QAREA

DESIGN QA

MITIGATE DESIGN ISSUES EFFICIENTLY

QAREA

TABLE OF CONTENTS

DESIGN DEBT AND WHY IT SHOULD BE TAKEN SERIOUSLY

Many projects today have design debt, a rarely discussed downside of iterative and incremental software development methods. As a term, **design debt** was based upon the more popular concept called **technical debt**, a metaphor, coined by a renowned American programmer Howard Cunningham. As a pioneer in both design patterns and extreme programming, Cunningham encouraged companies to perceive cutting corners in the course of development as getting yourself into financial debt. Like taking a large loan that you will later have to repay at a considerable interest rate, development short cuts and lack of proper verification procedures will inevitably lead to extra rework, hindering the addition of new features and slowing your company's growth.

High-quality design brings simplicity to solving user problems

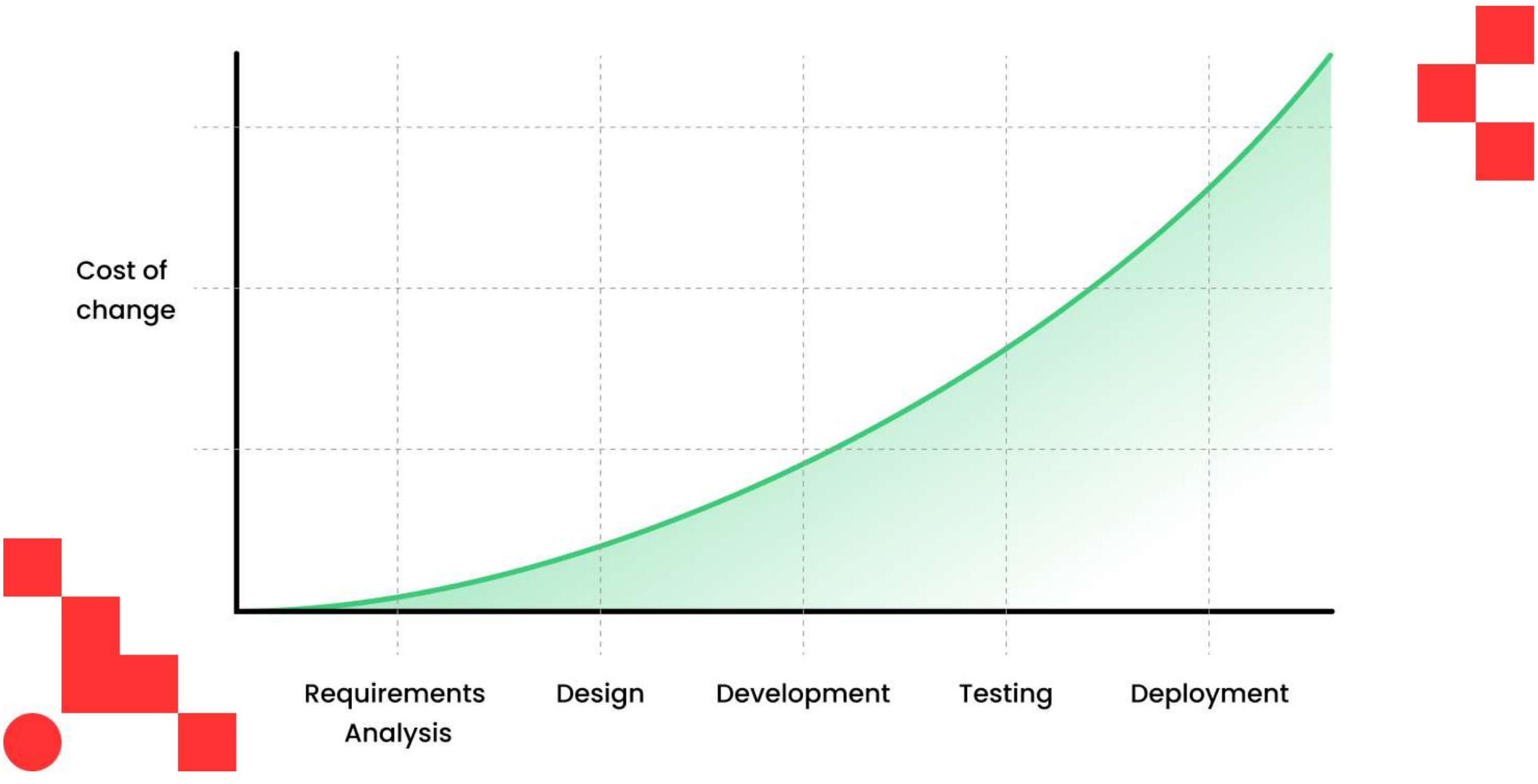
If technical debt is a result of rushed decisions and poorly written code that affect the integrity of your codebase and make it unwieldy, design debt is the outcome of hasty feature implementation and development compromises that damage the integrity of user experience. All the thoughtful design concepts you chose for the product are continuously ruined by the design verification procedures you skipped to save some time. Minor flaws and deviations accumulate with every sprint and eventually turn into an inconsistent, disjointed UI that delivers a disappointing experience. When that happens, no matter how many more features your product has, the target audience would still rather go with your competitor’s offer if it’s more aesthetically pleasing and provides a better, more compelling user experience.

QAREA

“Design debt is the outcome of hasty feature implementation and development compromises that damage the integrity of user experience. Minor flaws and deviations accumulate with every sprint and eventually turn into an inconsistent, disjointed UI that delivers a disappointing experience.”

WHERE DOES DESIGN DEBT COME FROM?

While a faster time to market may be a tempting goal, it should never be achieved at the cost of design quality. Otherwise, the usability and consistency of your design will naturally start deteriorating. Without proper design verification and validation procedures, every incremental change, every new element or feature introduced into the design will slowly ruin the structural integrity of your product. Old features will become stale and the whole thing will suddenly look like it had no design direction whatsoever—a Frankenstein monster of disjointed elements looking as though they were patched together without enough thought given to the long-term consequences.



