palinQA meetup

BUDAPEST

@palinQA

A11y & Agility For QA

Improving Workflows & Making Better Software for *EVERYONE*

Brief Introduction

I work doing a11y QA for Instructure. where we create Learning Management Software for the Education environment and Career Development Software for the Corporate environment. This means that both legally and ethically, we have a commitment to our products' accessibility.

What is Accessibility (a11y)?

Accessibility (a11y) is a measure of how accessible a computer system is to all people, including those with disabilities or impairments. It concerns both software and hardware and how they are configured in order to enable a disabled or impaired person to use that computer system successfully.

Accessibility is also known as assistive technology.

Who here uses an assistive technology on a daily basis?

Glasses are an assistive technology –

Henry Bemis faces the challenges of a11y & technology in *The Twilight Zone*.

Imagine This:

If any parts of the product that you built failed a11y – *meant it was **ALSO** incompatible with glasses*

(only allowed the user to see things with their natural, unmodified vision).

How confident are you that what you're releasing is something that *everyone* can easily utilize?

This next slide is only to be viewed without glasses.

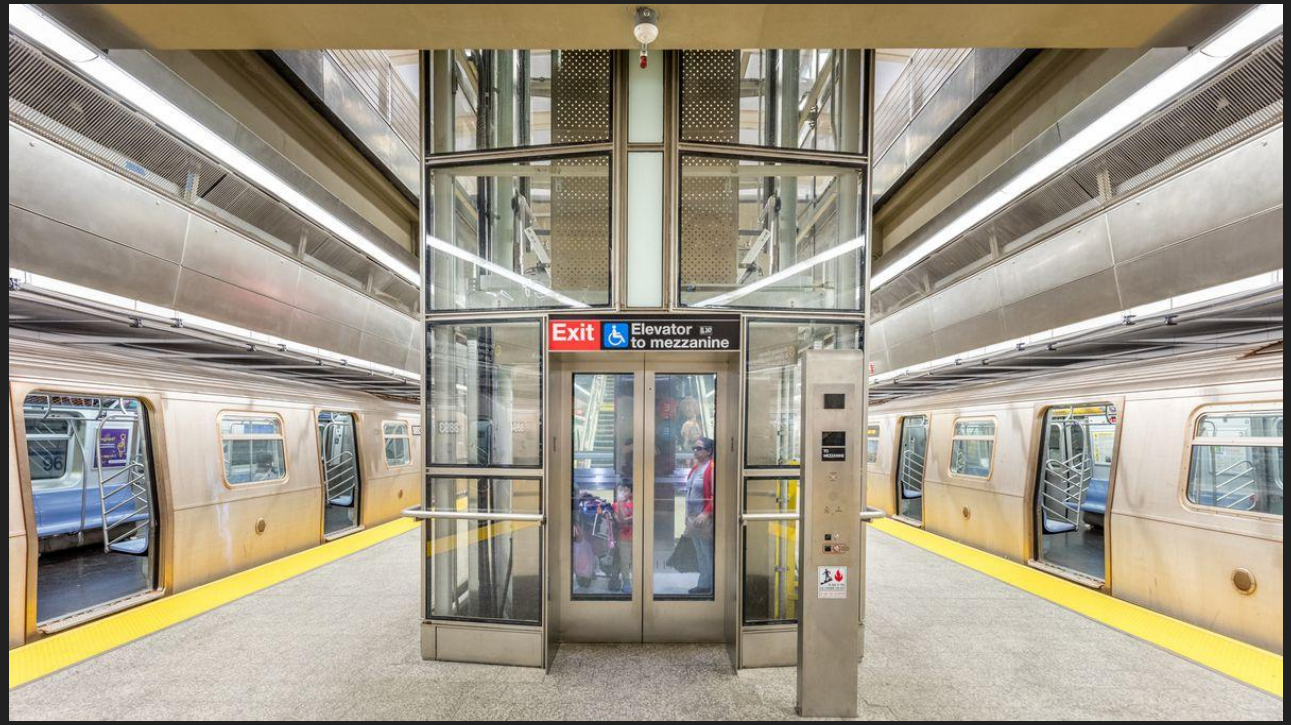

THAT'S NOT FAIR.

I think that everyone pretty much got the same message from that slide with or without glasses.

Regardless, hopefully it solidified my point.

(We'll wait until everyone's glasses are back on before continuing)

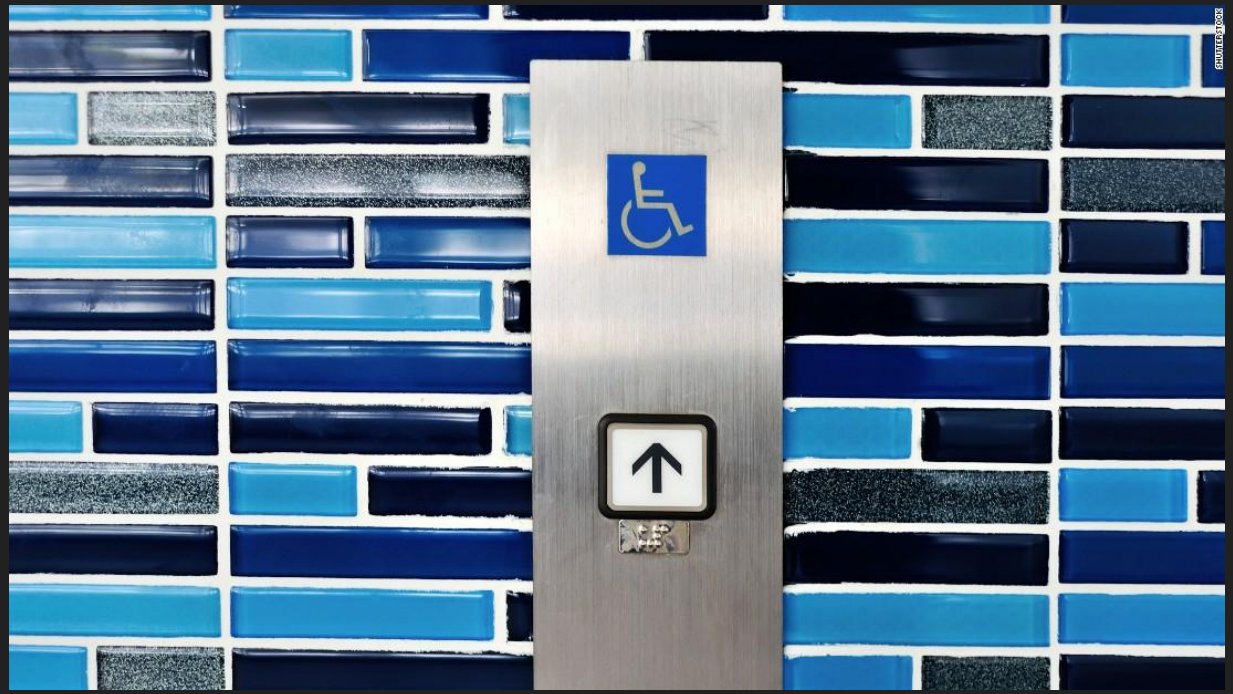
The Elevator Pitch



To help cover the initial points, we're going to focus on a recurring theme: **Elevators.**

Requirements in making accessible buildings can mirror the challenges in making software accessible (*and this makes it easier to remember the individual parts that they correlate to*).

A11y Rules & Guidelines



WCAG 2.1

The W3C (World Wide Web Consortium) is an international community that develop Web standards. The W3C, in cooperation with individuals and organizations around the world, they developed the

WCAG 2.1 (Web Content Accessibility Guidelines).

The **WCAG 2.1** is a standard with 3 levels of conformance (A, AA, & AAA), which is designed to meet the needs of individuals, organizations, and governments internationally. It's a set of four principles (with several sub points) used to determine the accessibility of a site, using the acronym: **P.O.U.R.**

- **P**erceivable
- **O**perable
- **U**nderstandable
- **R**obust

WCAG 2.1: P.O.U.R.

Perceivable: Information and user interface components must be presentable to users in ways they can perceive. – *This means that users must be able to perceive the information being presented (it can't be invisible to all of their senses)*

Operable: User interface components and navigation must be operable. – *This means that users must be able to operate the interface (the interface cannot require interaction that a user cannot perform)*

Understandable: Information and the operation of user interface must be understandable. – *This means that users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding)*

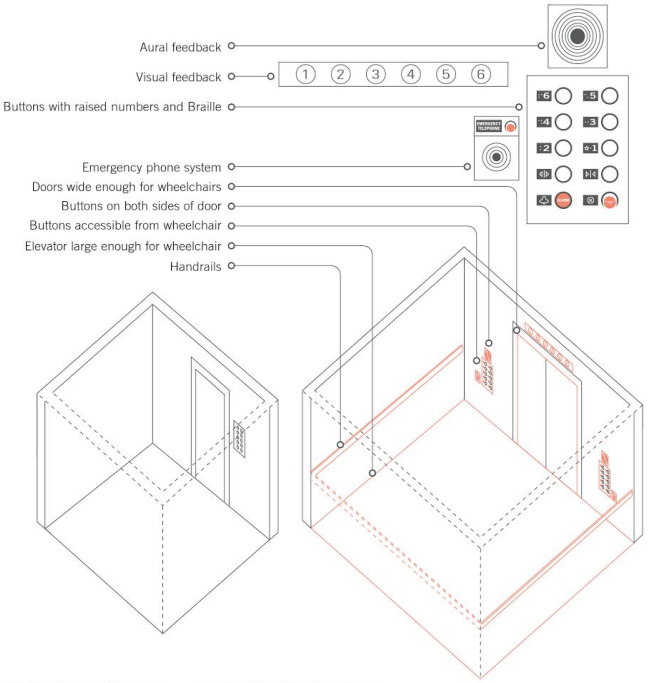
Robust: Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies. – *This means that users must be able to access the content as technologies advance (as technologies and user agents evolve, the content should remain accessible)*

EN 301 549

In 2005, there was a request from the European Commission (Mandate 376) to update, specify, and standardize the functional accessibility requirements for publicly procured ICT (Information & Communications Technology) products and services across the European Union.

The new European Standard for a11y requirements is detailed in: **EN 301 549**, also known as, **“The Web and Mobile Accessibility Directive”**. This was developed by an international team of experts, with the participation of the ICT industry, as well as organizations representing consumers, people with disabilities, and the elderly. – Like most standardizations, **EN 301 549** was built primarily on ensuring conformance to the updated WCAG 2.1 for public authorities and other public sector bodies, to ensure that websites, software, digital devices are more accessible.

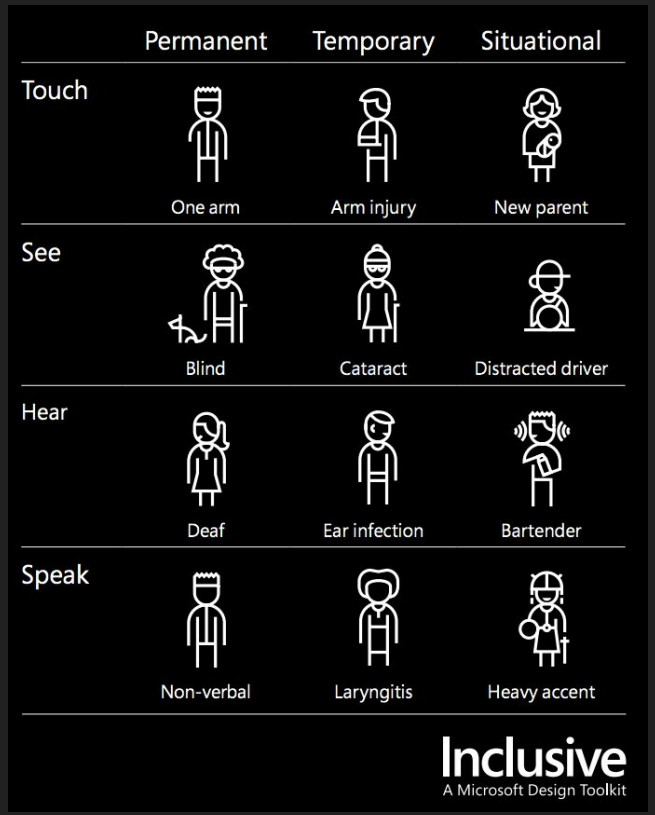
Universal Principles of Design



The large elevator has many features that make it more accessible than the small elevator: wide doors permit easy access; handrails help people maintain a standing position; two sets of controls are easily accessible from a seated position; controls are redundantly coded with numbers, icons, and Braille; feedback is provided visually and aurally; and an emergency phone system offers access to special assistance.

