SOFIA LEIVA: Thanks for joining this webinar, entitled "Mobile Accessibility Toolkit." I'm Sofia Leiva from 3Play Media, and I'll be moderating today. And today I'm joined by Gian Wild, CEO of AccessibilityOz. And with that, I'll hand it off to Gian, who has a wonderful presentation prepared for you all.

GIAN WILD: Hi, everyone. Thanks for joining me today. I'm in DC at the moment, and it's a beautiful day. I just want to start by saying you can follow along with the slides at pz.tt/3playmobile. And the presentation, as well as the actual toolkit itself, are available on that slide. So if you actually have a look at it, it looks a little bit like this.

Let me just make sure I'm sharing the right thing. Let's see. Excellent. So hopefully you can see that now. And this is what you do. You'll need to log in to be able to access the links. Create an account. But you can also ask questions if you come across things later. And they'll get sent to me.

And also, this PDF is not accessible. But if you do access the account itself, you can download the PowerPoint, which is fully accessible, which we'll also get Sofia to send out. So thank you very much for joining me today. And that URL again is pz.tt/3playmobile.

So a little bit about what we did. So what happened is, in 2017, I was a committee member of the ICT Accessibility Testing Symposium conference, which was in October in DC. And we talked at the end of the conference, as we do every year, about what was, I suppose, missing or what was difficult in the industry.

And one of the things that we came up with was that it was really difficult to know what to test when it came to mobile accessibility. And we're talking about mobile sites, as well as native apps. And so we actually put together a methodology. And it was a combination of Deque's methodology, the BBC Mobile Accessibility Guidelines, the AccessibilityOz guidelines, et cetera. And we had a committee that worked on a set of guidelines.

And we released them in 2018 in October. And then that was a few months after WCAG 2.1 was released. And we thought that WCAG 2.1 would probably address the issues. But unfortunately, that was not the case.

And so when we talk about WCAG 2.2, we know that all success criteria in WCAG 2 are

applicable to mobile, whether we're talking about mobile sites or native apps, because WCAG 2 was written to be technology neutral. However, not all aspects of mobile accessibility are specifically covered by WCAG 2. So for example, although WCAG 2 requires sites to be accessible to the keyboard user, it doesn't specify that it should also be available to the touchscreen user.

Now, of course, 2.1 does build on this and addresses more criteria related to touch screens, like pointer gestures and sensors and orientation, et cetera. However, we felt as a community that it didn't address enough things. And one of the perfect example is touch targets, which was relegated to AAA. And we thought that that was something that was really important and needed to be at the minimum level.

So we developed a second version of these guidelines. And we split them into two, one for mobile sites and one for native apps. And so I'm going to talk about the mobile site accessibility testing guidelines today.

And they're freely available. They're on Creative Commons. You can get them on that pz.tt/3playmobile link. You can get it on the ICT Accessibility Testing Symposium site and also the AccessibilityOz site. And the native app methodology will be released around Thanksgiving. And the plan is to iterate them quickly. And so we're actually looking for people to join our committee to actually iterate them and release new versions every year.

So one of the things that we first talked about was variations of mobile sites. So for example, WCAG 2.1 specifically talks about page variations. So a low-vision user who uses the zoom function inherent in the browser often end up with what is basically a mobile version of the site. WCAG 2 requires that zoom to 200% be included in regular testing. But on top of that, it's not just that you should test to that. But all your functionality needs to be available at 200% as well.

So an example of where this can be a problem is with YouTube. And this has been fixed. So at 100% screen size, you actually have your upload and notifications in your top-right-hand corner. And this is when you view it on a desktop. But when you zoom to 200%-- and as I mentioned, this is being fixed-- they disappear.

So now, why would this be the case? This is the case because they assume that at this point, you're viewing the YouTube website on your mobile device. And if you're going to upload videos, then you'd do so using YouTube mobile app. You wouldn't do it on the mobile browser. But of course, for someone using-- someone on a desktop that is just increasing the text size

due to accessibility issues, they're not going to be able to actually upload that content.

So the first thing we talked about was the fact that every variation needed to be tested, but also that all functionality needed to be available on all variations of the page. So that was one of the things about mobile sites. And we did look at native apps. And we talked about how we determine what to test on native apps.

So basically, the first step is defining the application functionality. So when you go into a website, it often has set up quite a number of different purposes. You might be going to, say, a banking website to find out what the latest mortgage rates are or to open an account or to find a branch location or an ATM location.

That's not really the case when you're talking about a mobile app. There's often very specific functionality. So you might have an online banking app which is specifically for people that have bank accounts and they want to transfer their money across. And that mobile banking app is not going to have information on mortgages or how to contact someone in a specific location.

