Jonathas Castilho

Agile as Rule-Governed Behavior: A Radical Behaviorist Examination of Software Development Practices in AR environment

São Paulo

14 de dezembro de 2025

Jonathas Castilho

Agile as Rule-Governed Behavior: A Radical Behaviorist Examination of Software Development Practices in AR environment

Work submitted to the Computer Science Program at the University of São Paulo, as a partial requirement for obtaining the Bachelor of Computer Science degree.

Universidade de São Paulo

Instituto de Matemática, Estatística e Computação

Bacharelado em Ciência da Computação

Orientador: Professor Dr. Eduardo Colli

São Paulo

14 de dezembro de 2025

Abstract

The development of augmented reality applications is characterized by high variability, technical constraints, and frequent environmental change. In the absence of an explicit development process, such variability tends to generate disorganized practices that fail to produce software increments with observable value for the client. This work examines agile software development as a set of rule-governed cultural practices that organize variation and increase the probability of functional outcomes. Drawing on radical behaviorism, agile practices are analyzed as arrangements of contingencies that shape developer behavior through short feedback cycles, rapid contact with consequences, and reinforcement of effective responses. The analysis is grounded in a case study of MatematecAR, an augmented reality application developed for a mathematical exhibition. By applying agile artifacts and ceremonies to the project, this study evaluates how reinforcement-based practices support the selection and retention of productive behaviors in an AR development environment, reducing the disruptive effects of change while preserving adaptive variation.

Keywords: agile; software development; punishment; radical behaviorism; augmented reality.

Sumário

1 Introduction

Matemateca is an interactive exhibition that presents mathematical ideas through direct manipulation of physical objects. Visitors come to the Institute of Mathematics, Statistics and Computer Science to engage with this environment, but they are often faced with many possible interactions, which can confuse newcomers. To orient them, augment the environment, and expand the range of viable experiences, Professor Eduardo Colli initiated the development of an augmented reality application that overlays 3D models onto the exhibition.

In experimental environments of this kind, variability is necessary for the selection of effective solutions. At the same time, such variability requires a controlled workflow to avoid maladaptive consequences. Prototypes must be produced and tested rapidly so that ineffective variants are extinguished and more functional ones are retained through selection by consequences. The MatematecAR application emerged from this iterative process.

This project focuses on managing the multidisciplinary group responsible for developing the application. At the time, the software was still immature, and contributors modified it with few constraints. The absence of structured practices weakened the group's ability to generate successive increments that consistently increased the product's value over time.

Therefore, this work proposes a method to reduce the disruptive effects of change while preserving alternative developmental paths that can later be selected. Formal development processes are treated as cultural practices that stabilize variation and enhance the likelihood that the product will achieve broad client acceptance.

1.1 Context

Augmented Reality (AR) can be understood as a specific instance of what Fred Brooks termed *Intelligence Amplification* (IA): the use of computational systems as tools that reduce the effort required for humans to perform tasks. As Azuma notes, AR systems do not replace human activity but extend it by providing contextual computational support (Azuma 1997). Despite their potential for engagement and entertainment, AR applications present substantial technical

challenges.

According to Milgram's taxonomy (Milgram 1994), applications that combine real and virtual elements belong to a *virtuality continuum*. At one extreme lies a completely real environment; at the other, a fully virtual environment. The degree to which the system must model the world in order to maintain the experience determines its position along this continuum. Virtual Reality (VR) applications lie closer to the virtual extreme, while Augmented Reality applications lie closer to the real-world end, as they depend heavily on environmental sampling rather than complete simulation.

In virtual reality systems, the observer is placed within a fully simulated environment. Under these conditions, developers maintain total control over spatial relations, interaction rules, and physical dynamics. The system has complete access to its own state, allowing predictable and consistent responses. To sustain the illusion of realism, the system must emit responses analogous to those expected from real-world physics. For example, a virtual plate placed on a table must be graspable, movable, and subject to gravity when released. If dropped from sufficient height, the plate should fracture according to the simulated force applied. These responses are not derived from the real world but from internally modeled contingencies.

Augmented reality applications operate under fundamentally different constraints. Rather than simulating the entire environment, AR systems must continuously sample the real world through sensors and adjust virtual content accordingly. The system must update object position, scale, lighting, and occlusion in real time to maintain coherence between virtual and physical elements. Here, the illusion depends on precise tracking and rapid feedback: virtual objects must respond appropriately to changes in the physical environment, including movement, obstruction, and illumination.

Although current AR systems fall short of full immersion, they have been applied across multiple domains. The level of immersion achievable depends on the available hardware and sensing capabilities. Smartphones, for instance, rely on relatively low-precision sensors, which constrains spatial accuracy and limits the reliability of environmental data. As a result, developers must carefully evaluate whether accessible technology can support the contingencies required by a given project. As reported by Azuma (Azuma 1997), contemporary AR applications have been

deployed in areas such as:

- Repair and maintenance: architecture realtime visualization of building projects ; internal mapping of building, showing wires and pipes. Industry building lines, internal structure of machines, etc; (Cuperschmid, Ruschel e Freitas 2012)
- Telepresence: projecting robots and simulating its path during missions; projecting your body to a meeting in another place far away from the office;
- Medical visualization: simulation of surgical procedures; using sensors like Magnetic Resonance Imaging (MRI); ComputedTomography scans (CT); or ultrasound imaging. It is possible to have a 3D model of the insides of a patient, combining the data with their real body giving doctors a live "X-ray".
- Education: interactive books or projecting 3D models of animals, molecules in the educational space; simulation of high risk or high coast training;
- Entertainment: games, such as Pokemon GO; playstation's dual sense controller, which gives tactical feedback according to the game's context and increases the force needed to press the buttons. (DualSense Wireless Controller)
- Sound equipment: noise cancelling headphones, which block outside sounds by adding (augmenting) the ambient sound with the exact frequency needed to superimpose waves and cancel outside noise sounds.

1.2 Motivation

Initially, the project was approached as a game development effort. In this domain, many software engineering practices rely on weak or inadequate development methodologies. According to Aleem (Aleem, Capretz e Ahmed 2016), the game industry commonly adopts approaches derived from the Waterfall model, in which developers prototype, implement, and test with little or no client involvement. This practice is often justified as a way to prevent spoilers and to preserve the intended player experience by avoiding prior exposure to the game narrative. However, MatematecAR should not have been treated under these assumptions.

Due to the absence of a defined development methodology, the project evolved in a monolithic manner. Minor changes in any component required a comprehensive understanding of the entire system. This pattern is typical of immature projects developed by multidisciplinary teams. Initially, the team consisted of students and professionals from Mathematics, Engineering, Education, Design, and related fields.

When presented as alternative, agile methodologies were perceived as unfamiliar and uncomfortable by non-developers. In multidisciplinary teams such as MatematecAR, there was a strong tendency to avoid agile practices altogether (Aleem, Capretz e Ahmed 2016). The introduction of structured software development methodologies was not positively received, and replacing an undefined process proved difficult. The team lacked empirical metrics and work traceability, which reinforced a generalized perception of effectiveness. This perception functioned as a misleading reinforcement, maintaining ineffective practices despite the absence of measurable progress.

As noted by Ken Schwaber, when a project claims adherence to a development methodology but fails to follow it in practice, unpredictable outcomes are produced (Schwaber). In MatematecAR, for example, several months elapsed before any tangible progress was observed or any increase in project value was detected. This is a strong reason to fixate a proper method, so the project would achieve better results in perceivable time.

Mainly, agile would reduce the inherently game development anti-patterns. Games are often developed within rigid pipelines, which promotes strong aversion to change. A narrative typically prescribes a beginning, climax, and end, and gameplay mechanics must conform to the established lore. As a result, development decisions are frequently constrained by early design choices, and deviation from the initial plan is treated as detrimental. The underlying assumption is that a concise and rigid plan defined at the outset will lead to a successful product. Empirical evidence, however, contradicts this assumption, as demonstrated by the MatematecAR project.

Similarly, the development of the MatematecAR application followed a strict, unchanging plan derived from an initial game concept. This proved to be a critical flaw. After two years of development, the outcome was limited to a poorly implemented prototype. The prototype consisted of a single scene for the “Equidecomponibilidade” board, featuring buttons labeled by

color and corresponding modeled puzzle pieces. A physical board was marked with a QR Code, and the application relied on camera-based image tracking to infer the position of virtual objects relative to the real board.

Basically, the application was reduced to interaction by clicking buttons that moved geometric pieces between different shape configurations. This implementation failed to validate the core concept: accurately aligning virtual objects with physical artifacts to effectively support visitor learning.

