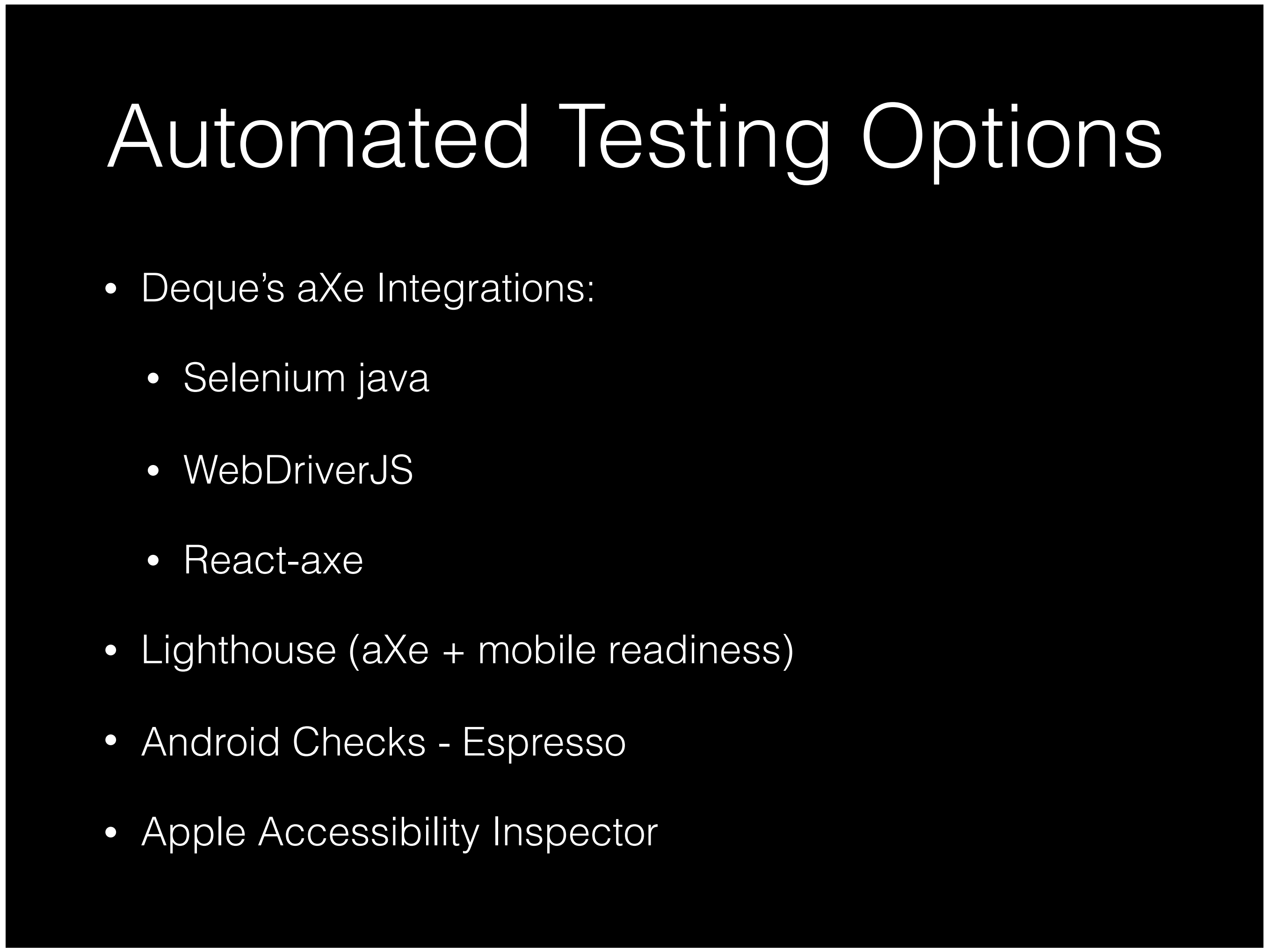
Statistics On Software Defects

- **56%** of all software defects emerge during the requirement phase
- **27%** in the design phase
- **7%** during the development phase.

Critical to start testing way ahead in the Software Development Lifecycle.

