

Chapter 1 / Capítulo 1



Primum non nocere: Software engineering
for medicine
ISBN: 978-9915-9876-8-2
DOI: 10.62486/978-9915-9876-8-2.Ch01
Pages: 1-26

© 2026 The authors. This is an open access
book chapter distributed under the terms of
the Creative Commons Attribution (CC BY)
4.0 License.

Chapter 1. The Digital Health Ecosystem

Annier Jesús Fajardo Quesada  

Eduardo Antonio Hernández González 

René Herrero Pacheco 

Rolando Javier Álvarez Pérez 

1 Granma University of Medical Sciences. Granma, Cuba.

2 University of Medical Sciences of Pinar del Río. Pinar del Río, Cuba.

“Technology in medicine is like a scalpel: in the right hands, it saves lives; in the wrong ones, it endangers them. The difference lies not in the instrument, but in the person wielding it and the respect with which they approach the human body.”

When a doctor listens to a patient with a stethoscope, they do not need to wonder if the instrument is calibrated. They know it's designed to amplify sounds, that it's been around for decades and that its function is clear to everyone. However, when the same doctor opens an app on their tablet to view an X-ray or when an algorithm suggests a possible diagnosis based on specific symptoms, the nature of the tool changes radically. We are no longer dealing with a metal tube and a diaphragm but with lines of code that make decisions, process sensitive information, and, ultimately, can influence clinical judgment. That collection of instructions written by programmers becomes, in fact, a medical device. In addition, as such, it must be treated.

Digital health is not a future promise but an everyday reality that is transforming medicine at breakneck speed. Apps that monitor chronic diseases, algorithms that interpret radiological images, artificial intelligence systems that predict complications, and telemedicine platforms that bring care to remote locations: all these technologies are already present in many of our hospitals and clinics. However, this new ecosystem is not just a collection of tools; it is a complex landscape where a wide variety of stakeholders converge—patients, professionals, regulators, developers, and managers—with often divergent interests, languages, and priorities. Understanding this landscape, its boundaries, and its unwritten rules is the first step toward navigating it safely and ethically.

1.1 SOFTWARE AS A MEDICAL DEVICE

The official definition of software as a medical device, or SaMD, is not a bureaucratic whim but a necessity for protection. The International Forum of Medical Device Regulators defines it as software intended for medical purposes that is not part of a physical device.⁽¹⁾ This means that an app that interprets heart rate using a smartphone's camera is a medical device, whereas a digital calendar for tracking appointments is not, even if it is used in a hospital. The line seems clear on paper, but in everyday practice it becomes blurred. What about an app that helps diabetic patients keep track of their meals? What if it also suggests insulin adjustments? Suddenly, the same tool has crossed an invisible line, and its consequences become potentially serious.

The implications of this distinction are enormous for everyone involved in the process. For the doctor who recommends an app to a patient, it means the peace of mind that comes from knowing that the tool has passed basic safety and efficacy checks. For the developer who builds it, it means assuming legal responsibility that goes beyond typical programming errors. In addition, for the patient—the most vulnerable link in the chain—it means that their health is not left in the hands of a program someone wrote in their garage with no guarantee other than their good intentions. That's why it's so important that anyone involved in creating health technology understands from the very beginning what they're actually building.

1.1.1 WHERE THE APP ENDS AND THE MEDICAL DEVICE BEGINS

Let us consider a specific example to help visualize this boundary. Imagine an app designed to help patients with hypertension keep track of their blood pressure readings. If the app merely stores the values that the user enters manually and displays them in a graph, it likely would not be considered a medical device. However, if that same app connects to a digital wristband, receives readings automatically, and, on the basis of them, advises the patient when to go to the emergency room or adjust their medication, then we are dealing with software that performs a medical function. The code hasn't changed drastically, but its purpose has—and with it, the responsibilities it entails.

This distinction is not new in the history of medicine, although the technological support is. For centuries, medical instruments have evolved from purely mechanical tools to complex electronic systems, and each technological leap has brought new questions about safety and regulation. What makes software different is its intangible nature and its ability to change without the user being aware of it. A dull scalpel is noticeable at the first cut; an algorithm that has been subtly modified may behave erratically without anyone noticing until it is too late. That invisibility of change is perhaps the most unsettling aspect of medical software and the one that demands

constant vigilance.

Regulatory bodies around the world have been trying for years to bring order to this frontier territory. The FDA in the United States, the European Medicines Agency, and other national agencies have published guidelines and documents intended to help developers classify their products and understand what level of scrutiny applies to them.⁽²⁾ However, these guidelines change, evolve, and sometimes become obsolete before they are even published because technology advances faster than regulators can analyze it. This mismatch between innovation and regulation is one of the great challenges of our time and a constant source of frustration for those trying to do things right.

For healthcare professionals entering this field for the first time, it can be overwhelming to hear terms such as regulatory classification, CE marking, or 510(k) clearance. However, it is important for them to understand these concepts because they will ultimately be the ones using these tools with their patients. Just as a doctor would not prescribe a drug without knowing its mechanism of action and side effects, they should not use medical software without understanding what guarantees it offers and what limitations it has. This analogy with pharmacology is more accurate than it seems: both are medical devices, both require studies to support their efficacy, and both can cause harm if used incorrectly.

Perhaps the most important point to take away from this introductory chapter is that medical software is not a special type of software but rather software that has been subjected to additional scrutiny because people's health depends on its proper functioning. A programmer can create a buggy game app, and the worst that will happen is that users will become frustrated and uninstall it. However, if a programmer makes a mistake in an algorithm that calculates radiation doses for cancer treatment, the consequences can be devastating. That asymmetry in the impact of errors is what justifies all the regulatory complexity we will explore in the coming chapters.

As we move through this book, we delve deeper into each of the aspects that make medical software such a fascinating yet demanding field. We will see how it is designed, how it is tested, how it is regulated, and how it is continuously improved. However, before all that, we needed to establish this foundation: to understand that when we talk about software in medicine, we are not talking about entertainment or productivity; we are talking about health. Health, as is well known, does not tolerate errors.

1.1.2 THE THREE FACETS OF RESPONSIBILITY: PHYSICIAN, DEVELOPER, AND PATIENT

When an app becomes a medical device, the chain of responsibility extends and becomes more complex in ways that are not always obvious at first glance. The physician who prescribes or recommends an app is not exempt from responsibility simply because it involves digital technology. In fact, their position becomes more delicate because they must evaluate not only the clinical utility of the tool but also its technical reliability—which is something for which many professionals have not received specific training. This duality of competencies—medical and technological—represents one of the greatest challenges of contemporary medicine and demands a new form of literacy that goes beyond traditional training.

The developer, for their part, enters a realm where their coding decisions have direct consequences for real people. It is no longer a matter of optimizing a database query or improving the user interface to enhance the appeal. Every line of code can affect a patient's diagnosis, treatment, or even survival. This pressure transforms the nature of programming work, which ceases to be a purely technical activity and becomes a practice with inherent ethical responsibility. The medical software developer must learn to think like a healthcare professional, anticipating usage scenarios that go far beyond standard test cases.

The patient occupies the final link in this chain but, paradoxically, is the one with the least information to assess the risks they are taking. When they download a health app onto their mobile phone, they rarely read the terms of service, much less verify whether the software has the appropriate health authorization. They trust—or should trust—that the regulatory system has filtered out potentially dangerous apps. However, the reality is a mo: there are thousands of health apps available in digital stores that operate in gray regulatory areas, offering functionalities that come dangerously close to medical practice without the corresponding safeguards. This situation of information asymmetry is particularly concerning in regard to vulnerable or low-income populations, who may rely exclusively on these tools for their healthcare.

The interaction between these three parties—the doctor, developer, and patient—creates an ecosystem of shared responsibility that is not always well defined in current legal frameworks.

When an incident related to medical software occurs, determining who bears responsibility can become a legal maze. Is the doctor who recommended the app responsible? The hospital that purchased it? The developer who programmed it? The platform that distributed it? These questions have no simple answers and vary significantly by jurisdiction, adding an additional layer of complexity to an already intricate field.

1.1.3 THE BLURRED LINE BETWEEN ASSISTANCE AND DIAGNOSIS

One of the most intense debates in the regulation of medical software revolves around the distinction between applications that provide information for the user to make decisions and those that make decisions for the user. This seemingly subtle difference has enormous regulatory implications. An app that displays a diabetic patient's glucose data and allows the patient to interpret trends operates on a different plane than one that, upon detecting an abnormal reading, automatically sends an alert to the emergency department or adjusts the insulin dose of a connected pump. In the first case, we use an information tool; in the second case, we use an automated clinical decision-making system.

The line between these two categories has become particularly blurred with the advent of artificial intelligence and machine learning. Modern algorithms can analyze complex patterns in health data and offer recommendations that, although presented as suggestions, carry an implicit authority that users—both professionals and patients—tend to accept without question. When an artificial intelligence system recommends a diagnosis based on the analysis of a medical image, the radiologist using it may be tempted to confirm that interpretation without the critical scrutiny they would apply to a human opinion. This phenomenon, known as automation bias, represents one of the most insidious risks of contemporary medical software.

Regulators have attempted to address this complexity by establishing criteria on the basis of the software's intended purpose and context of use. However, these classifications quickly become obsolete in the face of the speed of technological innovation. An application initially designed as a simple symptom log can, through a software update, acquire predictive analysis capabilities that place it in a completely different regulatory category. This inherent mutability of software stands in stark contrast to the stability of traditional medical devices, where a design change requires visible and quantifiable physical modification.

This issue becomes even more complicated when we consider the phenomenon of interconnected health ecosystems. A single functionality can be distributed across multiple applications and devices, each of which, when considered in isolation, might not qualify as a medical device but whose interaction produces significant clinical effects. For example, a nutrition app that analyzes food intake, combined with a physical activity monitor and a connected glucose meter, can generate personalized recommendations that are functionally equivalent to a medical treatment plan. Determining where regulatory responsibility lies within this distributed architecture is among the most pressing challenges for health authorities worldwide.

1.1.4 THE WEIGHT OF EVIDENCE IN A DIGITAL WORLD

The transition from software as a consumer product to software as a medical device entails a fundamental transformation in how evidence of its performance is required and evaluated. When a company develops an app for booking restaurant tables, it can launch it on the market, collect user feedback (interactions), and improve it iteratively. This agile development model, which is so efficient in the world of conventional software, clashes head-on with the principles of evidence-based medicine. A medical device cannot be improved on the fly on the basis of user complaints; it must demonstrate, prior to marketing, that it meets rigorous safety and efficacy standards through studies specifically designed for that purpose.