The cost of design fixes rises exponentially with every step of the SDLC

To make it more relatable, here are a few examples of the most common causes of design debt:

1. Your team starts the project with assumptions. User goals and the problem you're trying to solve with your product are not properly researched or tested during the planning and design stages. This leads to floating specifications, confusing navigation, and poor user journeys.
2. The scope of the project is not properly defined, managed, or documented. Your team is dealing with changing/growing requirements and tight deadlines at the same time. This makes it extremely hard for them to work on every feature with the same care.
3. There is no unifying plan directing the project. Designers have to conceptualize the product following their personal viewpoints which leads to conflicting opinions, inconsistencies, lack of cohesion in design, or misinterpreted product vision.
4. Chasing short-term goals at the expense of design hurts your product's long-term viability. The design gradually becomes stretched beyond its original intent (elements are added without due consideration and feel forced into layouts).
5. The current state of your product's UX/UI and the general design direction are disregarded when designing a new feature (e.g. your team is too focused on experimenting with the feature, trying to revolutionize it without paying attention to whether it fits in well with the rest).
6. Due to the lack of resources, poor communication, or errors in specifications, inexperienced designers are tasked with a job beyond their capabilities which leads to poor design choices, undue complexity, and logical flaws.
7. You lack proper communication and workflows between designers, developers, and QA: designers hand off a feature to devs but are not involved in its implementation processes. This causes potential design deviations and flaws to be left unnoticed or addressed at the last moment, causing delays, extra costs, and rework.

WHEN BAD DESIGN CAN COST YOU

When left unchecked, design debt can sometimes cause a lot more trouble than a bad first impression, hurt user satisfaction, or loss of profit. And the problem is not nearly limited to software development. Here’s one amusing real-life example for you.

Constructed by Uruguayan architect Rafael Viñoly in spring 2014, the commercial skyscraper on 20 Fenchurch Street in London was nicknamed “The Walkie Talkie” for its distinctive design. And while the building’s appearance is rather debatable, it’s not the way it looks that made it so notorious back in the day.

The reason behind all the trouble the 38-floor skyscraper caused was its concave design. The building was designed in a way to expand towards the higher floors, which basically turned it into a huge curved mirror. During the building's construction in summer 2013, this huge magnifying glass started reflecting concentrated sunlight that was six times stronger than normal, effectively unleashing a “death ray” of up to 243 °F (117 °C) onto the streets below. For about two hours each day, the incredibly powerful beams were capable of effectively cooking everything in their way. There were numerous reports of parked vehicles being horribly distorted with paintwork completely melted off.

20 Fenchurch Street aka the “Walkie Talkie”

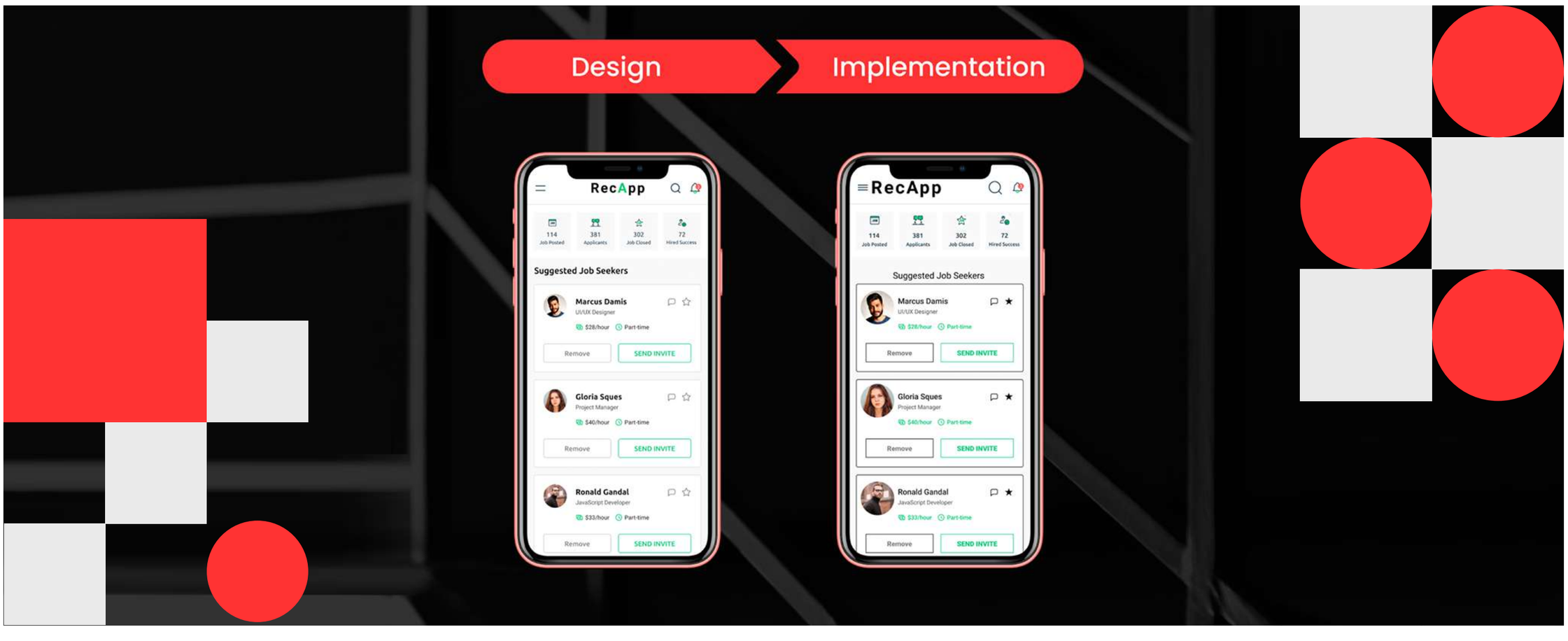
Shortly after the incident, the skyscraper was nicknamed “Walkie Scorchie” and several parking bays in the area were temporarily closed as a precautionary measure. In 2014, a series of vertical fins were installed on the higher floors of the tower as a long-term solution to the scorching problem. Integrated to the outside of the skyscraper’s windows, the fins could be angled to stop the beams from burning through unsuspecting locals and their property. But even with one problem out of the way, the terrible design choices behind the Walkie Talkie were not finished terrorizing the city just yet.

In July 2015, another issue revealed itself when Rafael Viñoly's skyscraper was accused of creating a severe downdraught effect. Apparently, the very same concave design had an unexpected impact on wind strength. When strong gusts of wind collided with the curved facade of the Walkie Talkie head on, the wind got redirected downwards at incredible speed and pressure. The downdraught was reported to have blown people over and ripped signs off nearby buildings.

The Walkie Talkie building is a great example of complete disregard for design verification and the consequences it can have for your business and reputation. Following all the distress caused by the building’s faulty design, the City of London Corporation has even started demanding independent assessment and verification of property developers' design reports at the planning stage of the project. Royal Town Planning Institute described the building as “a daily reminder never to let such a planning disaster ever happen again.”