So that's the first step when you're testing native apps, is to define application functionality, and then also to test common elements. And common elements include things like navigation, menus, headers, footers, landing pages, emergency alert pages, login pages, settings, accounts and profiles, contact us, real-time updates like on eBay and Uber, privacy policy, terms and conditions, interactional and transactional processes-- so selecting a product, adding to cart, payment, live chat, help, Q&A-- widgets such as calendars and date pickers, and third-party integration such as geolocational maps and chat, and, of course, high-traffic areas.

So basically, when it comes to testing native apps, we determined there's really-- the first step is asking the question. So how would the experience be impacted if the functionality failed, the content couldn't be reached, or the experience caused a barrier to the user? And then prioritize. So all functionality needs to be accessible within the native application. But you do really need to define and include the critical functionality for each app to prioritize testing.

And the other thing that we talked about was how we need to test with real devices. So I do a lot of travel. I spend about six months of each year in the States. And for a long time while I was doing that, there was no Wi-Fi on airplanes. And so we'd often get to LAX flying in from Melbourne. It's a 15-hour flight. We'd often get to LAX and find that we'd have to sit on the

tarmac for a few hours because often we didn't arrive exactly when planned.

But you could actually access LAX Wi-Fi at that point, which was great, until I came across this problem, where it says it's Wi-Fi internet access for LAX International Airport. And it says, "This page will redirect, so content doesn't really make sense to have here." And so all of a sudden, I don't really feel safe giving you my credit card details or using the internet.

And so this is why you need to test with real devices. It's absolutely essential to test with real devices. And we did decide when we were developing the second version of the toolkit that we would not allow simulations, so browser simulators or mobile simulators.

So the mobile testing methodology itself has a number of steps. So this is the-- step one is, identify what needs to be tested. So the first step is to identify the devices that you need to test with. And so basically you should test with an iPhone, an iPad, and an Android phone. But you really need to look at your Google statistics and your Google analytics because that will inform what devices you want to test with.

So for example, if you have a website that's aimed at the East, then they are actually much more likely to use Android devices than iOS, whereas people in the West tend to use iOS devices. So definitely look at your Google statistics. We tend to test-- "we," I mean AccessibilityOz-- with Samsung Galaxy phone and also a Pixel phone.

And we always test with Chrome. So that's the other thing. If you are testing with an Android device, and especially a Samsung device, we would recommend not testing with the internet browser that comes packaged with the device but instead installing Chrome and testing with that because that's a much better representation of what people will see across devices. Whereas if you test with just the internet browser, you'll have a very good representation of how it looks on that particular phone. And you want to see across a number of things.

So for a mobile site, you also need to identify the site type and variations of the page. So there are three site types. There's basically a desktop site, which doesn't reflow when you look at it in a smaller-sized window or on mobile. We also found M dot sites, which-- there aren't a lot of them around anymore. One of the most famous ones would probably be m.facebook.com-- and then responsive sites.

And the thing is, just because there is a M dot site doesn't mean there won't be also a responsive site. So there's a lot of information in the toolkit around how you determine what

type of site it is and then, once you determine the site, how many variations of the page. So usually you find there's one for-- a variation of the page for a mobile-sized portrait, a mobile-sized landscape, a tablet-sized portrait, and then your desktop.

But there can be more than that. And you do definitely need to figure them out. So it's often not something that the owners of the site know unless they are actually the ones that developed it. So you need to test with the different devices on the different variations of the page.

And then the second step is to conduct specific mobile tests. And we broke them up into four issues-- critical issues, mobile-specific issues, mobile assistive technology and feature support, and mobile and desktop relationship issues.

There are four main testing methods in mobile testing. We found-- we decided devices, testing on mobile and tablet devices. Devices with assistive technology or mobile features, so testing on mobile and tablet devices with assistive technologies. Responsive window, so testing on responsively sized window on desktop. And also testing on desktop, because it is important that there is some consistency between a desktop version of the site and the mobile site.

When it comes to native apps, we identified two main testing methods, one testing with devices and one testing with devices with assistive technologies. And I do understand there are some testing tools out there. But we didn't feel that they were sufficiently mature enough to include in this methodology.

So the other thing to remember is that you need to meet WCAG 2 and this mobile testing methodology. We didn't include anything that was already in WCAG 2. So Name, Role, Value or coded headings and things like that are not included in this methodology.

However, WCAG 2.1 issues are included. But it's very clear in the methodology what's a WCAG 2.1 requirement because we refer back to 2.1. So if you're already meeting 2.1, you could ignore those issues.

So I'm just going to take you through the four groups of issues and what we identified. So the first is critical issues. And we found a whole bunch of what we called "traps."