This requirement for premarket evidence creates an inevitable tension between the speed of technological innovation and the caution inherent in the healthcare sector. Software developers, accustomed to weekly or even daily release cycles, find medical regulations to be frustrating and costly obstacles. The timeframes for obtaining marketing authorization can extend over months or years, during which the technology has evolved significantly. This lag presents the paradox that authorized medical software may be technically obsolete before it reaches the first patient—a situation that has no parallel in traditional hardware medical devices.

The regulatory response to this challenge has been the creation of alternative authorization

pathways that seek to balance the need for evidence with the reality of technological innovation. Programs such as the FDA's Pre-Cert or the software change assessment pathways in Europe aim to establish frameworks where developers with a proven track record of quality can update their products more quickly.⁽³⁾ However, these initiatives are still in experimental phases and spark debates over whether they excessively relax safety safeguards. The medical community, legitimately concerned about patient protection, views any mechanism that might allow the commercialization of healthcare software without proper clinical validation with suspicion.

The quality of evidence required also varies significantly depending on the risk associated with the software. An app that calculates body mass indices from manually entered data does not require the same level of testing as an algorithm that analyzes electrocardiograms to detect arrhythmias does. Regulatory systems have established risk classifications that determine the intensity of scrutiny needed, but these categorizations face the challenge of anticipating emerging risks from technologies that do not yet exist. The history of medical device regulation is full of examples in which products initially considered low-risk turned out to have unexpectedly serious consequences, leading to a general trend toward regulatory caution.

• LESSONS FROM HISTORY: FROM THERAC-25 TO HOSPITAL RANSOMWARE

There is a human tendency to believe that technology's problems are new and that previous generations cannot imagine the ethical dilemmas and risks we face today. However, this belief, in addition to being naive, is dangerous because it deprives us of the wisdom accumulated through mistakes that cost suffering and lives. In the field of medical software, that history is written with tragedies that should be mandatory in any training program for both engineers and clinicians. Knowing them is not an exercise in erudition but a vaccine against the arrogance of believing that we will not make the same mistakes.

The most emblematic—and perhaps the most chilling—case is that of the THERAC-25 machine, a linear accelerator designed to administer radiation therapy to cancer patients. In the mid-1980s, this machine, manufactured by Atomic Energy of Canada Limited, incorporated software for the first time that fully controlled its operation, replacing the mechanical safety systems that had protected patients for decades. What no one could have imagined was that this software contained errors as subtle as they were lethal—errors that would cause patients to receive radiation doses up to a hundred times higher than prescribed, resulting in severe burns and, in at least three cases, death.

1.2.1 WHEN SOFTWARE LIES: THE THERAC-25 TRAGEDY

The most shocking aspect of the THERAC-25 case is not just that the software failed but also how it failed and how those responsible responded. The errors occurred under very specific circumstances: when the technician entered the treatment data in a certain way, the system entered an unexpected state and fired the electron beam without the device that was supposed to attenuate it. Worse still, the screen displayed an incomprehensible error message and, in some versions, indicated that no radiation had been administered when in fact it had. The operators, trusting what they saw on the screen, administered the treatment again, accumulating lethal doses on the same patient.⁽⁴⁾

Behind this tragedy lie lessons that remain relevant decades later. The first is that software does not tolerate half-measures: either it works exactly as it should, or it can fail in ways in which no engineer imagined during its design. The second is that blind trust in technology is a luxury medicine cannot afford. The technicians operating the THERAC-25 trusted what the screen showed them and that trust prevented them from suspecting that the system might be lying. The third lesson, perhaps the most uncomfortable, is that when a tragedy of this magnitude occurs, the natural tendency of organizations is to look for individual culprits rather than examine the systemic failures that made it possible.

Subsequent investigations revealed that software errors were not new to the THERAC family of machines. Earlier versions, THERAC-6 and THERAC-20, experienced similar failures, but those models had mechanical safety systems that prevented the administration of incorrect doses. When the THERAC-25 was designed, the software was deemed reliable enough to eliminate these backup mechanisms—a decision that saved costs but removed critical layers of protection. This phenomenon, known as the substitution of safety with software, remains a common pitfall

in the design of contemporary medical systems, where overreliance on the ability of the code to manage risks can lead to the removal of physical or human safeguards that previously provided redundancy.

The manufacturer's initial response to the first incident was characteristic of a defensive organizational culture. When hospitals reported that patients had suffered burns during treatment, the company attributed the problems to operational errors or specific patient conditions, systematically denying that the software could be the cause. Even after engineers identified the code error related to the condition check, the company underestimated the severity of the problem and delayed the implementation of fixes. This dynamic of denial and downplaying is not unique to the medical industry, but in this context, it had fatal consequences that could have been avoided with a more agile and transparent response.

The THERAC-25 case also illustrates the critical importance of the user interface in high-security systems. The error messages displayed by the machine were cryptic and technical and were designed from the perspective of the programmer who understood the internal code, not from that of the technician who needed to make urgent clinical decisions. When the system detected an anomaly, it displayed codes such as "MALFUNCTION 54" or simply "TREATMENT PAUSE," without clearly indicating that a potentially lethal overdose had occurred. This lack of effective communication between the system and the human operator transformed recoverable technical errors into irreversible tragedies—a lesson that remains relevant in the design of contemporary medical interfaces.

Ultimately, the THERAC-25 tragedy highlights the need for a safety culture that transcends organizational silos. The software engineers who wrote the code did not interact with physicists who understood radiation or with physicians who knew the clinical consequences of an overdose. Each group operated on assumptions about the work of the others that turned out to be incorrect or incomplete. This fragmentation of knowledge, combined with the pressure to bring the product to market, created a breeding ground where errors could propagate undetected until it was too late. The lesson that safety in medical software requires genuine integration across multiple disciplines remains fundamental decades later.

1.2.3 HIDDEN BIASES: WHEN ALGORITHMS PERPETUATE INEQUALITY

In this century, errors have evolved alongside technology, but their potential to cause harm has not diminished. In 2019, an algorithm widely used in the United States to identify high-risk patients and allocate healthcare resources was found to systematically discriminate against Black patients. The software, developed by Optum (the business services arm of the UnitedHealth Group), uses previous healthcare costs as a proxy to determine patients' health needs. For years, thousands of Black patients were denied care management programs and additional resources because the algorithm deemed their risk to be low, when in fact it was not. The algorithm had no malicious intent, but its consequences were real and preventable. (5)

This incident represents a new generation of medical software failures: those that arise not from explicit programming errors but from the characteristics of the data used to train artificial intelligence systems. Unlike traditional bugs that produce incorrect behavior in specific circumstances, biases in training data generate systematic discrimination that affects entire population groups on an ongoing basis. Optum's algorithm functioned technically correctly: it made accurate statistical predictions on the basis of available historical data. The problem was that historical data reflected decades of disparities in access to healthcare, where Black patients, despite having the same health needs as white patients did, had historically spent less on healthcare because of systemic barriers related to access, transportation, knowledge of healthcare systems, and socioeconomic inequalities. The algorithm learned that lower spending equaled lower needs, creating a cycle of invisible exclusion automated by technology.

The gravity of this type of error lies in its apparent objectivity. While human doctors may be aware of their own biases and consciously compensate for them, an algorithm presents its conclusions with the authority of numbers, without indicating the limitations of its database or the underrepresented populations in its training. Black patients affected by the Optum system never knew that they had been misclassified; they simply did not receive the preventive interventions and care management programs they would have received had they been white patients with the same actual risk profile. This opacity in decision-making, combined with the perceived authority of computational systems, makes algorithmic biases particularly difficult to detect and challenging.

The case takes on additional dimensions when we consider its scale and duration. Unlike a traditional software error that affects specific users at specific times, a population health algorithm can influence decisions about millions of people for years before anyone identifies the underlying bias. In the case of Optum, the algorithm had been operating for years before academic researchers at the University of California, Berkeley, discovered the racial disparity. During that period, harm accumulated silently in the form of unmanaged chronic diseases, higher healthcare costs due to avoidable hospitalizations, and, presumably, premature deaths that could have been prevented with early intervention. This study revealed that this racial bias reduced the number of black patients identified for additional care by more than 50%.

The institutional response to this discovery illustrates both the progress and the limitations of our algorithmic governance mechanisms. Once the problem was identified, the researchers worked with Optum to reformulate the algorithm, adjusting it so that it no longer used costs as a proxy for health needs. The reformulation eliminated 84% of the racial bias in the results. However, the correction could not undo years of accumulated harm or identify the specific patients who had been harmed. Furthermore, the review focused on the identified racial bias, without systematically examining other potential biases related to gender, socioeconomic class, or geographic location that might have been operating simultaneously within the same system.

This incident forces us to reflect on an uncomfortable truth: software is not neutral, even though many programmers believe it is. Every line of code incorporates the assumptions, biases, and limitations of those who write it and the data used to train it. An algorithm that determines which patients deserve priority care, which treatments are most effective, or which diagnoses are most likely, is not an objective tool but a mirror that reflects our own limitations. Recognizing this is the first step toward trying to correct them, or at least toward not unconsciously perpetuating injustices that already exist in the healthcare system.

1.2.4 DIGITAL WARFARE: WHEN HOSPITALS BECOME TARGETS

Alongside these well-intentioned failures, a threat of a very different nature has emerged: cyberattacks targeting healthcare infrastructure. In 2017, the WannaCry ransomware paralyzed dozens of hospitals in the United Kingdom, forcing the cancellation of thousands of appointments and surgeries and putting patients who depended on networked equipment at risk.⁽⁶⁾ The attackers were not seeking to harm patients directly but rather to make money; however, the collateral effect was the same: systems that stopped working, doctors without access to medical records, and operating rooms that had to close. The incident demonstrated something that until then had been merely theoretical: cybersecurity is not a matter of data theft but of patient safety.

Hospitals have become prime targets for cybercriminals for a very simple reason: they cannot afford to be offline. A store can close for a day because of a cyberattack; an emergency department cannot. This asymmetry makes healthcare institutions easy targets, and the response must be not only technical but also organizational and cultural. Doctors and nurses must be aware that a phishing email is not just a nuisance but also a potential gateway to a clinical catastrophe. In addition, hospital administrators must understand that investing in cybersecurity is not an administrative expense but a clinical safety measure as important as having fire extinguishers or evacuation plans.