20 Fenchurch Street aka the “Walkie Talkie”

DESIGN QA AND HOW IT HELPS IN PREVENTING DESIGN DEBT



Example of low-quality implementation of the UI design, showing noticeable deviations from the expected output

After helping numerous businesses and software development companies around the world deal with their quality assurance and software testing challenges, we realized that design debt is a very common but rarely talked about problem. It is something many projects that came to us seeking help didn't take seriously before it hit them hard—small design issues, ignored from iteration to iteration, accumulated and suddenly creeped up on them. Here's how it usually happens:

Preparing for the next sprint, the project manager realizes that there is something wrong with the UI: the layouts look wonky; font sizes and colors are not what they should be; alignments, animations, and micro-interactions are broken. The team gathers a meeting. They take the build, compare it to the mockups and prototypes, trying to figure what's wrong. They see that the actual UI in development is way off what it was originally designed to look and feel. The project manager is confused, the stakeholders are pissed, the development team is demoralized, and the only question on everyone's mind is: "Where did it all go wrong?" The project ends up in a tight corner where they either invest considerable resources to rework the whole thing, or knowingly deploy a faulty design with a risk of being booed and refused by the customers.

Many software development projects today find themselves in the same situation asking the exact same thing. They wake up to an important lesson of just how essential the design QA procedures are to avoiding being caught between a rock and a hard place.

QAREA

WHAT IS DESIGN QA?

“Design QA is a verification and validation process designers conduct during every iteration to check if the UI of the actual build has any inconsistencies or deviations from the expected design output. In short, this means making certain that the design didn’t get messed up in the course of its implementation by developers.”

Maintaining consistency in design is one of the main responsibilities a designer carries on a software development project. To produce software of desired quality, a designer’s job should never end with handing off features to the development team and moving on to designing the next. As your product is being developed, designers need to actively observe its implementation and participate in the verification and testing processes during every iteration. They should cross-check the UI in the build against the handed-off designs. And in case they find any inconsistencies, they should make sure the developers refactor the UI code, prioritizing the necessary design fixes before deployment.

Design verification should be treated as an essential part of software development workflows on par with other quality assurance procedures. If a company does not incorporate design QA in the development and verification processes, pile after pile, small design issues will infest the code until the project has no other choice but to do a costly redesign, undermining all of the research and brainstorming conducted prior to development.

WHY GOOD UX/UI DESIGN MATTERS

The main reason why most teams focus on sprint speed, prioritizing feature delivery over visual integrity, is that they don't understand the real value of design. Lots of projects think that people won't see the difference between a well-designed UI and its poorly coded doppelganger. But they are wrong. Good UX/UI design can make a huge difference between a successful product with great ROI and a failed, struggling one. Companies like Amazon, Apple, Google, and Facebook invest a lot of time and effort in refining their designs since every dollar invested in usability brings a return in the range from 2 to 100 dollars.

This is best illustrated with a quote from Dr. Claire-Marie Karat, from the IBM Thomas J. Watson Research Center.

"With its origins in human factors, usability engineering has had considerable success improving productivity in IT organizations."

Dr. Clare-Marie Karat, IBM Thomas J. Watson Research Center

She continues with an example:

"A major computer company spent $20,700 on usability work to improve the sign-on procedure in a system used by several thousand people. The resulting productivity improvement saved the company $41,700 the first day the system was used. On a system used by over 100,000 people, for a usability outlay of $68,000, the same company recognized a benefit of $6,800,000 within the first year of the system's implementation. This is a cost-benefit ratio of 1:100."

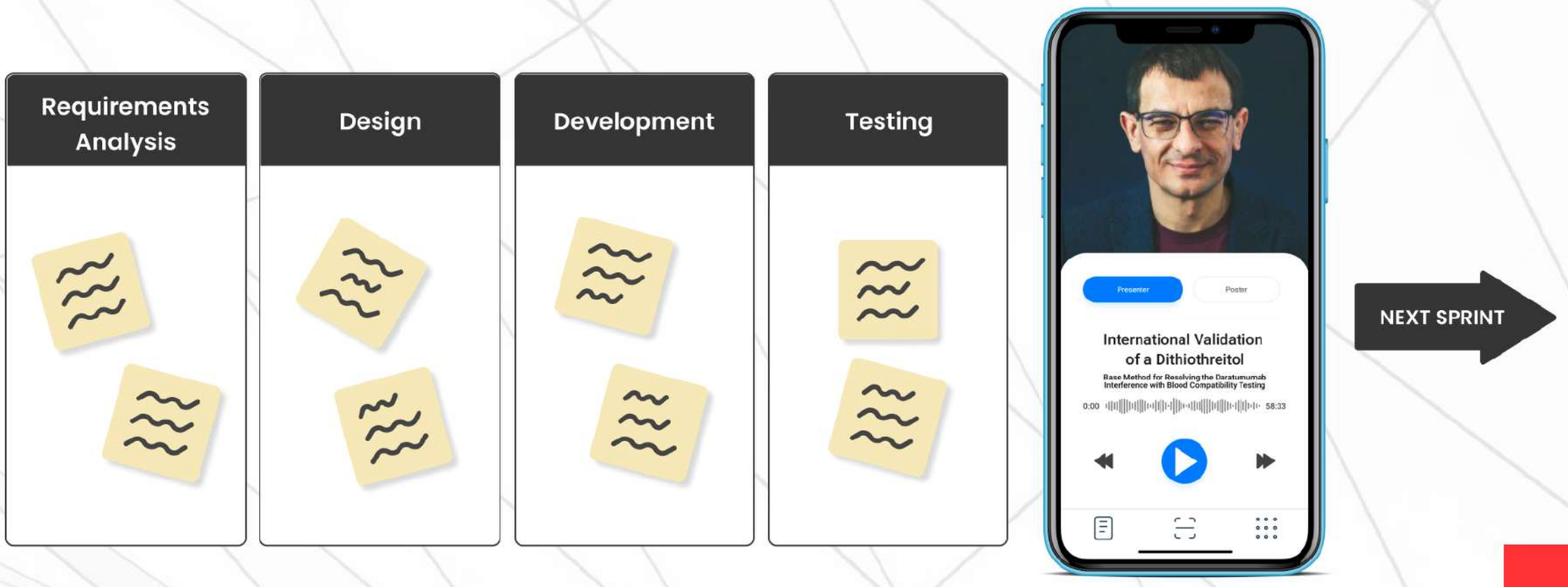
And while it may be a tough challenge to fix all the inconsistencies during every sprint, a proper proactive approach and timely reactive measures can help your team effectively deal with design debt and improve the end value of your work.