Page 17

Who are Accessibility Users:



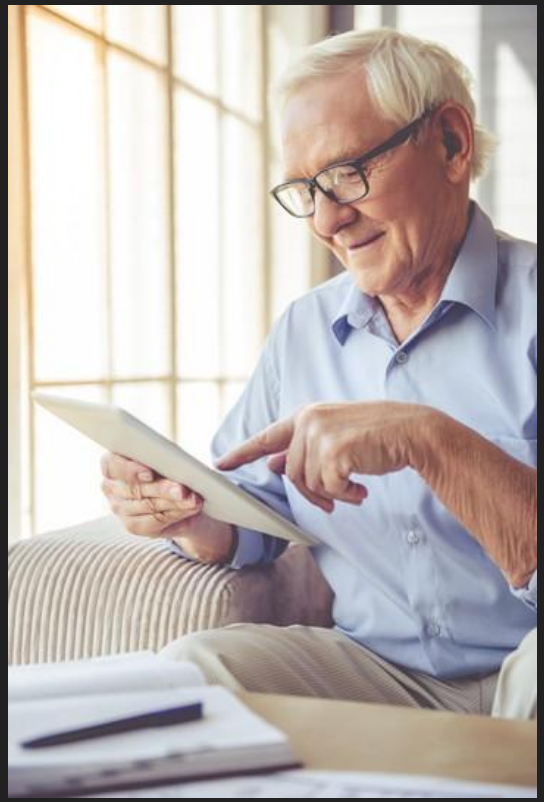
Accessibility Users come in many forms, and some a11y users may need to learn about their technologies suddenly, or only use them some of the time.

This is why having workflows remain consistent for all users is important in inclusive design.

Mobility Disabilities

Anyone else remember these days?

Mobility Disabilities



Mobility is a disability that will also impact more and more users as the population ages.

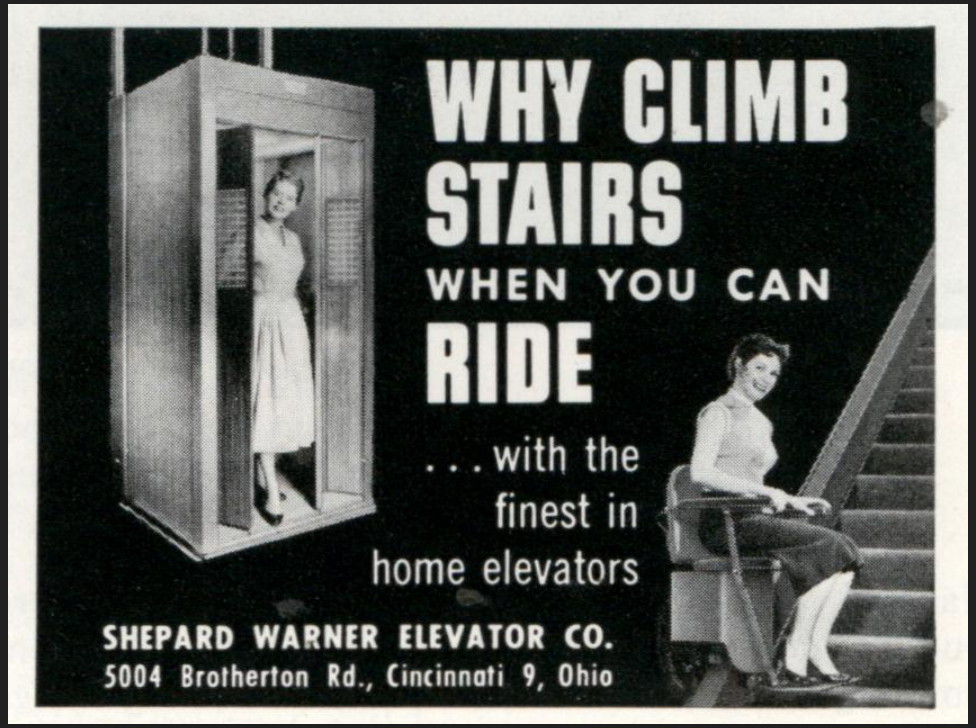
Things like Parkinson’s can make using a mouse or precise touch navigation anywhere from difficult to outright impossible.

Additionally: Stress exacerbates dyskinesia symptoms, so even having poorly implemented mouse or touch-based controls can create additional difficulties for users.

Mobility Disabilities

Tools like sip & puff devices, switches, as well as eye-trackers utilize keyboard or gesture navigation controls, so by supporting that, you're passively supporting *all of those different devices*.

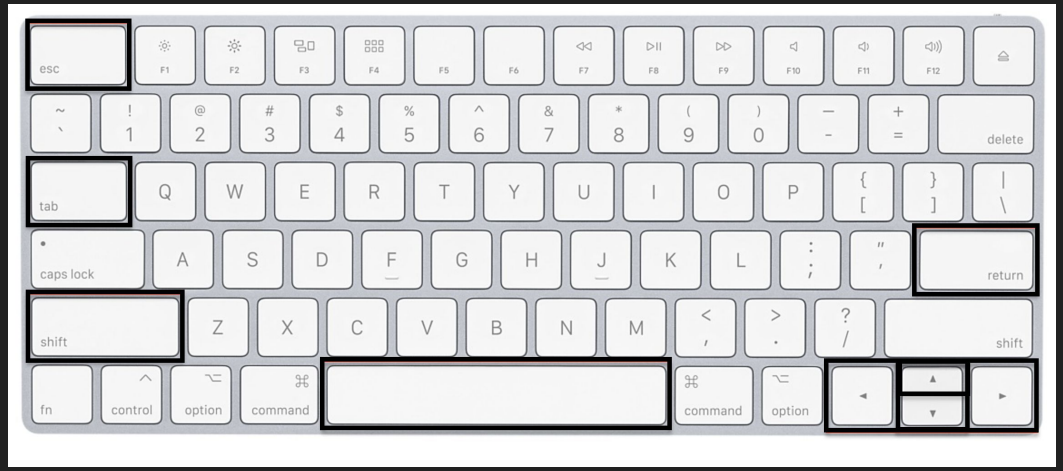
Mobility Disabilities

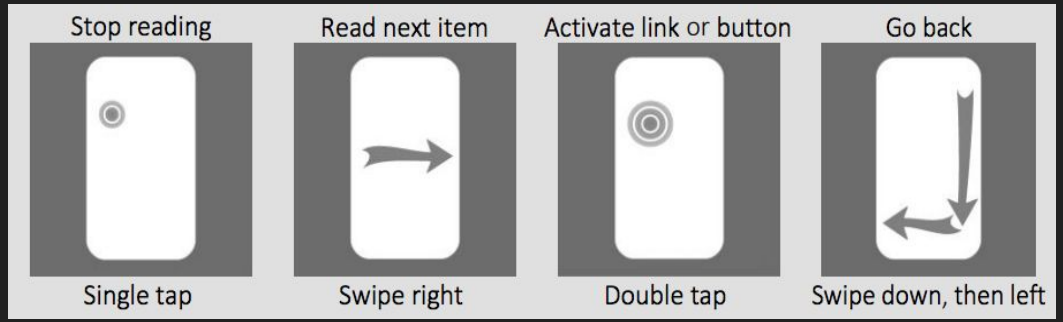


Conveniences for some are necessities for others.

Well-implemented Shortcuts, Keyboard / Gesture Controls are a convenience for many users, but they are first and foremost a necessity for accessibility users.

Basic Device Controls





Basic Device Controls

Elevator Operation

- Call Button
- Door Open
- Door Close
- Individual Floor
- Emergency Stop
- Phone

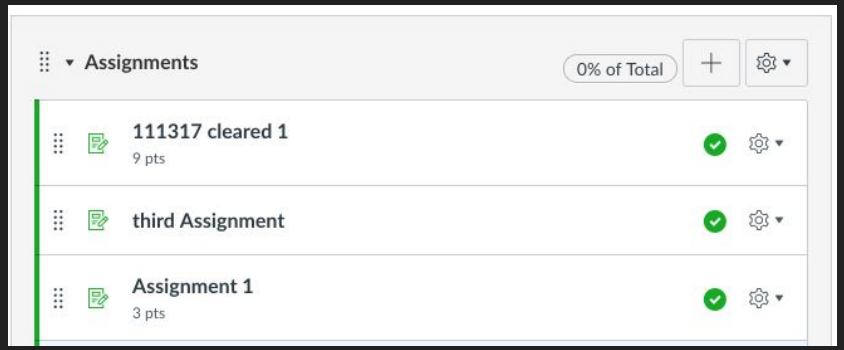
Keyboard Navigation

- Enter
- Space
- Tab
- Arrow Keys
- Escape
- Shift

Considering these basic controls and understanding that *they have to be prioritized for accessibility users* ensures that all users can get where they need to be, and that the controls also provide convenience for others.

Focus, Order, & Shortcuts

Ensure the navigation is visible, the order makes sense, & is streamlined for the most frequent use cases.

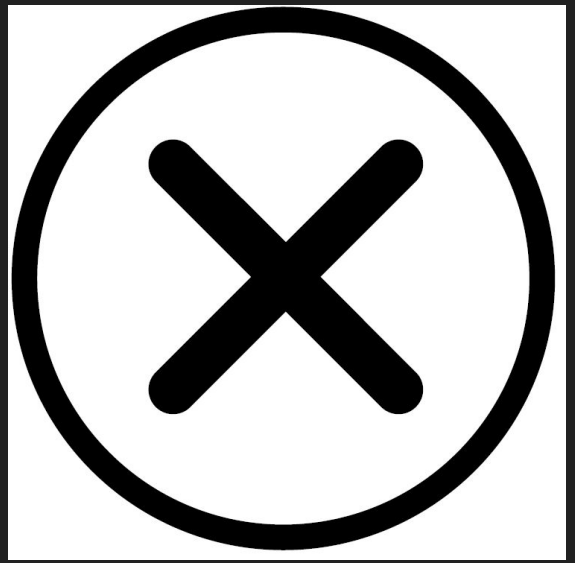


Matching Workflows

Ensuring that your standard & a11y workflows match closely for **both** accessibility requirements & also general shortcut convenience ultimately refines a better workflow that is *also consistent for **all users***.

The best designs work the same regardless of whether you're using only one or a combination of both.

Also: *Consistent* Workflows

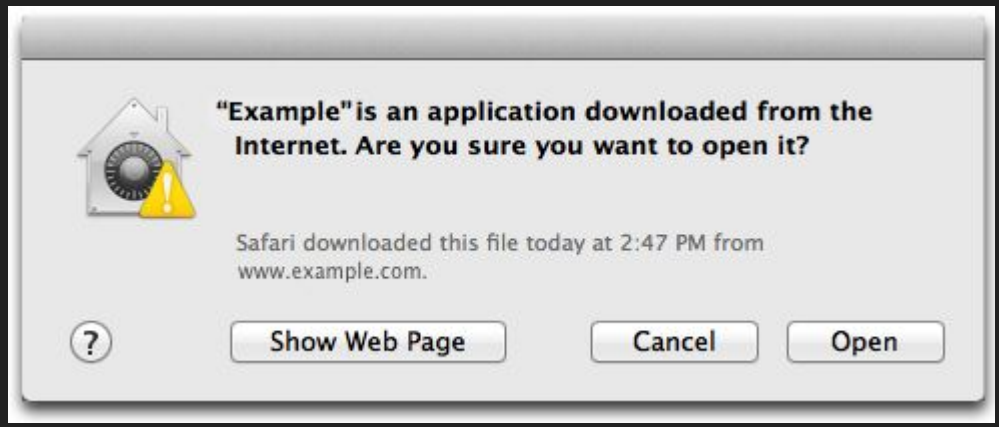


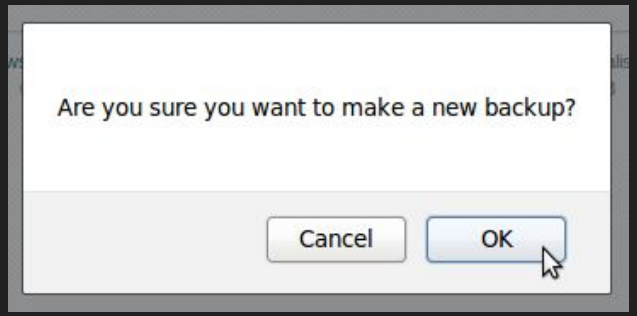
“X” buttons can be many things:
Cancel, Close, Remove, Archive, Delete

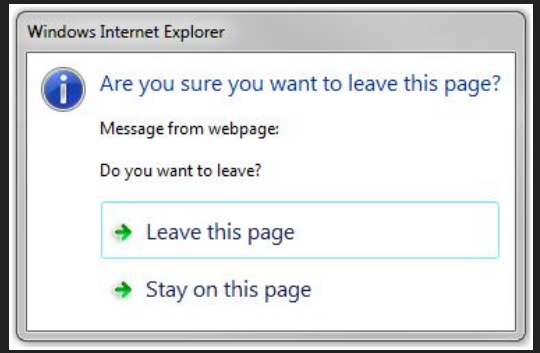
Make sure that it’s clear which thing *your* controls do, and also that buttons and patterns are internally consistent in their placement and usage.