The spread of WannaCry in the British National Health Service revealed systemic vulnerabilities that went beyond individual technology. The malware exploited a known vulnerability in Windows systems that Microsoft had patched months earlier, but many hospitals had not applied because of the complexity of updating legacy systems interconnected with specialized medical equipment. This tension between the need to patch vulnerabilities and the risk that updates might interfere with the critical functionality of certified medical devices represents an ongoing dilemma in healthcare infrastructure management. Hospitals are caught between exposure to known threats and the possibility that a software update might disable a patient monitor or a medication dispensing system.

The clinical impact of WannaCry was immediate and tangible. In several hospitals, radiological equipment is affected, preventing urgent diagnoses. Bed management systems stopped working, hindering the coordination of admissions and discharges. Electronic health records became inaccessible, forcing professionals to work with pen and paper, without access to complete histories or recent test results. In some cases, ambulances had to be rerouted to unaffected hospitals, increasing response times for critically ill patients. Although no deaths directly attributable to the attack were documented, the deterioration in the quality of care during

the incident exposed the fragility of modern healthcare systems in the face of digital disruptions.

The response to the attack also highlighted deficiencies in cyber incident preparedness and coordination within the healthcare sector. Many hospitals lacked specific contingency plans for ransomware attacks, and communication between affected facilities and authorities was initially chaotic. Recovery took days or weeks in some cases, during which healthcare services operated in degraded mode. This episode catalyzed an awareness of cybersecurity as an essential dimension of patient and security, leading to significant policy and budget reviews in healthcare systems worldwide.

However, five years after WannaCry, the situation remains concerning. Attacks on hospitals have intensified and become more sophisticated, with criminal groups developing ransomware specifically designed for healthcare environments and aware of their vulnerability and willingness to pay. The COVID-19 pandemic exacerbated these risks by forcing accelerated digitization of services and an expansion of remote connectivity that was not always accompanied by commensurate security measures. Hospitals' attack surface has grown exponentially with the proliferation of connected medical devices, many of which were designed without robust security considerations and are difficult to update or replace.

There is one constant in all these incidents that is worth highlighting: None was caused by malicious intent. The engineers who designed the THERAC-25 did not want to kill patients. The programmers of the biased algorithm did not intend to discriminate against anyone. The developers of the system that deleted medical records believed that they were making technical improvements. The problem is not malice but ignorance, a lack of perspective, and the inability to imagine how a technical system will interact with a complex human environment. Moreover, ignorance can be combated only through education, humility, and the conviction that in medicine—unlike other fields—there is no room for trial and error.

That is why this book does not begin with abstract theory but with these stories. Before learning to program for medicine, before understanding technical standards or validation methodologies, one must develop a special sensitivity to potential harm. We must understand that every design decision, technical choice, and line of code can have consequences that we will never know but must anticipate. That sensitivity, that awareness of risk, is what separates a good software developer from a good medical software developer.

History also teaches us that mistakes tend to be repeated when they are not properly documented or when the lessons learned are not passed on to new generations. Every serious incident involving medical software has generated reports, investigations, and new regulations, but this knowledge often remains confined to specialist circles. Doctors who use these systems rarely know about past failures, and young programmers are often unaware that there was a time when a linear accelerator could kill patients because of software error. Breaking down this information silo is one of the motivations behind this book.

When we look back and see this succession of errors—from the most basic programming flaws to the most sophisticated algorithmic biases, including deliberate cyberattacks—an inevitable question arises: are we better prepared today than we were thirty years ago? The answer is mixed. Technically, yes, we have better tools, better methodologies, and a better understanding of the risks. However, we also have systems that are far more complex, interconnected, and invisible, where a failure in a small component can propagate in ways that we are only beginning to understand. In addition, we still have the same human frailty as always: the tendency to place too much trust in what we build and to forget that, behind every line of code, there are people whose health depends on our having done our job well.

1.3 MAPPING THE LANDSCAPE: SAMD, TELEMEDICINE, IOMT, AND BIG DATA

Let's imagine for a moment that we travel back in time to the late 20th century and explain to a doctor of that era what his practice will look like twenty years later. We'll tell him about apps that remotely monitor his patients' hearts, algorithms that analyze X-rays with greater precision than the human eye, and devices that collect millions of data points from each person and process them to predict diseases before the first symptoms appear. They would likely look at us with skepticism, perhaps even disbelief. However, the everyday landscape of contemporary digital health is not science fiction. A vast, sometimes chaotic, yet fascinating territory that we should understand in some detail before venturing to build upon it.

The first concept we must place on this map is that of software as a medical device, the acronym SaMD that we introduced in the first section. It is now worth delving deeper into its practical

implications because not all healthcare software deserves the same regulatory consideration. The classification proposed by the International Forum of Medical Device Regulators ((1)) distinguishes four categories based on two variables: the severity of the disease it addresses and the importance of the information it provides for clinical decision-making. Software that helps in the diagnosis of mild illness faces fewer requirements than software that guides intensive treatment does, and an algorithm that simply informs the physician will require fewer controls than one that makes automatic decisions without human supervision does.

This gradation is not an unnecessary bureaucratic complication but a rational attempt to align the level of scrutiny with the level of risk. Consider, for example, an app that helps patients with seasonal allergies track their symptoms and identify seasonal patterns. If it fails, the worst that can happen is when the patient experiences minor discomfort or confusion. However, now consider software that controls insulin infusion in an intensive care unit. A failure in that system can cause severe hypoglycemia with irreversible neurological consequences within minutes. It makes sense, therefore, that the second is subject to much stricter controls than the first, even though both are technically health software.

1.3.1 SAMD: WHEN CODE BECOMES PRESCRIBER

The classification of software as a medical device represents one of the most significant regulatory advances of recent decades, although its practical implementation continues to generate intense debate. What distinguishes SaMD from other software is its ability to directly influence medical decisions without being part of specific hardware. This seemingly clear definition becomes murky when we examine borderline cases that abound in today's market. An app that uses a phone's camera to analyze the color of a skin lesion and suggests whether it might be malignant operates in a gray area where information is confused with recommendations and is recommended for diagnosis.

The International Forum of Health Regulators' classification matrix establishes four risk levels ranging from basic information software to critical treatment systems. At the lowest level are applications that manage clinical information without offering interpretation, such as medical calendars or symptom-tracking systems without analysis. At the highest level, we find software that controls intensive care therapies, such as smart infusion pumps or cancer treatment planning systems. Among these extremes lies a continuum of functionalities where determining the appropriate regulatory class requires expert judgment and, frequently, negotiation with health authorities.

The importance of this classification goes beyond mere bureaucracy because it determines the product's entire lifecycle. Software classified as Class I can reach the market relatively quickly, with documentation and testing requirements that are manageable for a startup with limited resources. Class III or IV software, however, will face years of development, clinical trials, quality audits, and regulatory reviews that demand millions of investment and experienced multidisciplinary teams. This disparity in entry costs has shaped the current digital health ecosystem, where wellness and management apps abound, but solutions for serious diseases requiring strict controls are scarce.

A particularly relevant phenomenon in the field of SaMD is the trend toward the progressive medicalization of applications that were initially marketed as consumer products. Many technology companies have discovered that the health and wellness market is enormously lucrative but also that the boundary between general wellness and medical health is permeable. A meditation app that helps reduce stress can evolve into a treatment program for anxiety disorders. A fitness tracker can incorporate algorithms for detecting atrial fibrillation. Each of these shifts transform the product, change its regulatory classification, and necessitate a complete reevaluation of its development, validation, and marketing.

The regulatory response to this dynamic has been the creation of specific pathways for software that allow for some flexibility without compromising safety. In the United States, the FDA has implemented precertification programs for companies that, by demonstrating a strong quality culture, can update their algorithms with greater agility. In Europe, the Medical Device Regulation introduces specific considerations for software that recognize its evolving nature. (3) However, these regulatory adaptations remain insufficient to keep pace with the speed of technological innovation, creating a constant tension between the need to protect patients and the risk of delaying potentially beneficial technologies.

1.3.2 TELEMEDICINE: SOFTWARE AS A CLINICAL SPACE

Parallel to the development of SaMD, we have witnessed an explosion of telemedicine over the past decade, a term that encompasses very diverse realities under a single semantic umbrella. In its most basic sense, telemedicine simply involves performing medical procedures remotely using communication technologies. However, this broad definition masks an enormous diversity of situations, from a video call consultation with a primary care physician to robotic surgery performed by a specialist on another continent to the remote monitoring of chronic patients or the remote interpretation of diagnostic tests. Each of these modalities has its own technical requirements, its own limitations, and its own risks.

What makes telemedicine particularly relevant in our context is that it places software at the center of the clinical relationship. When a doctor and a patient share the same room, technology is an auxiliary tool that supports human interaction. However, when they are separated by hundreds of kilometers, software becomes the sole channel of communication, and its quality directly determines the quality of the medical procedure. A-based video call that cuts out, an image that becomes pixelated at a critical moment, or data transmitted with a delay are not mere technical annoyances but barriers that can compromise a diagnosis or treatment. This is why the reliability and usability of telemedicine are far superior to those of other consumer applications.

The COVID-19 pandemic accelerated the adoption of telemedicine in a way that would have been inconceivable just months earlier, revealing both its potential and its vulnerabilities. In a matter of weeks, health systems around the world had to implement remote consultation solutions that would have previously required years of planning. This forced transformation demonstrated that many medical consultations can be effectively conducted remotely, reducing geographical and time-related barriers to accessing care. However, it also revealed digital divides that exclude populations without access to adequate technology, diagnostic limitations when a direct physical examination is not possible, and the risks of depersonalizing the doctor–patient relationship.

Telemedicine modalities span a spectrum of increasing complexity that determines their technical and regulatory requirements. At the simplest end, we find asynchronous teleconsultation, where the patient sends information and the physician responds at a later time, as is the case with secure messaging systems for prescription renewals or the follow-up of stable chronic conditions. This model is technically accessible but poses challenges regarding continuity of care and clinical responsibility when unforeseen complications arise. Synchronous teleconsultation by videoconferencing adds the dimension of real time, allowing for richer interaction but requiring robust network infrastructure and strict privacy protocols.

Tele-triage and telemonitoring represent a higher level of complexity, where software not only facilitates communication but also continuously collects and analyzes clinical data. Weight, blood pressure, and oxygen saturation of a patient with heart failure can be monitored daily from home, with algorithms that alert the healthcare team to significant deviations. This model transforms care from episodic to continuous and from reactive to predictive, but it also creates new forms of technological dependence and risks of information overload for healthcare professionals. A physician receiving dozens of alerts daily, many of them false positives, may develop alert fatigue that causes them to lose sensitivity to truly important signals.