Reference - https://www.bmc.com/blogs/what-is-shift-left-shift-left-testing-explained/


we don't automate
because
because
we don't have time



Automated Testing Options

- Deque's aXe Integrations:
  - Selenium java
  - WebDriverJS
  - React-axe
- Lighthouse (aXe + mobile readiness)
- Android Checks - Espresso
- Apple Accessibility Inspector

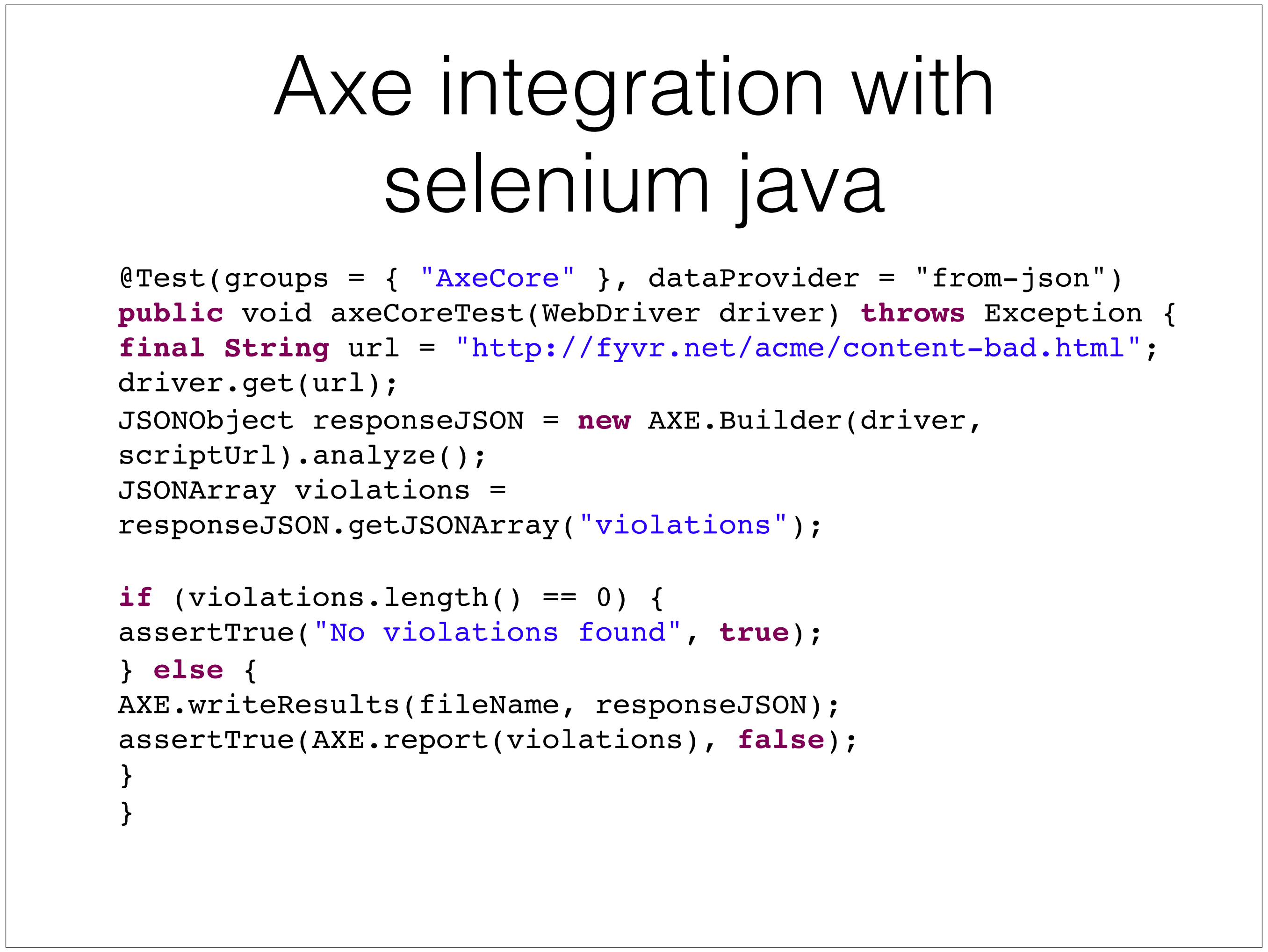
Grunt + HTML CodeSniffer: https://github.com/yargalot/grunt-accessibility
AccessSniff https://github.com/yargalot/AccessSniff

What Is aXe?

- WorldSpace aXe (The Accessibility Engine) is free, open-source JavaScript accessibility rules library.

