Software Testing for Absolute Beginners

Learn how to test a real Bootstrap website step by step

What you will learn

- What software testing really means
- Manual testing vs automated testing
- Writing test cases for a real website
- Functional, UI, responsive and accessibility tests
- Cross-browser and cross-device testing
- Reporting bugs that developers can actually fix
- Intro to automated testing with the browser console

egotechworld.com

Beginner-friendly programming tutorials

Table of Contents

1.	What is Software Testing?	3
2.	Why Test the Bootstrap Website?	4
3.	Types of Testing You Need to Know	5
4.	The Test Case: Your Most Important Tool	7
5.	Functional Testing the Website	8
6.	UI / Visual Testing	10
7.	Responsive Testing (Mobile, Tablet, Desktop)	11
8.	Cross-Browser Testing	12
9.	Accessibility Testing	13
10.	Performance Quick Checks	14
11.	How to Report a Bug Properly	15
12.	Intro to Automated Testing	16
13.	Final Test Checklist	18

1. What is Software Testing?

Software testing is checking that a piece of software does what it is supposed to do. Nothing more complex than that. You compare what the software *actually* does against what it *should* do, and you note every difference.

A simple example

On our Bootstrap website, the navbar has a link called "About". The expected behaviour is: clicking it scrolls the page to the About section. If you click it and nothing happens, that gap between expected and actual is a **bug**. Finding it is testing.

Two key questions every tester asks

- **Does it do what it should?** (positive testing)
- **Does it fail gracefully when something goes wrong?** (negative testing)

Tester mindset

Developers build. Testers break. A good tester thinks like a confused user, an angry user, an unlucky user, and someone using the site on a 5-year-old phone with bad internet. The goal is to find bugs *before* real users do.

> **TIP:** Testing is not about proving the software works. It is about trying to make it fail. If you cannot make it fail after honest effort, you have built confidence.

The cost of skipping testing

A bug found while coding takes minutes to fix. The same bug found in production after launch can cost hours, hurt your reputation, and lose users. Testing saves time — it does not waste it.

2. Why Test the Bootstrap Website?

In our previous tutorial we built a one-page Bootstrap site with a navbar, hero, three feature cards, and a footer. It looks fine when you open it on your laptop. But that is just one screen, one browser, one user.

What could go wrong?

- The navbar links might not actually scroll to the right section.
- The hero button might be unclickable on mobile.
- Cards might overlap on small screens.
- The Bootstrap CDN could fail and the page becomes unstyled.
- Text might be too small for older users to read.
- The page could load slowly on a 3G connection.
- Colour contrast might fail accessibility rules.
- It might look different in Chrome vs Safari vs Firefox.

Each of these is a real risk that testing finds.

The website under test

Throughout this tutorial we use the same `index.html` file from the Git/GitHub tutorial. To recap, it has these elements you will test:

Element	What it should do
Navbar brand	Display "MySite", stay visible at top
Nav links (3)	Scroll smoothly to Home, About, Contact sections
Hero section	Show title, lead text, and Learn More button
Hero button	Scroll page down to the About section
3 feature cards	Display in a row on desktop, stack on mobile
Footer	Show copyright text at bottom

NOTE: Open `index.html` in your browser now. Keep it open as you read. Every chapter from here is hands-on.

3. Types of Testing You Need to Know

Testing is a big field. Beginners get overwhelmed by the names. Here are only the types that matter for a website like ours.

Type	What it checks	Example for our site
Functional	Does each feature work?	Clicking "Learn More" jumps to About
UI / Visual	Does it look right?	Cards have shadow, navbar light gray
Responsive	Looks good on every screen size	Cards stack vertically on phone
Cross-browser	Works in different browsers	Same look in Chrome and Firefox
Accessibility	Usable by everyone	Tab key navigates all links
Performance	Fast enough?	Page loads in under 3 seconds
Compatibility	Works on different devices	Renders on iPhone and Android

Manual vs Automated

Two ways to actually run tests:

- **Manual testing:** A human opens the site and clicks through. Slow but catches things automation misses (visual oddness, confusing flow).
- **Automated testing:** A script runs tests for you. Fast, repeatable, but only finds things you wrote tests for.