In an environment characterized by unpredictability and simultaneous user interaction, concerns naturally arose regarding system robustness. Local testing within the team revealed severe limitations. Occlusion of the QR Code, rapid movement, or low camera resolution disrupted tracking and compromised the sense of virtual presence. The registration problem remained unresolved.

2 Methodology

This investigation followed a sequential procedure that combined the examination of specialized literature with the direct production of a software artifact. The procedure began with a review of Agile methods interpreted through principles of behavioral psychology, emphasizing the impact of rule-governed behavior on an imature project; and a discussion on the effectiveness of reinforcement and punishment in the process. Agile artifacts and ceremonies were examined through Skinner's perspective as arrangements that shape developer behavior through immediate feedback, short reinforcement cycles, and continuous contact with task-relevant stimuli.

After this review, the procedure advanced to the application of the examined practices in the development of MatematecAR. The development environment involved two programmers. Only the author's tasks and his contribution to the code was analyzed and gathered as data.

User stories was written as registration of improvement. Each of them followed the traditional model of answering "who, what and why". When viewed through a behavior-analytic approach, this template resembles a contingency, since it specifies relations between an organism's responses and the environmental conditions that select them (Skinner 1969). It is important to mention the differences as well and clarifies that a user story intertwines two distinct contingencies: it organizes the client's sought-after reinforcement conditions while simultaneously arranging the developer's required responses-in most cases the implementation of the solution.

The team held weekly meetings, which functioned as regular occasions for evaluating the outcomes of the previous cycle and arranging the tasks for the next one. Outside these meetings, communication occurred through social media groups, used only when relevant to coordination or progress updates were required.

The project adopted weekly sprints as the shortest interval for delivering increments. Each sprint began with the specification of observable objectives and ended with the delivery of verifiable results.

A simple Scrum board structured this workflow. It included three columns—Sprint Backlog, In Progress, and Done—allowing the continuous tracking of task movement.

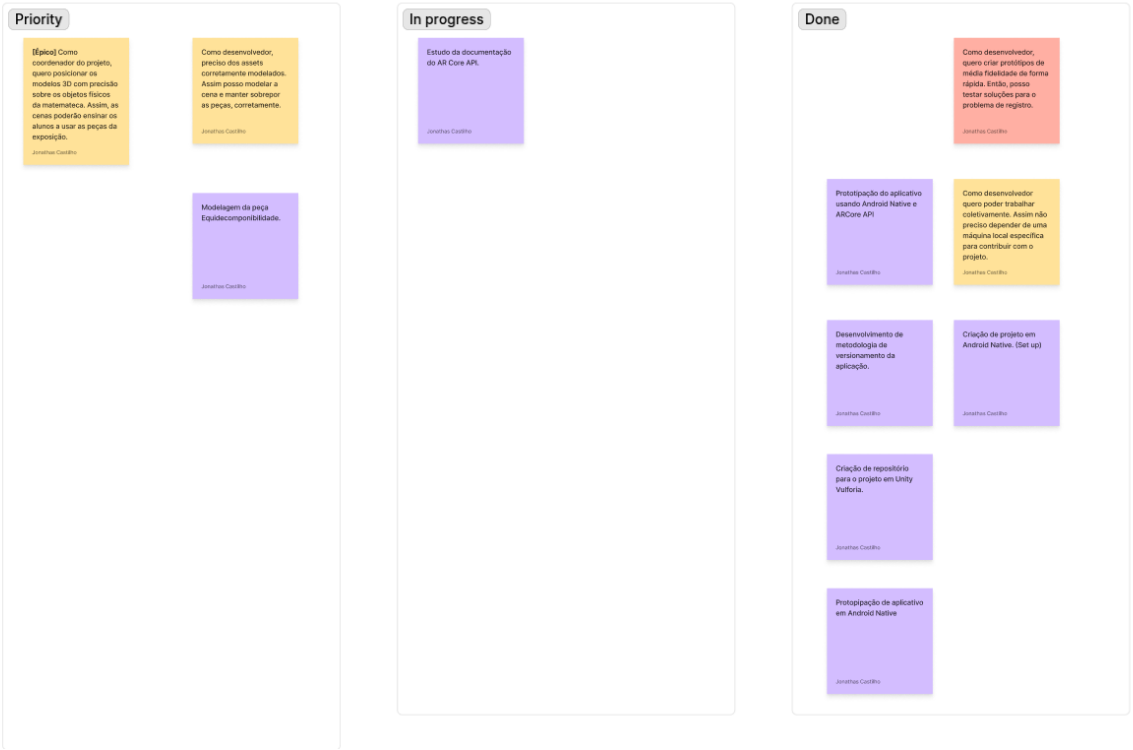

Figura 1 – Scrum board used in the development process of MatematecAR. It containers three columns. Namely, sprint backlog (priority), in progress and done.

Each post-it represented a differnt aspect of development. Yellow cards are user stories; purple ones are atomic tasks related to the completion of a particular story; and red represents overdone user stories.

Evaluation of increments occurred with the participation of the real client, Professor Eduardo Colli. The client's validation acted as an external selective agent by providing immediate consequences for each software iteration (Skinner 1981).

Project wise, MatematecAR was developed using Unity's 6000.1.10f1. It follows a component-based architecture, using Scriptable Objects to manage game data and the MVC pattern to separate logic from UI.

Different software were produced. The one that succeed was Unity with AR Foundation, which is based on an open source Google's API and Apple's ARKit. Both have two core feature for the working of the application: plane detection and fiducial marker detection.

The main constraint for the implementation was the use of mobiles phones, particularly Android OS. This is why operation system must be at least in version Android 7.0 Nougat (or iOS 11) and meet the sensors requirements as follows: one camera sensor (two for more precision),

accelerometer and gyroscope.

Although not a requirement, if the user desires to get the best experience, the phone should have ToF (time of flight sensor-common on top-line Android Phones), LiDAR (Light detection and ranging sensor-only available on Pro version of Apple's phones) or support Depth API.

3 Results

3.1 General obstacles

Augmented reality has advanced alongside modern technologies, yet the registration problem remains as a complex aspect to solve, largely because effective implementation depends on client devices equipped with multiple sensors and substantial capacity for transferring and processing data. For developers, it is also required environments capable of handling the computational demanding software for development.

Given these demands, arranging the available implementation paths becomes essential; without such organization, resources are easily expended on efforts that fail to produce functional outcomes.

3.1.1 Client support

Compatibility for final users also acted as a restrictive variable. Expecting that visitors will have the best phones would exclude many ones from the experience developed. Sponsoring a complete set of Head Mounted Displays (HMDs) for all visitors was also infeasible, and widespread public access to such devices remains limited. These factors selected a different course of action for the project, which culminated in the adoption of smartphones as the primary interaction tool. Their broad availability enables consistent delivery of software to a diverse audience.

Still, they introduce constraints related to processing capabilities, sensor precision, and data capture.

3.1.2 Presence

Presence is a fundamental concept in mixed–reality applications. Milgram's continuum (Milgram 1994) highlights *Real Time Imaging* as the maximal point of presence, a condition in which individuals experience no observable distinction between computer–generated stimuli and the natural environment. According to his formulation, such an effect would be approached through the use of

non–mediated Head Mounted Displays (HMDs). In practice, approaching this limit is nearly impossible, but reducing reliance on resynthesized or sampled imagery tends to bring the virtual and natural environments closer together.

Azuma (Azuma 1997) cite some events which can reduce the sense of presence:

- Optical distortion: cellphone cameras often present radial and tangential distortions due to compact lens assemblies. These distortions reduce the accuracy of feature detection and require calibration procedures and correction models based on mappings (Rolland, Hua e Gao 1994).

- Errors in the tracking system: tracking based on feature extraction and sensor fusion is susceptible to drift, noise, and unstable feature sets. In mobile devices, these errors increase because illumination changes, motion blur, and rapid device movement degrade the stability of the point sets used for pose estimation (Schubert, Qin e Cremers 2021, Billinghurst, Clark e Lee 2015).

- Mechanical misalignments: small displacements among the camera, inertial sensors, and display create fixed offsets that require compensation. These offsets behave as fixed transformations under compositions in the form $T = R \circ S$, and inaccurate estimation leads to cumulative positioning deviations (Rolland, Hua e Gao 1994).

- Incorrect viewing parameters: mobile displays differ in pixel density, field of view, and aspect ratio, generating mismatches between the projection model and the physical display. Without proper calibration, the projection mapping $P : R^3 \rightarrow R^2$ deviates from the intended geometric relation (**??**).