We defined a trap as where a user is trapped within a component and cannot escape without closing the browser or the app. And there are many more traps in mobile and native apps than

on desktop. On desktop, there's really a keyboard trap and an inverse keyboard trap. And there's not much else.

So the first one we found was the hover trap. Content must be able to be dismissed if activated on touch. And so, often these are actionable items that are activated on mouse hover on a desktop. And these apply to touch users.

The example we have here is-- it's a dress shop. And if you look at this site on desktop, if you move your mouse over the dress, you get a zoomed-in portion of the dress on the right-hand side. You move the mouse away from the dress, that zoomed-in portion disappears.

However, if you're looking at it on mobile, if you tap on the dress, you get a zoomed-in portion of the dress, which overlaps a whole lot of content. But even if you tap elsewhere, you can't actually get that zoomed-in portion to disappear. So the only way to get that zoomed-in portion to disappear is to close the browser and start again or re-enter the URL.

This is one I'm sure we've all come across, an on-screen keyboard trap. On-screen keyboard must be able to be dismissed. And this applies to on-screen keyboard users.

And so this is Trello. There's no actual field or form that you can enter text into at the moment. But the keyboard is up. And although it looks like you can drag it to close, you can't do that. So you can't actually dismiss the keyboard. And of course, this is really only an iOS issue because most Android devices have a button that allows you to dismiss the keyboard that's part of the device.

Then we have the screen reader swipe trap. Screen reader users must always be able to activate an item on the current page or move back to the previous page. And this applies to screen reader users.

So this is an example of a LinkedIn post. And they've fixed this now. But when you go to post an update, you get this page, which gives you the option of selecting Public or Connections. And if you're a touch user, then you can select one of them. And it will post it to the correct users.

However, a screen reader user can't select any options here. So they can't select Public or Connections. They can't go back. And there's no way to go forward, either. So we call that a screen reader swipe trap.

Then we have a touch trap, which is, user must always be able to scroll or swipe to move up and down the page. And this applies to touch users.

So for example, this is-- we're at the bottom of a page. And there's an arrow at the bottom-right-hand corner that doesn't meet color contrast requirements. And the only way to scroll back up to the top of the page is to activate that arrow. If you actually try and swipe with your finger, nothing happens. So we call that a touch trap.

A zoom trap, which is, do not replace the entirety of the page with a picture that overrides static mobile functions, such as swiping and scrolling. And this applies to touchscreen users.

So this is an example of a zoom trap where you have to-- your finger has to hit the small areas of white around the map to actually swipe up and down the page. If you actually hit anywhere in the map, then you scroll the map. And of course, this is less of an issue now with two-finger scroll options and things like that.

Text-to-speech trap. If the app has an ability to provide content via text to speech, the screen reader user must be able to pause or stop the app speaking in a simple manner-- i.e., by performing a swipe on a screen. So this applies to screen reader users.

And so, for example, this is a text-to-speech trap where you can activate Pocket to start talking. But once activated, the screen reader user can't stop the text-to-speech. And of course, they need to be able to stop it in order to navigate the page and choose something else because they're relying on their screen reader to read things aloud to them.

Then we have a swipe trap, which is, any swipe gesture that is superseded by the screen reader must have an alternate gesture. You must be able to perform the same action by using a link. And this applies to screen reader users or other assistive technology users which capture the swipe feature.

So this is an example of a Facebook friends list. So you get to the Facebook friends. And the only way you can go back to your main page is to swipe from left to right. So you can't go back. I think they've actually fixed this issue without swiping. And of course, a screen reader user can't swipe because the swipe is interpreted by the screen reader.

This is another example. Lufthansa has an option at the bottom, Swipe to Enter. And so you have to swipe right to left to go to the next page. It's not a link. And of course, a screen reader user can't do that.

Headset trap. Headset users must always be able to pause media-- audio or video content-- by using the Pause/Play control on the headset. This applies to screen reader users and headset users.

So this is an example here where you've got some video playing at the bottom of a website on mobile. And a touch user can just activate the Mute button if they want to. However, a screen reader user can't pause the audio using headset hardware because the pause on the headset pauses the screen reader, not the content that's playing.

Exit trap. Ensure there is always an accessible actionable item-- e.g., a Close button that meets color contrast requirements and has an accessible name that closes any feature that overlays the current page, such as a full-page ad. And this applies to all users.

So this is an example of an exit trap. This is the YouTube app. And when you get to the YouTube app, you have the search at the top that's kind of in focus. And the rest of the content is greyed out. As a touch user, you can tap the greyed-out content and basically go to the home page of YouTube. But a screen reader user can't actually do that. And they're just stuck in the search box.

And this is another example of an exit trap. You've got an ad here for the HP EliteBook Folio which overlays the entire page. And there's actually no way to close it.