TIP: Beginners should start manual. Once you understand what to test and why, automation makes more sense.

Black box vs White box

Two ways of looking at the software:

- **Black box:** You test as a user. You do not look at the code. You only care about input and output.
- **White box:** You read the code and write tests for specific functions and code paths.

For a small static website, black box testing is enough. White box matters more when you write JavaScript or backend logic.

Levels of testing

Level	What it tests
Unit testing	One small function in isolation
Integration testing	Two or more parts working together
System testing	The whole application end to end
Acceptance testing	Does it match what the user wanted?

Our static site has no functions to unit test, so we focus on system testing (the whole page) and acceptance testing (does it match what we set out to build?).

Positive vs Negative testing

Positive test: the user does the right thing, the system responds correctly. Example: click About, page scrolls to About.

Negative test: the user does something wrong or unusual, the system handles it gracefully. Example: click About when the section was deleted from the HTML — the link should fail safely, not crash the page.

4. The Test Case: Your Most Important Tool

A **test case** is one written instruction that says: do these steps, and this exact thing should happen. Without written test cases, testing is just "clicking around". With them, testing becomes professional and repeatable.

Standard test case format

Field	Meaning
Test ID	Unique number, e.g. TC-001
Title	One short sentence: what is being tested
Pre-condition	What must be true before starting
Steps	Exact actions, numbered
Test data	Any input values needed
Expected result	What should happen
Actual result	What did happen (filled in during testing)
Status	Pass / Fail / Blocked

A real test case for our website

Field	Value
Test ID	TC-001
Title	Hero "Learn More" button scrolls to About
Pre-condition	index.html is open in a browser
Steps	1. Scroll to top of page 2. Click "Learn More" button
Test data	None
Expected result	Page smoothly scrolls down to the "What I Learned" section
Actual result	(filled in when you run the test)
Status	(Pass / Fail)

TIP: Write test cases **before** testing. Writing them after is rationalising clicks you already did. Write first, then execute.

5. Functional Testing the Website

Functional testing answers: do all the features work? Here is a complete set of test cases for our Bootstrap site.

Test cases for navigation

ID	Title	Expected result
TC-001	Click navbar "Home" link	Page scrolls to top hero section
TC-002	Click navbar "About" link	Page scrolls to feature cards section
TC-003	Click navbar "Contact" link	Page scrolls to footer section
TC-004	Click "MySite" brand text	No action, or scroll to top (define expected behaviour)
TC-005	Click "Learn More" hero button	Page scrolls to About section

Test cases for content

ID	Title	Expected result
TC-006	Verify hero title text	Reads "Welcome to MySite"
TC-007	Verify hero lead text exists	Description paragraph is visible below title
TC-008	Count feature cards	Exactly 3 cards: HTML, Bootstrap, Git
TC-009	Each card has title and description	All 3 cards show heading and paragraph
TC-010	Footer text	Reads "Built by Me · 2026"

Negative test cases

What happens when something goes wrong? These tests check graceful failure.

ID	Scenario	Expected behaviour
TC-011	Disable internet, reload page	Page still shows text, but without Bootstrap styling
TC-012	Resize browser to 320px wide	No horizontal scrollbar, all content visible
TC-013	Open page with JavaScript disabled	Content still visible (our site has no JS, should work)
TC-014	Click navbar link to a section that does not exist	URL changes but page does not crash

How to record results

Make a simple table in any spreadsheet (Google Sheets or Excel) with these columns: **ID, Title, Expected, Actual, Status, Notes**. After running each test case, fill in the actual result and mark Pass or Fail. This is your **test report** — the deliverable you give the developer.

> **TIP:** Always mark Pass or Fail. "Mostly working" is not a valid status. If you are unsure, write a clearer test case.