- Dynamic errors caused by system delays: delays arise from sensor sampling intervals, frame processing, and rendering operations. These delays generate systematic lags in the temporal sequence, altering the correspondence between the organism's movement and the device's produced visual consequences (Steed 2016).

- Reduce system lag: decreasing the end-to-end latency requires optimized pipelines, hardware acceleration, and reduced computational overhead during feature processing and

rendering (Steed 2016).

- Reduce apparent lag: apparent lag can be mitigated using techniques such as late-stage reprojection or pose prediction, which adjust the rendered frame immediately before display refresh (Steed 2016).

- Match temporal streams (with video-based systems): sensor streams must be synchronized so that inertial readings, camera frames, and rendering operations correspond to the same temporal interval. Without synchrony, the composed output represents mismatched events (Wen, Schwerdtfeger e Klinker 2013).

- Predict future locations: prediction techniques estimate future device poses using motion models defined on sets of past observations. These predictions reduce perceived delay by aligning rendered output with the organism's upcoming movement pattern (**??**Steed 2016).

- Depth: Finally, occlusion is the strongest depth clue (Azuma 1997), but most mobile devices do not have support for algorithms, api or sensors which would allow applications to create this effect.

All those points would be easier to solve using proper technology focused on mitigating the registration problem in AR. However, phones are not optimal for that.

3.1.3 Registration

Initially, the system depended exclusively on image matching and tracking to establish the model's registration in the environment. By locating and maintaining an anchor in physical space, the application could render the 3D objects in the scene. However, occlusion events or abrupt movements weakened the sense of presence by disrupting registration. A single strategy proved insufficient, and the solution required combining two techniques: image tracking and plane detection. The image established the anchor position, while the plane provided stability for the virtual object once placed.

Presently accessible technologies remain imperfect in guaranteeing a consistent superposition between real and virtual elements. Even high–end systems fail to generate a fully continuous

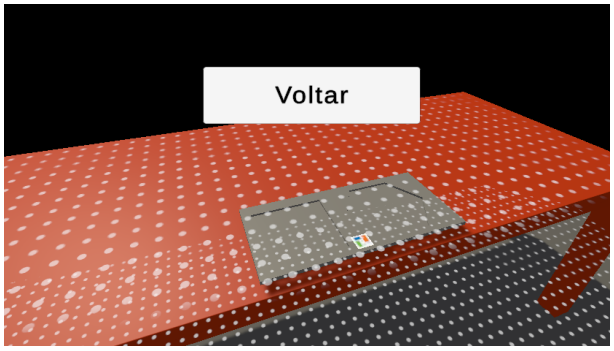

Figura 2 – Simluated AR environment. Firstly, the application detects all the plane. The white dots over the 3D modeled table and floor represent the mesh plane detected in real-time.

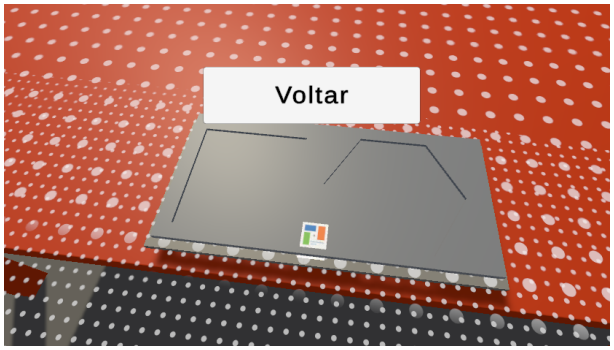

Figura 3 – Once the image is detected, using fiducial tracking tecniques, the virtual version of the Equidecomponibilidade board is place onto the simulated one.

illusion. When designing an augmented reality solution, minimizing registration and sensing errors becomes essential (Azuma 1997). In this project, the application targeted Android devices, which differ widely in sensor configurations, precision, processing power, and data–handling capacity. These variations generate unreliable knowledge of the world (KOW) (Milgram 1994), weakening the degree of presence attainable in the XR environment.

3.1.4 Documentation

Further complications emerged from the state of AR development kits. Documentation is often sparse, outdated, or not actively maintained. For Unity in particular, platform updates are not promptly reflected in official learning materials. Numerous e–books are available, but many describe interfaces or workflows no longer present, creating confusion for beginners.

3.1.5 Hardware

Hardware diversity also influences performance: differences in cameras, GPUs, and CPUs among mobile devices limits the rendering pipeline. It would be good to have sensors which alloweded depath effects and real-time filters to make the experience more realistic. However, most graphical systems rely on the pinhole model, which renders every element in uniform focus. Any advanced effect must be applied as a post-render detail, which is computational consuming. Then, the field of view for real-time rendering is fixed. This disrupts the immersion.

Environmental lighting also operates as a critical aspect for the illusion of presence. So, AR APIs estimate the surrounding light over the virtual objects. However, excessive brightness reduces detection accuracy, whereas insufficient lighting prevents tracking altogether. This is mainly caused by the hardware of the camera or the computational power of the device, compromising the exprience.

3.1.6 Software

The central concerns in the project involved selecting a platform with sufficient resources to reduce the overall workload. Initially, this motivated the adoption of Vuforia, an extension package for Unity that provides a broad set of functionalities capable of addressing the project's requirements. However, its heavy dependence on specific hardware and software environments significantly limited the development process.

Indeed, the software is fully supported only on Windows, which conflicted with the Institute's preference for Linux-based and open–source systems. Consequently, reliance on proprietary constraints became an obstacle to productive work.

In addition, several of Vuforia's features were restricted behind paid tokens. Even after deployment, developers were required to comply with usage constraints imposed by the software's license terms. These conditions created an environment in which both development and distribution were shaped by financial coercion rather than by the project's objectives.

3.1.7 Versioning

The extensive use of assets, scenes, and other non-textual resources introduces significant obstacles when applying traditional version control methodologies. Unlike software projects in which most changes occur in text files, game-related assets are typically stored as binary data, making conflict detection and resolution impractical or impossible. As a consequence, the project adopted a centralized version control system (CVS), which better accommodates binary versioning at the cost of flexibility.

For these reasons, many game development companies rely on centralized version control tools. Such systems require constant connectivity and enforce exclusive access to assets, ensuring that only one individual can modify a given resource at a time. This blocks entirely asynchronous and offline development. Hence, it prevents true concurrent work and introduces bottlenecks in the process.

3.1.8 Missing assets

The project was developed mainly by non-artistical students. Then, asset creation slowed the development process. Then, there was the need to manually model them. This was a bottleneck in the production process, because a lot of time was spent measuring and modeling each scene.

Lack of precision during modeling was also problematic. Superposition won't be possible if it is not precise. It would be much easier to use scanners to recreate the physical objects, but the excessive number of polygons would lead to optimization problems.

3.1.9 Optimization

Optimization in AR is crucial for greater imersiveness. Lag during renderization can make the 3D not reliably superimposed in the scene. The poor performance, then, breaks the sense of presence. Augmented reality requires a much more accurate registration than virtual environments. It means that small inconsistencies can confuse users. According to Azuma (Azuma 1997), the human visual system allures other senses, that is, humans "believe" in their eyes more than the other stimuli.

This is why optimization was not overlooked since the beginning.

3.1.10 Tests

Classical software for smartphones can be tested by running automated tests. Aspects of a software that are costly to implements in automation can be built locally and manually tested. In the case study, developers needed to have physical objects from the exhibition to make sure the application was running as intended. To solve this, it was necessary to create a simple virtualy simulated scene and run the AR application on it. Unity provides a 3D simulation of the real world, allowing developers to model the environment, such that it is possible to test if everything is working properly.

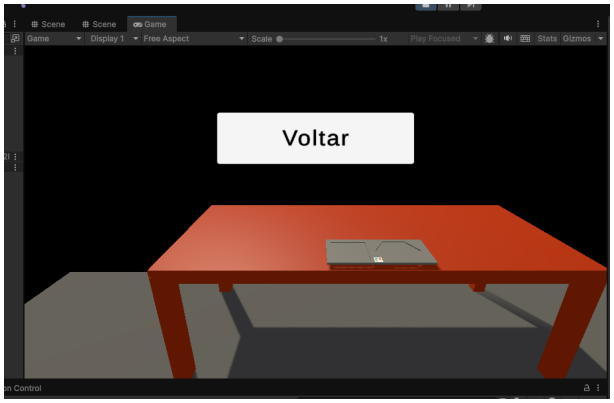

Figura 4 – Simulation of a Matemateca setting modeled in 3D. It uses Unity's XR Environments to run the application over the simulated environment.

This kind of tests should not be an excuse to avoid testing the application in the environment it is suppose to work on, but saves developer's time when they need to test if a part of the code is working properly.