PROACTIVE SOLUTIONS TO DEALING WITH DESIGN DEBT

Make design QA a part of your software development workflow

A typical workflow in iterative software development looks something like this:

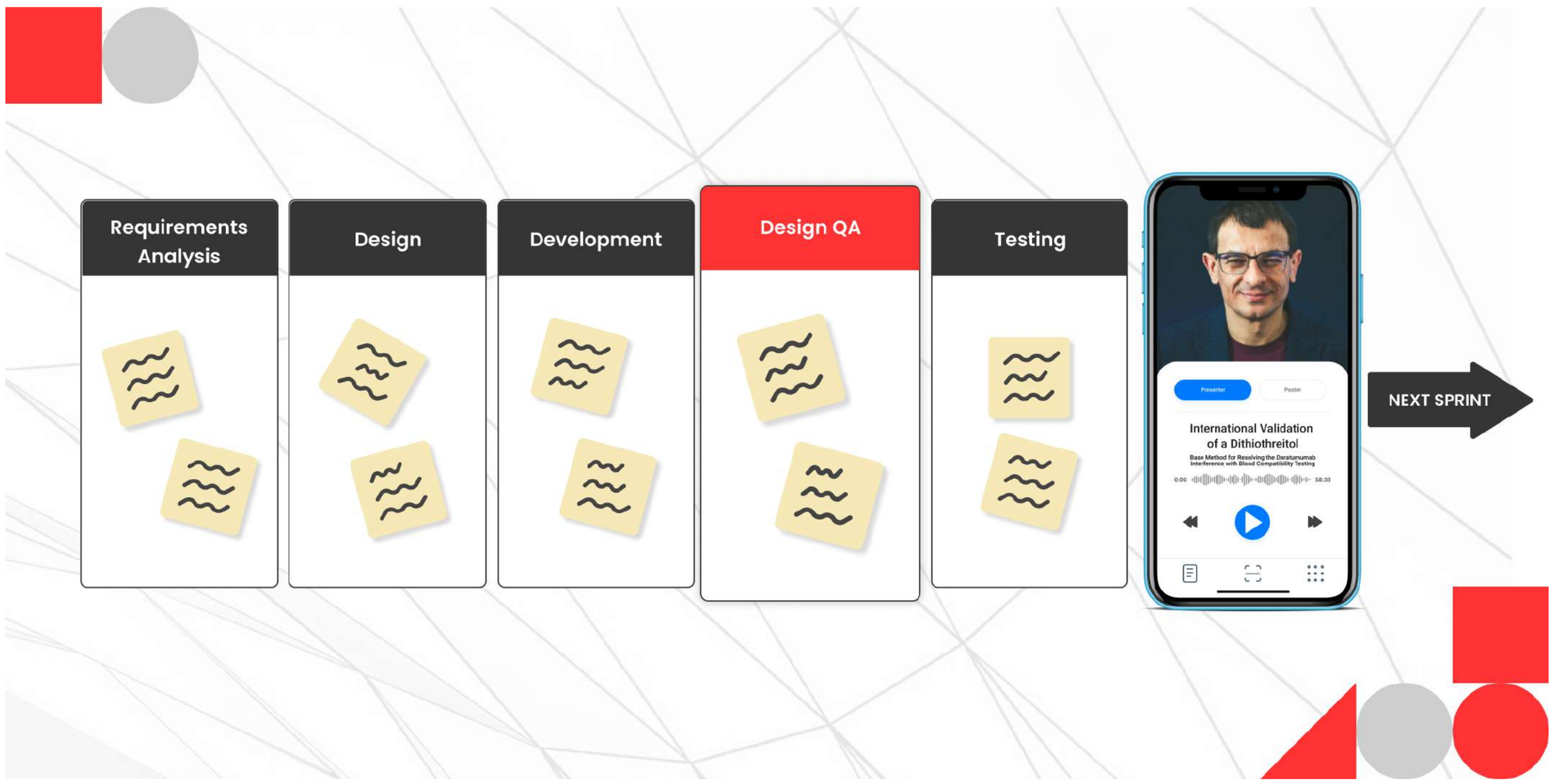
1. The team receives the requirements for a new feature.
2. Designers create mockups/prototypes with design specifications and hand them off to the devs.
3. Developers implement the feature and pass the ticket on to QA.
4. QA conducts a round of code reviews and testing to verify the feature.
5. Developers fix some of the issues found and backlog the rest for the next sprint.



A typical workflow in iterative software development

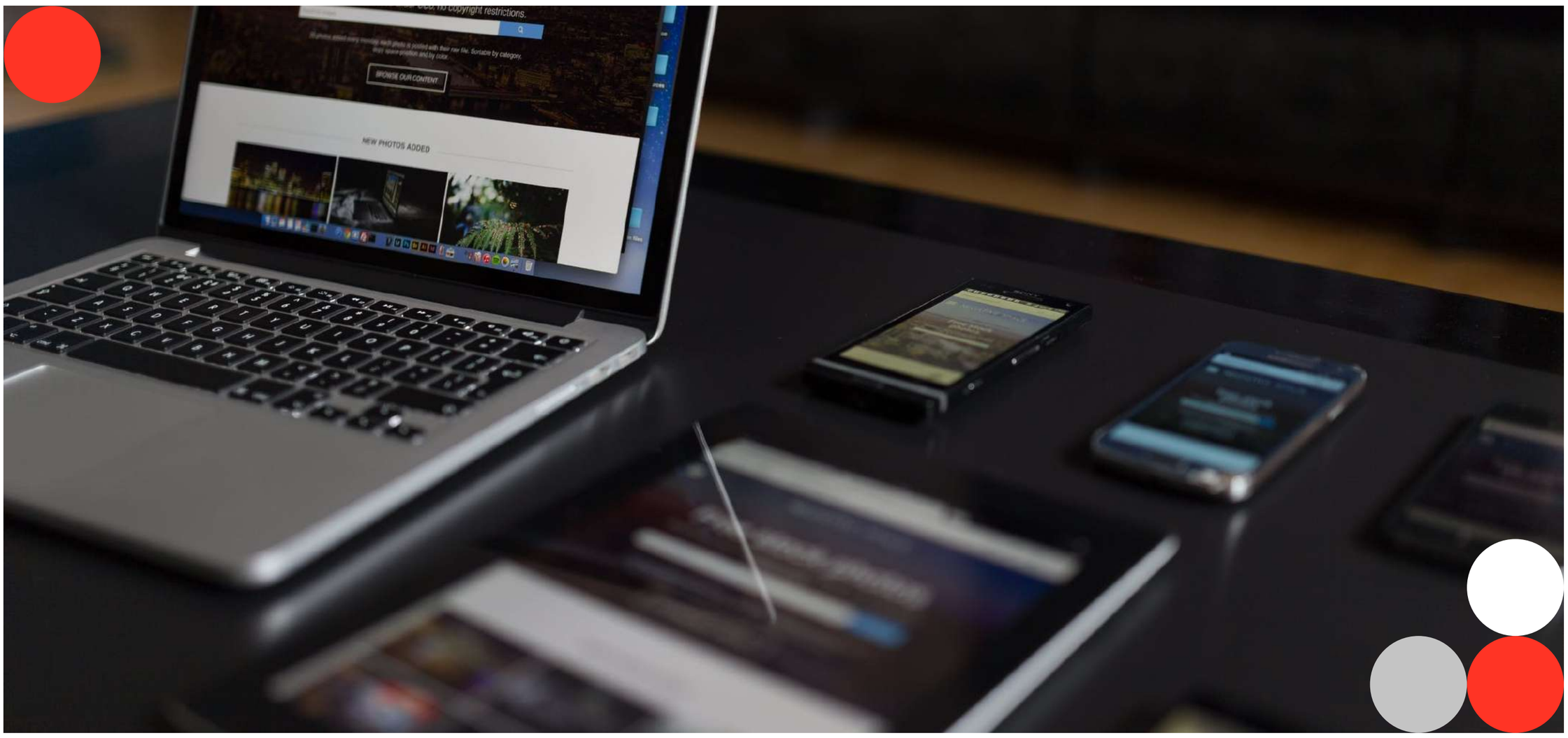
The downside of such workflow is that the ticket is simply being relayed from one team to the other. When the team is done with one task, they immediately move on to the next. Thus, designers provide the mockups and specifications to the devs but do not follow the implementation process. Then, the ticket goes straight from development to testing without any verification from designers. In a cycle where designers have no say during the development and QA processes, it is pretty hard to guarantee consistency and integrity of design in a finished product. While a good QA team will guarantee optimized code structure, your user interface may still get out of place without your designers' overwatch.