At the most sophisticated end of the spectrum, we find telesurgery and other remote interventional procedures, where software directly controls physical devices acting on the patient. Although still not widely adopted because of technical and regulatory limitations, this modality illustrates the highest degree of trust that can be placed in a technological system. The surgeon operating remotely is absolutely dependent on zero-latency connectivity, the millimeter precision of the robots, and the total reliability of the control software. A failure in any of these elements is not a technical glitch but a potentially lethal adverse event that demands levels of redundancy and safety comparable to those in aviation or nuclear power.

1.3.3 IOMT: PATIENT'S DIGITAL ANATOMY

The third major vertex of this technological triangle is the internet of Medical Things, known by its acronym IoMT. Behind this somewhat cryptic term lies an everyday reality: connected devices that monitor vital signs, sensors that track physical activity, smart patches that measure glucose, and pacemakers that send automatic reports to the cardiologist. It is estimated that tens of millions of these devices are already in operation worldwide, generating a continuous stream

of data that, if properly managed, can transform preventive medicine and the management of chronic diseases.

The promise of IoMT is enticing: imagine a patient with heart failure whose signs of decompensation are detected by sensors days before symptoms appear, allowing for early therapeutic intervention that prevents hospitalization. Alternatively, an elderly person living alone whose movements are discreetly monitored to detect falls or changes in activity patterns that may indicate an emerging health problem. Or an athlete whose training and recovery data are integrated with their physiological parameters to optimize performance without jeopardizing their health. All these applications are technically feasible today, and many are already being implemented in advanced healthcare systems.

However, alongside the promises, the IoMT also presents formidable challenges. The first is managing data volumes that would have been unimaginable a decade ago. A single patient with a continuous glucose monitor generates hundreds of readings daily. Multiply that by thousands of patients, and you have a torrent of information that threatens to overwhelm any system designed to be managed by humans. The second challenge is integration: devices from different manufacturers use proprietary protocols, incompatible formats, and standards that do not communicate with one another, turning the task of gathering all that information into a major technical puzzle.

The IoMT architecture consists of three interconnected layers that must work in sync for the system to be useful. The perception layer includes all physical devices that capture data, from consumer wearables to highly complex medical implants. These devices vary greatly in reliability, accuracy, and security, ranging from equipment certified as medical devices to wellness gadgets without specific health regulations. The network layer transports these data from the patient to processing systems, using infrastructures ranging from short-range Bluetooth to wide-area cellular networks, each with its own limitations regarding bandwidth, latency, and security.

The application layer is where data are transformed into useful clinical knowledge and where the greatest potential—but also the greatest risks—of IoMT lie. Analytical algorithms must be able to distinguish between true signals and artifacts, between significant physiological changes and normal variations, and between emergencies requiring immediate intervention and situations that can wait until the next scheduled appointment. This capacity for contextual interpretation is what distinguishes an intelligent medical system from a mere data collector, but it is also where errors most frequently occur—errors that can have serious clinical consequences.

IoMT security is a particular concern given the vulnerable nature of many of these devices. Unlike traditional computers, which receive regular security updates, many connected medical devices are designed without remote patching mechanisms and have lifespans spanning decades. A pacemaker implanted in a patient cannot be easily removed to update its software, nor can it rely on network connections that could compromise its basic operation. This tension between connectivity and security, between updating and stability, defines many of the current dilemmas in the design of smart medical devices.

1.3.4 BIG DATA: THE OCEAN OF INFORMATION AND ITS HIDDEN CURRENTS

This is where the fourth element of our map comes into play: Big Data applied to healthcare. Because there is no point in generating vast amounts of data if we are unable to process it, analyze it, and extract useful knowledge for clinical practice. Healthcare Big Data are not just a matter of volume but also of variety, velocity, and veracity. Clinical data come from a wide variety of sources: electronic health records, monitoring devices, diagnostic images, genetic analyses, pharmacy records, and patient surveys. Integrating this entire mosaic into a coherent picture of a person's health is among the most exciting technical and conceptual challenges of our time.

The applications of Big Data in medicine are already numerous and promise to increase in the coming years. There are predictive models that identify patients at high risk of readmission to the hospital, enabling personalized preventive interventions. There are diagnostic support systems that compare a patient's case with millions of similar records to suggest rare diseases that the physician might not have considered. There are population health tools that identify epidemiological patterns in real time, as some systems did during the COVID-19 pandemic by detecting outbreaks even before health authorities were aware of them.⁽⁷⁾ There are, of course, research algorithms that uncover new associations between genetic and environmental factors and diseases, accelerating the pace of medical knowledge.

However, we should not be blinded by these promises without also examining the problems.

Healthcare big data raises uncomfortable questions about privacy and autonomy that society is only just beginning to ask. Who owns the data a patient generates? Can an insurer access it to adjust premiums? Should researchers be able to use anonymized clinical records without explicit consent? To what extent is it truly possible to anonymize data when the combination of just a few variables can uniquely identify a person? These questions have no simple answers, and the answers we provide as a society will largely determine the future of data-driven medicine.

An additional problem, less debated but equally relevant, is the quality of the data feeding these systems. Electronic health records were originally designed for billing and administrative management, not for research or prediction. They contain errors, omissions, inconsistencies, and biases that are silently passed on to the algorithms that use them. A famous study showed that sepsis prediction systems in U.S. hospitals performed worse for patients from ethnic minorities simply because their clinical records were less complete.⁽⁸⁾ Data do not lie, but it does not tell the whole truth either, and blindly trusting it can perpetuate inequalities that the healthcare system should be combating.

The infrastructure needed to manage healthcare big data exceeds the capacity of any single institution. Population health data repositories, shared medical image banks, and genomic research platforms require massive investments in storage, processing, and analysis that can be undertaken only by large-scale public or private consortia. This concentration of data in the hands of a few creates risks of monopoly and technological dependence, where decisions about what research is possible and what treatments are developed rest with entities not subject to direct democratic control.

When we look at these four elements—SaMD, telemedicine, IoMT, and Big Data—together, a fascinating yet unsettling picture emerges. Fascinating because of the opportunities it has opened up for more precise, more preventive, more personalized, and more accessible medicine. Unsettling because of the complexity it introduces into a realm—that of the clinical relationship—which for centuries has been based on interpersonal trust and human judgment. Software is not going to replace the doctor, at least not in the foreseeable future, but it is profoundly transforming their work, their tools, and their relationships with patients. Understanding this new territory is the first step toward navigating it consciously and ethically.

The players in this landscape are numerous and diverse. There are public hospitals and healthcare systems that must decide which technologies to adopt and how to integrate them into their workflows. There are technology companies, ranging from small, innovative startups to the giants that already dominate other sectors of the digital economy. There are regulators who are trying to put up fences while technology gallops are unchecked. There are healthcare professionals who receive these tools with a mix of hope and skepticism. There are, above all, patients who are both potential beneficiaries and subjects of involuntary experimentation in this vast global laboratory.

To navigate this complex territory without being lost, it is wise to always keep an ethical compass in mind to guide us. Not every technological innovation is desirable simply because it is possible. Not all the data that can be collected should be collected. Not all automation that saves the doctor time necessarily improves patient care. Technology in medicine must serve the profession's fundamental values: beneficence, nonmaleficence, patient autonomy, and distributive justice. If a technological innovation contradicts any of these principles, no matter how brilliant it may be from a technical standpoint, it is likely not worth it.

The map we have drawn in this section is necessarily incomplete and provisional. Digital health is evolving at a dizzying pace, and what is cutting-edge today will be obsolete tomorrow. However, one thing remains: the need to understand this field from an integrative perspective, one that does not separate technology from ethics, innovation from safety, or code from people. Those who enter this field with this broad perspective will have a much better chance of building truly useful tools and, above all, of not causing harm in the process.

1.4 KEY PLAYERS: HOSPITALS, REGULATORS, STARTUPS, PATIENTS

There is a recurring image in science fiction films showing a lone genius locked away in his garage, creating revolutionary technology that will change the world. It is a powerful narrative that fuels the collective imagination about innovation and contains a grain of truth: many great ideas have been born in humble settings and owing to brilliant minds working against all odds. However, when we discuss innovation in digital health, that image is not only incomplete but dangerously misleading. Unlike others, in this field, ideas do not travel alone: they must make their

way through a complex ecosystem of stakeholders with very different interests, languages, and priorities. Understanding these stakeholders, understanding their motivations, and respecting their legitimate concerns is just as important as the idea itself that one wishes to develop.

Let us start with the most visible and, paradoxically, most heterogeneous actor of all: hospitals and healthcare systems. When we speak of hospitals, we are not referring to a monolithic entity but rather to an organizational microcosm where professionals with diverse backgrounds, cultures, and objectives coexist. Administrators focus on financial statements and operational efficiency. There are clinicians—doctors and nurses—whose priority is patient care and who often view technology as an additional burden rather than a tool. There are the IT managers who grapple daily with the nightmare of integrating technologies from dozens of different manufacturers. There are patients, of course, although within the hospital structure, their voice is often the weakest. A developer who does not understand this internal complexity is doomed to fail, no matter how brilliant their technical solution may be.

Hospitals also have a unique characteristic that sets them apart from other environments where software has been successfully implemented: they are organizations where errors have immediate and serious consequences. A failure in a factory's inventory management system may delay production; a failure in an electronic prescribing system can kill a patient. This asymmetry creates a risk aversion that is often mistaken for resistance to change but has deep and legitimate roots. Healthcare professionals have seen too many unfulfilled technological promises and too many tools that promised to save time but ended up multiplying administrative work and too many solutions that failed to understand the reality of clinical practice. They are rightly suspicious and that suspicion is not overcome with technical arguments but with facts, patience, and a deep respect for their expertise.

1.4.1 HOSPITALS: A WORLD OF WORLDS

Hospitals are like complex living organisms where technology must be integrated into workflows that have evolved over decades, often organically and not always rationally. An emergency room doctor does not need a faster app; they need one that works with wet gloves and is under stress. A ward nurse isn't looking for a sleek interface; they're looking for one that doesn't force them to make unnecessary clicks when they have ten patients waiting. A hospital administrator does not value innovation for its own sake; they value cost reduction and improvements in quality metrics that they can defend before their board of directors. Each of these potential users speaks a different language, has different priorities, and experiences technological frustration in unique ways that the external developer must learn to map out.