3.2 Contribution

A plethora of studies was needed to understand the domain of the problem. The knowledge required was named above and shows the author understanding of what he was developing. Although technical skills are important to the success of a product, solely knowledge of steps necessary to make a design implemented is not enough. The lack of management skills can be an obstacle to a successful project.

The evolution of human product is not about the evolution of product itself, it is the evolution of cultural practices which were selected by their contribuitons to a reinforcing product from

human behavior (Skinner 1986).

Therefore, not only the AR application was developed, but the process to develop was also expanded and studied to get better results.

3.2.1 Adequacy to requirements

Professor Eduardo Colli proposed a straightforward objective: overlay 3D models onto Matemateca's physical artifacts so that visitors could readily understand how to interact with the exhibition. Achieving this required highly precise registration, capable of sustaining reliable performance under multiple simultaneous interactions and within the variability typical of a museum environment.

The initial scenes' designs followed a classical imitation approach. Teaching through imitation represents one of the simplest procedures for shaping behavior (Skinner 1986). From a development perspective, it is also the least costly strategy, as it avoids the need for elaborate scene orchestration or complex codding. By presenting a modeled action that resembles the target interaction, the system establishes a clear contingency between stimulus and response (Sidman 1971).

Demonstrating the use of a Matemateca artifact through a similar, though not identical, action can effectively evoke the correct behavioral repertoire in the visitor. According to Skinner (Skinner 1986), when an organism reproduces the behavior of another, it increases the probability of contacting the same reinforcement. In fact, "organisms must have profited from the behavior of each other at a very early stage through imitation" (Skinner 1986). In this sense, imitation functions as an efficient mechanism for rapidly establishing effective interaction with the exhibited artifacts.

3.2.2 Advancements

The most significant advancement in the implementation process was the migration to a broader and more stable support platform through ARCore, an open-source library. This transition enabled the team to rely on a wide range of built-in tools while extending compatibility to

multiple devices, including Android phones, iPhones, and head-mounted displays supported by the platform.

Another essential development was the creation of a script designed to stabilize the 3D model within the environment. Professor Colli emphasized the need for strong registration, which was achieved through a combined strategy of image tracking and plane detection. Together, these techniques ensured precise alignment of 3D elements in the scene, even under occlusion. Rapid camera movement and shifting visual context were also addressed by this combination.

Even though it was not the perfect solution, there was an improvement. The initial attempt to use solely QR Code images as tracking techniques. Its precision depended heavily on the variability of sensor accuracy across different smartphone models. Devices with lower-resolution cameras performed inconsistently, and maintaining marker visibility and quality was not always feasible. Because the exhibition's interactive objects were frequently handled, the markers could not be guaranteed to remain intact or preserve the resolution of their original source images.

By using fiducial marker and plane detection, constraints that initially appear insurmountable was addressed through a combination of design decisions and technical strategies. In the Matemateca project, software alone could not ensure the level of precision required for release, especially under conditions of limited compatibility and inconsistent sensor performance. The solution emerged from balancing user-guided actions—such as detecting planes—with automated processes like image detection of fiducial markers. Although the goal was originally to track a physical object and place a 3D model directly on it, this hybrid sequence produced the precision necessary to meet the project's requirements. These adjustments illustrate how technical constraints shape the final workflow, often requiring the addition or reordering of steps to achieve the intended design outcome.

3.2.3 Agile in practice

After the adoption of agile practices, several alternative solutions to the initial problem were proposed. One of the first attempts involved replacing the proprietary Vuforia–Unity integration with an Android native application developed using the ARCore API.

The outcome, however, was not satisfactory. Many functionalities previously handled by

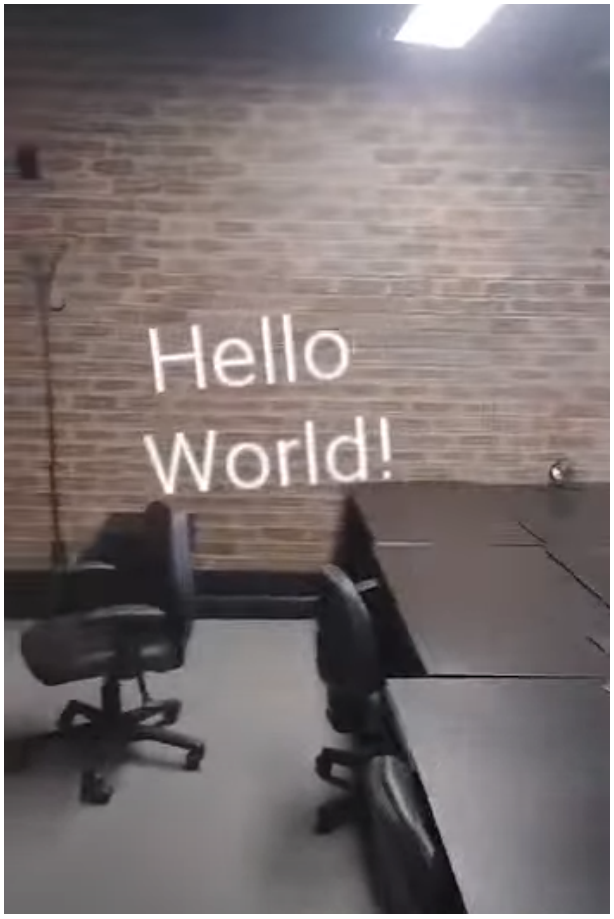

Figura 5 – Android native implementation of MatematecAR using Kotlin and the ARCore API.

Unity required full re-implementation, including API communication, 3D projection handling, and pose-correction mechanisms. The development effort increased, and progress slowed.

To address this, a simpler framework was used. Then, by using react—widely adopted in the developer community— and AR libraries from the community there were a simplification of the development. It offered a more accessible environment, faster iteration cycles, and access to a robust ecosystem of open-source tools. This shift contributed directly to the agility of the project.

Using this framework, early prototypes of the enhanced augmented-reality features were produced. The library Viro React enabled rapid integration of 3D content using plane detection and direct manipulation gestures.

Further iterations led to more refined models. Blender was used to create accurate 3D assets based on existing Matemateca phsyical objects, and animation libraries for React facilitated interactive implementations.

Beyond technical prototyping, agile processes encouraged the exploration of learning strate-

Figura 6 – User interface developed with the React Native framework. HTML-like tags and Tailwind-style utilities improved responsiveness and layouts.

gies before committing to software implementation. Low-fidelity experiments helped validate interactions and conceptual structures.

To demonstrate the increase of delivered value, agile metrics were collected. The burndown chart shows how work progressed throughout the sprints.

The cumulative flow diagram provides an additional view, illustrating the relationship between new stories and completed ones.

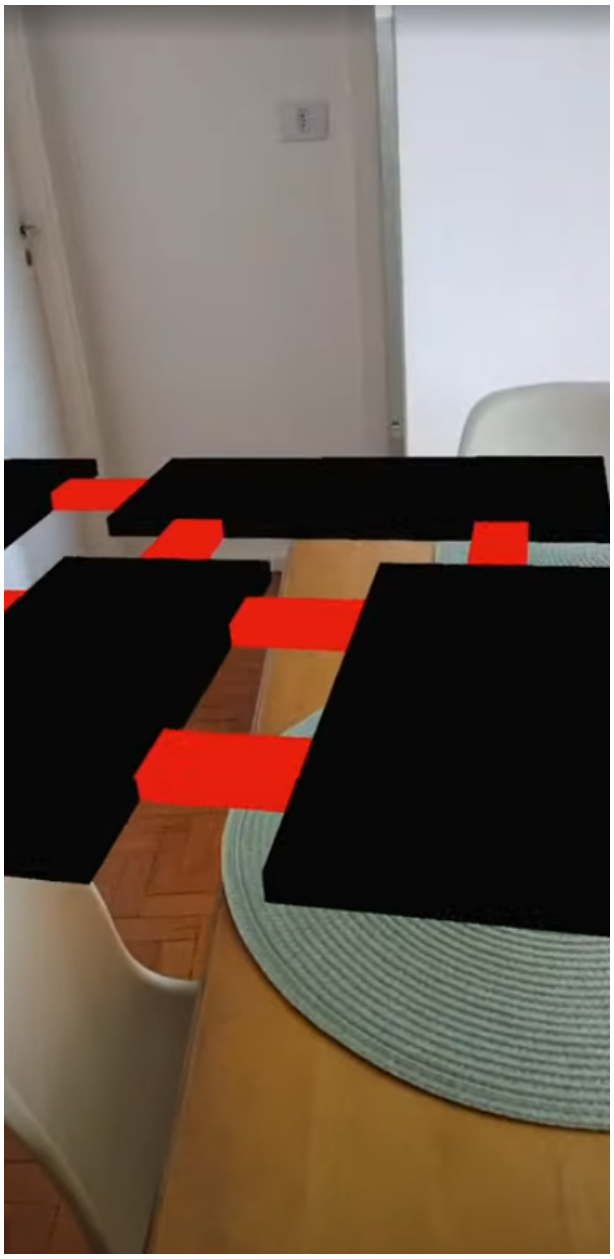

Figura 7 – Initial overlay prototype for the Königsberg Bridge model using Viro React. Plane detection positioned the 3D object, allowing users to drag the model across the detected surface.