- It was developed to be fast and to return zero false errors or duplicate results, and it is available as a GitHub repository, browser plugin, or framework integration.

Reference - https://www.deque.com/axe/



Axe integration with selenium java

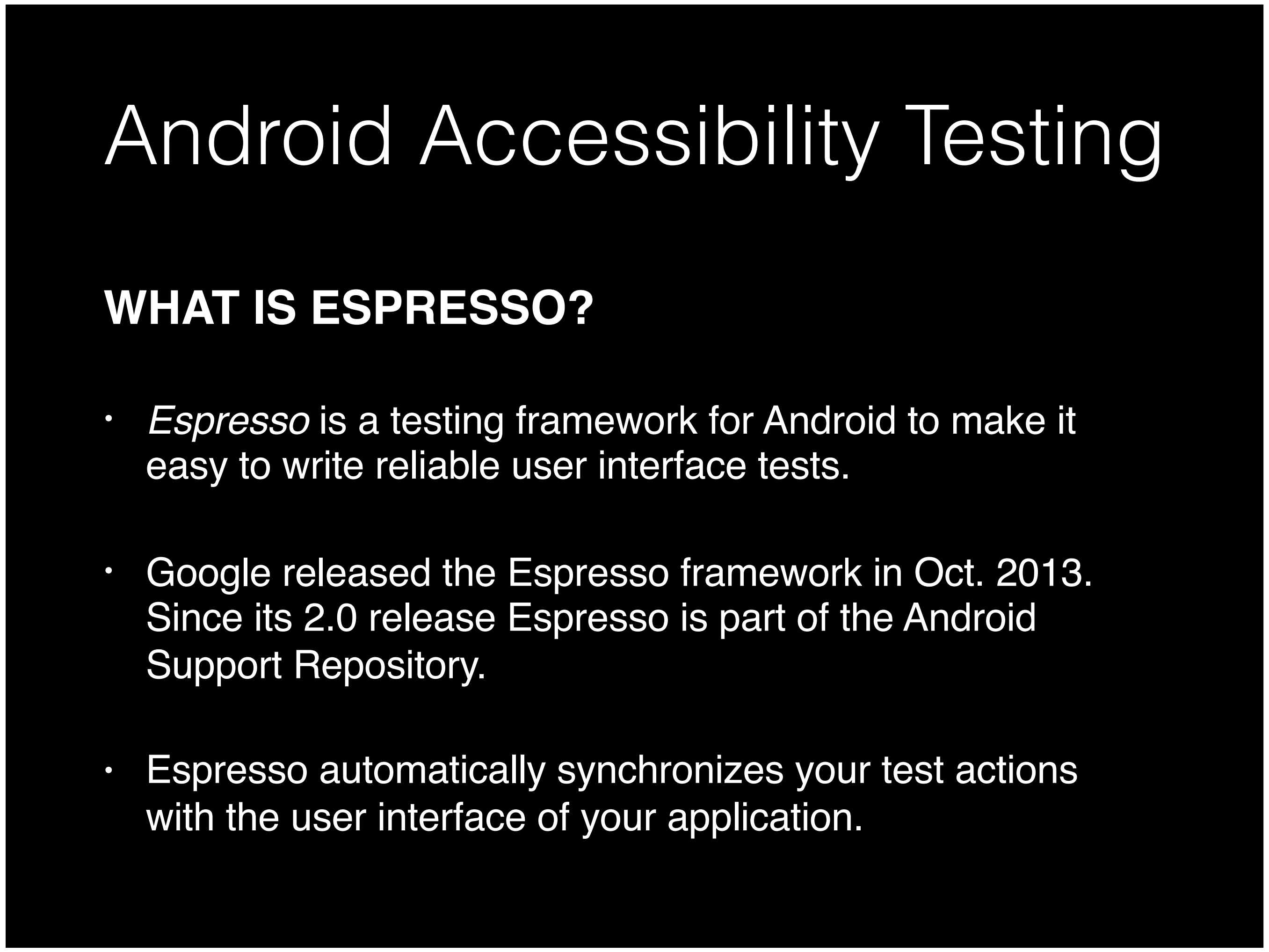
```
@Test(groups = { "AxeCore" }, dataProvider = "from-json")
public void axeCoreTest(WebDriver driver) throws Exception {
final String url = "http://fyvr.net/acme/content-bad.html";
driver.get(url);
JSONObject responseJSON = new AXE.Builder(driver,
scriptUrl).analyze();
JSONArray violations =
responseJSON.getJSONArray("violations");

if (violations.length() == 0) {
assertTrue("No violations found", true);
} else {
AXE.writeResults(fileName, responseJSON);
assertTrue(AXE.report(violations), false);
}
}
```

What is axe-webdriverjs?
An open source library that injects axe-core into Selenium Webdriver instances

DEMO

- VIDEO



Android Accessibility Testing

WHAT IS ESPRESSO?