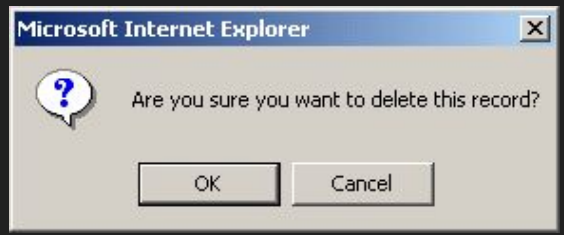
Inconsistencies cause issues for...

Cognitive Disabilities







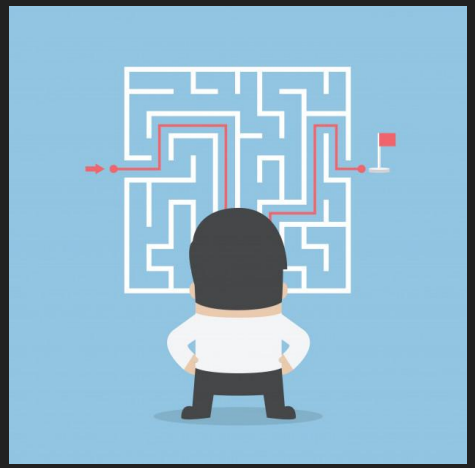


Cognitive Disabilities

Most programs have some universal & some custom iconography. Ensuring there are ways to check the labels of those icon controls before just pressing them to see what they do is very important.

This allows both users who have someone assisting them, and users who make use of speech-to-text software to be able to clearly indicate which control they need to use.

Cognitive Disabilities



Checking that workflows aren’t confusing ensures that your users don’t get distracted or just give up.

Important: Stress Case Testing

The “typical” user is likely somewhat stressed. Ensuring that workflows are clear in *stressful* scenarios helps to consider issues that can cause a11y users issues in completing tasks, or just giving up altogether.

Cognitive Disabilities

“Go to the 8th Floor, the Red Level.”

This provides multiple clues for users to get to the correct location quickly & efficiently.

However, this also brings up an important point about **colour usage** – *The information that it provides a user is:*

always supplemental never primary.

Visual Disabilities

A colourblind user given that information could still read the correct floor, however…

Once they select their floor, they're faced with the following two buttons:

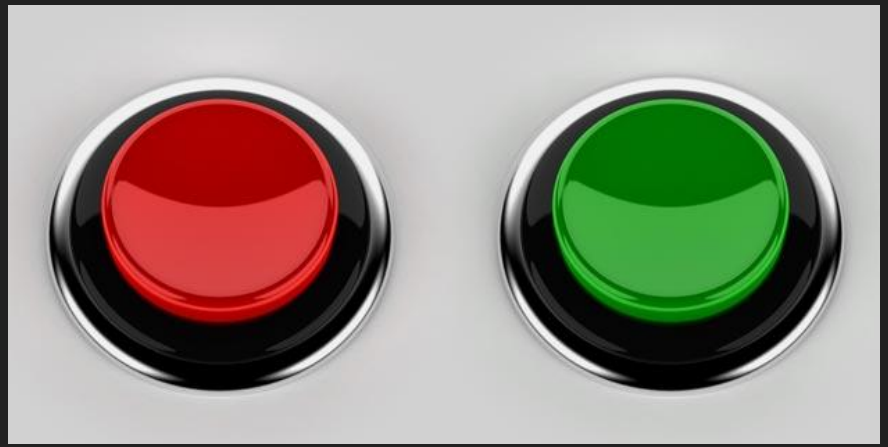
Visual Disabilities

Press the red button to light the elevator on fire.
Press the green button to go up a floor.

Visual Disabilities

So… did you go up a floor?

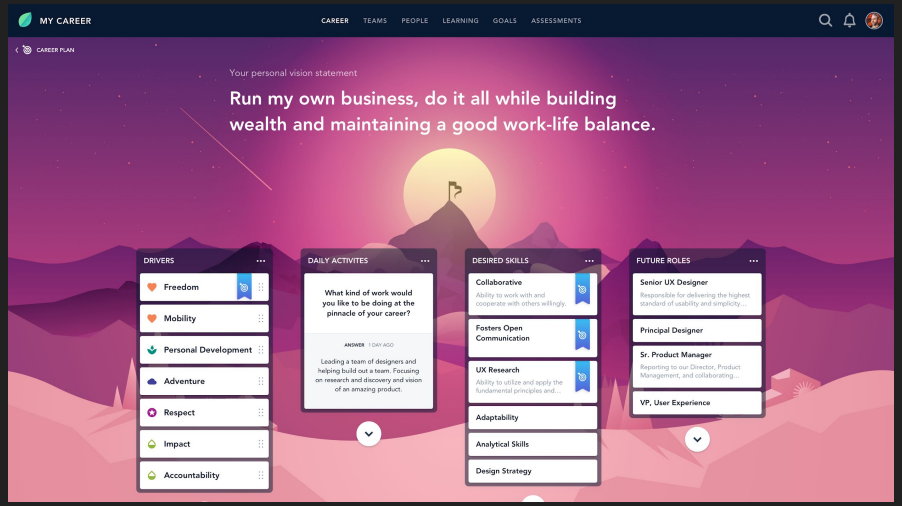
Hopefully that's an easy reminder for no colour-only indication
– *but there are more colour considerations to check!*

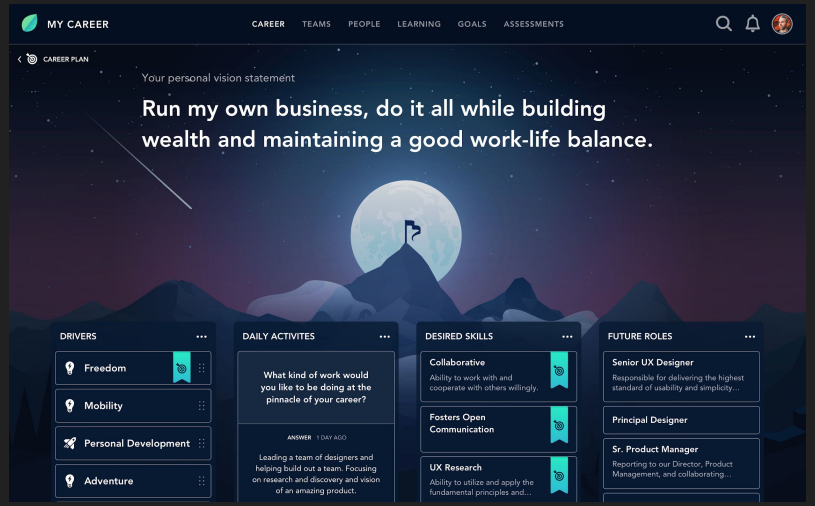


Red Green

Visual Disabilities

Colour Contrast Requirements (& Dark Modes)





Adequate contrast is a requirement. Having a dark mode to help achieve that, or other adjustable settings can be very helpful to users with different visual disabilities.

Visual Disabilities

Charcoal #404040	Text AAA 10	Text AAA 9.2	Text AAA 7.6	Text AA 6.4
Black #000000	Text AAA 21	Text AAA 18.7	Text AAA 15.4	Text AAA 13
Effective on Extremes #2F78C5	Text AA 4.5	Text AA18 4	Text AA18 3.3	Text DNP 2.8
Effective on Lights #0F60B6	Text AA 6.2	Text AA 5.5	Text AA 4.5	Text AA18 3.8

Colour Contrast Levels
Range from **1:1** to **21:1**

Contrast Guidelines:

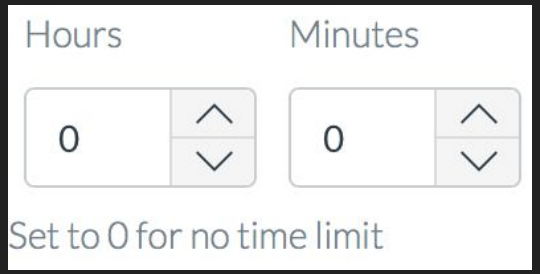
- **3:1** Large Text
- **4.5:1** Standard Text
- **7:1** (Extra Good)

These contrast values are set so that even after anti-aliasing and display inconsistencies that the content is still visible.

Visual Disabilities

Contrast < 4.5:1

Blue on Grey
1.41:1



Thin Grey on White
4:1

Contrast > 4.5:1

Dark Green on White
5.52:1

Blue on Dark Background
7.06:1

Visual Disabilities

The Text Below this line has a contrast of 1:1

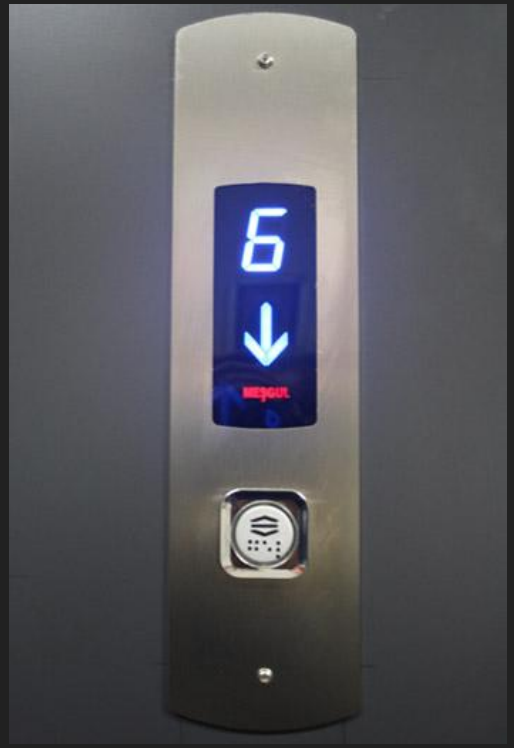
Visual Disabilities

Users who are anywhere from legally to fully blind don't perceive most-to-any visual information, but it still exists as text that can be read by a Screen Reader.

This translates all of the on-screen controls from visual into auditory information, which means that Screen Readers can *also* assist users with dyslexia or other literacy issues who may be fully sighted – so it's important that descriptors still match what's being displayed visually.

Screen Readers need to relay information clearly:

Visual Disabilities



A blind person & three other people get into an elevator on the 6th floor. The blind person presses to go to the 3rd floor & two of the other people also press buttons before the doors close and the elevator moves down.

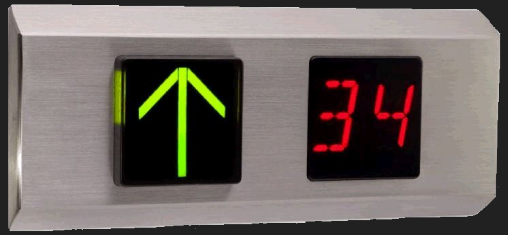
The elevator dings & the doors open.

– *Should the blind person exit the elevator?*

Auditory Disabilities

There is a lot of overlap between audio-only & visual-only information in videos, and ways to provide that to users:

- **Subtitles:** Synced text of dialogue
- **Closed Captions:** Synced text of dialogue & other sounds
- **Audio Descriptions:** Synced audio descriptions of visuals
- **Transcriptions:** Separate text of any of the above



Ensuring that there are equivalents of the information that can be conveyed both visually and by audio is important. Understanding both ways we consume that helps identify any gaps.

Even *if* all these a11y examples don't make up a majority demographic of your users –

They're NOT "edge cases"

THEY'RE PEOPLE.

A11y's Biggest Enemy

"I mean, how many people really need these options anyway? I'm sure that we can release this version and try to get that done afterwards & tell anyone who asks that we're working on it."