The organizational structure of a modern hospital reflects the historical tension between medicine as a clinical art and healthcare management as an administrative discipline. Clinical departments operate with significant autonomy, each with its own traditions, hierarchies, and ways of working that resist standardization. Admissions, pharmacies, laboratories, and radiology departments function as silos that exchange information in a fragmented manner. Procurement committees and information technology units impose vendor selection criteria that often prioritize stability over innovation. Navigating this labyrinth requires patience, diplomacy, and the ability to identify allies at every level who can facilitate the introduction of new tools without disrupting the institutional balance.

Technology adoption in hospitals follows learning curves that developers frequently underestimate. The pilot phase, where a product is tested in a controlled environment, is only the beginning of a process that can take years to achieve widespread implementation. During this period, the software must demonstrate not only that it works technically but also that it fits into existing workflows, does not increase the cognitive load on healthcare professionals, generates useful data for management, and withstands the variations in use that occur in real-world practice. A product that works perfectly in the lab can fail spectacularly in the hospital because it did not anticipate that doctors would use it in unforeseen ways or because it did not integrate with the existing electronic health records system.

The relationship between hospitals and technology providers is also shaped by asymmetrical power dynamics that hinder genuine innovation. Major medical equipment manufacturers establish long-term relationships with hospitals, creating barriers to entry for new competitors. Maintenance and upgrade contracts create technological dependencies that resist disruption. Public bidding processes, designed to ensure transparency and fairness, often favor established solutions over disruptive innovations that do not fit within predefined specifications.

Understanding these economic and political realities is essential for any startup aspiring to scale beyond isolated pilot projects.

1.4.2 STARTUPS: SPEED AND VULNERABILITY

At the opposite end of the spectrum are digital health startups—those small teams that typically combine a passion for technology with a desire to improve people's lives. The health tech entrepreneurial ecosystem has grown exponentially over the past decade, driven by the availability of capital, the reduction in technical barriers, and the growing awareness that technology can solve real healthcare problems. Many of these startups are founded by healthcare professionals who have experienced the system's limitations firsthand and want to change it from the outside. Others emerge from business schools or engineering departments, with teams that have barely set foot in a hospital but trust in their technical ability to transform medicine.

In addition to their small size and agility, startups are characterized by their risk tolerance and focus on rapid growth. While a hospital thinks in terms of years or decades, a startup thinks in terms of quarters and funding rounds. While a hospital seeks stability and predictability, a startup seeks innovation and disruption. These differences in culture and time horizons inevitably create tensions when the two countries try to collaborate. The hospital accuses the startup of failing to understand clinical complexity and of promising magic solutions that do not hold up in the real world. The startup accuses the hospital of being slow, bureaucratic, and resistant to innovation. Both are partly right, and the art of collaboration lies in building bridges that respect the legitimate needs of each party.

The life cycle of a digital health startup goes through critical phases where most projects fail before they reach the market. In the ideation phase, excitement over the technical solution often clouds understanding of the actual problem. In the validation phase, it is discovered that what doctors say they need does not match what they actually use. In the regulatory phase, the timelines and costs of obtaining health authorizations far exceed the initial estimates. In the commercialization phase, where the harsh reality is faced such that hospitals do not buy innovation, they buy proven solutions with solid track records. Each of these transitions eliminates a significant proportion of competitors, leaving only those that combine technical excellence with an understanding of the ecosystem.

The financing of healthcare startups introduces unique dynamics that shape their development. Venture capitalists seek exponential returns that are possible only with scalable business models, pushing startups toward solutions that can be replicated without extensive customization. However, the reality of healthcare is that every system, hospital, and specialty has specific needs that resist standardization. This tension between the logic of investment and the logic of clinical practice creates pressures that sometimes lead to insufficiently tested products or unfulfilled promises of functionality. The entrepreneurs who successfully navigate this tension are those who manage to find niches where standardization is viable without sacrificing clinical safety.

1.4.3 REGULATORS: GUARDIANS OF THE FRONTIER

Between the two, playing a role that is often uncomfortable but absolutely necessary, stands the regulators. Agencies such as the FDA in the United States, the European Medicines Agency, or national agencies in Latin America and Spain are responsible for ensuring that medical devices reaching the market are safe and effective. Their work is inherently conservative because their role is not to accelerate innovation but to protect patients from the harm that hasty innovation could cause. This focus on safety clashes head-on with startup culture, which is accustomed to "moving fast and breaking things," but it is indispensable in a field where the things that break can be human lives.

Regulators also face an unprecedented challenge: the speed of technological change far outpaces their ability to respond. By the time a regulatory agency takes years to develop guidance for a new product category, that category has already evolved or been replaced by another. This mismatch creates uncertainty and frustration among developers, who often do not know exactly which regulatory requirements their product will have to meet. To address this uncertainty, understanding the philosophy underlying regulation, beyond the specific details that are constantly changing, is essential. Regulation is not an arbitrary obstacle, but a set of

questions that any medical device must be able to answer: Is it safe? Does it work? Do its benefits outweigh its risks? Is it well documented so that its performance can be audited?

Recent regulatory developments reflect attempts to adapt frameworks designed for physical devices to the reality of software. The FDA has introduced company precertification programs that allow for some flexibility in algorithm updates. Europe has developed specific regulations for medical software within the general framework for medical devices. However, these adaptations remain insufficient to keep pace with the speed of innovation, and regulatory uncertainty remains one of the main barriers to entry for new competitors. Compared with startups that must learn on the fly, established companies with experienced regulatory departments have significant advantages in terms of the complex requirements for documentation, clinical trials, and risk management.

The relationship between regulators and the regulated does not have to be adversarial, although it is often perceived that way. Health agencies offer channels for dialog, presubmission consultations, and guidance programs that developers frequently underutilize because of a lack of awareness or fear of revealing strategic information. Understanding that the regulator is a partner seeking to fulfill its mandate of public protection—not an obstacle to being circumvented—fundamentally changes the development strategy. Companies that establish transparent and collaborative relationships with agencies tend to navigate the regulatory process more efficiently than those that approach it as a legal battle do.

1.4.4 PATIENTS: THE INVISIBLE CENTER

At the center of all these actors, as the *raison d'être* of the entire system, are the patients. It may seem obvious to point this out, but in the day-to-day practice of healthcare software development, it is surprisingly easy to lose sight of them. Patients appear in conversations as “users,” as “data,” and as “clinical cases,” but rarely as people with fears, hopes, limitations, and preferences. A patient is not a set of variables feeding an algorithm, nor a consumer choosing between different apps, nor a body to be managed efficiently. They are people living through an illness who trust their doctors, who have the right to understand what is done with their information, and who deserve to be treated with dignity in every interaction with the healthcare system.

Incorporating the patient's perspective into the development of medical software is not a paternalistic gesture but a requirement for quality. A glucose monitoring app that does not account for the fact that many diabetic patients have vision impairments or limited manual dexterity will be useless, no matter how technically precise it may be. An appointment scheduling system that does not consider that some patients lack internet access or are unable to use mobile apps will create inequalities rather than resolve them. An emergency room triage algorithm that does not incorporate subjective variables such as pain or discomfort will offer a distorted picture of clinical reality. Patients are not merely the end users of technology but also irreplaceable sources of knowledge about what actually works in practice.

There is also a category of patients who deserve special attention: those who actively participate in the development of health technologies, whether as advisors, evaluators, or research subjects. Their contribution is invaluable, but it also raises ethical dilemmas that are not always properly managed. How can we ensure that a patient participating in the design of an app does not feel pressured to consent to use their data that they do not fully understand? How can we prevent the excitement of contributing to a technological advancement from clouding their ability to critically assess the risks? Who safeguards their interests when the commercial interests of developers point in the opposite direction? These questions have no easy answers, but ignoring them does not make them go away.

Patient participation in digital health governance is evolving from symbolic representation to more substantive models. Some healthcare systems are experimenting with patient councils that review technological proposals before implementation. People who are associated with chronic conditions are developing the technical capacity to critically evaluate apps targeting their communities. Open data and personal data ownership movements are empowering citizens to better control who accesses their health data and for what purposes. These trends, although nascent, are gradually shifting the balance of power within the ecosystem, challenging the traditional asymmetry between those who design technology and those who use it.

Rounding out the ecosystem, although often in the background, are other actors whose influence is decisive. Insurers and public payers determine which technologies are funded

and, therefore, which technologies actually reach patients. Scientific societies and professional associations issue guidelines and recommendations that shape clinical practice. Universities and research centers train future professionals and generate the knowledge that fuels innovation. Patient organizations, which are increasingly organized and influential, advocate for the interests of those living with specific diseases. Together, they form a dense web of relationships, interests, and interdependencies that any digital health project must navigate skillfully.

The complexity of this ecosystem explains why so many promising initiatives fail to reach the market. It is not enough to have a good idea and a strong technical team. One must understand how hospitals think, what clinicians need, what regulators require, what patients value, and what investors seek. You must be able to translate between languages that often do not align: the language of efficiency and return on investment, the language of evidence and clinical safety, and the language of human experience and suffering. This task of translation is exhausting but also fascinating, and those who master it build bridges that benefit society as a whole.

For healthcare professionals entering this world for the first time, it can be overwhelming to see the number of stakeholders and interests involved in something as seemingly simple as developing an app. However, it's important to understand it, because you'll be part of that ecosystem, and the better you understand it, the more effectively you can help ensure that technology serves the purposes that truly matter. For the developer coming from the tech world, the culture shock can be even greater. In environments where innovation is the supreme value and the primary metric is the speed, they report that in digital health, other values carry more weight: prudence, safety, and equity. Learning to respect them does not mean giving up on innovation but rather practicing it in a more mature and responsible way.

Ultimately, all these stakeholders share a common goal, even if it is sometimes difficult to see amid tensions and disagreements. Everyone wants to improve people's health. Hospitals want to better care for their patients. Regulators want to protect them from preventable harm. Startups want to create solutions that solve real problems. Patients want to live longer and better lives, with as little disease burden as possible. Keeping this shared goal at the center of the conversation is the best way to ensure that inevitable disagreements do not become insurmountable barriers. It is also the best guarantee that the technology we build truly deserves to be called healthcare.