You can't make a successful product when everyone on the team is only concerned about their own piece of work. It's by taking collective ownership over the project, communicating and reviewing each other's work your team can be sure everyone is on the same page, delivering the desired result. What you can do to avoid messy design from happening is put design QA as a step on your task board. By doing so, you make design QA a deliberate part of your software development workflow—something your team cannot simply ignore and skip.



The high-quality design is simple and solves the user's problems

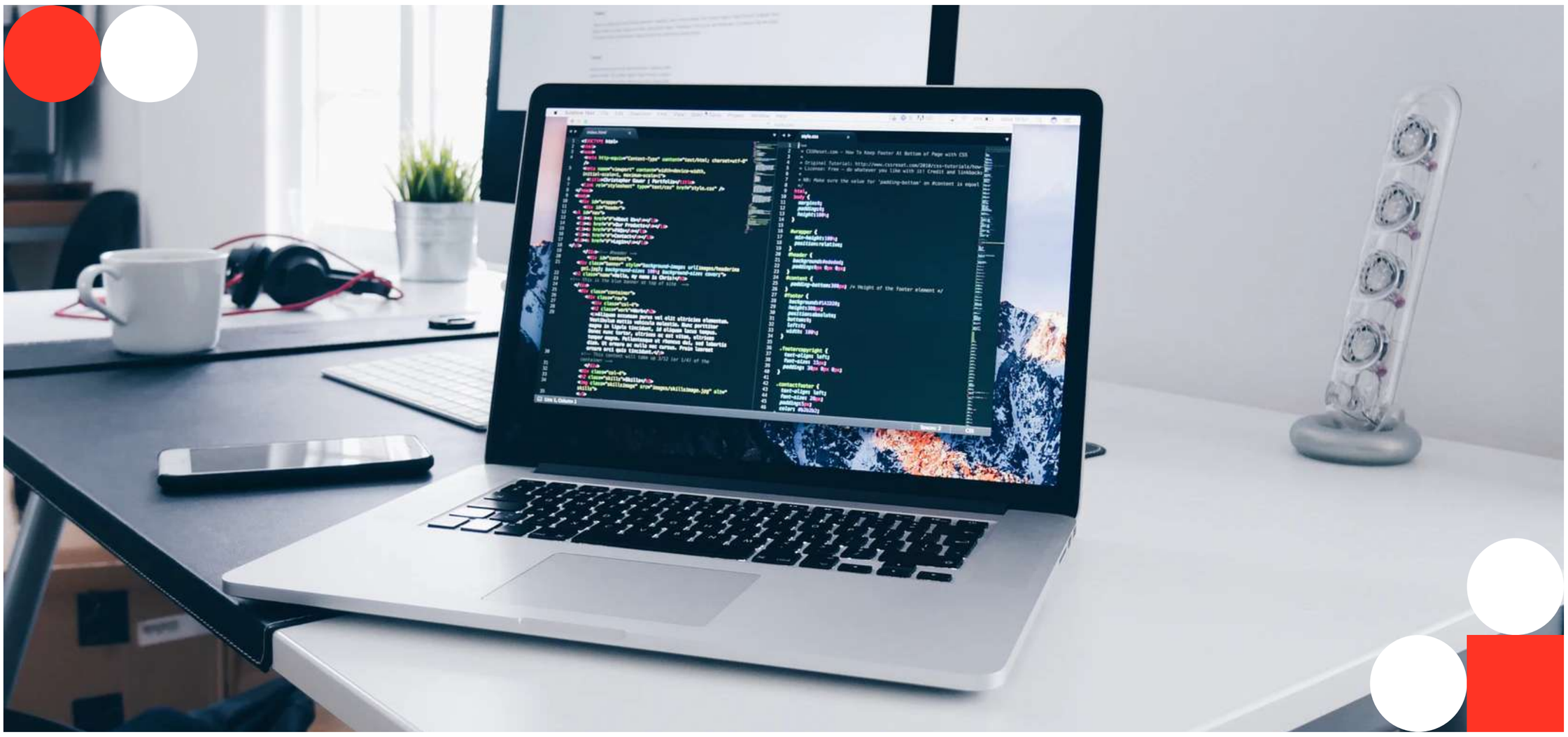
It may look like an additional level of testing—which it totally is—but it is also an additional layer of protection of the integrity of your UX/UI design. By making design QA a part of your workflow, you promote open and honest communication. The team becomes aware of a design issue as soon as it arises and together work out the best solution to it. Close collaboration and mutual support between designers and devs in the course of development will ensure your design choices don't lose their value after the release.



Your design output does not deteriorate from iteration to iteration, effectively saving you from the risks of costly redesign and reengineering. Thus, you can put these resources to good use in marketing or the future extension of your product with new features. This also means significantly less UI issues to deal with during the testing phase, which speeds up the QA process.

Involve developers in the design process

If designers can be a part of the feature implementation process, why can’t developers participate in design? By means of simple collaboration or actual pairing when working on a feature, you can make the design and development processes better aligned with each other.