And this is another example, where you go to a website like *The Washington Post.* And there's ads at the bottom of the screen. And you can close those ads. But then you get a thing that says, ad closed by Google. And this overlaps the simplified view option on Android. So the way that you get to the simplified view of a page is by activating a little link at the bottom of the screen. But if there's an ad over that link, then you can't activate it.

And this is another example, *San Francisco Chronicle,* where it's got a fall sale. And you can save now, et cetera. And it does have a close button. But it doesn't meet color contrast requirements. And it doesn't meet touch target requirements, either. So yes, you can close it. But a number of users won't be able to.

And then we have a layer trap, which is, the user should not be trapped on a non-visible layer. This applies to all users. But it's mostly encountered by screen reader users.

So this is an example, dispatch.com. When you activate the hamburger menu, the hamburger

menu pops up. But the screen reader user is actually trapped on the layer underneath the hamburger menu. So the screen reader will continue reading the main page and not the navigation.

So those are the critical issues. And of course, there are other critical issues that you need to be testing for as well, such as movement that can't be stopped and audio that can't be stopped, et cetera. But the next group of issues are mobile-specific issues. And we've broken these into groups.

So the first group is alternatives. So alternatives are provided for non-web mobile functionality that is mandatory-- for example, recording a specific gesture by the camera before functionality appears.

So this is an example here where it says, copy this gesture, which is basically making the peace sign. And you take a photo to prove that you're a human being. There is an option if you can't do it. It has a little picture of a wheelchair and it says, need another way? However, if you activate that, you, just get gray text on a gray background that says "Verification query. Type your message." So the alternative is not descriptive or sufficient in any way.

Another one under alternatives is important functionality and information is available in the reader or simplified view. So this is part of mobile. So reader is what you see on iOS. Simplified is what you see on Chrome, on Android, or Chrome and so on the left, you have *The New York Times.*

And you've got a pop-up at the bottom that says, show simplified view. And on the right you see the simplified view, which is basically just the article. It doesn't have your hamburger menu or your header, your subscribe. It is just the article. So it makes it a lot easier to read.

Then some other ones we have are changes of state of nonstandard controls, e.g., Hamburger menu and star ratings are clearly indicated. Audio cues have an equivalent visual cue. And horizontal or vertical swiping supports touch gestures as a fallback.

So here is an example of touch gestures as a fallback. You've got a-- this is the State Library of New South Wales. And you've got a little Next option at the bottom after the information about the collection.

Now, if you activate that-- if it's a link, but it can also be activated by swipe-- actually, the next

page loads all at once. So it doesn't actually operate as if you're swiping. However, it allows for swiping or activating via a link through a screen reader or another assistive technology.

Horizontal or vertical dragging support touch gestures as a fallback. And toggle and slider elements support touch gestures as a fallback. And these are both about 2.5.1, Pointer Gestures.

And this is an example. So rate this book. And it's got five stars that are not completed. And it says, drag to change. And so in order to give a book four stars, you have to put your finger on the first star and then drag to the fourth star. You can't just hit the fourth star as a link and give it four stars. So that's a failure of 2.5.1.

Active swipe elements support both horizontal scrolling and swipe gestures. Actionable elements are triggered only on removal of touch. On Touch Start and On Key Down have not been used. And that's from 2.1, Pointer Cancellation.

So this is an example of On Touch Down. So basically, if you go to swipe the page and you hit your finger-- when it hits the mobile first hits South Australia, or SA, it will take you to that link. And this is a page that lists all the Coles stores in Australia. And there's about 3,000 or 4,000 Coles stores.

And so if you accidentally just want to swipe but you activate the link instead, then you go to the list of South Australian stores. And in order to swipe up to the top again, that takes 12 seconds. So that's quite a lot of time.

Display is our next category. Do not present new content on hover over actionable element. For example, do not have a top-level menu item that displays sub-items on hover but also when tapped opens a new page. And this is similar to 1.4.13, Content on Hover or Focus.

Size of touch targets is at least 44 by 44 CSS pixels, approximately 7 to 10 millimeters. And that is referring to 2.1 2.5.5, Target Size. Touch targets have sufficient inactive space between them. Inactive space of at least 10 pixels should be provided around active elements.

So this is an example that fails both target size and inactive space. You've got the Airbnb website. And because you're viewing it on a mobile phone, it gives you an option to install the app.

But if you don't want to install the app and you just want to close that little pop-up, you have to

hit the very small gray cross in the top-left-hand corner that is only about 4 pixels by 4 pixels. And there is no inactive space between it and actually installing the app. So that's a failure of both target size and inactive space.

This is an example of passing for touch targets. So you've got an Outlook message. And the to, the BCC and CC, the subject, and the message all have sufficient touch targets and also inactive space between them.