• CROSS-LITERACY: WHAT DOCTORS NEED TO KNOW ABOUT SOFTWARE

It's a scene that plays out in hospitals around the world with minor variations, but with a recognizable pattern: a team of engineers has spent months developing an app they consider revolutionary—a tool that will aggregate data from different sources and present doctors with a dashboard containing all the relevant information about their patients. On the present day, the engineers proudly showcase their creation, and the attending doctors exchange glances that range from bewilderment to disappointment. Something isn't working; something doesn't fit. The app is technically flawless, but it does not answer the questions that doctors ask themselves; it does not integrate into their workflow, and it does not understand how they actually think when treating a patient. The problem is not technical; it is one of translation. As with any translation, there are two languages that must learn from each other.

Let us start with what a doctor needs to understand about software—not to become a programmer but to be a competent and demanding partner in projects requiring their participation. The first thing—and perhaps the hardest to accept for professionals accustomed to the precision of clinical diagnosis—is that software rarely works in a binary fashion. A drug is either effective or is not effective; it has side effects or does not. However, a computer program can function correctly ninety-nine percent of the time and fail catastrophically in that remaining one percent. It can do its job well for years and reveals serious errors when it is updated to a new version. It may be perfect for one type of patient and useless for another. This probabilistic nature of software is difficult for minds trained in classical clinical thinking to accept, but ignoring it leads to misplaced trust.

The second fundamental lesson for healthcare professionals is that software is not neutral. Behind every line of code are human decisions that reflect priorities, assumptions, and, at times, biases. When a programmer decides which variables to display on the main screen of an electronic health record, they are making an implicit statement about what is important in patient care. When a development team prioritizes one feature over another, it expresses its vision of what clinical work should look like. When an algorithm weights some factors more heavily than others do to make a prediction, it is incorporating a value judgment about what is worth considering.

Clinicians using these tools must be aware that they are not dealing with objective instruments but with artifacts laden with intentionality and must learn to critically question them.

1.5.1 ESSENTIAL VOCABULARY: CONCEPTS THAT PREVENT MISUNDERSTANDINGS

There is also a set of basic technical concepts that every physician involved in digitalization projects should be fluent in. You do not need to know how to program, but you do need to understand what a database is and how information is organized into tables and records. You must understand the difference between an integrated system and one that requires interfaces to communicate with other systems because that difference determines the smoothness of your daily work. It is in your best interest to know what it means for software to be open-source or proprietary because that affects the hospital's ability to modify or adapt it. In addition, you should have a basic understanding of client-server architecture, cloud storage, and the fundamentals of IT security. It is not about becoming an expert but about acquiring the minimum vocabulary needed to hold a meaningful conversation with the technicians.

Understanding databases is particularly important because it determines how the information a physician uses daily is structured. When a healthcare professional enters data into an electronic health record, they are not simply writing in a digital document; they are filling in specific fields of a predesigned structure that determines what can be searched, analyzed, and shared. Understanding that data are organized into relational tables, that there are primary keys identifying unique records and that SQL queries extract information on the basis of defined criteria allows the physician to ask more precise questions about the system's functionality. Knowing why certain information appears automatically on their screen while other information requires complex searches provides clues about the underlying architecture and potential improvements.

The distinction between open-source and proprietary software has practical implications that many physicians are unaware of. An open-source system allows the hospital to hire developers to modify it according to its specific needs, to fix bugs without relying on the original manufacturer, and to adapt workflows that the generic designer did not anticipate. A proprietary system generally offers greater stability and commercial support but at the cost of technological dependence and inflexibility with respect to specific needs. This choice is not merely technical but strategic and affects healthcare facilities' autonomy in the medium and long term. The informed physician can participate in these decisions with discernment, understanding that immediate convenience does not always offset future dependence.

The concept of client-server architecture explains why some applications continue to function even when the network fails, while others become unusable. In a traditional client-server architecture, processing occurs on central servers, and the physician's device displays results only, allowing access from any location but creating a dependency on connectivity. Modern hybrid applications store data locally and synchronize when possible, offering greater resilience against network issues but introducing complexity in version management. Understanding these architectures helps physicians assess whether a system is suitable for their particular environment, where connectivity may be intermittent or where the immediate availability of critical data is a priority.

Although it may seem the exclusive domain of specialists, cybersecurity requires a basic understanding from all users. Physicians must understand what data encryption in transit and at rest means, why strong passwords are mandatory and not optional, and what risks are involved in accessing clinical systems from public networks. They must understand the principle of least privilege, which limits access to information only to those who need it for their work, and why sharing credentials with colleagues or subordinates compromises traceability and security. This knowledge is not abstract: a physician who understands the risks is less likely to fall victim to social engineering, better able to identify anomalous behavior in the system, and more aware of their responsibility in the data protection chain.

1.5.2 BETWEEN THE USER AND THE PATIENT: DESIGN FOR REAL-WORLD PRACTICE

One of the most useful concepts a physician can incorporate into their mental repertoire is that of "user experience," known by its acronym UX. Behind this term lies a profound insight: software must not only do what it promises but also do so in a way that feels natural to the user. In the clinical setting, where time is scarce and interruptions are constant, a poorly designed interface

is not a minor annoyance but a source of potentially serious errors. A button that is confused with another, a screen that delays critical information, a workflow that forces unnecessary clicks: all of this affects the quality of care. A physician who can identify and communicate these issues will make a decisive contribution to patient safety.

The user experience in medicine has unique characteristics that distinguish it from other fields. In consumer applications, the user can take time to learn the interface, repeat actions if they make mistakes, or seek alternatives if the system does not respond. In the clinical setting, the physician operates under time pressure, is frequently interrupted, and often suffers from accumulated fatigue. An interface error cannot be undone with a simple “undo”: it can mean prescribing the wrong medication, overlooking a critical allergy, or delaying an urgent diagnosis. Medical software design must incorporate these extreme constraints, anticipating that the user will not be in optimal attention conditions and that any ambiguity will result in error.

An informed physician can critically evaluate the usability of the tools they use, applying principles that do not require specialized technical training. They can ask whether the most frequently needed information is accessible with the fewest possible interactions. They can observe whether the system provides clear feedback on actions taken, confirming that an order has been recorded or alerting them if something has failed. They can verify whether the terminology used corresponds to standard clinical language or imposes unfamiliar jargon that causes confusion. These observations, which are systematically documented and communicated to development teams, are valuable contributions that improve the quality of the software for all users.

Another aspect that often leads to misunderstandings is the difference between technical validation and clinical validation. For an engineer, for software to work means that it does what it is supposed to do, that it meets specifications, that it does not crash, and that it responds within acceptable time frames. For a physician, software to work means that it improves patient care, that it helps make better decisions and that it does not introduce additional risks. Both perspectives are legitimate, but they do not always align. A program can be technically flawless yet clinically useless or even harmful. A program can have technical flaws yet be clinically valuable. Physicians involved in development projects must learn to both demand and understand that clinical validation is not an optional add-on but the very core of what justifies the existence of healthcare software.

Clinical validation requires specific methodologies that physicians are uniquely qualified to design and execute. It is not enough to prove that the software does not crash; one must demonstrate that it improves outcomes relevant to patients, that it does not introduce treatment biases and that it is acceptable to those who must use it. This involves controlled trials, usability studies in real-world settings, and analyses of the impact on healthcare quality indicators. Physicians who understand the difference between technical and clinical validation can demand that both be carried out rigorously, preventing products prematurely launched onto the market from compromising patient safety.

1.5.3 HOW SOFTWARE IS MADE: PROCESSES PHYSICIANS SHOULD BE AWARE OF

It is also helpful for healthcare professionals to understand software development cycles and the methodologies used by technical teams. For example, they should know that many teams use agile methodologies, which involve delivering partial versions of the product in short cycles and improving them iteratively. This is very different from the traditional development model, where teams wait until the product is completely finished before it is presented. If a physician understands this dynamic, they will know that they can provide continuous feedback, that they do not have to wait to see the final result to point out problems and that their early involvement can save months of misguided work. They will also understand why engineers sometimes ask them to prioritize certain features over others: they are managing limited resources and need to know what is truly essential.

The agile methodology, with its two- or three-week sprints, daily stand-up meetings, and visual task boards, may seem chaotic to those accustomed to rigorous clinical planning. However, this apparent improvisation follows a logic of continuous adaptation that offers advantages in uncertain environments. The clinician who participates in agile projects must learn to articulate their needs in concrete user stories, to prioritize features in collaborative backlogs and to accept that the product will evolve through controlled trial and error. This flexibility is not technical laxity but rather a recognition that requirements change when confronted with reality and that it is more efficient to adapt than to try to foresee everything from the start.

The concept of technical debt is particularly useful for understanding why some medical systems seem obsolete or slow despite ongoing investments. When development teams take shortcuts to meet deadlines, they accumulate imperfect code that hinders future modifications. Over time, this debt grows until any change, no matter how small, requires disproportionate effort. A clinician who understands this dynamic may have a greater understanding of the current system's limitations but also be more insistent on the need for periodic refactoring to maintain the code's long-term health. They can distinguish between problems requiring quick fixes and structural issues that demand substantial investment.

Version management and change control are processes that directly affect clinical practice but are rarely explained to medical users. When a system is updated, it does not happen magically: There are testing processes in parallel environments, scheduled maintenance windows, and recovery protocols in case something goes wrong. A clinician who understands these procedures can better plan their work, anticipating periods of downtime or changes to the interface. They can also participate in pilot tests of new versions, providing valuable feedback before general deployment. This active participation improves the quality of updates and reduces the frustration of encountering unexpected changes in tools they use daily.

A particularly sensitive area is that of data and its quality. Doctors often assume that if data are on the computer, it is correct. This lack of trust in the infallibility of computer systems is among the main sources of errors in contemporary clinical practice. The reality is that health data are riddled with impurities: misrecorded values, misinterpreted fields, duplicates, omissions, and updates that are not reflected correctly. A physician involved in designing a system must be relentless in asking where the data come from, how it is cleaned, who validates it, and how often it is updated. They must understand that an algorithm trained on dirty data will produce dirty predictions and that relying on them without questioning them can have serious consequences.

1.5.4 INTEROPERABILITY, ALERTS, AND THE ETHICS OF CRITICAL USE

Another concept that deserves attention is interoperability, a long word that describes simple reality: the ability of different systems to exchange information and use it in a coherent manner. In practice, this means that the electronic health records should be able to communicate with the laboratory, that the laboratory should be able to send results to the prescribing system, and that the prescribing system should alert the physician to drug interactions on the basis of the medications the patient is already taking. When this does not happen, the physician becomes the manual integrator of all that information, copying data from one screen to another, wasting time and increasing the risk of errors. An informed physician knows that interoperability is not a technological luxury but a basic prerequisite for digitization to truly improve care.