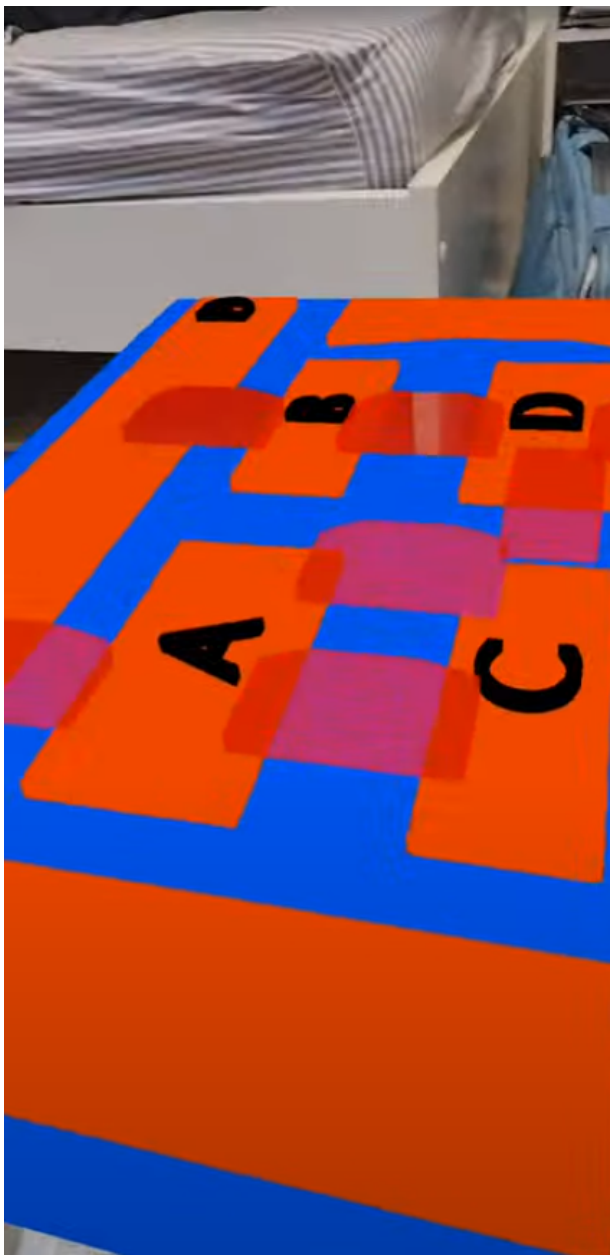

Figura 8 – Accurate 3D representation of a Matemateca object, modeled in Blender and animated using React-based libraries.

Figura 9 – Paper prototype illustrating graph reasoning being introduced to Matemateca's visitors. They connected nodes to determine shortest paths, later incorporating weighted edges to also determine the shortest path. It introduced them to ideas related to the Eulerian bridge problem.

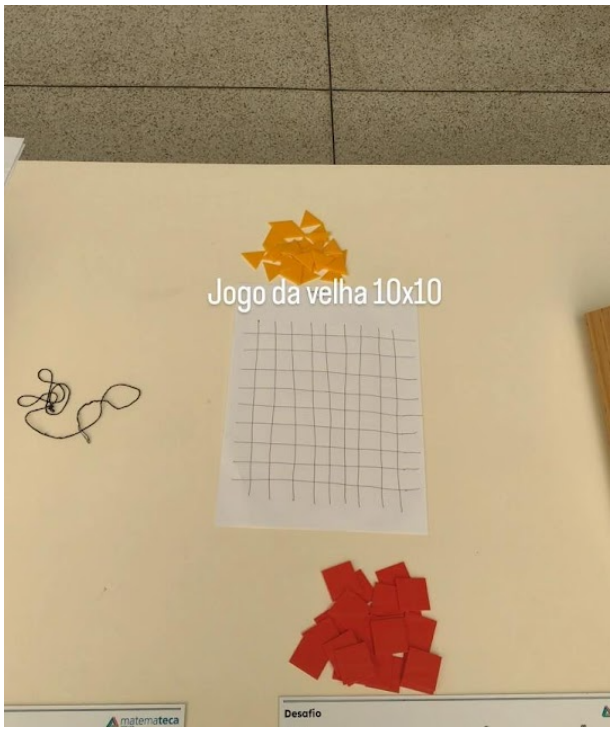

Figura 10 – Prototype illustrating the Gomoku game. The objective was to identified arrangements of five aligned positions, expanding the traditional 3x3 tic-tac-toe. Visitors contact this prototype to acknowledge variation of the game.

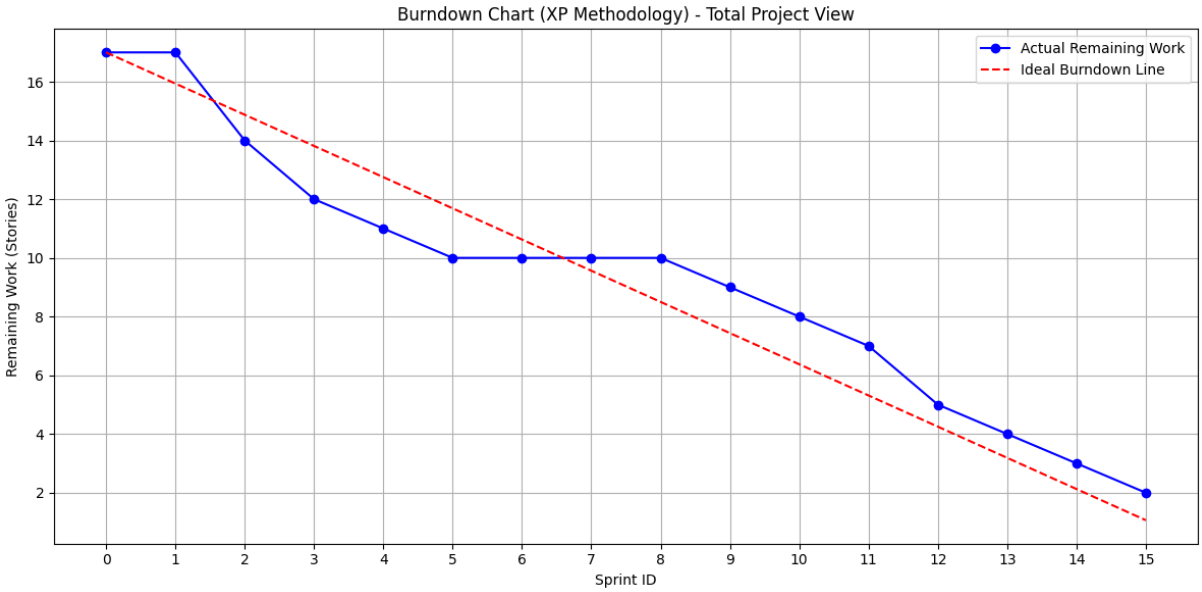

Figura 11 – Burndown chart displaying user-story completion over time.