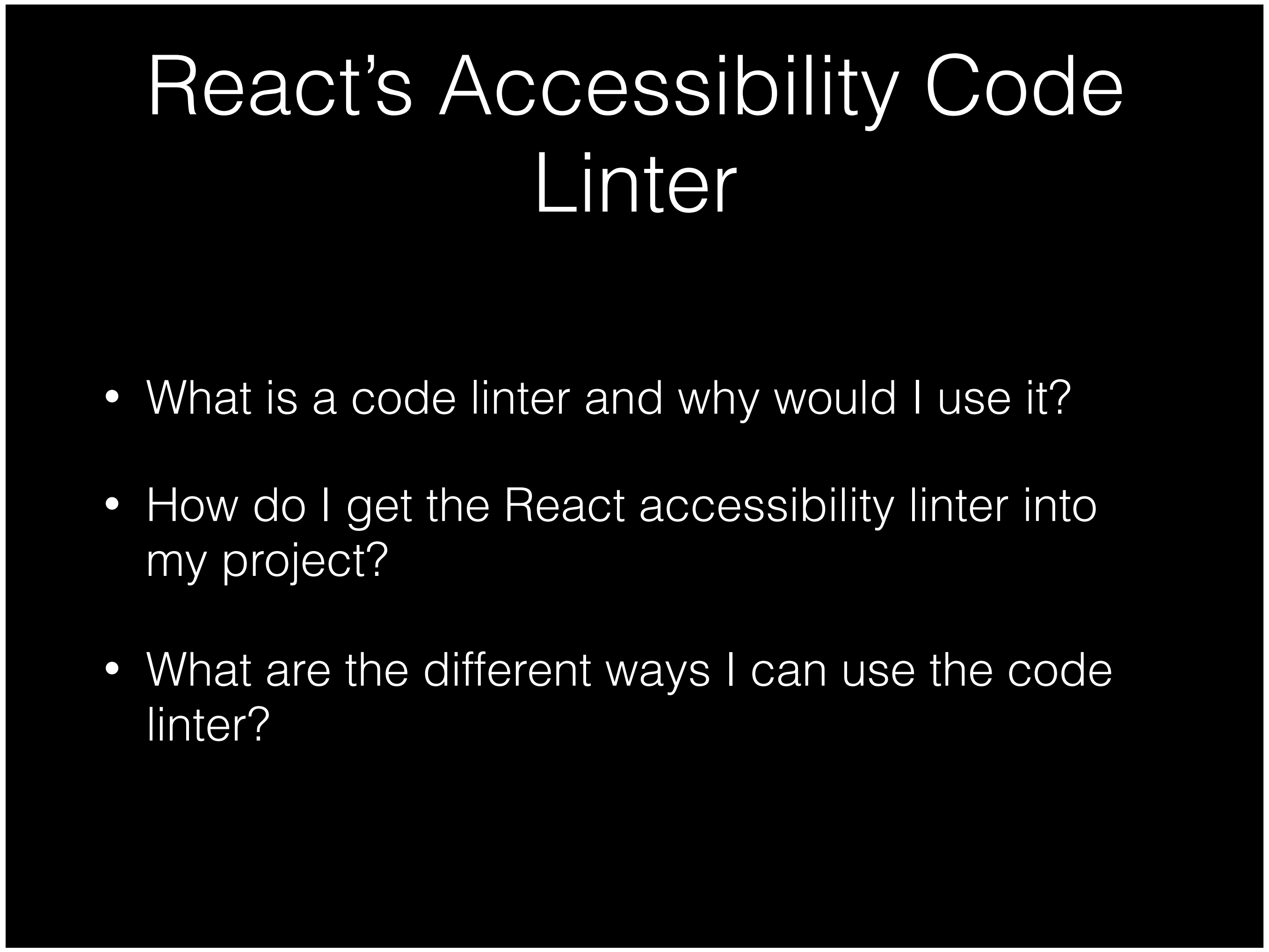
- *Espresso* is a testing framework for Android to make it easy to write reliable user interface tests.
- Google released the Espresso framework in Oct. 2013. Since its 2.0 release Espresso is part of the Android Support Repository.
- Espresso automatically synchronizes your test actions with the user interface of your application.