This is another example of failure of touch target and inactive space. So this is Asana. And the top-right-hand corner, you have two buttons, Edit and Mark Complete. And they don't have enough inactive space between them. And so it's very easy to accidentally activate Marked Complete instead of Edit.

And of course, once it's marked complete there's no way to undo it, which is another violation of WCAG 2. And so this is really a problem on mobile that we don't see on desktop because we don't have as much fine motor control on mobile as we do on desktop with using a mouse.

The next one is, horizontal scroll bars do not appear at all when the page is resized. And this is referring to 2.1 1.4.10, Reflow. And so reflow in WCAG 2.1 actually allows for tables to be excluded from reflow. And we don't agree. We had quite a discussion. And we don't agree that you should be-- that tables should be excluded because this is the kind of thing that you see.

So this is *Time* magazine. And it's got a table of how much things cost and the platforms and things like that. And half of it is cut off by the side of the screen. And there is no horizontal scroll bar.

But we don't mean something like this, which is a post at another newspaper where it's fit all on one screen. But each column-- and you've got 0 days, 14 days, 28 days, et cetera-- each column is one or two letters wide. So it's basically impossible to actually interpret.

Pinch zoom is operable unless an accessible font resizing feature has been included in the website that allows the user to increase the size of content at least two times the size of the standard font size. And this is similar to WCAG 2.1 1.4.4, Resize Text.

So this is an example of where resize text has failed. So this is the Virgin Australia app when you fly from Melbourne to LA. And the bottom-right-hand corner has the date, the local time at destination, at origin, et cetera. And I can read that now. But that's because I'm on a desktop.

But you can't actually read that if you're on a mobile phone. And I took the screenshot using an iPhone 6 Plus. And there was no way to actually increase the text size. And it didn't allow for pinch zoom. So basically, it was impossible for everybody to read.

Be careful if you have an app that inherits text size from the system. So this is the ABC app. And you've got two options here, one with the smallest text set in iOS and the other with the largest text set on iOS. And there's not a huge difference between them. One's probably 6 point, and one's probably 12 point.

But if you scroll up to the top of the article, even there they both look exactly the same-- the ones that have the system set to the smallest text or the one that has the system set to the largest text. So when you do inherit the system font size, you need to make sure you do so consistently across all issues.

The system can be used in portrait mode. And the system can be used in landscape mode. So this is WCAG 2.1, Success Criteria, and 1.3.4, Orientation. And so this is an example of a failure with Instagram. At portrait, it displays in portrait. In landscape, it displays in portrait. And this is a pass in Twitter. In portrait, it displays in portrait. And in landscape, it displays in landscape.

Now, one thing-- when testing this it's important that you have your device in the specific orientation, open the app or the site, and then close it entirely, change your phone's orientation, and then reopen the app. Because we don't expect the apps to inherit orientation once they're opened. And a lot of them rely on what the orientation of the device is when it's opened and won't check after that. So make sure you close your apps before you test this.

The next section is actionable items. So native UI controls, objects, alerts, and elements have been used. And this is probably the most important requirement in the mobile testing methodology.

Native UI elements, whether they be select boxes or submit buttons or anything like that-- they all have accessibility built in. If you're building some new, flashy widget, you have to add in a whole bunch of accessibility features that you basically get for free if you use HTML or the native elements. So if you take one thing from this toolkit, it's use the native elements that are available in iOS and Android or in HTML.

When direct input by the keyboard is not required, provide options for the user to achieve the

same result. This basically means that if you've got a finite number of options-- say you are listing the states in the US. There's 50 states. As far as I'm aware, there's 50 states. Correct me if I'm wrong.

There are only 50 options that people can choose. So give people a select box to choose them. Don't make them type in their state because the keyboard is more difficult to use. And there's more chance of error when you're allowing a free text box. And there's a finite number of options so you know exactly what they're going to choose.

This is an example of a non-standard UI. So basically what has happened is someone has coded it as a free text field. So the mobile has pulled up the on-screen keyboard. But it's actually a dropdown. And because it's a date, the year-- so it gives you 14, 15, 16, 17. And then the rest of it is overlaid by the on-screen keyboard.

Now, if you're talking about an expiration date for a credit card, once again, it's a finite number of options. So you should just use a select box. Because if you do that, then you'll get all the inherent accessibility features with that text box. And you know it will work with the accessibility features, the assistive technologies, and the mobile features inherent in mobile.

Functionality that can be operated by device motion or user motion can also be operated by user interface components. And responding to the motion can be disabled to prevent accidental actuation, except for certain situations. And that is very similar to 2.1, Success Criteria, 2.5.4, Motion Actuation. And infinite scrolling has not been used.