Involving developers in the design process will give them a better understanding of your design decisions, help them find better development approaches to perfectly match your design intentions. Developers’ feedback in the process of brainstorming and conceptualizing for a new feature, as well as during design reviews, can provide valuable insights—help further refine the designs, work out solutions for the smoothest, debtless implementation. Such collaboration creates transparency, improves communication on the project, and helps avoid any potential inconsistencies.

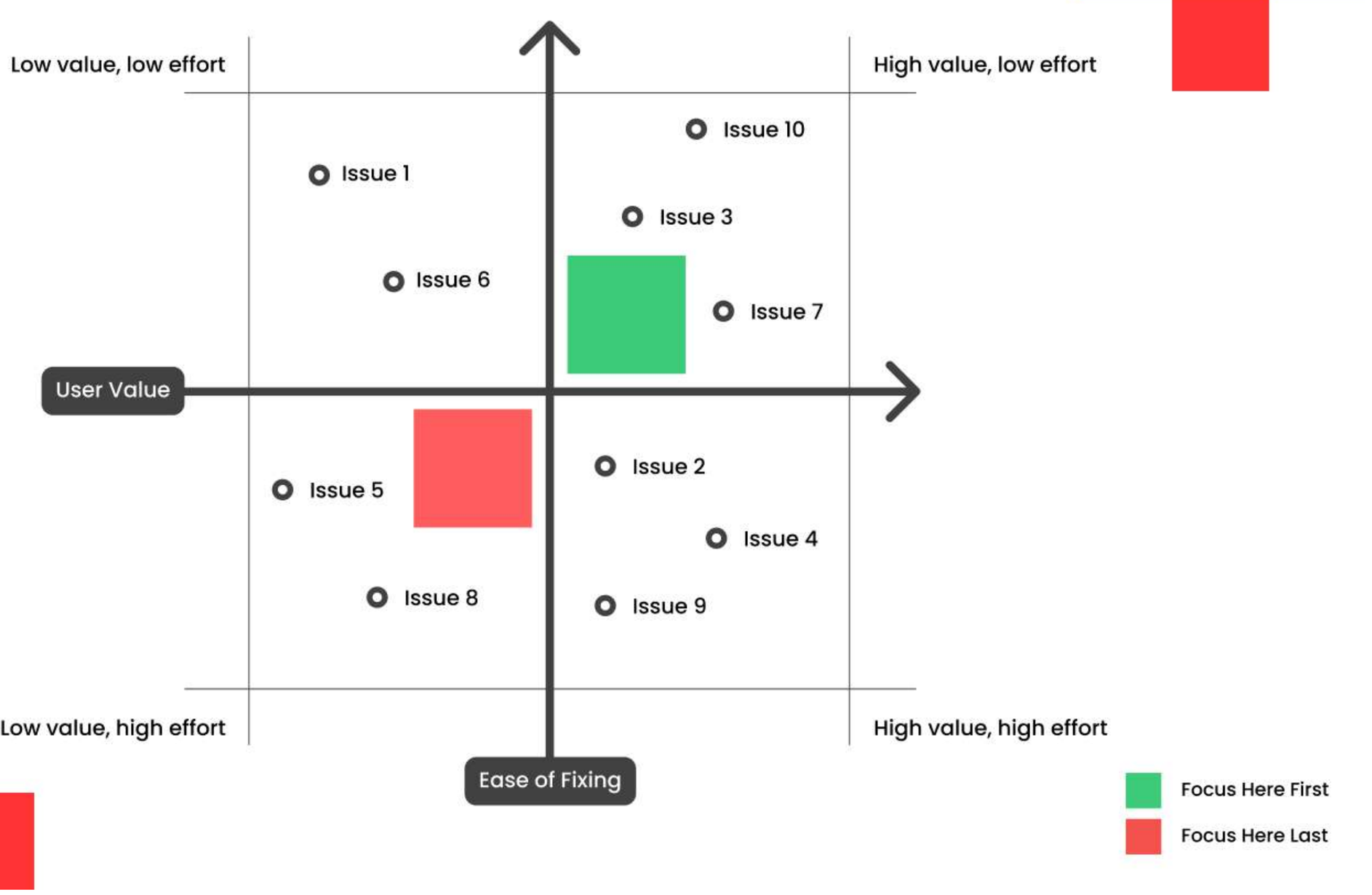
With tools like Figma, Zeplin, and InVision, your design and development teams can bridge the gap between design and CCS by sharing precise data and timely feedback on the same platform. Using these collaborative design tools, you can drastically improve the quality of your design hand-off process. A design spec document becomes much faster and easier to prepare since a huge chunk of these specifications (alignment, padding, font sizes, etc.) is already accessible through a convenient tool.

Access to both the design and the CSS attributes in a shared file saves a lot of time for both designers and developers while constant communication improves your workflows and overall team efficiency. It helps your team make better design decisions and provides you with better development results.

Measure design debt and set the right priorities for fixes

The ability to assess the state of your design and decide what fixes must come in first is the most important part of keeping your design debt in check. It shouldn't take your designers and developers a ton of time if they do it together. Right after or during code review, they can go through the UI to see if there are any obvious design bugs or inconsistencies. Since developers are quite often on a very tight deadline, they will probably find a few here and there. Identifying them before testing will help QA engineers pay closer attention to the affected areas of the codebase and make related issues easier to find.

Instead of creating a feature in one sprint and fixing all the design issues in the next, designers can help developers identify these issues on the spot and prioritize their fixing in the order from the smallest to the most visible and impactful ones.



Severity prot for UX debt prioritization

When prioritizing what needs to be fixed ASAP and what can be carried on to the next sprint, consider two factors: **user value** of the affected area and the **ease of fixing**. It's all about urgency and importance. If the impact of the issue is high but it's pretty easy to patch up, fixing this issue right here right now should be your top priority. In case the flaw has low impact on usability but requires a ton of work to fix, it is better to backlog it in favor of fixing a more critical issue. The ability to effectively balance in-between these extremes will help you minimize design debt and stick to the expected output.

Build and maintain a design system

Designers' collaboration and communication with engineers, product managers, and QA as a part of the development workflow will help the team build a robust design system with well-refined style guides and pattern libraries that support every element in the product. Having a set of clear rules and convenient building blocks decreases the likelihood of design debt. It makes it easy for your development team to add new features avoiding design inconsistencies.

This will help your team keep focus on the entire project, help them avoid blindly chasing the development of new features and stacking up design issues in the backlog. Involved in the development and verification processes, a designer will provide a constant stream of timely feedback during every sprint. If there are any signs of UI decline, the designer will be the first to notice and immediately call for a round of refactoring to create the most value for users.