Interoperability in healthcare faces technical and organizational challenges that explain why it remains elusive despite decades of effort. Messaging standards, such as HL7 or FHIR, provide common formats for information exchange, but their implementation varies among vendors. Databases from different systems use medical terminologies that do not always correspond and require complex mappings. Economic incentives often work against interoperability: software vendors prefer that customers rely on their closed ecosystems. Physicians who understand these barriers can push for their removal, demanding compliance with open standards in tenders and participating in information architecture initiatives that promote integration.

It is worth pausing for a moment to consider the phenomenon of alerts and notifications, one of the main sources of friction between physicians and computer systems. Software designers, with the best of their intentions, fill out screens with alerts: potential drug interactions, out-of-range lab values, and reminders of pending tests. Physicians, overwhelmed by hundreds of daily alerts, develop what is known as “alert fatigue” and ultimately ignore them all, including those that might be truly important. Understanding this dynamic and understanding how to contribute to the design of systems that alert physicians only when truly necessary is a valuable contribution that physicians can make to development teams.

Managing clinical alerts requires balancing safety with usability so that neither is sacrificed. Alerts must be context specific: a known and manageable drug interaction does not require the same urgency as a potentially lethal combination does. They must be graded by severity, using different levels of intrusiveness based on risk. They must incorporate smart suppression options, allowing the clinician to indicate that they have assessed an alert and do not wish to see it repeatedly for the same patient. The clinician who participates in the design of these logic rules can contribute clinical knowledge that engineers lack, creating systems that are truly useful

rather than merely well intended.

There is also an ethical dimension to a physician's digital literacy that must not be overlooked. When healthcare professionals use software, they are entrusting their clinical judgment to a device designed by others, and that trust must be informed and critical. They must know how to ask: How has this system been validated? In which populations has it been tested? What biases might it contain? Who is accountable if it fails? These questions are not hostile but rather responsible. A physician who asks them is exercising their duty to protect their patients, and developers should receive them for what they are: an aid to building better products, not a threat to their work.

Physicians' ethical responsibility extends to understanding algorithmic biases and their implications for healthcare equity. Artificial intelligence systems trained on historical data reproduce past treatment patterns, which often reflect inequalities based on sex, race, or social class. A transplant prioritization algorithm trained on historical decisions can perpetuate forms of discrimination that an individual physician would not endorse. An informed professional must be able to question the system's recommendations when they contradict their clinical judgment, request information about the training dataset, and demand equity audits to verify that the software performs equally well for all patient groups.

Finally, it is important for physicians to understand the nature of multidisciplinary teamwork. Developing healthcare software is not like writing a scientific article, where each author contributes their part and the whole is assembled at the end. It is an iterative, sometimes chaotic, process in which ideas are collectively built and rebuilt, where disagreements are frequent, and where the ability to listen and translate is more valuable than specialized knowledge. A physician sitting at a table with engineers is not there to give orders or to receive them but to build together a shared understanding of the problem and possible solutions. That collective construction is slow and sometimes frustrating, but it produces results that neither party can achieve on its own.

A doctor's digital literacy is not, therefore, a mere curricular embellishment or a concession to modernity. It is an indispensable working tool in a healthcare system that can no longer be understood without technology. It is also a way to fully practice the profession, maintaining control over the tools used rather than being controlled by them. The physician who understands software is not a physician who programs but a physician who retains their professional autonomy in an environment that is increasingly mediated by machines. Ultimately, this is what benefits their patients the most.

If the physician needs to learn the language of software to avoid losing professional autonomy, what about the engineer who enters the world of medicine without understanding that here, errors are not fixed with patches but can cost lives and that clinical precision demands a humility that technological culture often does not foster?

1.6 CROSS-LITERACY: WHAT ENGINEERS NEED TO KNOW ABOUT MEDICINE

Let us now imagine the scene from the opposite perspective. A team of doctors has identified a recurring problem in their department: the delayed detection of deterioration in hospitalized patients. With the best of their intentions, they sit down to design a technological solution and draft a specifications document that they hand over to the engineers. The document is exhaustive: it describes screens, functions, and algorithms—all with a level of detail that the doctors consider sufficient. When engineers receive that document, however, they experience a mix of perplexity and unease. The instructions are precise on the surface, but they omit exactly what the engineers would need to know to build something useful: what data is actually available? How do they relate to one another? What exactly does “deterioration” mean in measurable terms? What decisions will be made on the basis of the information the system provides? Again, the problem is not technical but one of translation, and this time, it is the engineers who must learn the language of their clinical counterparts.

The first thing an engineer must understand when entering the world of medicine is that clinical practice is unlike any other field they have worked in before. It is not like developing software for the banking sector, where transactions follow precise and predictable rules. It is not like creating an e-commerce application, where user behavior can be modeled with reasonable accuracy. Medicine is a field governed by uncertainty, where two patients with the same diagnosis may respond completely differently to the same treatment, where the clinician's intuition and experience carry as much weight as objective data and where decisions are made with incomplete information and under time pressure. An engineer who fails to internalize this reality will build systems that are elegant on paper but useless in practice because they will be

based on assumptions that do not hold true in the real world.

The second fundamental lesson for the technician is that in medicine, the context is everything. A blood pressure reading of 140 over 90 may be normal in one patient and alarming in another. A heart rate of 110 beats per minute may indicate a dangerous arrhythmia or simply that the patient is nervous about the visit. A slightly abnormal lab result may be irrelevant or may be the first clue to a serious illness, depending on the complete medical history. Engineers, accustomed to working with data that are the same thing in all situations, must learn that in medicine, meaning emerges from the integration of multiple sources of information, many of which are not digitally encoded. A system that ignores that context will not only be less useful but also potentially dangerous because it will present as certainties what are merely clues.

1.6.1 CLINICAL VOCABULARY: MORE THAN MERE DEFINITIONS

There is also a set of basic clinical concepts that every engineer involved in healthcare projects should be fluent in. They do not need to diagnose diseases, but they do need to understand the difference between signs and symptoms, between sensitivity and specificity, and between incidence and prevalence. They must understand how healthcare is organized: what a primary care physician does, what a specialist does, what role nurses play, and how responsibilities are distributed in a hospital. It is beneficial to know the basic structure of a medical record, the most common types of diagnostic tests, and the basic vocabulary of pharmacology. In addition, you should have a basic understanding of clinical reasoning—how doctors arrive at a diagnosis, the heuristics they use, and the biases to which they are subject. This knowledge won't turn you into a clinician, but it will allow you to have conversations where both parties feel heard and respected.

The seemingly simple distinction between signs and symptoms has profound implications for the design of medical software. Symptoms are what the patient subjectively reports: pain, dizziness, and fatigue. Signs are what the doctor objectively observes: a measured fever, an abnormal auscultation, and a lab result. A system that fails to distinguish this fundamental difference can make serious errors: treating subjective data as if it were objective, ignoring the importance of the patient's perception, or vice versa. The engineer must understand that both types of information are valuable but of a different nature and that the software must allow them to be recorded, weighted, and related in ways that reflect the complexity of clinical reasoning.

The concepts of sensitivity and specificity are particularly critical when the software includes diagnostic or detection components. Sensitivity measures the system's ability to correctly identify those who have the condition; specificity measures the ability to correctly rule out those who do not. A sepsis detection algorithm with high sensitivity but low specificity will generate many false alarms, consume resources and cause fatigue. Cases with high specificity but low sensitivity will miss actual cases, with potentially fatal consequences. The engineer must understand that there is no universal optimal configuration: the choice depends on the severity of the condition, the costs of false positives versus false negatives, and the system's capacity to handle alerts. These are clinical decisions that require dialog with professionals, not isolated technical optimizations.

An understanding of the healthcare organization prevents design errors that assume nonexistent structures or ignore operational realities. The engineer must know that primary care filters and refers patients, that specialists have waiting times that dictate the urgency of decisions, that nurses manage much of chronic care and that pharmacists are key partners in treatment adherence. A system designed without this understanding can create impossible workflows: requiring specialized decisions within timeframes that cannot be met, omitting key actors in the care chain, or assuming the availability of resources that do not exist in reality. Mapping actors and processes must precede technical design; it should not be an afterthought.

1.6.2 CLINICAL TIME: RHYTHMS THAT CANNOT BE COMPRESSED

One of the most difficult concepts to convey to engineers is the nature of time in clinical practice. In the world of software development, time is often conceived as a resource that can be organized and optimized. In medicine, time is often an enemy: the time that passes while a patient waits for a test, the time running out for someone with internal bleeding, and the time lost when a computer system runs slowly. However, it is also an ally: the time that allows an infection to resolve, the time that provides perspective to assess the progression of a chronic disease, and the time at which doctors need to explain a complex diagnosis to a frightened patient. An engineer

who designs healthcare systems must constantly ask themselves, “Does my tool speed up what needs to be sped up and slow down what needs to be done calmly? Does it respect the natural rhythms of care or impose an artificial pace that undermines the clinical relationship?”

Urgency in medicine is neither uniform nor predictable. There are moments when every second counts: a cardiac arrest, massive bleeding, and a severe asthma attack. In these scenarios, the software must be invisible, fast, and reliable, adding no friction to actions that should be automatic. However, there are also moments when haste is counterproductive: the consultation of a patient with a complex chronic illness, decision-making regarding cancer treatments, and the delivery of bad news. Here, the software must allow for reflection, the consultation of additional information, and the necessary pause for ethical deliberation. A system that imposes the same pace on all these situations—one that optimizes for speed without distinguishing the context—will be detrimental regardless of its technical efficiency.

Interruption is another temporal dimension that engineers frequently underestimate. In software development, the state of flow is valued and protected; interruptions are considered detrimental to productivity. In medicine, interruptions are inevitable and often necessary: an emergency call, a consultation with a colleague, and a monitor alarm. Medical software must be designed for this fragmented environment, allowing clinicians to leave and resume tasks without losing context, maintaining the state of incomplete sessions, and providing summaries that facilitate quick reentry. A system that requires uninterrupted sequences of interaction ignores the reality of clinical work and generates frustration or, worse, errors when users attempt to speed up processes to compensate for interruptions.