The framework also ensures that your activity is started before the tests run. It also let the test wait until all observed background activities have finished.
It is intended to test a single application but can also be used to test across applications. If used for testing outside your application, you can only perform black box testing, as you cannot access the classes outside of your application

Accessibility Test Framework for Android

- Consists of rules around accessibility to help evaluate different applications.
- Any team using Espresso for Android Automation can integrate this library easily into their existing automation suite!

https://github.com/google/Accessibility-Test-Framework-for-Android



React's Accessibility Code Linter

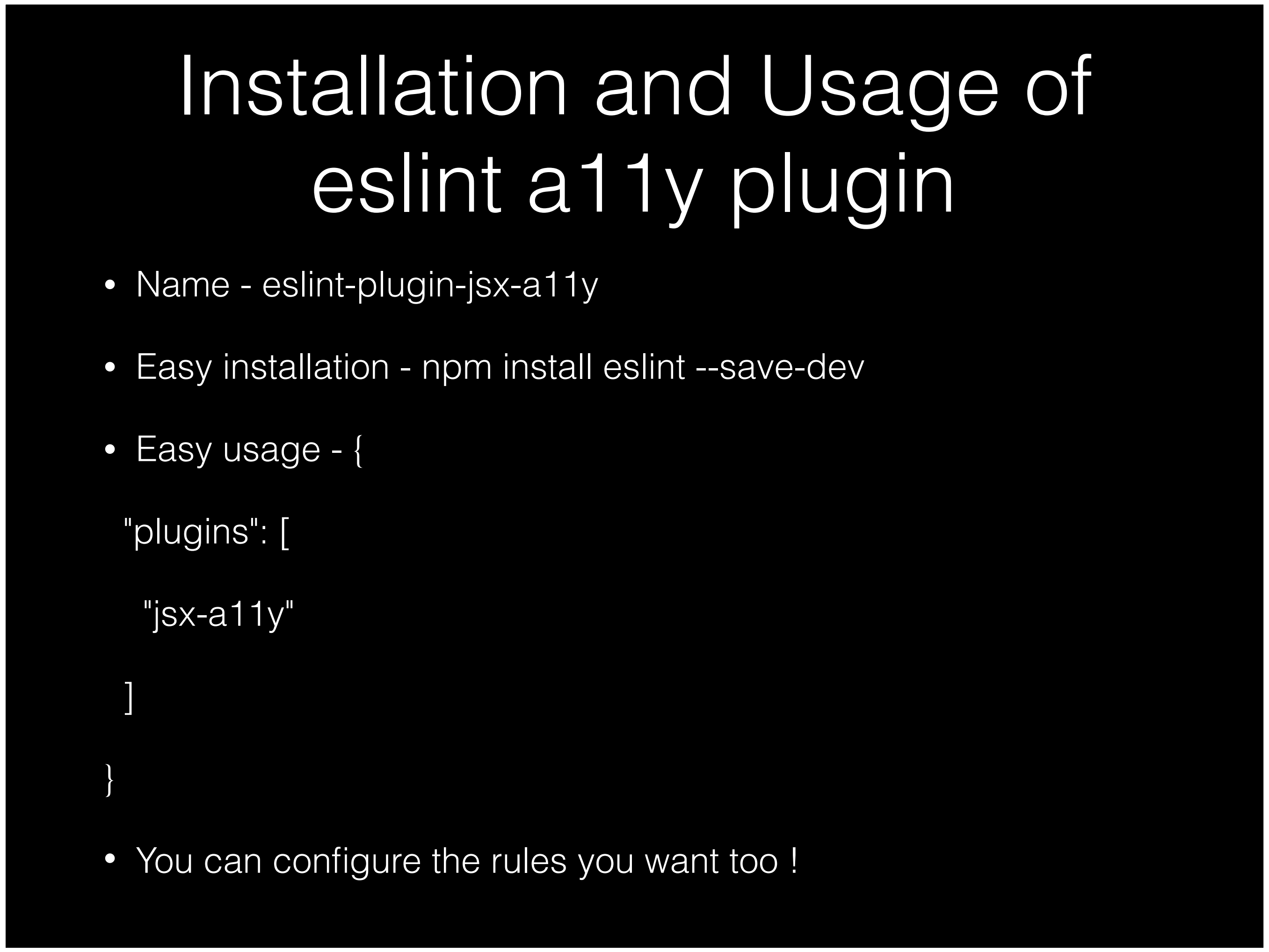
- What is a code linter and why would I use it?
- How do I get the React accessibility linter into my project?
- What are the different ways I can use the code linter?