This mentality is ***not only*** consistently detrimental to making accessibility core value changes stick throughout your company, **but is constantly one of the biggest hurdles when trying to get these things done – because it creates a *mentality* & *workflow* of *de-prioritizing a11y to an afterthought*, and *treating it like POLISH – rather than a core function*.**

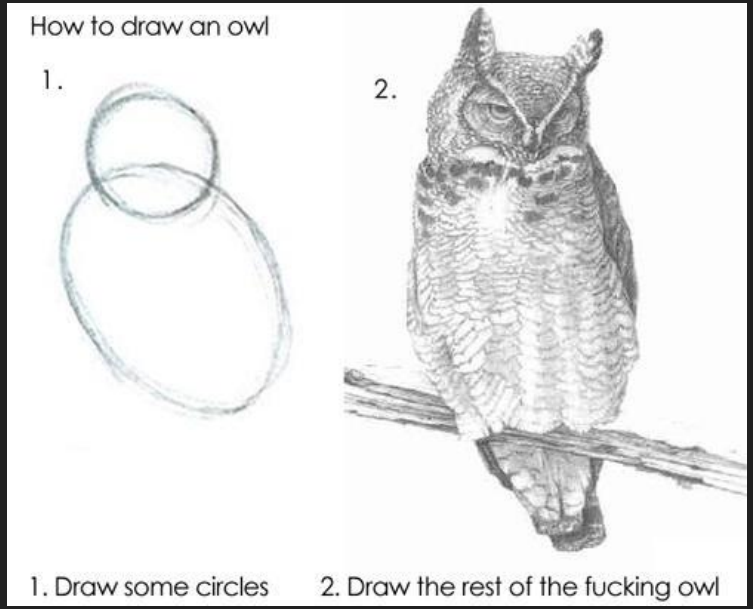
So! How do you QA all this?

And not just that – how do you QA for all of this, while also ensuring that you release things, and that you're not going to bottleneck the entire development process with tons of very specific and meticulous verification?

Both Avoiding This Feeling:

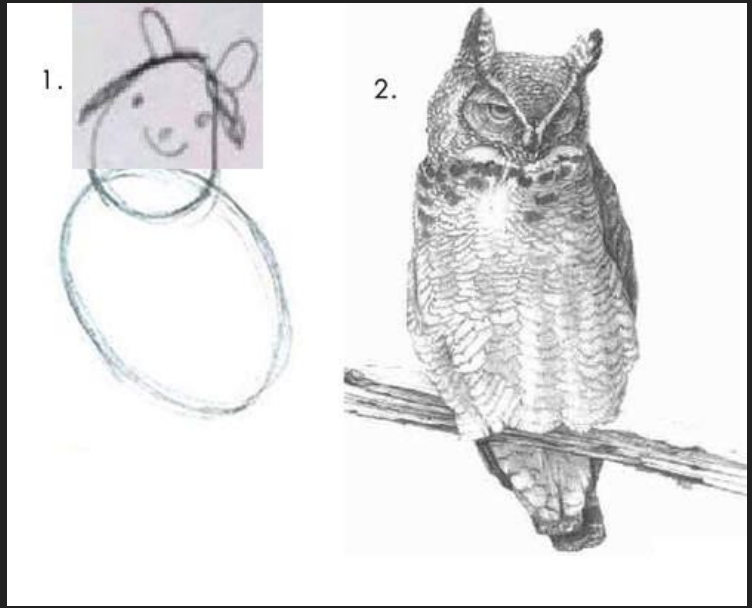
(How do you start to get this “MVP” to a “V2”?)

And Avoiding This Feeling:



(I *am* going to summarize, but hopefully not *too much*)

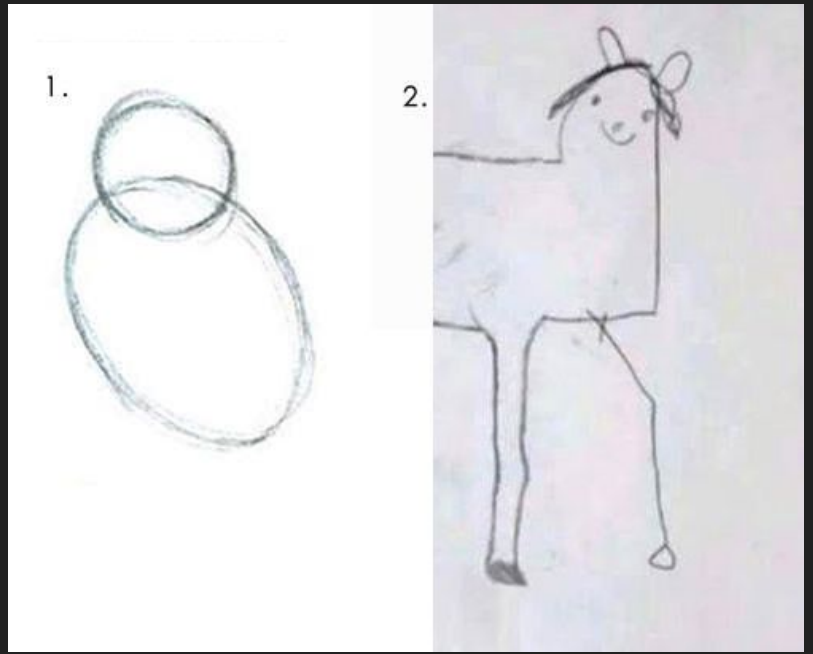
How To Do Agile A11y:



(Getting from what we ***don't want*** to what we ***do want***)

How To Do Agile A11y:

What to know *BEFORE* you start testing:



Things to Remember:

After the Fact Change = Hard
Last Minute Change = Stressful

Only a11y QA testing just before release generates stress and frustration around a11y for the new thing that you should be proud of – *which is the opposite of what **anyone** wants.*

What kind of process is left?

What kind of process is left?

A Good Warning Sign

Due to the fact that a11y is most heavily tested at the QA stage, **a11y pain functions as a litmus test for where gaps in the *overall development process* exist.**

The gaps in a11y around understanding, ownership, planning, & accountability can help point to deeper issues felt (to varying degrees) interdepartmentally for **a wide range of related issues that all have *the same core unhealthy patterns.***

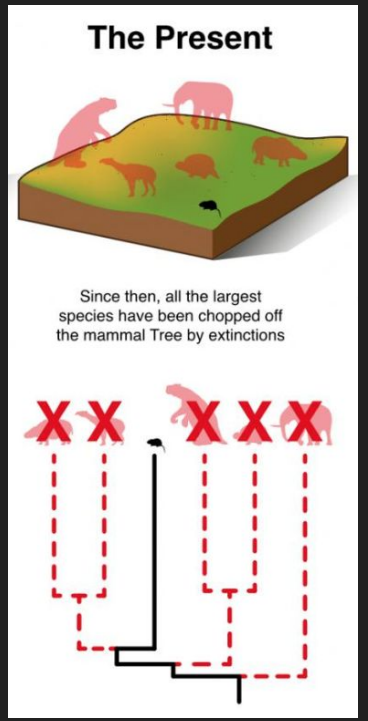
Iterative Process Patterns

How well does a Products' a11y happen?

- Well-managed patterns are *self-sustaining*
- Poorly managed patterns are *self-defeating*

Normally, this means that any type of iterative processes that work well survive, whereas ones that don't work gradually get eliminated as better & more sustainable paths arise…

...However –



Iterative Process Patterns

How well does a Products' a11y happen?

Sometimes, it's preserving a pattern that needs a lot of help, and has a lot of self-defeating qualities.

But... just leaving the self-defeating elements the way they are *is not a long-term solution to the problem.*

It needs to *become* self-sustaining, Instead of being *kept* self-defeating.

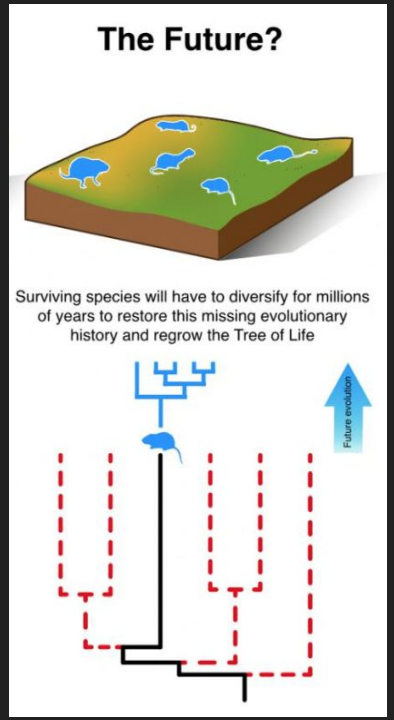
Iterative Process Patterns

How well does a Products' a11y happen?

QA can help to *actively guide the development* of what's being made *BEFORE* warning pain becomes something completely broken and unsustainable.

To start: Let's look at this general iterative perspective to discern the *core* of what being agile actually even *means* in this process.

Then, we'll cover the path to achieving it for a11y.

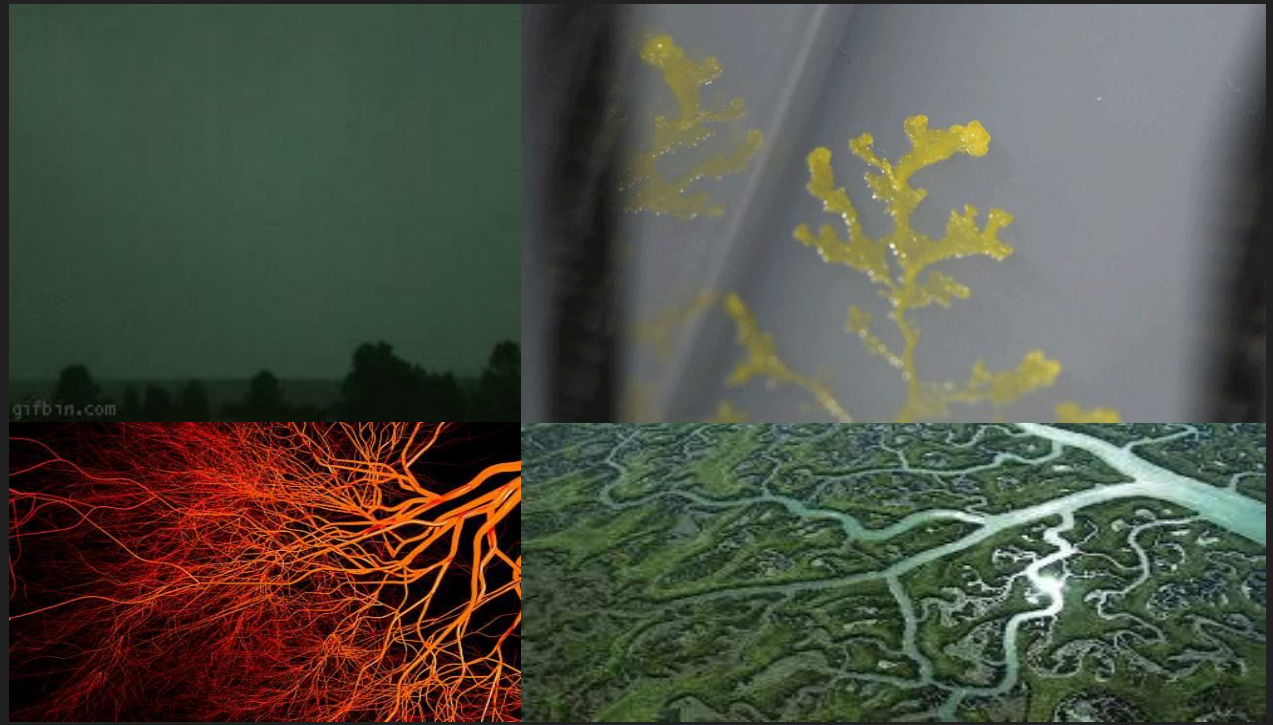


Branching & Agile Patterns

The natural world vs. the software world

Branching is all about finding the correct path(s), and then utilizing the one(s) that accomplish the necessary action:

Branching works *even without any direct guidance.*



Branching & Agile Patterns

The natural world vs. the software world

The addition of *external guidance* alters the efficiency of this system:

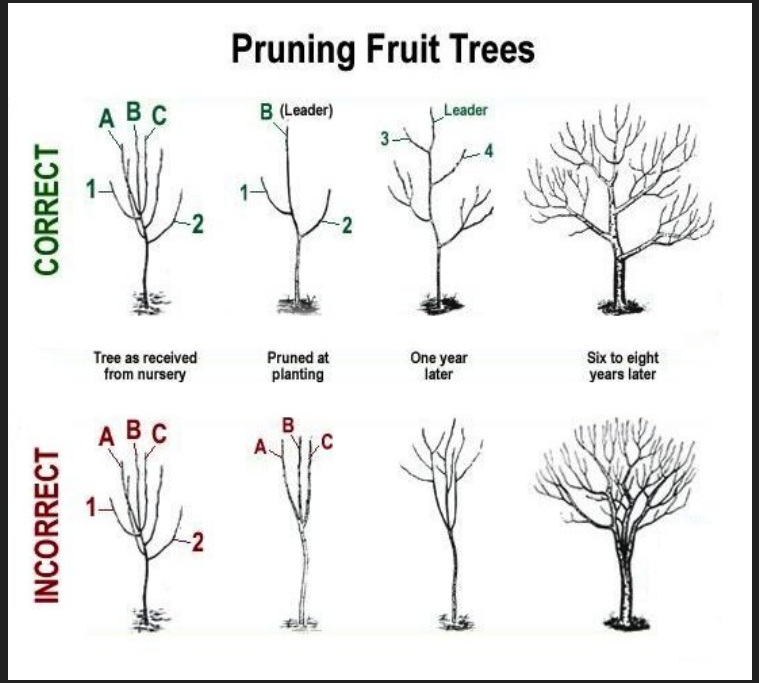
- **Agile: Removes self-competition** – speeds maturity & growth
- **Stagnant: Exacerbates self-competition** – slows maturity & growth

Maturity & Polish Patterns

Those patterns are like growing & pruning a tree

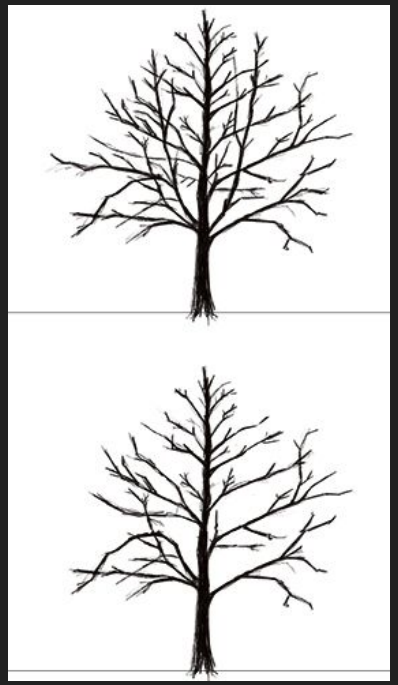
Every new Feature (Branch) needs its *own* room to grow on the Product (Tree).

If that's done *poorly*, the Features all become entangled too tightly to easily manage individually, & they compete internally for the available space & resources.



Maturity & Polish Patterns

Early Direction & Space = Health & Maturity



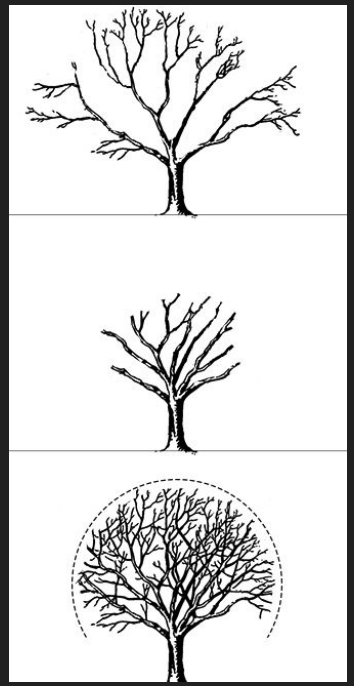
Small issues *will continue to arise naturally* over time. When there're too many leaders competing for space, pruning is necessary to maintain the health & growth of the Product (Tree) itself.

"Pruning"

This should clearly define the leader & give all of the Features (Branches) their necessary room for growth. There can be branching overlap that is still healthy.

Maturity & Polish Patterns

Poor techniques *hurt the whole tree's growth*



"Thinning"
Cuts out mature and growing features for space in an attempt to define the leader that's become lost.

"Topping"
Cutting off *all* the immature growth, and *only* focusing on anything that's already got a certain level of maturity

"Shaping"
Making the *outside* pretty to look at & interesting from a distance... while growing under the surface is an ever-more condensed, inter-tangled, resource-starved, chaotic mess.

Maturity & Polish Patterns

Ensure *each feature* grows sustainably *in phase*:

Agility **exists when feature changes are clear actions and feel like light trimming with a sense of direction...**

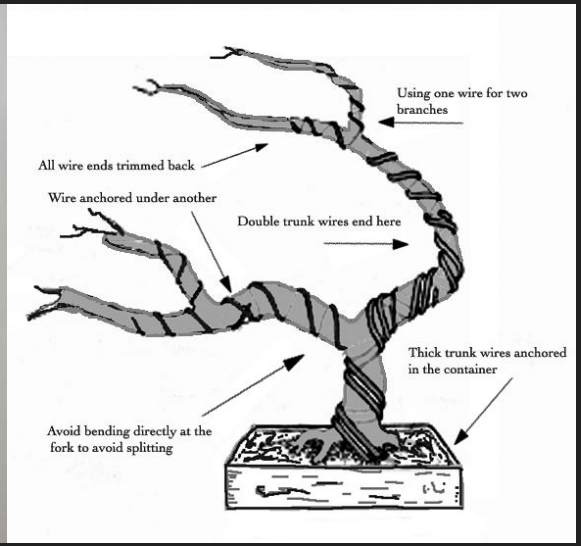
Stagnation **becomes apparent when feature changes are hard to reach, and adjustments require uprooting...**

Maturity & Polish Patterns

Long-term stability from earlier flexibility

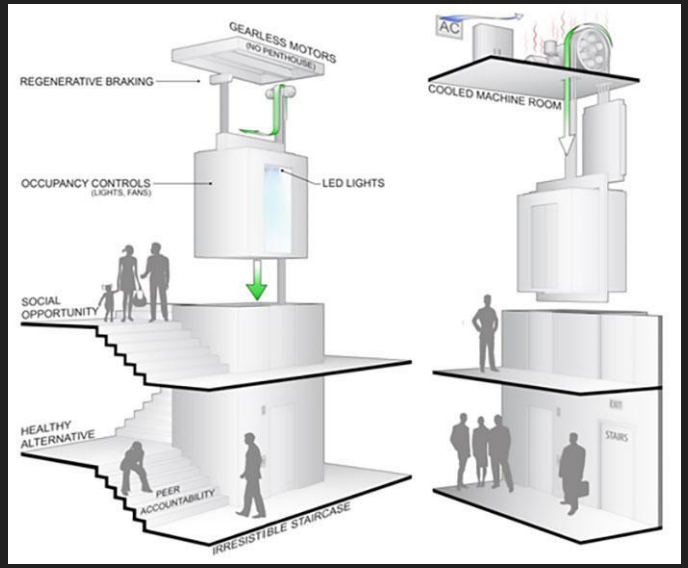
Good a11y has several different considerations that shapes a product at multiple different levels.

Understanding the core concept, the wireframe, and adapting & shaping to enable future growth will lead to success.



...but how do you accomplish that?

Workflows – Not Dragnets



An integrated process accounts for accessibility verification *at regular intervals during multiple stages* – ***corresponding to when those things can be accounted for.***

Don’t have a process that leads to *one* final checklist to trawl the unexplored depths for a11y *ONLY* at the end of the development cycle in the QA stage – ***especially only just before release.***

Recognizing The Dragnet Process

How do you spot the difference between the two?

Recognizing The Dragnet Process

How do you spot the difference between the two?

It’s subtle... but also easier than you might think.

The Dragnet Process Early On

This feels like running a race with an open parachute attached to you.

At the start, you move and maneuver super quickly. You're driven by a singular focus & vision forward towards a goal, and can shift to hit that mark.

– Then, the chute snaps open.

Suddenly, all of the same drive and vision doesn't get you as far & it takes way more effort. You burn all your energy just to get forward, you stop being able to easily maneuver & feel like you're sliding backwards.

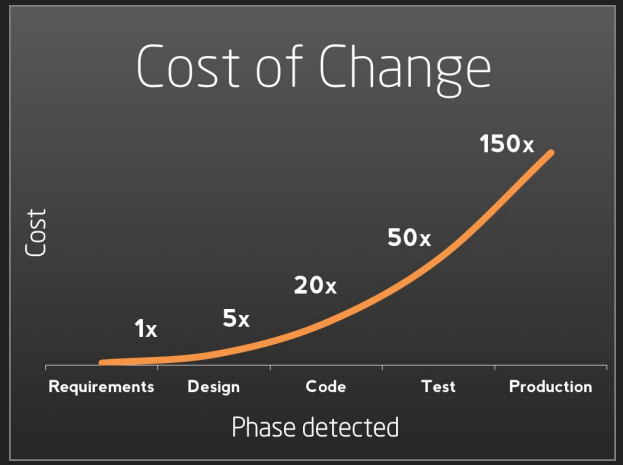
The Dragnet Process Later On

This feels like someone just equipping you with an open parachute.

Let's be honest: We've *ALL* had a deadline-approaching release that's felt like this.

Sure you released on time… but how's that whiplash feel?

The further down the process you are, the more *actively* painful and apparent this is, because you're working overtime (oftentimes literally) to keep up… *which sometimes keeps focus forward & not on the open chute.*



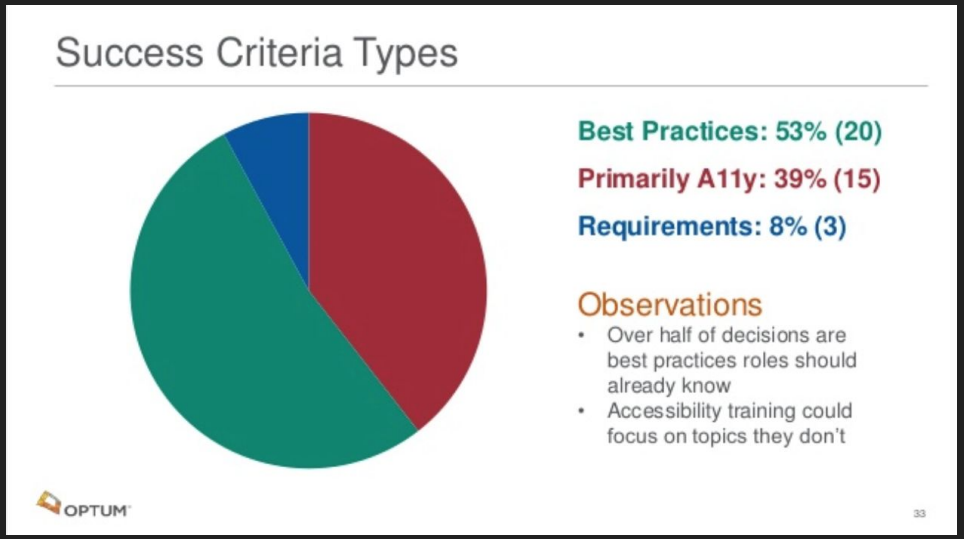
Phase-inflated cost of change!

Just continually running with that open parachute slows you down & pulls you off course, so you only ***technically*** hit the mark you're aiming for… *(face first)*

Inflating “a11y” issues

Failing **early** best practices (we often do) *can result in more than doubling the size/amount of “a11y” work…*

ProTip: Don’t double the size of an open parachute.



The *Parachute* Isn't the Problem

Remember: Equipping everyone with a parachute is part of meeting a11y needs. You don't have the option to just not have a parachute – it's a key safety feature.

So, *having* the parachute itself isn't the issue… Then what can make this such an exhausting, terrible experience?

The *Process* Itself.

Not having a process that has a *sustainable* plan for running with a parachute is why you're left with a *massive open parachute… <u>in the wind</u>*

Instead*, there's just a whole team of people working to stabilize the situation – *rather than everyone moving forward smoothly together.

Building On A Bad Foundation

Building software & then testing with a dragnet (running with an open chute) results in software like constructing a house & checking the foundation after:

It doesn't matter how good it looks, if isn't <u>stable</u> & <u>sustainable</u>.