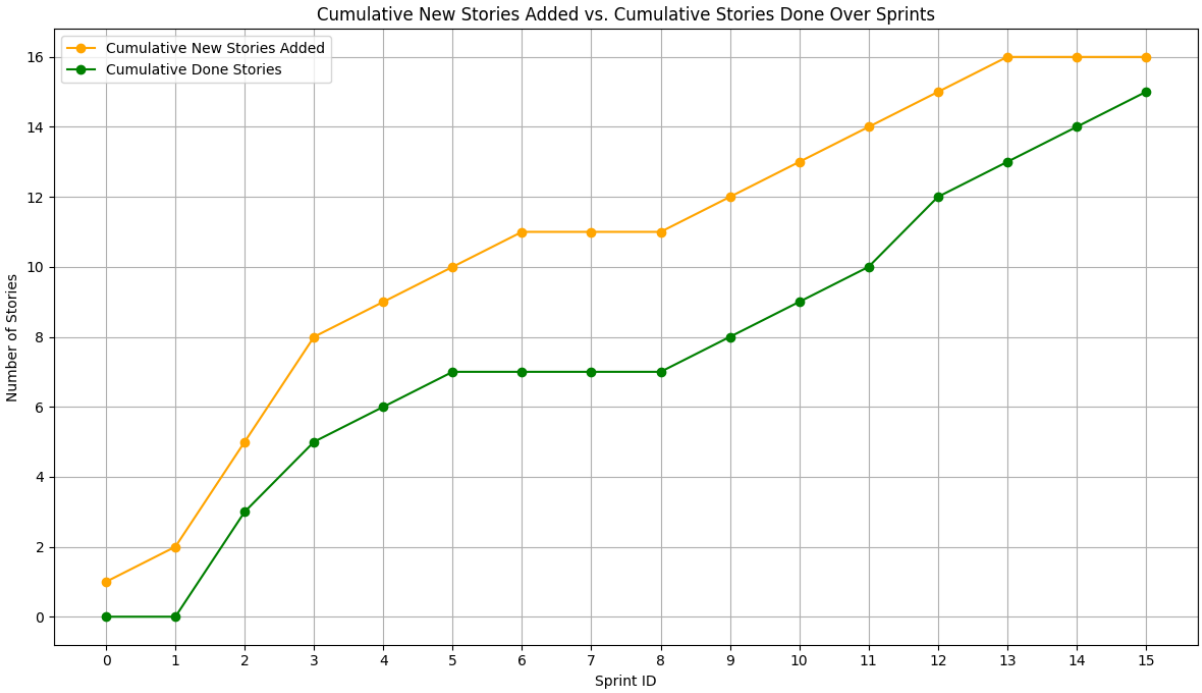

Figura 12 – Cumulative chart showing rate of new story creation and completion across the project timeline.

4 Discussion

"Evolution favors those that operate with maximum exposure to environmental change and have optimised for flexible adaptation to change", says Ken Schwaber (Schwaber). In his view, software development resembles a system continually shaped by external conditions, in which creators are responsible for the success of the artifacts they generate.

Scrum, XP and Lean are examples of agile methodology which accounts for the change Schwaber describes. Their artifacts and cultural practices vary, yet all rest on a set of foundational principles. These principles are based on empirical patterns of behavior whose procedures increase productivity, the very definition of ruled-governed behavior (Skinner 1953).

In fact, when studying Skinner's behaviorism (Skinner 1953), there is a strong parallel with the practices and how to understand them by behavioral psycology perspective.

Therefore, blindly following the practices will not produce a change in the environment. They are rule-governed behavior. That means that, they describe contingencies - relation between organism and environment -, but solely cannot be effective to maintain an agile behavior. For that, there are crucial parts of the process that by skipping those will produce no change on how the team treats software develompent.

The following discussion examines these principles in terms of how they may function as punishment and how such consequences can be prevented. The analysis follows Skinner's definition of behavior. Then, only reinforcement alters the frequency of an organism's response and that this alteration supports genuinely agile and productive conditions (Sidman 1989).

As it will be exposed, punishment do not help teaching developers how to respond accordingly to agile (Sidman 1989).

Two categories of punishment will be utilized to describe how agile can help punishing devlopers, as definded by Sidman and Skinner:

- **Positive punishment:** the introduction of an aversive event following a response.
- **Negative punishment:** the removal of access to something that functions as reinforcement.

It is essential to note that the terms “positive” and “negative” carry no moral or ethical meaning. Both are ineffective on behavioral change and may cause byproducts.

4.1 Agile manifesto

Tech companies organize their activity around the success of the development process. As a collection of organisms whose primary reinforcer is financial return-which is by itself coercive-, their managers and project owners hold the responsibility of arranging conditions so that stakeholders contact beneficial outcomes and the product attains commercial viability. Within this settings, each organism emits responses that follow its own access to what benefits them. Then, behavior is maintained not by moral constructs but by the contingencies arranged in how these relations are socially played.

To establish conditions that increase the likelihood of favorable outcomes, a group of developers formulated the Agile Manifesto. Agile organizes activity through a set of recurring practices—such as daily meetings, weekly sessions, and retrospectives—and through procedures like code review and pair programming.

Lets analyze each practice and discuss how they can be interpreted through behaviorism.

4.1.1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

While developers work on the requirements, there is no access to validation of what was implemented. Then, they are likely to make incorrect assumptions, which can turnout the product useless. If the cycle is short enough, it allows the client to ask for changes the earliest possible. This reduces the need waste complex implementation. Since little was done, little will be eliminated.

These rapid adjustments function as discriminative stimuli (Skinner 1953). When an organism emits a variation—such as a new design alternative or a fresh piece of code—the variation must contact consequences at once. If consequences are delayed, the relation between response and outcome weakens, and no effective pairing occurs. In software development, when develo-

pers present a solution, the client selects the variation that produces better results, much like how animals get their behavior selected under environmental pressure. This selection clarifies which practices are effective and which require further shaping (Skinner 1953).

Continuous delivery, therefore, operates as a behavioral pattern maintained by a reinforcement schedule that favors small, testable increments. It also sustains code quality. A codebase created under a rigid, long-term plan tends to form tight couplings because the structure evolves in the absence of frequent corrective consequences. When a feature requires extensive time and resources, its rejection becomes unlikely, since a lot of resources was spent until then. Code should be produced only when it reinforces the client's goals—similar to how, in an evolutionary model, traits persist only when they enhance survival conditions.

To avoid the drift toward developer-centered value, increments must carry observable utility. Judgments made only from the local repertoire of the developers may select responses that satisfy personal habits rather than client needs. A feature may be shaped by convenience instead of actual customer demands.

Regular client meetings counter this tendency. In these sessions, users provide feedback that acts as immediate consequences for the increments delivered, validating their usefulness, and guiding the next selective step in the cycle.

4.1.2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Unlike immature software or stiff development, agile embraces change. This is one of the best advances of adopting an agile software development methodology. It helps developers to be exposed to a variety of reinforcements. Selecting those that better suit the client's request, leading to a more adequate product.

Ontogenetically, when an organism is exposed to a variety of reinforcements, it will acquire more repertoire, that is, experience, allowing them to act more adequately and generalize their behavior. The same happens on software development process.

Particularly, in the case study, for example, there has been no resistance for adopting different

technologies and alternatives to propose a solution to the software. At the beginning, it was studied the possibility to implement the whole project using Android Native and call the AR Core API natively. This was much more complex, but allowed developers to select the best alternative: develop in a game engine, but use AR Foundation, which is based on AR Core studied before. Then, not all work was a waste.

4.1.3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

In many domains of knowledge, product development follows a rigid pipeline in which a beginning, a middle phase, and an end define completion. Software development, on the contrary, does not conform well to this structure, as functionality must be continuously tested and validated. The shorter the interval between implementation and client evaluation, the greater the likelihood that changes can be effectively selected and retained.

For selection to occur, consequences must follow responses with minimal delay. Organisms do not rely on deferred internal processing to maintain behavior; rather, behavior is shaped when reinforcement is contiguous with the response. As temporal distance increases, the effectiveness of a contingency diminishes, as it was pointed before, making it more difficult for a response to be selected. Immediate pairing between response and consequence increases the probability that the behavioral pattern will be maintained. In the case of software engineering, the client keeps paying developers and developers maintain their deliverings.

This principle is directly exerted by continuous integration. By rapidly integrating, testing the effect of changes, and presenting those changes to real users, the development process produces immediate consequences for each implemented feature, allowing ineffective variations to be discarded and effective ones to be retained.

4.1.4 Business people and developers must work together daily throughout the project.

Developers carry technical excellence shaped by repeated contact with computational tasks, tools, and design practices. This repertoire enables the construction of what the client requires, but it does not ensure that the product will produce value when put in production.

Business representatives have experience related to product value. When they participate as members of the team, their contributions align developers' perspective with client needs. Under these conditions, the team learn about the domain of the problem and the financial dynamics of the market, determining what should be built and in what order. Through repeated cycles of exposure, feedback, and adjustment, the developers' repertoire expands to include domain-specific patterns of action.

This interaction benefits business representatives as well. Increasing product value is a cooperative task carried out by the entire development team. Developers know the operational steps required when a feature is prioritized in the backlog, and their communication about difficulties, constraints, and missing prerequisites helps business representatives adjust expectations regarding development duration.

For example, producing a highly interactive scene in the case study would yield considerable value, but implementing it would require extensive time and design work, which was not available in a short cycle of Sprints. Foundation needed to be set before moving foward. Alas, more urgent tasks were necessary to be implemented prior advanced results. This is why constructing a simpler scene—with well-positioned text to guide the users on the use of a Matemateca object—produced more immediate utility and was prioritized.