Going to cover these questions in the next slides
JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it.

What is a11y plugin?

- In the simplest terms, a utility which helps you write accessible code.
- It’s a plugin to be used with ESLint
- Reports issues directly in your development terminal.
- Can also be hooked to your build systems.

a very popular and well made JavaScript code linter which is supported by all the popular code editors of today.



Installation and Usage of eslint a11y plugin

- Name - eslint-plugin-jsx-a11y
- Easy installation - npm install eslint --save-dev
- Easy usage - {

```
  "plugins": [
    "jsx-a11y"
  ]
}
```

- You can configure the rules you want too !

This avoids the overhead of the console bombarding with errors. We can limit it to showing us only the critical one’s and others can be warnings.

DEMO - React A11y Website

- REACT ACCESSIBLE DEVELOPMENT

Build Time Warnings

```
./src/AboutBad.js
  Line 150:  img elements must have an alt prop, either with
meaningful text, or an empty string for decorative images

  Line 152:  Invalid alt value for img. Use alt="" for
presentational images

  Line 159:  <input> elements with type="image" must have a text
alternative through the `alt`, `aria-label`, or `aria-labelledby`
prop

  Line 177:  img elements must have an alt prop, either with
meaningful text, or an empty string for decorative images

  Line 179:  Each area of an image map must have a text alternative
through the `alt`, `aria-label`, or `aria-labelledby` prop Anchors
must have content and the content must be accessible by a screen
reader

  Line 198:  Do not use <blink> elements as they can create visual
```

Testing Tools

- Six Accessibility Testing Tools for Your Toolbox -

1. WAVE
2. aXe
3. Tenon
4. a11yTools
5. tota11y
6. HTML_CodeSniffer

- Do check them out !

These are five of the accessibility testing tools that I like to use on a regular basis for accessibility testing and reporting. Each tool has some pros and cons and each has certain accessibility issues they're great at identifying.

1. WAVE browser extensions are much more powerful than the WAVE website because you won't be able to use the online WAVE behind password protected websites and sometimes the online version does not work on very complex websites. The WAVE extensions work behind a firewall, are faster to run tests, and handle complex websites.

2. The most popular way aXe is used is likely via the Chrome and Firefox Extensions that provide a user interface for the accessibility errors output.

3. Tenon:

4. a11y tools No Chrome or Firefox extension. -biggest con

5. tota11y - Headings testing tool is awesome for testing incorrect heading levels and nesting or not starting page with an H1 element. but ARIA Landmarks testing tool seems less useful than some others.

6. pros - Iframe testing, Tests for duplicate ID attribute values.

Concluding notes

- Automation is key.
- Encourage use of keyboard testing as a tool.
- Incorporate linters to ease accessible development.
- Shifting left to go the right way !

Key Takeaways

1. Identify and Plan Testing Lifecycle
2. Integrate Development and Project Management Process with Testing
3. Define Quality Standards & Controls for all SDLC stages
4. Create Process and Operation driven test cases and framework
5. Induce Developers to Code with Testability in Mind
6. Define Continuous Feedback Mechanism

Links for research

- https://24ways.org/2017/automating-your-accessibility-tests/
- https://24ways.org/2014/integrating-contrast-checks-in-your-web-workflow/
- https://www.24a11y.com/2017/reacts-accessibility-code-linter/
- https://www.24a11y.com/2017/accessibility-testing-tools-desktop-mobile-websites/
- https://www.24a11y.com/2017/build-cloud-hosted-accessibility-testing-windows-computer-using-amazon-workspaces/
- https://www.24a11y.com/2017/writing-automated-tests-accessibility/
- https://www.npmjs.com/package/eslint-plugin-jsx-a11y
- https://www.deque.com/product/axe/
- https://www.cherwell.com/blog/want-to-shift-left-with-itsm-change-management
- http://extreme-accessibility.karlgroves-sandbox.com/

@sagarbb160