So this is an example of 2.5.4 where you shake your iPhone, and it gives you the option to undo typing. Now, this fails because 2.5.4 requires two things. One, it requires that you be able to turn off the feature. But it also requires that you have an alternative. So this undo typing on iOS can be turned off in Settings. But it doesn't have an alternative.

Compare it to this example, which is the other way around. This is Facebook on an Android phone. And if you shake it, it comes up with this thing that says, report a problem. Your feedback helps us improve Facebook.

And so it fails because, yes, you can report a problem. There is an alternative to do this if you can't shake the phone. But you can't turn off this feature. So if you have problems with shaking, you could activate this feature accidentally and not be able to turn it off.

Our next section is links. Link text should have a minimum color contrast ratio of 3 to 1 when

compared with the surrounding body text. And this is similar to success criterion 1.4.11.

Color alone should not be used to indicate links if not underlined. A secondary method, such as underlines, should be used in addition to color. And that's because when you're talking about desktop, you can use the mouse. So you can tap. And that's how you can find your links. The mouse changes display if you hover over a link. But there's no option like that on mobile.

And so here you've got text in a dark gray. And you've got the links in the red. And of course, black text on a white background and red text on a white background look the same to people who are red-blue colorblind. So people who are red-blue colorblind would not be able to identify which of these are the links.

Then we have navigational aids. Arrows and Next and Previous buttons have been used to indicate swipe or scroll areas. And that's the same as 2.5.1, Pointer Gestures. Navigational aids such as Back buttons, breadcrumbs, Next and Previous buttons are provided. and ARIA document landmarks are being used to appropriately describe document structure.

So we added that last one, although we did have quite a discussion about it, because of the rotor on iOS and how you can navigate through landmarks. So that's why that was included.

Audio and video. Basically there's only one additional requirement to WCAG 2. And that is that all video and audio have an accessible transcript. We felt that captions and audio descriptions were not sufficient when you're on a mobile device, that you needed to provide a transcript as well.

And forms. Field labels for all fields are positioned adjacent to the input field. And the following HTML5 input types are used appropriately-- EMAIL, TEL, DATE, DATETIME, MONTH, and SEARCH. And that's because these input types change the display of the on-screen keyboard. EMAIL will show the at symbol on the keyboard that pops up. TELL will give you the number keyboard. And SEARCH will actually give you the option to activate the search in the keyboard.

So this is an example where field labels are positioned incorrectly. You've got Yes, followed by a selected radio button, then No. And the selected radio button is actually a little bit closer to No than to Yes. And you can't really tell which it refers to because the No radio button doesn't meet color contrast requirements. So it's really unclear as to what that is.

The user should be able to dismiss on-screen keyboards that display when the user focuses on an input field. Additionally, any pop-up dialog or warning must be able to be dismissed. So this is similar to 1.4.13, Content on Hover or Focus.

And then the next section is mobile assistive technology and feature support. So there are two requirements here. It's important that all actionable items can be accessed and activated by the following assistive technologies or when the following feature is enabled. And all important content can be accessed by the following assistive technologies or when the following feature is enabled.

And those technologies and features are voiceover on iOS, keyboard on iOS, keyboard and switch on iOS, zoom on iOS, invert colors on iOS, grayscale on iOS, and reader view on iOS. On Android, we're talking about TalkBack, keyboard, keyboard and switch, magnification, invert colors, grayscale, increase text size, and simplified view, which is in Chrome.

And then the last section is testing mobile and desktop relationship issues. This is, of course, only applicable to the mobile site. There are three requirements. Item labeling across mobile and main site is consistent. It's important that there is consistency across the mobile and main site.

Links between mobile and the full version of the website have been provided. And we know that that's not necessarily possible with a responsive website. So we're going to add some additional information about that in the methodology. And users are not restricted to a particular version dependent on device. Users should be able to use the mobile version on the desktop and the desktop version on the mobile, et cetera.

So this is an example of failure of consistency. So this is Asana. And on the desktop, you have My Tasks, Inbox, Dashboard, and then a list of projects. And I would operate off My Tasks. If anything wasn't assigned to me, then I wouldn't do it. I didn't look at the projects all that often. But if you look at the mobile app, all it has is the projects. It doesn't have My Tasks, Inbox, or Dashboard. So that's a failure of consistency.

This is another example. This is Row New York City. And so on the desktop, you've got your menu. But it also has a section that says, how would you create your New York City? Start with Times Square, the heart of Manhattan 24/7, et cetera.

And basically, if you go through that, it will actually allow you to put together a timetable. You

got two days. These are the kind of things that you see. However, if you look at it on a mobile, all you see is the menu, the navigation. You can't actually do that on the mobile device.