Explaining these limitations to business related people reduces unnecessary workload and developer's overwork. It prevents breakdowns in correctness, as well as avoiding defects introduced when rushed implementations attempt are done solely to satisfy fixed deadlines.

In fact, sustained contact with the application's domain is essential for effective participation in the project. And domination of it is acquired through immitation (Skinner 1986). It shapes the developers' technical variations so that they match the environmental demands of the product from the expirience of business people, increasing the probability that each increment survives

operational use and contributes to correctness.

4.1.5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Motivation is not an inner force that appears without environmental conditions that supply reinforcing consequences. Alas, it is not a behavior sustained prior to the observable behavior, but after. As Skinner emphasized, behavior occurs because of its history of reinforcement, not because of an inferred internal state that energizes action (**??**). Behavior that has contacted reinforcing consequences in similar contexts tends to occur again, and when an organism emits responses at higher rates in the presence of particular discriminative stimuli, observers commonly label this pattern as "being motivated."

Like all organisms acting under operant contingencies, developers contact reinforcement when implementing a feature. Social approval functions as a potent reinforcer in many work environments. Simple verbal consequences—such as acknowledging effective performance and encouraging continuation of successful practices—maintain desirable response classes.

In contrast, punishment produces counterproductive effects and decrease the motivation. As Sidman warns, "Yes, we can lead people to do what we want through punishment or the threat of punishing them for doing anything else, but when we do so, we plant the seeds of personal disengagement, social isolation, neurosis, intellectual rigidity, hostility, and rebellion" (Sidman 1989).

Another aspect that can contribute on motivating operations is deprivation and satiation. They alter the effectiveness of consequences (Michael 1982). Reduced access to a reinforcer increases its strength; extensive access decreases it. Reinforcers should therefore not be delivered in excess, but their accessibility should vary in ways that increase their effectiveness when desirable behavior occurs. This arrangement strengthens the response classes required for the project.

Importantly, decreased access to reinforcers should not be arranged as coercion. Sidman characterizes negative reinforcement contingencies that require compliance to avoid loss of reinforcement as coercive (Sidman 1989). Although not all negative reinforcement is coercion,

establishing a condition of “comply or lose access” functions like punishment and does not reliably strengthen long-term behavioral patterns. Then, although scarcity can increase the value of a reinforcer, its reduction as coercison should be avoided. Doing so can also establish competition for reinforcers, generating conflict and social pressure that reduce cooperative functioning.

When applied non-coercible, satiation can establishing an operation which increases the effectiveness of a reinforcer. As a consequence, behaviors that produce that reinforcer occur more frequently (Michael 1982). Under these conditions, what is labeled as motivation enhances the organism’s ability to acquire the precise response patterns required for high-quality work. Consistent engagement with a task increases the likelihood of contacting reinforcing consequences, strengthening effective response classes and sharpening discriminations between productive and unproductive actions.

Likewise, fear of punishment generates avoidance behaviors, which reduce what is labeled as motivation. One major difficulty in maintaining efficient and committed performance is the presence of competing responses that deliver more immediate reinforcement. This pattern often appears as procrastination. Except where neurological divergences alter sensitivity to contingencies, procrastination emerges when alternative responses offer more reinforcing stimulation than the task at hand, functioning as avoidance of an aversive environment. This avoidance weakens the target behavior and disrupts sustained engagement.

As a discleimer, what appears to come out of nothing as a motivated individual is due to its history of reinforcers. Individual reinforcement histories strengthen some response classes and weaken others (Skinner 1974). These histories create patterns that may appear “goal-directed,” yet the controlling variables lie in environmental contingencies and the organism’s past contact with consequences. Some individuals possess a broader repertoire of events that function as reinforcers, and thus show what is labeled as “higher motivation.” This does not reflect superior character, virtuous personality trait or greater dedication; it reflects differential learning histories. Considering that it is not all developers that have had this apparent "innate"motivation allow the teams to stablish truly motivated individuals.

Across evolutionary history, traits that increased survival and reproduction were selected (Darwin 1859). Classes of responses such as foraging, constructing shelters, cooperative activity, and mating displays trace their persistence to phylogenetic selection. What is labeled "motivation" frequently reflects an interaction between these inherited behavioral tendencies and present contingencies. Thus, praise, recognition, and other reinforcing consequences that maintain productive work increase the probability that such behavioral patterns continue.

Therefore, to keep a team motivated, the development team, which includes the client, should avoid punishment and assumption that motivation happens prior to a behavior. Only reinforcements can make someone motivated to implement something truly innovative and well-developed. There is no use in blaming developers for lack motivation, for there is no internal processes of the mind, such as mindset to drive someone's behavior. As Skinner noted, "it is of no help, in solving a practical problem, to say that some aspect of a man's behavior is due to frustration or anxiety; we must also know how the frustration or anxiety was induced and how it can be altered" (Skinner 1969).

In practice, this means that to produce what is called motivation, the organism must have a history of reinforcement with that consequence and current access to the relevant establishing conditions (Michael 1982). Punishment will decrease participation, produces avoidance, and reduces the probability of sustained engagement.

4.1.6 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Emails or text messages can generate ambiguous interpretations. Human action includes posture, tone, facial movements, and other nonverbal responses that function as additional discriminative stimuli. Research in communication demonstrates that nonverbal cues substantially influence how messages are discriminated and acted upon (Mehrabian 1972). Written communication often removes these cues, reducing the precision with which team members contact the client's needs and adjust their responses accordingly.

Asynchronous communication also competes with numerous distracting stimuli in the client's environment. In contrast, a face-to-face setting establishes a shared stimulus field that reduces

competing reinforcers and increases the probability that relevant cues will be observed. For instance, after a succession of aversive events—a pattern commonly described as “a bad day”—a client may emit stronger aversive behavior in response to a minor defect than they would in a more controlled setting. Synchronous interaction allows the team to discriminate such contextual variables immediately. Research on communication grounding shows that synchronous exchanges support faster error correction and mutual alignment (Clark e Brennan 1991).

In online environments, synchronous meetings still function more effectively than asynchronous message exchanges. Studies comparing synchronous and asynchronous collaboration show that synchronous interactions reduce misinterpretation and support faster convergence on shared understanding (Kock 2005).

Because face-to-face meetings are not always feasible, the environment should be selected according to the client’s repertoire. If the client persists in preferring asynchronous communication, arranging punitive contingencies to force face-to-face interaction would be counterproductive. Under radical behaviorist principles, reinforcement—not coercion—should guide the interaction.

Thereafter, face-to-face conversation functions as an efficient medium for bringing the team into contact with the controlling variables of the project. It strengthens coordinated action, increases the probability of producing useful increments, and reduces errors produced by incomplete or distorted communication.

4.1.7 Working software is the primary measure of progress.

In agile development, progress is not inferred from plans, documentation, or isolated components, but from software that is demonstrably operational. Working software is defined as an integrated set of behaviors that can be executed, observed, and evaluated in its intended context. Partially implemented features, mockups, or non-integrated modules do not constitute progress, because they do not yet participate in the functional system. These artifacts are valuable to developers, but has no value to the client.

From a behaviorist perspective, only behavior that contacts the environment can be selected by its consequences. Code that is written but not executed, or features that exist in isolation, cannot be exposed to variation, evaluation, or reinforcement. As a result, such artifacts provide

no empirical evidence of improvement. In contrast, working software produces immediate and observable effects, allowing stakeholders to respond to its outputs and shape subsequent development.

Non-integrated features also delay feedback. When components are developed independently and combined only at later stages, the interval between implementation and evaluation increases. This temporal gap weakens the contingency between developer actions and their consequences, increasing the probability of persisting ineffective design choices, as it was explained earlier. Integration, therefore, is not a final step but a continuous requirement for meaningful progress.

Measuring progress through working software also constrains overproduction. Plans, specifications, and partially completed features may accumulate without ever contacting real use conditions. This accumulation creates the appearance of advancement while failing to increase the functional value of the system. Working software, by contrast, must satisfy minimal criteria of correctness, usability, and reliability, even in early iterations.

In practice, this principle demands that development cycles prioritize end-to-end functionality. Each increment should result in a version of the system that can be run, tested, and experienced by its real users on the real environment the software is suppose to run. Even when functionality is limited, its execution allows rapid evaluation and selection of effective variations. Progress, therefore, is not the quantity of implemented code, but the degree to which the system's behavior functions successfully in its environment.

4.1.8 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Excessive delivery of reinforcing reduces their effectiveness through oversupply. This was also pointed out before. It weakens the selection power of such consequences (**??**). In an agile arrangement, the same selection dynamic appears: when stimuli intended to shape productive behavior are emitted without calibration, their discriminative function diffuses, and the continuity of effective responding decreases.