6. UI / Visual Testing

UI testing is checking that the site *looks* right. The feature might work, but if the button is hidden behind another element or the text is unreadable, the user cannot use it.

What to check on every page

- All text is readable and not cut off.
- Colours match the design (blue hero, light navbar).
- Buttons look like buttons (not flat text).
- Spacing between sections is even, not cramped.
- Card shadows are visible.
- Hover states change appearance (links underline, buttons darken).
- Nothing overlaps anything else.
- Images (if any) load correctly with no broken icons.
- Fonts load — no fallback to plain Times New Roman.

UI test cases for our site

ID	Visual check	Expected
TC-015	Hero background colour	Bootstrap blue (bg-primary)
TC-016	Hero text colour	White, readable on blue
TC-017	Navbar background	Light gray (bg-light)
TC-018	Card alignment	All 3 cards same height, evenly spaced
TC-019	Card shadow	Subtle drop shadow visible (shadow-sm)
TC-020	Hover on "Learn More" button	Background darkens slightly
TC-021	Footer alignment	Text centred, light background

NOTE: Compare visually with screenshots of the design (if you have any). For our site, the "design" is whatever the Git tutorial showed. Use that as your reference.

7. Responsive Testing (Mobile, Tablet, Desktop)

Your site might look great on a 1920px monitor and totally broken on a 360px phone. Responsive testing catches this.

Use Chrome DevTools to test sizes

- Open the site in Chrome.
- Right-click anywhere → **Inspect**.
- Click the small phone/tablet icon at the top of DevTools.
- Pick a device from the dropdown, or drag to resize.

Standard sizes to test

Device	Width	What you check
Small phone	320px	No horizontal scroll, all readable
Phone	375px	Cards stack vertically
Phone (large)	414px	Buttons easy to tap
Tablet	768px	Layout transitions to medium view
Laptop	1024px	Cards start showing in a row
Desktop	1440px	Full row layout, hero centred
Wide	1920px	Content does not stretch too wide

Responsive test cases

ID	Test	Expected
`TC-022`	View at 320px width	All content fits, no side scrolling
`TC-023`	View at 375px width	3 cards stack one below the other
`TC-024`	View at 768px width	Cards may show 1 per row or 2 per row
`TC-025`	View at 1024px+ width	All 3 cards in single horizontal row
`TC-026`	Rotate phone to landscape	Layout adjusts cleanly
`TC-027`	Tap navbar links on touch device	Tap targets large enough to hit

TIP: If you can, test on a real phone too. Open your GitHub Pages URL on your phone browser. Real touch targets behave differently from a mouse simulation.

8. Cross-Browser Testing

Different browsers render the same HTML and CSS slightly differently. A page that looks perfect in Chrome can break in Safari or Firefox. Bootstrap removes most of this pain, but you still test.

Browsers to test (priority order)

Browser	Why it matters
Google Chrome	60% of all users worldwide
Safari	All iPhone and Mac users
Microsoft Edge	Default on Windows 11
Firefox	Privacy-focused users, developers
Samsung Internet	Default on Samsung phones (huge in Asia)

How to test on multiple browsers

- Install Chrome, Firefox, and Edge on your computer (all free).
- On a Mac, Safari is built in. On Windows, install Chrome and Edge.
- Open your site in each browser, run your test cases.
- Note any differences in spacing, fonts, button styling.

If you cannot install all browsers, free online services let you test:

- BrowserStack (free trial) — test on real devices
- LambdaTest (free tier) — 100 minutes per month free
- Browserling (free single tab) — quick checks

What to look for

- Fonts render the same size and weight.
- Buttons have the same border radius and padding.
- Hero gradient or solid colour matches.
- Card shadows appear consistently.
- Smooth scroll animation behaves the same.

NOTE: Internet Explorer is dead. Microsoft ended support in 2022. You do not need to test IE anymore unless your client specifically asks.