Another crucial aspect is the hierarchy of information. In a typical software system, all the data usually carry the same weight or at least a weight that can be determined by explicit rules. In medicine, not all data are equal: there is critical information that must always be visible, secondary information that can be consulted if necessary, and irrelevant information that only adds noise. A doctor treating an emergency needs to know immediately if the patient is allergic to any medication but can wait to learn their educational background or profession. A system that does not distinguish these levels of importance—one that presents all information with equal emphasis—will force the clinician to search for what is relevant amid a sea of data and increase the risk of overlooking something crucial. The engineer must learn to ask: What does the doctor need to know at any given moment? What can wait? What is best left unseen so as not to distract them?

1.6.3 UNCERTAINTY AS DATA, NOT AS ERROR

Uncertainty is another concept that engineers must learn to handle with respect to humility. In the world of software, uncertainty is often viewed as a problem to be eliminated: tests must always pass, responses must be consistent, and errors must be corrected. In medicine, uncertainty is inherent: it is part of the very nature of clinical work. A doctor can never be completely sure of a diagnosis until it is confirmed by tests and sometimes not even then. A treatment that works for most patients may not work for a specific patient. A prognosis is always a probabilistic estimate, not a certainty. Systems that ignore this uncertainty—which presents as certain predictions that are only probable—will mislead clinicians and lead them to make worse decisions than they would make without assistance. Good medical software does not eliminate uncertainty but helps surgeons navigate it in an informed way.

Presenting probabilities in medical interfaces requires special care that engineers often overlook. Humans, including doctors, are intuitively poor at probabilistic reasoning. Presenting a risk as “15% mortality” does not have the same emotional impact or imply the same decisions as presenting it as “85% survival,” even though they are mathematically equivalent. The algorithms that produce risk scores must communicate not only the numerical value but also its accuracy, its limitations, and the factors that most influence it. Engineers must collaborate with psychologists and clinicians to design representations that are honest about the underlying uncertainty without paralyzing decision-making.

Managing uncertainty also involves knowing when the system should not make a decision. There are situations where the information is so incomplete or ambiguous that any algorithmic recommendation is unwise. A good medical software system must recognize these limits, alert the clinician to the lack of data, and control the human judgment by hand. This contradicts the engineering tendency to seek solutions for every scenario and to view edge cases as technical problems to be solved. In medicine, acknowledging uncertainty is sometimes the safest approach,

and the software must allow for this option without penalizing the user with error messages or blocked workflows.

The issue of clinical workflows deserves special attention, as it is among the main sources of friction between doctors and computer systems. Engineers tend to think in terms of linear processes—first you do this, then that, then the next thing. However, real clinical work is inherently interrupted and multitasking. A doctor might be writing a patient's notes, receive an urgent call, attend to a nurse who needs a prescription, and then return to the notes, all within a matter of minutes. A system that does not support this type of interruption—one that forces a task to be completed before starting another or that loses its state when the doctor switches screens—will be perceived as an obstacle rather than a help. Engineers who observe real clinical work, who spend time in clinics and on wards and who understand the organized chaos of a hospital, are far more likely to design tools that integrate naturally into that workflow.

Ethnographic observation of clinical work should be mandatory for every engineer designing healthcare systems. It's not about reading procedure manuals or interviewing doctors in conference rooms but about spending hours in the real environment, seeing how decisions are actually made, what information is consulted and when, what shortcuts are used, and what frustrations arise. This immersion reveals discrepancies between the official and actual workflows, identifies opportunities for improvement that are not evident from the outside, and builds genuine empathy between the designer and the user. Projects that skip this phase often produce systems that meet formal specifications but fail to be adopted because they do not align with the complex reality of practice.

1.6.4 THE THERAPEUTIC RELATIONSHIP: TECHNOLOGY AT THE SERVICE OF THE ENCOUNTER

Communication with patients is another dimension that engineers often underestimate. To a technician, the doctor–patient relationship may seem like an incidental add-on, something that occurs on the margins of the system. However, it is precisely the heart of medicine. Patients who do not trust their doctor will adhere less to treatments, share less relevant information, and have poorer health outcomes. Any technology that interferes with that relationship, that is, that comes between the doctor's gaze and the patient's gaze, that turns the consultation into an exchange of data rather than a human encounter, will cause harm that is difficult to measure but very real. The engineer must ask: Does my tool facilitate communication or hinder it? Does it free up time so that the doctor can listen better, or does it tie them up with administrative tasks? Does it help humanize care or dehumanize it?

The design of interfaces for medical consultations requires specific considerations that go beyond general usability. The screen must be positioned so that it does not physically stand between the doctor and the patient, allowing for eye contact. Data entry must be possible without the doctor taking their eyes off the patient and using input devices that do not require full visual attention. System response times must be fast enough to avoid creating awkward silences that disrupt the flow of conversation. The terminology displayed must be understandable to the patient if the screen is visible or hidden if it contains sensitive information that could be alarming without context. These seemingly minor details determine whether technology facilitates or hinders the therapeutic relationship.

Another essential concept is the variability of clinical practice. Engineers, accustomed to seeking universal solutions, often assume that there is one correct way to do things and that all doctors should adapt to it. However, the reality is that two excellent clinicians may approach the same problem in very different ways, and both may be equally valid. Personal preferences, training received, accumulated clinical experience, and even personality influence how each doctor works. A rigid system that imposes a single way of doing things will generate resistance, be perceived as an imposition, and likely be sabotaged in more or less subtle ways. Flexibility—the ability to adapt to different work styles—is a virtue that engineers must incorporate into their designs.

Workflow customization does not mean a lack of structure but rather configurability within safe parameters. The engineer must identify which aspects of the process are critical for safety and must be standardized and which are legitimate preferences that can be adapted. The sequence of an electronic prescription has mandatory components for safety reasons, but the order in which data are entered, the presentation of interaction information, and the method of confirming the dose can vary according to user preferences. Offering customization options, saving user-specific settings, and allowing shortcuts that do not compromise safety are ways to respect professional

autonomy without sacrificing standardization where necessary.

The legal and ethical dimensions of clinical work are also uncharted territory for most engineers. In the world of software, errors typically result in consequences limited to financial losses or user inconvenience. In medicine, an error can mean the death of a person and that possibility is always present, fostering a culture of prudence and responsibility that permeates every decision. Engineers working in this field must understand that their systems will be used by professionals who are afraid of making mistakes, who know that the lives of others depend on their decisions, and who carry the emotional burden of past errors. A system that ignores this reality—one that adds more pressure or more opportunities for error to an already stressful job—will be doomed to failure and will cause harm in the process.

There is also an emotional aspect to medicine that engineers tend to overlook: suffering. Patients are not just bodies with illnesses; they are people with fears, hopes, families, and stories. Doctors are not merely implementers of protocols; they are human beings who choose every day to confront the pain of others as a way of life. Software that treats patients as mere sets of data, which reduces their experience to coded variables and ignores the human dimension of the clinical encounter, will be betraying the very essence of medicine. The engineer who designs it may never have seen a patient receive a devastating diagnosis, may never have been in an oncology waiting room, and may never have witnessed the relief in a mother's eyes when she is told her child is well. If they haven't had those experiences, they should at least listen to those who have and let them guide their work.

Finally, it is important for engineers to understand something about their own place in this ecosystem. They are not casual visitors arriving with prefabricated solutions but rather other members of a team that share a common goal: improving people's health. Their technical knowledge is valuable, but it is not the only valuable knowledge. Doctors know things that engineers do not and recognize that asymmetry is not a form of humiliation but rather the first step toward fruitful collaboration. The engineer who arrives with humility, who asks questions, who listens, who observes, and who respects clinical knowledge even if they do not fully understand it, will be welcomed. The one who arrives with arrogance, convinced that technology will solve everything, will be rejected—and rightly so. At the intersection of medicine and engineering, as at all intersections, courtesy and mutual respect are just as important as technical knowledge.

This cross-disciplinary literacy that we have explored in the last two sections is, therefore, neither a luxury nor a courtesy. It is the basic infrastructure upon which a genuine collaboration can be built between two worlds that need each other. Doctors need to understand software so as not to be mere passive recipients of technology designed by others. Engineers need to understand medicine so that they do not build tools that look brilliant on paper but are useless in practice. When both make the effort to learn each other's language, when they sit down at the table with respect and curiosity, then truly extraordinary things can happen. Afterward, the software ceases to be a foreign body implanted in the healthcare system and becomes a natural extension of human intelligence and compassion.

If we have established that doctors and engineers need to learn each other's languages to collaborate effectively, how do we build a common technical vocabulary that allows both worlds to communicate precisely when discussing safety, quality, and risk—concepts that have profoundly different yet equally valid meanings in each discipline?

REFERENCES

- Software as a Medical Device | International Medical Device Regulators Forum [Internet]. [cited May 13, 2026]. Available from: <https://www.imdrf.org/working-groups/software-medical-device>
- Medical Device Software Guidance Navigator | FDA [Internet]. [Cited May 13, 2026]. Available from: <https://www.fda.gov/medical-devices/regulatory-accelerator/medical-device-software-guidance-navigator>
- Digital Health Software Precertification (Pre-Cert) Pilot Program | FDA [Internet]. [Cited May 13, 2026]. Available at: <https://www.fda.gov/medical-devices/digital-health-center-excellence/digital-health-software-precertification-precert-pilot-program>
- Gotterbarn D. Software Engineering Ethics. *Encycl Softw Eng* [Internet]. January 15, 2002 [cited May 13, 2026]; Available from: <https://ethicsunwrapped.utexas.edu/case-study/therac-25>
- Obermeyer Z, Powers B, Vogeli C, Mullainathan S. Dissecting racial bias in an algorithm used to manage the health of populations. *Science* (80-) [Internet]. October 25, 2019 [cited May 13, 2026]; Available from: [doi/10.1126/science.aax2342?download=true](https://doi.org/10.1126/science.aax2342?download=true)
- Collier R. NHS ransomware attack spreads worldwide. *C Can Med Assoc J* [Internet]. June 5, 2017 [cited May 13, 2026];189(22):E786. Available from: <https://pmc.ncbi.nlm.nih.gov/articles/PMC5461132/>
- MacIntyre CR, Lim S, Quigley A. Preventing the next pandemic: Use of artificial intelligence for epidemic monitoring and alerts. *Cell Reports Med* [Internet]. December 20, 2022 [cited May 13, 2026];3(12):100867. Available at: <https://www.sciencedirect.com/science/article/pii/S2666379122004311>
- Cross JL, Choma MA, Onofrey JA. Bias in medical AI: Implications for clinical decision-making. *PLOS Digit Heal* [Internet]. November 1, 2024 [cited May 13, 2026]; Available from: <https://doi.org/10.1371/journal.pdig.1009000>