REACTIVE MEASURES TO ELIMINATING DESIGN DEBT

When your UI has already become somewhat inconsistent, the elements disjointed, and the new features look like they were added out of the context of each other, it may seem that redesigning the whole thing is your best and only option at this point. But that's not entirely true. Yes, a costly redesign may be the obvious last resort in dealing with design debt. However, you should never jump to such conclusions without considering your other options first.

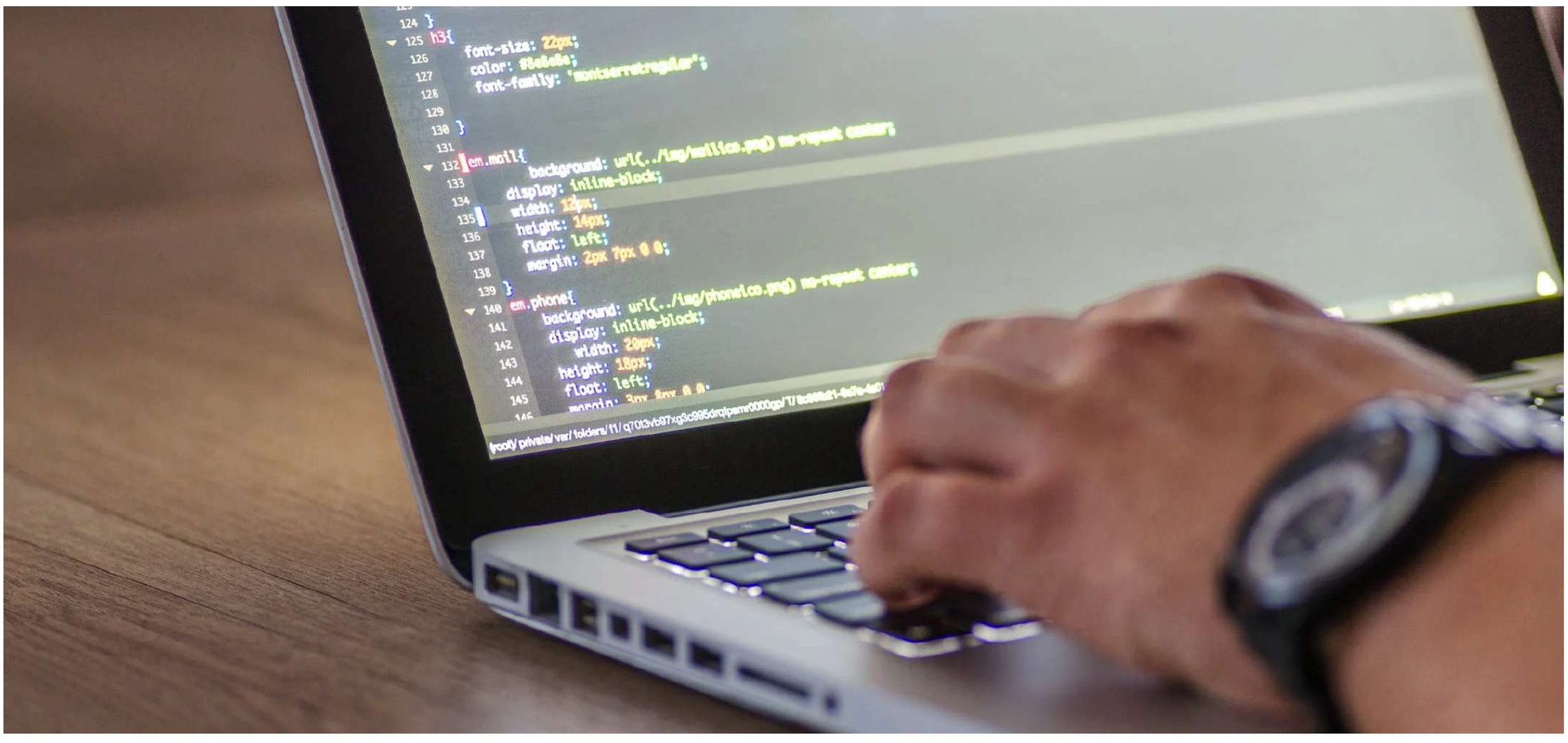
UI refactoring

You may have already heard about code refactoring. As a renowned software developer Martin Fowler put it in his book, "**Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure**." When you refactor, you are improving the design of your codebase after it has been written. But if you can refactor the code to improve its internal structure, why can't you improve the internal consistency of your UI by refactoring its design along the way?

Same as your codebase, over time the UI design can also suffer flaws, experiments, and modifications that will inevitably affect its integrity. But it doesn’t mean you have to take radical steps like a complete redesign to pay the debt down. Instead, plan for refactoring to save you the trouble.

With refactoring, you can take a corrupted UI and refine it into a well-designed, enjoyable experience without altering the software’s essential functionality in the process. The focus of design refactoring is to identify and remedy usability flaws and visual inconsistencies—elements that look unaesthetic, lack cohesiveness, and can cause user frustration. While keeping the user interface enjoyable and intuitive for your users, refactoring will also help the team refine your design direction, making future changes to the UI much easier to design and smoother to implement.

UI refactoring can be viewed as a large process conducted in a series of small iterations, baby steps you take to gradually introduce changes that will restore the planned look and feel of your UI without altering the essential functionality. Adjusting font and button sizes, aligning elements, fixing modals, animations, and micro interactions, balancing colors, rewording the texts, et cetera—bit by bit, your designers and developers consolidate the design patterns and reorganize the structure of the UI. Instead of a costly redesign, refactoring enables you to continuously smooth out rough edges, identify and remedy visual and usability flaws to keep your UI from taking on a substantial amount of unnecessary design debt.



Since refactoring is a complex transformation process that involves a lot of changes in the code, every step you take needs to be verified. This makes testing an imperative part of the refactoring process. You should treat refactoring as a maintenance project where the last thing you want to do is mess up the functionality of your software with a bunch of cosmetic fixes.

Therefore, every small iteration during refactoring must be followed by a scheduled round of testing. And while more frequent testing may seem drawn out, in fact, it makes the whole QA process a lot faster—smaller iterations mean a smaller set of changes introduced to the code. The issues can be isolated and fixed on the spot, which leaves you fewer errors to deal with in the final round of testing.

Unit testing

Successful UI refactoring is hardly possible without unit testing, a method used by developers to find and fix defects in individual units of code to ensure they meet their design objectives and behave as intended. In this case, a unit can be anything from a small button or micro-interaction to a feature as a whole. Units are tested independently to make sure every issue that may arise in the process is isolated and doesn’t affect the rest of the UI. Without proper unit testing, you risk breaking the functional components at one end of the UI when refactoring an element at the other.