9. Accessibility Testing

Accessibility (often shortened to **a11y**) is making sure people with disabilities can use your site. Blind users, users with motor difficulties, colour-blind users — they all deserve to read your content. It is also legally required in many countries.

Quick accessibility tests anyone can do

Test	How to do it
Keyboard navigation	Press Tab repeatedly. Every link/button should be reachable.
Visible focus	Tabbing should show a visible outline on the active element.
Image alt text	Every <img> tag must have meaningful alt attribute.
Heading order	H1 first, then H2, then H3. No skipping levels.
Colour contrast	Text should be clearly readable against background.
Zoom test	Press Ctrl + = until at 200%. Layout should still work.

Use the free Lighthouse tool

- Open your site in Chrome.
- Press F12 to open DevTools.
- Click the **Lighthouse** tab.
- Tick **Accessibility**, click **Analyze page load**.
- You get a score from 0–100 with a list of issues.

Likely issues you will find

BUG EXAMPLE: On our website, the navbar links are inside <a> tags but the brand link goes to # which is an empty link. Lighthouse may flag this. Fix: change to `#home`.

BUG EXAMPLE: The cards have no `aria-label` or headings inside semantic landmarks. For a small site this is minor, but worth knowing for bigger projects.

TIP: Aim for a Lighthouse accessibility score of 90+. Perfect 100 is hard for complex sites, but 90+ means you are doing the important things.

10. Performance Quick Checks

A slow site loses users. Studies show that if a page takes more than 3 seconds, half the users leave. For a static site like ours, performance is mostly about file size and CDN speed.

Run Lighthouse Performance test

Same Lighthouse tool from the accessibility chapter, but tick **Performance** instead. You get four key numbers:

Metric	What it means	Good target
First Contentful Paint	When user first sees something	Under 1.8s
Largest Contentful Paint	When the main content is visible	Under 2.5s
Total Blocking Time	How long the page is frozen	Under 200ms
Cumulative Layout Shift	How much things jump around as page loads	Under 0.1

Test on slow internet

- Open Chrome DevTools.
- Click the **Network** tab.
- Find the throttling dropdown (says "No throttling").
- Choose **Slow 3G** and reload the page.
- Time how long until the page is usable.

Things that slow our site

- Loading Bootstrap from a CDN — fast on most networks but adds a request.
- If you add images later, large unoptimised photos.
- If you add fonts from Google Fonts, extra HTTP requests.

NOTE: For a tiny static site like ours, performance is not a big concern. But always run Lighthouse once before publishing — it catches obvious mistakes like uncompressed images.

11. How to Report a Bug Properly

Finding a bug is half the job. Reporting it clearly is the other half. A vague bug report wastes the developer's time and yours.

The 7 fields of a good bug report

Field	Description
Title	One sentence summary
Environment	Browser + version, device, OS
Steps to reproduce	Numbered exact actions
Expected result	What should happen
Actual result	What actually happened
Severity	Critical / Major / Minor / Trivial
Screenshot	Picture of the bug if visual

Bad bug report (do not do this)

BAD: "The button doesn't work."

Why bad: Which button? Doesn't work how? On what browser? What did you expect? The developer cannot fix this.

Good bug report

Field	Value
Title	"Learn More" button does not scroll on mobile Safari
Environment	iPhone 14, iOS 17.2, Safari
Steps	1. Open site on iPhone Safari 2. Tap "Learn More" button in hero
Expected	Page scrolls smoothly to About section
Actual	Nothing happens. URL changes to add #about but page does not scroll.
Severity	Major — affects core navigation on mobile
Screenshot	(attached: bug-001.png)

Severity levels explained

Level	Meaning	Example
Critical	Blocks all use	Page is blank, won't load at all
Major	Important feature broken	Navbar links don't work
Minor	Small issue, workaround exists	Card shadow missing
Trivial	Cosmetic, no impact	Footer has typo

12. Intro to Automated Testing

So far you have done everything by hand. Now imagine doing it 200 times for 200 tests. That is why automation exists. Even for a small site, you can write a few quick automated checks using just the browser.