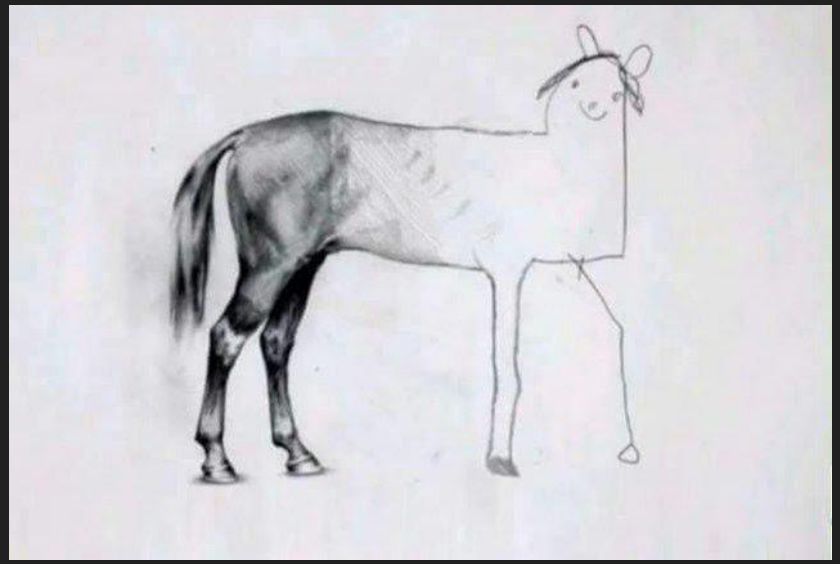
That means you're either left propping up something and hoping that you can build in a core that supports it properly…

Or making reactive changes after a lot of work has already been done & now needs to be undone.

Enabling Bad Foundations

Racing to finish *only works* if A11y NFRs are in place.

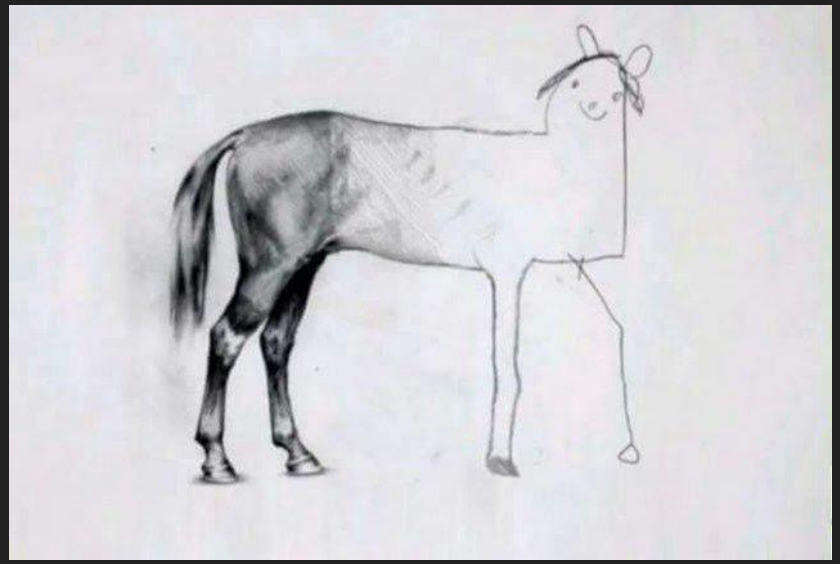
Lack of *adequate* early preparation – *opens the chute.*



Enabling Bad Foundations

Racing to finish *only works* if A11y NFRs are in place.

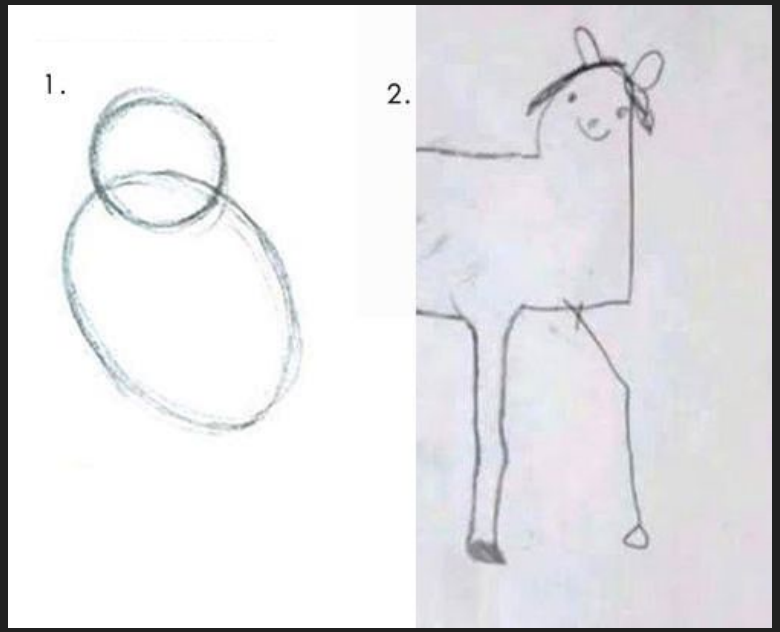
Lack of *adequate* early preparation – *opens the chute.*



"Horse Has A Face"
Was your Acceptance Criteria for GA release for this image.

In order to hit the deadline, a new *"Looks Like A Real Horse"* JIRA story got made (as polish work) ...and that work gets delayed.

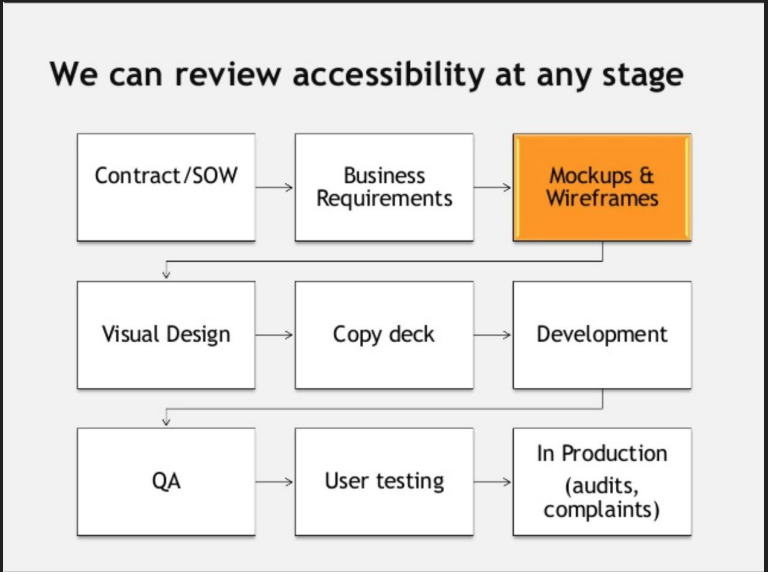
Being Proactive not Reactive



Ensure each phase's a11y NFRs ***actually*** *lead to the same a11y* that should be in the GA release.

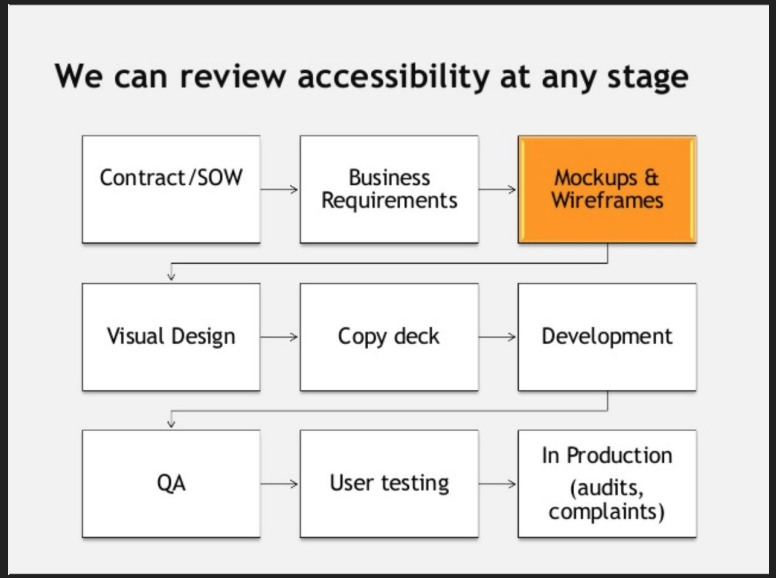
Then you won't be left needing to include a quirky little smiley face to let everyone know for certain that ***you really do care about them.***

Being Proactive not Reactive



Once you start breaking down a11y NFRs as ***a11y issues by phase***, you also start to get a better understanding how *VERY* predictable & simple most of them are *(when you start catching them at regular process intervals while they're still small).*

Being Proactive not Reactive



Having a proactive, phased approach gives a solid base to iterate *from.*

Ensuring that each previous phase is accounting for their own a11y NFR pieces slims down on the overall total a11y Phase/Cost work needed per release.

It’s no longer something that you need to overspend time/effort retrofitting fixes for.

So, Pack The Parachute First

Remember, this *isn't "The Tortoise Vs. The Hare."*

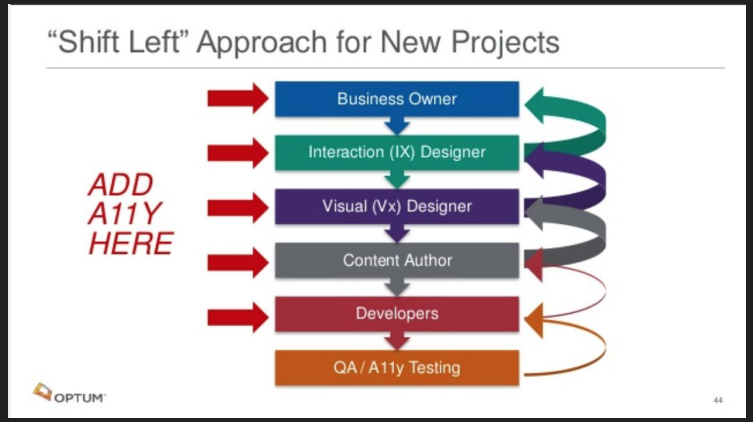
You're not packing a parachute to save you from diving out of a plane. You're preventing it from deploying while you all run with it in a backpack.

This requires a good a11y understanding & *regular synchronization* across multiple levels.

This allows for dynamic reactions: Slow down or speed up ***sustainably*.**

Sprint & Verify in Small Phases

There are handoffs like a relay race, not a continual rush.



Cooperation for verification is imperative: Everyone needs to know if the chute is coming undone in Planning, Design, or Development. That way, the a11y team can help to proactively address the processes where things *aren't* meeting phase requirements & *are* generating unnecessary drag.

Sprint & Verify in Small Phases

There are handoffs like a relay race, not a continual rush.
Move forward *as a team*, don't leave them behind just to *feel* fast.

At first this will feel like you're carrying around a lot of extra weight.

But then you'll realize that ***the same weight*** *was always still attached to you*.
It just felt like less on the ground – *but became* ***more*** *once the wind hit it.*

Sprint & Verify in Small Phases

https://c2experience.com/web-accessibility-checklists

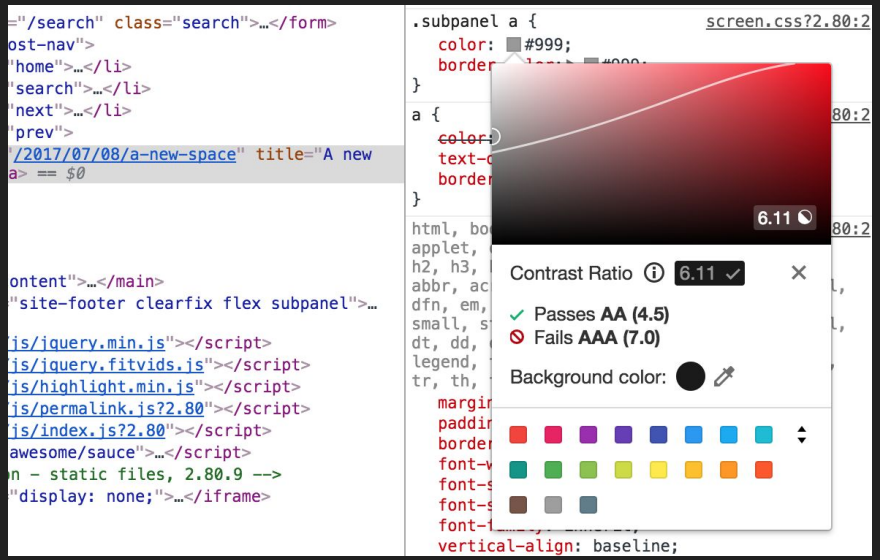
There are lots of well-defined checklists to help in every phase.