So that is the mobile site toolkit with a little bit of information about the native apps. We will be releasing the native app guidelines just before Thanksgiving. And I would like to thank our committees. We had two committees, one for mobile site and one for native app.

That included Brent Davis. Corbb O'Connor. Myself. Thanks to me. Jennifer Chadwick, who is co-chair of the native app. Karen Herr. Kathryn Weber-Hottleman. Kathy Eng. Laura Renfro. Megha Rajopadhye. Michael Keane. Morgan Lee Kestner. Peter McNally, who was co-chair of the mobile site committee. Rafal Charlampowicz. Ryan Pugh. Shane Anderson. Steve Sawczyn. Sunish Gupta. And Tom Lawton.

So would you like to join the committee? We are actually going to have three committees next year, one on mobile sites, one on native app, although we might merge those. And then we're actually going to write some guidelines on wearables as well.

So if you would like to join the committee or you'd just like to put forward your opinion, please go to www.surveymonkey.com/r/ictcommittees. And that link is also on the present link as well. So we'd love to know what you think of the methodology. Is there anything missing? And those slides, again, are on pz.tt/3playmobile.

And just before we get into questions, I just wanted to mention that there are some webinars that we're doing, that AccessibilityOz is doing. We've got the determining accessibility of a product in under two hours, which is next Tuesday. And in an hour and 15 minutes, I actually am doing a CCC webinar on social media and accessibility.

We're also doing workshops in San Francisco. So there's four workshops that we're doing in San Francisco-- Managing Accessibility in the Web Development Lifecycle. Mobile Accessibility Testing, where we go into this in a whole lot more detail. Accessibility in Design. And WCAG 2 for Developers and Content Authors.

And the last thing I wanted to show you was OzWiki. As an attendee of one of my presentations, you're welcome to have access to OzWiki. And so I'll get 3play to talk to you. And so OzWiki gives you all the errors that you could ever think of in terms of accessibility errors and an example and a solution.

So this is an error-- image title contains additional information that's not elsewhere on the

page. You have the WCAG 2 information and the technique. And you have some incorrect examples and some solutions. And you can also ask questions about the issue. So now I'll hand back to 3Play. And thank you very much for having me. And we can get started with questions.

SOFIA LEIVA: Thanks so much, Gian. If everyone-- you could keep asking your questions by typing them into the chat window or the Q&A, and we'll try to get to as many as possible. So the first question that we have is, with the responsive sites on mobile, test with different devices that have different screen sizes so that you are testing the actual CSS breakpoints?

GIAN WILD: Yes. So there's a couple of things that you can do there. The first one is, you can actually use the Chrome DevTools to specify the different size window. The methodology goes into it in a little bit more detail on how to find those breakpoints and the fact that you have to test them all. Does that answer the question?

SOFIA LEIVA: I believe so. But you can ask a follow-up question if you have more as well. Are there individuals with disabilities who were part of the testing that you did?

GIAN WILD: Hello?

SOFIA LEIVA: Hello?

GIAN WILD: Oh, sorry. I don't know what happened. Were there individuals or people-- yes, absolutely. There were people with different disabilities on both committees-- screen reader users, so people who are visually impaired, low-vision users, switch users, keyboard users. So yes. We did want to make sure that we included as many people with disabilities who had actual experience using the assistive technologies as possible.

SOFIA LEIVA: Thank you. The next question we have is, how do you test game apps? And what needs to be tested in these?

GIAN WILD: Gosh, that's a good question. So we haven't looked at game apps. I think there is some-- I think that there was a paper, actually, at the ICT Accessibility Testing Symposium about that. But basically, I would suggest that you look at it the way that we would look at native apps-- so the functionality, et cetera. And then I would be testing with the assistive technologies that we mentioned and the mobile features that we mentioned.

SOFIA LEIVA: Thank you. The next question we have is, what are the largest or most significant barriers to

testing on desktop via browser simulations with a touchscreen monitor?

GIAN WILD: So basically the problem with using-- I think the question is, what's the problem with simulators? And so I'll answer that question. The problem with simulators is that it doesn't actually simulate the device very well. So you're looking at a screenshot. Or you're looking at the device on a much larger screen, much larger resolution. You don't have any issues like screen glare or things like that.

So simulators can be problematic. And we see there are many, many, many issues that I've come across that are basically because someone used a simulator and never actually tested on an actual device. And it might not even just be the screen size or anything like that.

We've even found M dot sites that link to Flash catalogs, which they've obviously worked very hard to do. But of course, Flash has never been supported on iOS. So those kind of things are the problems with using simulators.

SOFIA LEIVA: Thank you. The next question we have is, can you explain some of the keyboard traps a bit more and how to avoid these?