Method 1: Browser console smoke tests

Open your site in Chrome. Press F12 to open DevTools. Click the **Console** tab. Paste this code and press Enter:

```
// Quick smoke test: do all key elements exist?
const checks = [
  ['Navbar', 'nav.navbar'],
  ['Hero section', '#home'],
  ['About section', '#about'],
  ['Footer', '#contact'],
  ['Learn More button', '.btn.btn-light'],
  ['Three cards', '.card', 3],
];

checks.forEach(([name, sel, expected]) => {
  const els = document.querySelectorAll(sel);
  const ok = expected
    ? els.length === expected
    : els.length > 0;
  console.log((ok ? 'PASS' : 'FAIL') + ' - ' + name);
});
```

You will see PASS or FAIL for each check. Run this any time you change the HTML to make sure nothing important got deleted.

Method 2: Test with real automation tools

When you are ready to grow, learn one of these tools. They open a browser, click buttons, and verify results — all automatically.

Tool	Best for
Cypress	Modern web testing, easy syntax, great UI
Playwright	Cross-browser testing (Chrome, Firefox, Safari) by Microsoft
Selenium	Industry classic, supports many languages
Jest	Unit testing JavaScript functions

Tiny example: a Cypress test

Cypress is the friendliest tool for beginners. A test for our site looks like this:

```
describe('MySite homepage', () => {
  beforeEach(() => {
    cy.visit('https://yourname.github.io/my-website/');
  });

  it('shows the hero title', () => {
    cy.get('h1').should('contain', 'Welcome to MySite');
  });

  it('has 3 feature cards', () => {
    cy.get('.card').should('have.length', 3);
  });

  it('Learn More button scrolls to About', () => {
    cy.contains('Learn More').click();
    cy.get('#about').should('be.visible');
  });
});
```

Read this in plain English: *Visit the site, the heading should contain "Welcome to MySite", there should be 3 cards, and clicking Learn More should make About visible.* That is how readable modern testing has become.

> **TIP:** Do not jump into automation before you can manually test well. Automation tests *your understanding* of the system. If you do not know what should happen, automation just runs wrong tests faster.

When automation pays off

- You run the same tests over and over (every release).
- The site has many pages or complex flows.
- Multiple developers change code daily.
- You need to test on many browsers/devices regularly.

When manual is enough

- Small static sites (like ours).
- Early prototypes that change shape often.
- Visual checks that need a human eye.

13. Final Test Checklist

Before you call any website "done", run through this list. Tick each item. If anything fails, file a bug and fix it before launch.

Functional

- All navbar links work and scroll to correct sections.
- Hero button works.
- All text content is correct (no typos, no Lorem ipsum left).
- All sections are present (hero, about, footer).

Visual

- Colours match the design.
- Spacing is consistent.
- Cards have shadows and equal height.
- Hover states work on all clickable elements.

Responsive

- Tested at 320, 375, 768, 1024, 1440 px widths.
- No horizontal scrollbar at any size.
- Cards stack on mobile, row on desktop.
- Tested on a real phone if possible.

Cross-browser

- Tested in Chrome, Firefox, and Safari/Edge.
- Layout matches across all browsers.

Accessibility

- Tab key reaches every link and button.
- Focus outline is visible.
- Lighthouse Accessibility score 90+.
- Page works at 200% zoom.

Performance

- Lighthouse Performance score 90+.
- Page loads under 3 seconds on Slow 3G.

Production

- HTTPS works (automatic on GitHub Pages).
- Page title is meaningful, not "Document".
- Live URL works on a fresh browser with no cache.

You are now a tester

test plan, execute it, find bugs, and report them well. The same principles apply to apps, APIs and bigger projects — only th

Test early. Test often. Test honestly.

More tutorials and free resources: ***egotechworld.com***