Sprint & Verify in Small Phases

Chrome Dev Tools & Axe Core

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

https://github.com/dequelabs/axe-core



Chrome Dev Tools uses Axe by Deque, and makes it easy to do some simple verifications just with Chrome.

Axe Core can be added as a linter for code checking on commit, and handling it as automation.

Sprint & Verify in Small Phases

A11y Insights

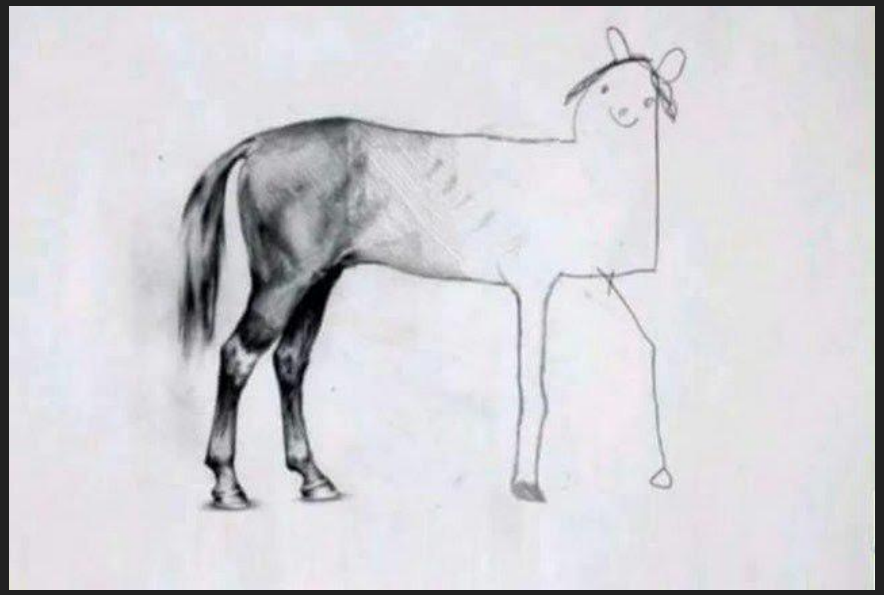
https://accessibilityinsights.io/

Accessibility Insights for Web is an open-source tool that was built in a partnership between Deque and Microsoft for helping to test software for a11y requirements.

It also has a number of ad hoc tools that non-QA members can use on their own. Any tools that can extend beyond QA help maintain a solid connection through each phase.

Sprint & Verify in Small Phases

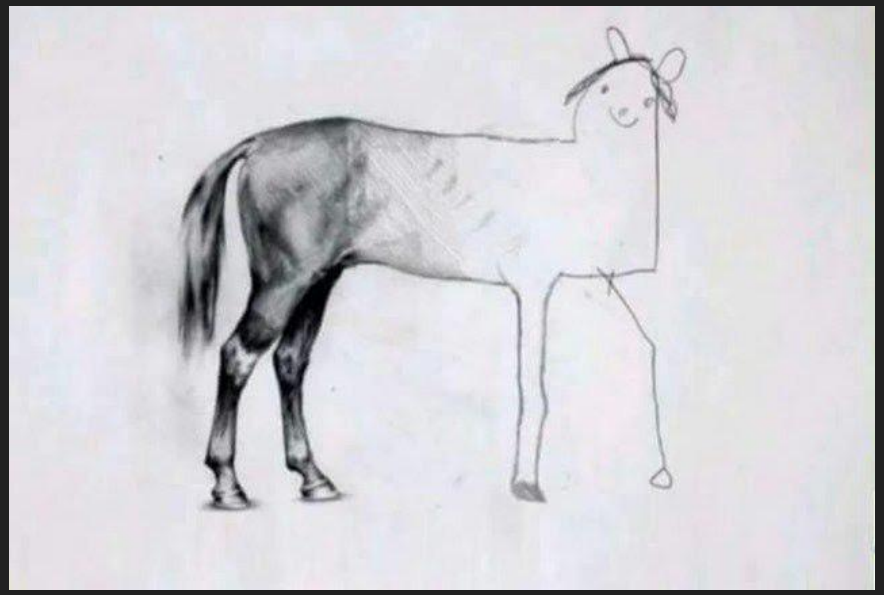
Don’t let the chute deploy as you hand off to the next phase



If you don’t account for *ALL* the phases’ a11y time & needs ***accurately***…
any ***agility*** *you had gets lost –* ***even though the start looked good.***

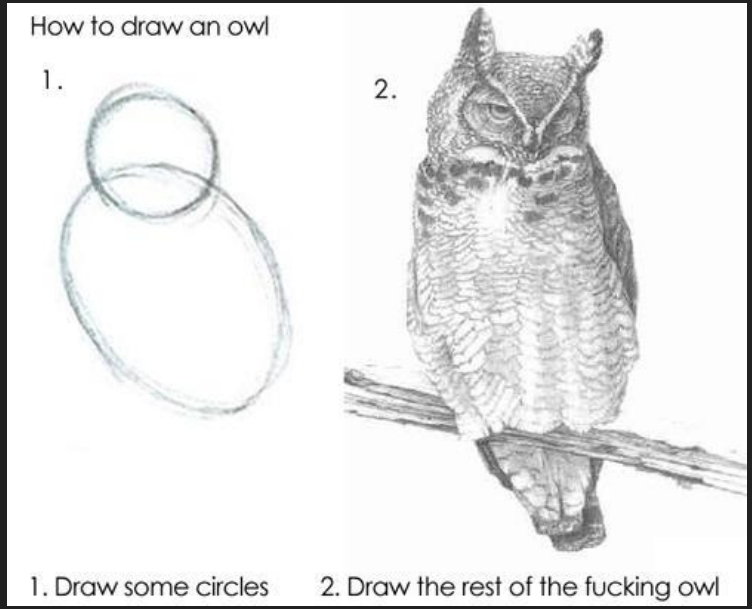
Sprint & Verify in Small Phases

Don’t let the chute deploy as you hand off to the next phase



Run regular project *Pre-Mortems* (that assume failing) before hand-offs,
Think through how it could fail & ensure those issues are well-covered.

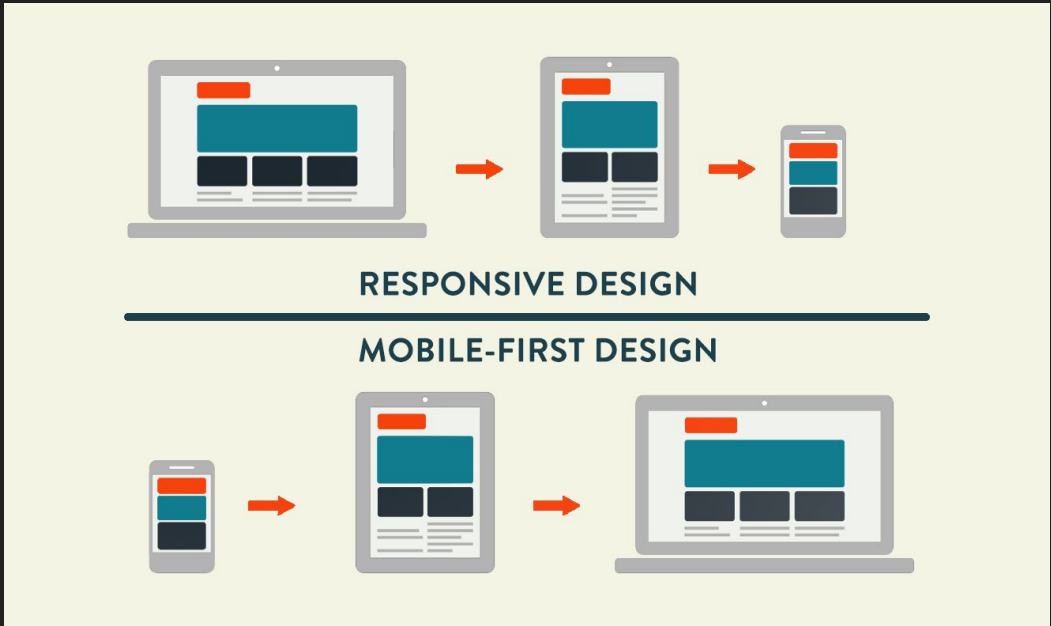
Sprint & Verify in Small Phases



What're some of the a11y needs that exist *during & between phases* *and how do you go about testing for them*?

Design & QA Verifications

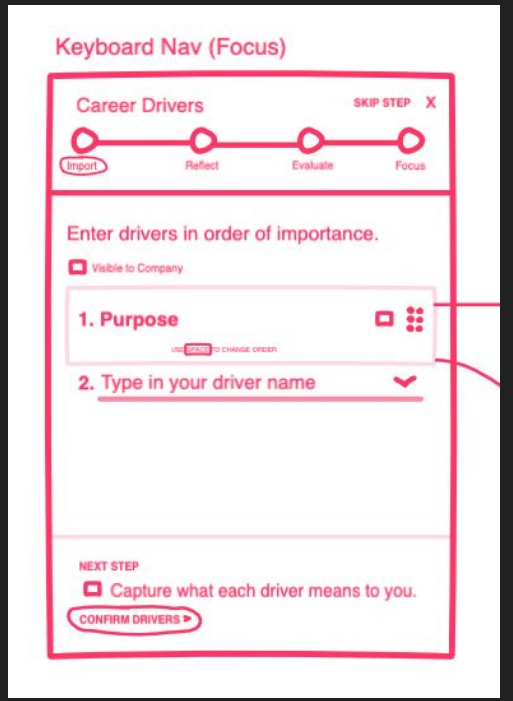
Responsive & Mobile-First wireframes can be QA'd for a11y, and are *paramount* for WCAG 2.1



Design & QA Verifications

Responsive Wireframe-level a11y callouts

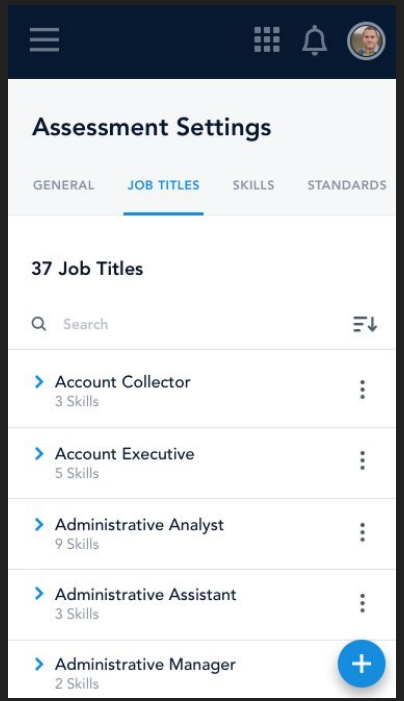
- **No Hover-Only:** Responsive considers touch devices
- **Heading Structures:** Wireframes are simple outlines, which assists in explicitly defining heading levels for content & the hierarchy of page objects
- **Landmark Regions:** Blocking out content into specific areas & knowing what their purpose is on the page
- **Focus Order/Workflow:** Logical flow is well-understood as the content has to stack vertically in a specified arrangement, and there is an ability to easily bypass / minimize / collapse content groups
- **Parent/Child Labels:** Which controls are specific vs. which are generic & contextual labels are needed



Design & QA Verifications

Some Prototype-level a11y considerations

- **Focus Placement:** Where does it start? Where does it move when X action is taken? Should it be contained?
- **Announcements:** Is there a silent update on the page? Does it need an announcement? Which one?
- **Component Usage:** What type of component does it use? If it's something new, is it outlined specifically?
- **Color Usage:** What do the HCM or Dark Mode versions look like? Are there issues with contrast or the information conveyed by color?
- **Content & Context:** As-is, does the information make sense? Does it need anything additional for SR/KO?
- **Workflows:** Are there complications during the steps?