GIAN WILD: Yes. So there are two types of keyboard trap. They occur on both mobile and desktop. The first is where you're actually trapped within a component. And a very well-known one is, Firefox video players generally were keyboard traps in Firefox probably about five years ago. So basically you could tab into the video player and operate the video player. But then you couldn't tab out. You couldn't escape from the actual video player.

The other one that we've found is where you're actually just-- you are trapped basically on a layer underneath the component. So we've found this in the video player testing we've done recently, where basically you make the video player full size or full screen. And the keyboard actually remains on the page underneath. So you can't actually activate the video or play the video or anything like that or even close the full screen because your keyboard is just tabbing on the page underneath.

The other one that we found is where you basically open a tab, and a component opens, like the Add This, Share feature. So it's kind of like a pop-up. But you can't close that pop-up with the keyboard. So it overlaps on the content.

SOFIA LEIVA: Thank you. The next question we have is, what are the benefits of joining the committee?

GIAN WILD: Well, there are benefits to the committee, and there are benefits to you. So the benefits to the committee is that basically, if you have been doing a whole lot of mobile site testing or native app testing, then we want to hear from you. And we're meeting every two weeks for two hours. So there's probably, I'd say, half a day's work every two weeks if you join the committee.

But if you can't meet those kind of requirements if you don't have that kind of time-- and a lot of us don't have that kind of time-- it still would be great for you to reach out to us. Because if you have very specific knowledge-- say with a specific assistive technology or something like that-- then we could maybe call on you to advise us when we get to that point. So we definitely want people who know what they're doing.

And also, we are desperately looking for native app developers because we don't-- we've got a couple on the committee. But we don't have a lot. And so we definitely need more expertise around what can and can't be done in native apps. So that's the benefit to the committee.

The benefit to you, and the benefit that I found, is that you're working with some really, really intelligent people. And you're learning stuff that-- there's just really no other way to find out this stuff of how requirements are utilized across different devices and things like that. And it gives you an opportunity to see the testing methodology before it gets released and see how feasible it is.

So one of the things that we've been talking about for a while is the whole testing each variation of a page. If you test each variation of a page on every device with each assistive technology, then you're talking about quite a lot of work. So we're going to be talking next year about how we can identify exactly what it is that needs to be tested on each variation. And that can hopefully inform your testing and make it easier for your own testing.

SOFIA LEIVA: Thank you so much. The next question we have is, what is the recommended browser to test using mobile devices?

GIAN WILD: So on iOS, I would say Safari. And on Android, I would say Chrome.

SOFIA LEIVA: Thank you. The next question we have is, for those building websites that are going to be viewed on desktop and mobile, what type of testing can be done to make sure it's accessible on both?

GIAN WILD: There is-- I would say, make sure it's mobile first. So the biggest thing you can do is make

sure your functionality and content is available on all variations. But I would say when it comes to actually testing, whether you're doing waterfall or agile, make sure you test your mobile designs and templates, as well as your desktop.

There is a presentation on our website, on the AccessibilityOz website. It's under Resources, Webinars on prioritizing accessibility in the building of a website. And that might be something worth looking at in terms of what you should test when you're actually building a website.

But I'd also say, send this information to the developers, the project managers. And make them aware that they need to be involving people with disabilities when they can and really thinking about accessibility at the main stages, which is functional specifications, design template. And perhaps some training needs to be run on what they need to do.

SOFIA LEIVA: Thank you. We have a couple more questions. In discussions, we pitch WCAG as being technology agnostic. Yet you suggested device-specific testing. How do we explain that to clients when we audit per WCAG?

GIAN WILD: That's a great question, Peter. In my experience, clients don't necessarily understand that WCAG is technology agnostic. So I've never actually had to explain it. I've basically said, this is what we do. And they go, OK, excellent. You're the experts. You should know what you're doing.

However, if you do have a client that says, hey, WCAG is technology agnostic, I would say, yes, but-- and I spent six years with the W3C contributing to WCAG 2. We did try to make it technology agnostic. It was released, and it was basically finished around 2006, 2007. The iPhone had only just been released. I think it was released in 2007.

So we really had no idea how people were going to serve content the way that they do now, especially on a device as small, say, as a wearable. So yes, it is true that WCAG is technology agnostic. But WCAG is also 11 years old. And yes, 2.1 is great. But it really doesn't go far enough.

So I would say-- and this is the reason why we developed this methodology-- is that it doesn't do enough and that hopefully WCAG Silver or 2.2 will address these issues. But until then, this is kind of like the industry standard that you recommend to ensure that your content is actually available to all people on mobile devices.

SOFIA LEIVA: Great. Thank you so much. Gian, thank you so much for joining us today and talking about

mobile accessibility. And I hope everyone has a great rest of their day.