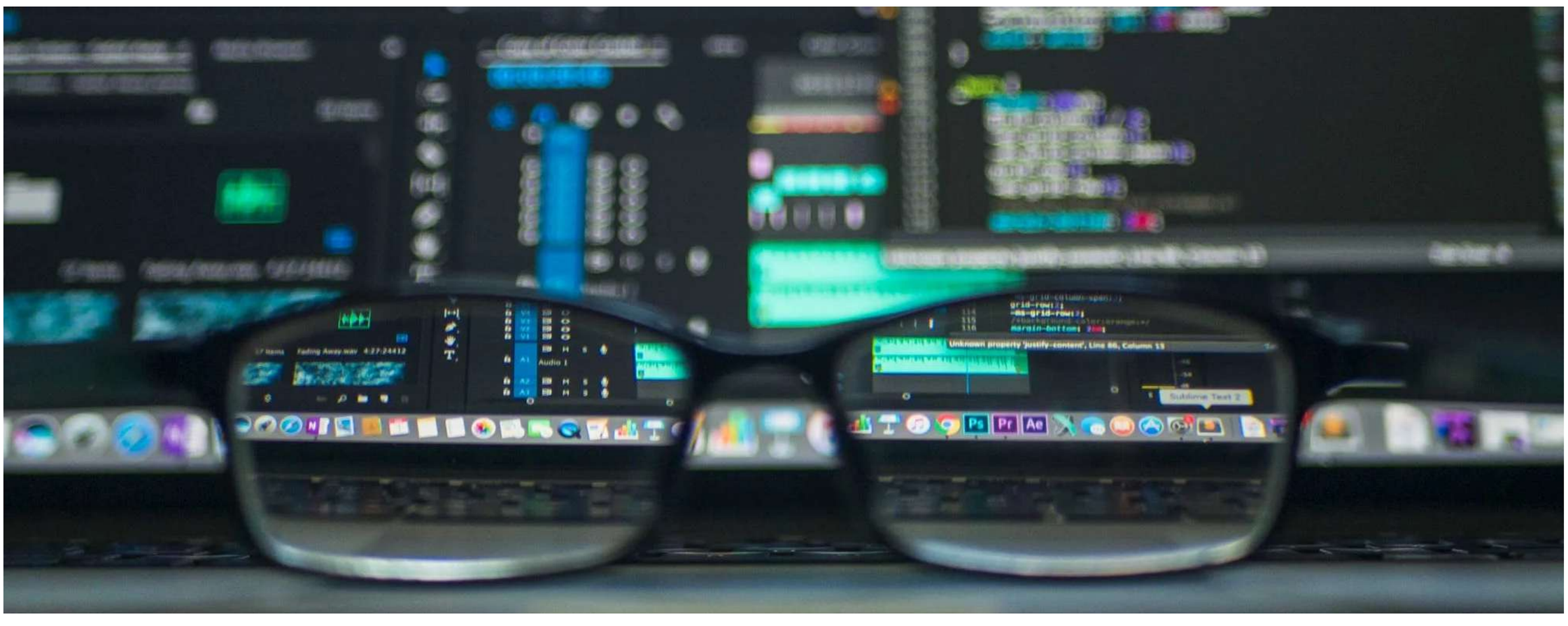
Unit testing provides the correct scope of testing to UI refactoring, making it an iterative process of timely and frequent changes to small units of code. It brings speed and stability to even large refactoring projects. Through many small steps, you can gradually fix the look and behavior of every visual element, pushing your design quality to where you want it to be.

And even though unit testing your UI can’t be automated, it’s better to facilitate further unit testing of the code behind your UI with automated testing tools like xUnit, Selenium, or Cucumber. This saves the dev team a lot of time with faster test feedback, which is crucial in agile software development.

Regression testing

After rounds of unit testing and refactoring the UI, it is crucial to test the impacted areas for possible regression. You want to make sure that the implemented changes and bug fixes in the units of code did not unintentionally break the looks, performance, and functionality of any other UI components. Your QA team takes the test cases from the previous version of your software solution, checks the validity of these tests, and works to improve their quality before re-running them against the new, refactored version.

A great way to approach regression testing is to maintain a proper version management system with clearly defined use cases. This enables your team to determine and create an appropriate subset of both positive and negative test cases to cover a use case. They can easily see which parts of the code affected by the refactoring belong to which use cases and test against these user flows to make sure no issues are left undetected.



After your team has generated a comprehensive suite of regression tests, you can automate the testing process by means of tools such as Git, Jenkins, and Bamboo. When set up, your team can use such testing systems to automatically re-run the tests, making defects easier to fix as they are identified faster. Such automated tests can mimic the behavior of real users and simulate their interactions with the UI. The program loops through user inputs in the user flow to verify that all the visual elements and interactions are in place and work well together after the changes. Regression testing automation can free up a significant part of your QA team to focus on other important tasks.

FINAL THOUGHTS

Iterative development and experiment-driven design approaches are undisputed champions of creating software that truly shines. After all, great products don't come from simply meeting business requirements and acceptance criteria. They come from getting to the core of user perception, needs, and wants and facing them with a simple, understanding, and kind user interface. However, it is important to not get too carried away in the process. Otherwise, you risk taking on a considerable design debt that you will inevitably have to pay through either UI refactoring or a complete redesign.

It's a common problem among today's software development companies since they put design verification at the very bottom of the to-do list. After designers finish prototyping and user testing the feature, they just hand it off to developers and move on to designing the next one. When built, the feature undergoes several rounds of QA and fixes where only after every functionality-related issue has been dealt with, designers get to verify if everything looks good on their part. This means the potential UX/UI deviations are addressed in the last place. Thus, fixing them costs a lot more of your time and budget than if a designer was involved in the QA process from start to finish—reviewing the implemented changes during every sprint to see if they had any negative effect on the design.

Companies need to understand the gargantuan value of having a high-quality UI and what it can bring to their business. Sure, making design QA a part of your workflow could be a pretty challenging endeavor at first. But do it the right way and you will succeed in both effectively avoiding accumulating design debt and reaping great rewards in the long run.

ABOUT US

www.qarea.com
business@qarea.com

As a software outsourcing company with 20 years of experience in delivering complex, custom solutions, we've built strong expertise in reliable architecture, efficient development practices, and transparent communication with clients.

ABOUT AUTHOR

Andrew M.
Technical Writer at QArea

Andrew is an insatiably curious geek who loves writing about technological innovation, business development, and digital transformation in the globalized world. Throughout more than 5 years of experience as a writer for different media, startups, and tech companies, Andrew has meticulously studied every aspect of the tech industry and loves sharing his knowledge with the international community.