Eng & QA Verifications

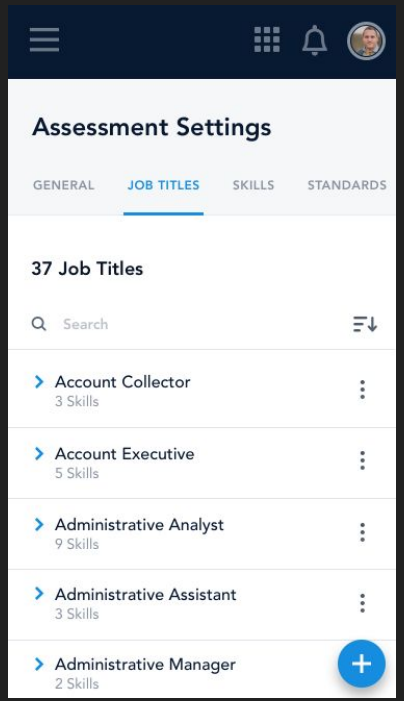
Ensure everything actually functions... *together*.

(Unexpected behaviours can emerge when combining individually stable components)

Eng & QA Verifications

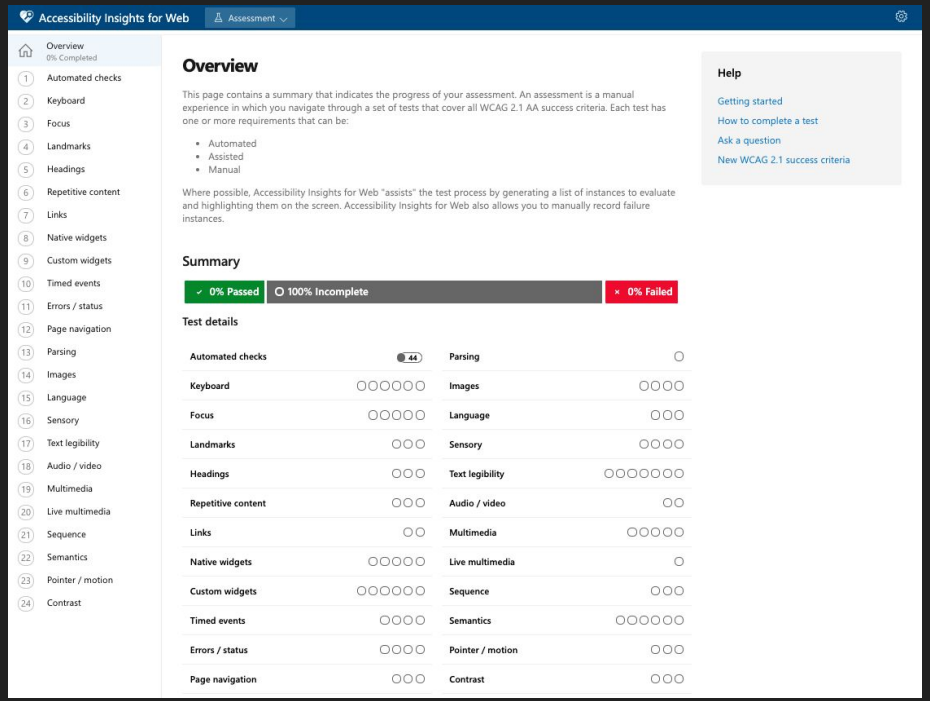
Some Alpha-level a11y considerations

- **Keyboard Nav:** Does it move in a logical order? Does it reach all (and only) the actionable objects? Is Focus always visible? Do all the Keyboard Controls work?
- **Page Structure:** Do Automated Tests pass? Are the Heading and Landmark Regions configured correctly?
- **Screen Readers:** Do the SRs work as they should? Do repeated controls have sufficient contextual labels? Do announcements exist for on-page updates?
- **Responsive:** Does the Responsive version work properly, and do browser zoom / text resize work?
- **Windows High Contrast Mode:** Do the styles still function with Windows HCM options enabled?



QA Verifications

The “Accessibility Audit”



While each team has their own checklists that they can verify, and each stage has things that QA can assist with, the final step of the “a11y audit” is the full end-to-end check before release.

A11y Insights for Web is *especially fantastic* for this.

QA Verifications

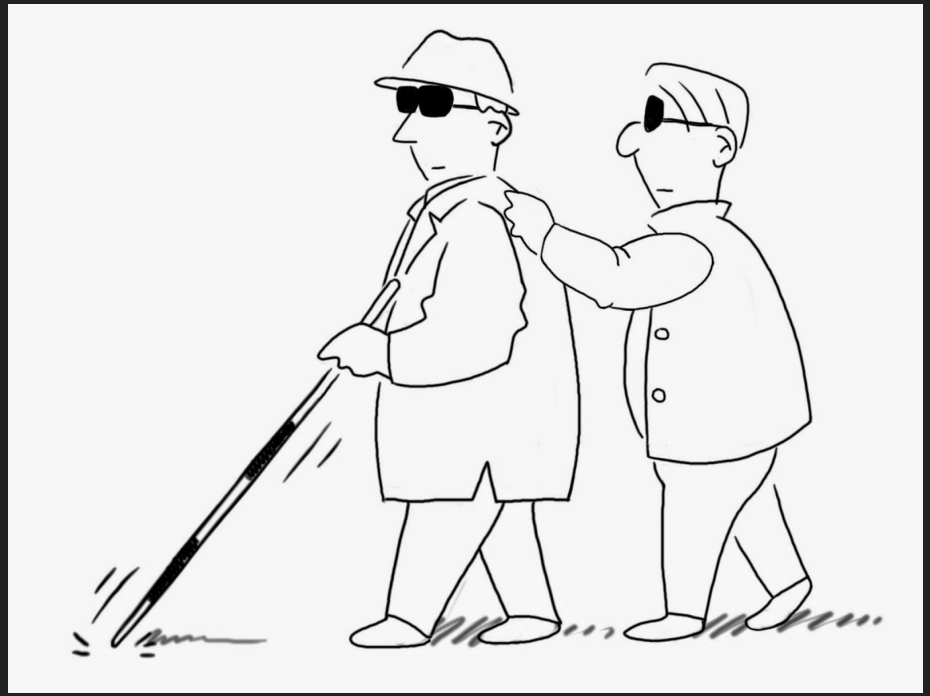
The “Accessibility Audit”

This is where you’ll learn if your process is working smoothly... or if it needs some help.

Identify the types of bugs being found & *adapt accordingly.*

A11y User Testing & Support

A11y User Testing should always be guided



This ensures that you get accurate data, remove unnecessary stressors, and understand the feedback.

Also anything wholly inaccessible can be missed otherwise – like an important image that's tagged as hidden with no alt text.

A11y User Testing & Support

Support has a unique & beneficial perspective

Like a11y, Support doesn't belong to any one specific development sprint team. Their focus is on the *entirety of a product*. Their active understanding is very wide, and they can help catch basic design, structural, & functional inconsistencies across the app. The more they know a11y the better, too.

Bringing Support In Early

Support has a unique & beneficial perspective

QA is good at ensuring features are very solid against being broken:

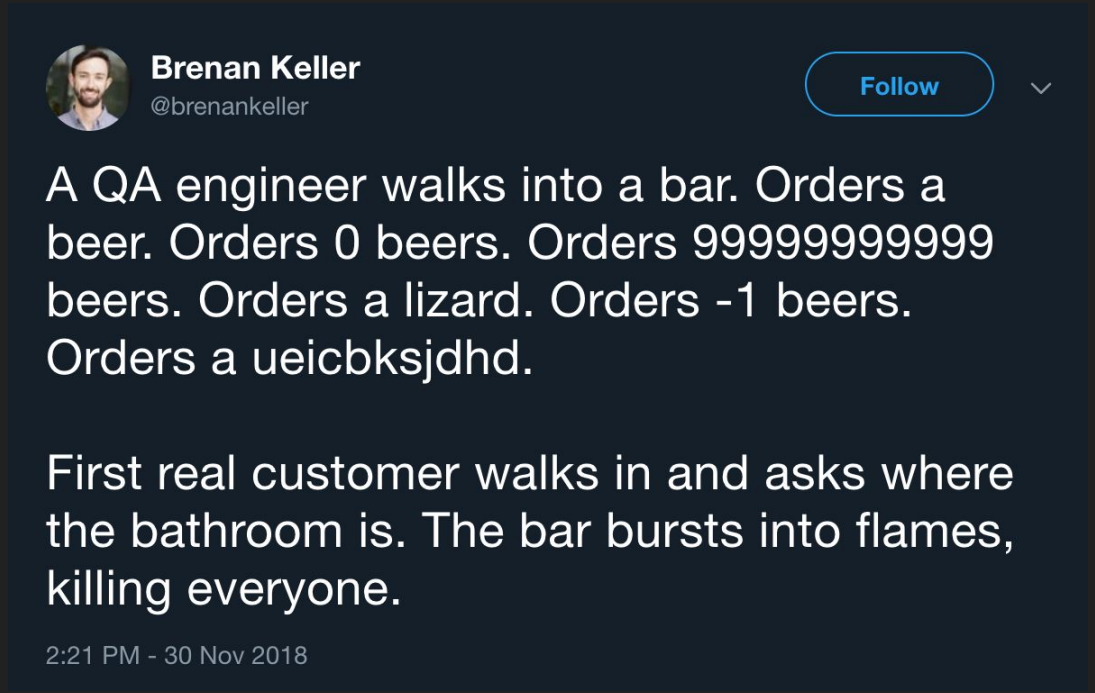
Support is good for knowing how your users are most likely to (mis)use a feature:

Bringing Support In Early

Support has a unique & beneficial perspective

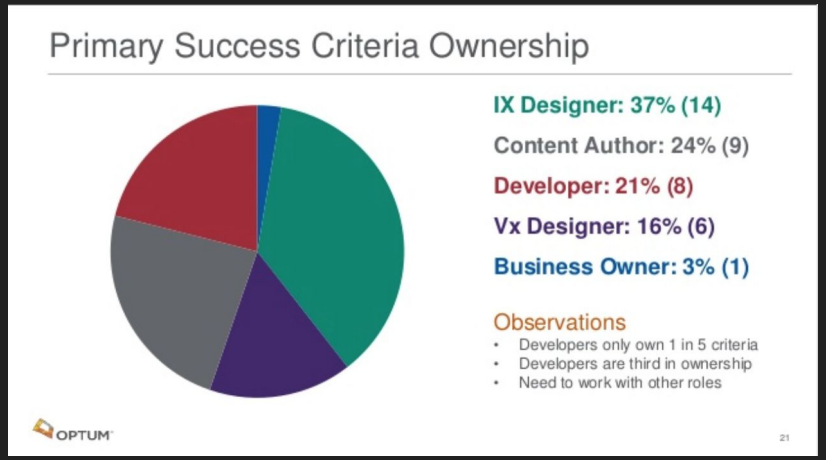
QA is good at ensuring features are very solid against being broken:

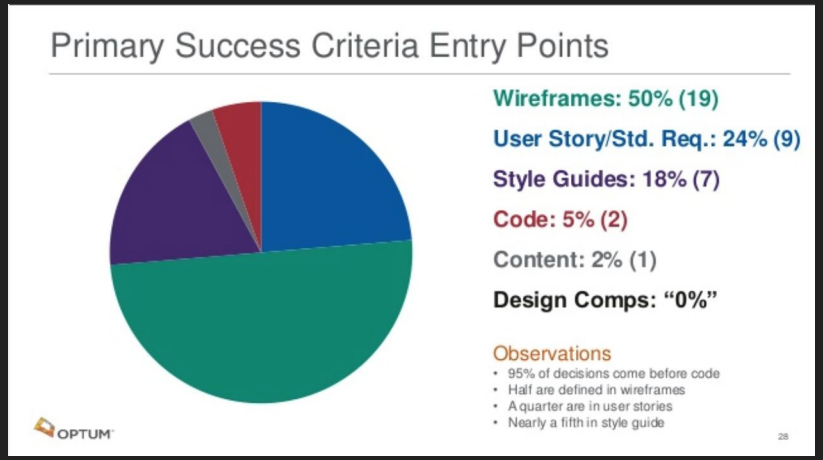
Support is good for knowing how your users are most likely to (mis)use a feature:



Owning A11y Responsibilities

- Outline *stronger* & *more specific* a11y criteria by phase
- Ensure *each team* is actively guiding and driving them
  - Regularly *check*, *find*, & *address* any phase gaps





More Resources on Owning It.

Shift Left Testing & A11y Ownership:
https://bit.ly/2HJea3e

Designer's A11y Responsibility:
https://bit.ly/2HeLylP

Performing A11y QA on Wireframes:
https://bit.ly/2SJ3qdL

Resources For Testing.

Team-Specific Checklists:
https://c2experience.com/web-accessibility-checklists

Axe Core:
https://github.com/dequelabs/axe-core

Accessibility Insights for Web:
https://accessibilityinsights.io/