Abrupt removal of reinforcement can also be problematic. A characteristic increase in

response variation happens when there is an extinction event. This is called extinction burst. When expected consequences stop happening all at once, behavior usually becomes more varied for a short time. Skinner showed this with rats that pressed a lever: when the usual lever-press no longer produced anything, the animals began emitting a wider range of actions built from the old ones plus new variations (**??**). In development cycles, analogous patterns appear as irregular performance, stalled tasks, or abrupt shifts in work allocation—operant variations emitted as the group contacts the absence of expected consequences.

This is why keeping commitments within feasible boundaries prevail nonproductive development. Accepting only the workload that can be reliably implemented ensures that reinforcement schedules remain within a stable range and that costumers will be satisfied with the resources invested in the project.

4.1.9 Continuous attention to technical excellence and good design enhances agility.

Design is a broad concept applied across multiple domains, but its central function remains consistent: to produce structures that are both effective in operation and clear in communication. Good design prioritizes human interaction and comprehension, ensuring that artifacts can be understood, modified, and extended without unnecessary effort. As emphasized in the literature, "any fool can write code that a computer can understand. Good programmers write code that humans can understand" (**??**).

Within agile development, technical excellence and good design are not optional refinements but enabling conditions for agility itself. Agility should not be mistaken for improvisation or the absence of careful planning. While agile processes encourage frequent increments, continuous delivery must not function as justification for weakened coding standards or careless architectural decisions. Poorly designed systems accumulate structural complexity that constrains future variation, making change increasingly costly and slower.

Sustained attention to design quality reduces friction in the development process. Code that is readable, modular, and well-structured allows new behavior to be introduced with minimal disruption to existing functionality. In this sense, technical excellence increases the range

of viable future responses, supporting rapid adaptation when requirements change. Agility, therefore, emerges not from neglecting design, but from maintaining it consistently throughout the development cycle.

4.1.10 Simplicity—the art of maximizing the amount of work not done—is essential.

Simplicity serves to prevent unnecessary development and as a mechanism for maintaining adaptability. Rather than encouraging premature expansion, agile practice emphasizes reducing scope to what is strictly required. As Fowler states, “Simplicity—the art of maximizing the amount of work not done—is essential” (Fowler 2001). This principle, commonly expressed in the software community as the KISS guideline, limits structural growth and reduces long-term maintenance costs.

By minimizing complexity, each component is more likely to remain functional under changing conditions. Simple implementations are easier to evaluate, modify, or discard when they fail to produce the expected outcomes. When a component is small and isolated, its removal does not impose extensive costs on the rest of the system, allowing ineffective variations to be rapidly eliminated.

Achieving simplicity depends on clearly defined user stories with explicit completion criteria. These constraints restrict development to observable behavior that satisfies specific requirements, preventing speculative features from being introduced without justification. In this way, simplicity supports continuous variation and selection, preserving system flexibility while directing effort toward functionality that produces measurable value.

4.1.11 The best architectures, requirements, and designs emerge from self-organizing teams.

Self-organizing teams operate without rigid hierarchical control, relying instead on distributed responsibility and shared understanding of objectives. Independence in decision-making and practical awareness of what must be done reduce coordination overhead and accelerate deve-

lopment. When authority imposes behavior through social coercion, behavioral variation is constrained, and adaptive responses are suppressed. As Sidman argues, coercive control limits freedom by restricting the conditions under which behavior can be shaped (Sidman 1989).

In a self-organizing team, members orient their actions toward shared contingencies rather than toward compliance with authority. When individuals understand their roles and act accordingly, coordination emerges through mutual adjustment rather than command. This arrangement enables rapid variation in architectural and design decisions, allowing effective patterns to be selected through their consequences.

Collective monitoring and collaboration arise naturally under these conditions. Team members observe the effects of each other's actions and adjust their own behavior in response, producing reciprocal reinforcement for practices that align with agreed principles. Over time, this process shapes more effective architectures and requirements, not through top-down compulsory prescription, but through the continuous selection of behaviors that produce functional outcomes.

4.1.12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Agile Manifesto stated above that "the best architectures, requirements, and designs emerge from self-organizing teams" (**??**). In such teams, behavioral relations are shaped by ongoing feedback rather than by rigid external compulsory control. Regular moments of reflection create conditions in which the team can contact the consequences of its own practices, allowing ineffective patterns to be weakened and effective ones to be strengthened. This recurring evaluation resembles a Darwinian selection process: behavioral variations arise during work, and those that increase reliability, maintainability, and delivered value are retained.

When difficulties arise in the software development process, the manner in which adjustments are addressed is critical. Harsh criticism and personal accusations do not produce improvement, because such interactions function as punishment. Punitive consequences reduce participation and suppress behavioral variation, rather than increasing the probability of effective change. Blaming individuals for errors does not establish conditions under which alternative behaviors

can be selected.

To increase effectiveness, the team must analyze the contingencies that give rise to inefficiency. By identifying the antecedent conditions and consequences maintaining ineffective practices, developers can deliberately modify the environment in which work occurs. Adjusting stimuli and reinforcement patterns makes it more likely that behaviors consistent with agile principles will occur.

Instead of attributing fault, the focus should be on proposing immediate, actionable solutions. By arranging new contingencies that support collaboration, experimentation, and rapid feedback, teams can actively shape behavior toward more effective practices and sustain continuous improvement.

4.2 Punishment in Agile

Punishment appears in agile environments even when teams attempt to emphasize collaboration. In practice, coercive contingencies operate through many socially accepted forms, largely because the selective pressures of both phylogeny and culture have favored punitive practices that suppress behavior perceived as disruptive to group activities (Sidman 1989).

In software organizations, these contingencies frequently regulate developer activity through harsh or intimidating client interactions; ironic or disdainful remarks during meetings; exclusion of adequate performers in favor of exceptional ones; reprimands during daily ceremonies; ambiguous or unstable task demands that increase aversive load—particularly for neurodivergent individuals; repeated formal warnings; and the social assignment of undesirable tasks.

Positive punishment contingencies can also occur when teams schedule meetings in which communication failures are publicly exposed; or when developers are compelled to lead client demonstrations which exposes flaws in the implementation. These events often occur during daily ceremonies or retrospectives and operate by presenting aversive stimuli following a target response. Additional forms include documenting resistance to change during retrospectives; assigning documentation or code review as a consequence for opposing new practices; or delivering corrective feedback that frames dissent as incompatible with organizational values. Some teams may even require dissenting developers to participate in refining the very requirements

they previously resisted, thereby introducing further aversive conditions.

Moreover, negative punishment appears in routine team activity. Exclusion from client discussions; removal from collaborative decision-making; withholding assistance when developers request support; or restricting autonomy in task selection; which serves as contingency of removing access to reinforcing environments. Organizations may further discount a developer's broader contributions by recognizing only work that aligns with imposed changes.

Finally, punitive outcomes extend into broader organizational practices. Developers may be dismissed, and performance metrics—such as burndown charts, version control or delivery measurements—produced by agile routines can become components of the selection environment that contributes to termination. Additional aversive practices include removing bonuses, restricting choice of tasks, excluding individuals from client meetings, or employing competitive gamification systems that regulate access to privileges. These contingencies are widely reported across development communities and often coincide with descriptions of abusive or high-strain workplaces.

Although such contingencies can suppress behavior rapidly, they do not strengthen the reinforced patterns of action needed for long-term project stability. On the contrary, exposure to many aversive conditions frequently generate countercontrol, in which the organism emits responses that oppose or weaken the coercive source (Sidman 1989). A developer subjected to repeated criticism may withdraw from collaboration, reduce interaction, or minimize visible effort. These patterns are predictable outcomes of coercive environments and not indicators of individual inadequacy.

Hence, coercive contingencies reduce productivity because both employer and employee continually adjust their actions to avoid further aversive stimulation. As Sidman demonstrated, punishment does not create effective behavior; it suppresses activity, induces escape and avoidance, and contributes to the progressive weakening of an organism's behavioral repertoire (Sidman 1989). Skinner likewise emphasized that punishment produces only temporary suppression and, when extended across a system, moves the organism toward behavioral extinction rather than durable improvement (Skinner 1953). Under such conditions, activity collapses into the minimal output required to escape or avoid aversive events, and functional variability—the

substrate upon which selection operates—declines.

4.3 Reinforcement Instead

Punishment does not establish durable behavioral repertoires. Only reinforcement produces stable changes in response probability over time. In effective development teams, behavior is shaped by selectively reinforcing variations that increase project value, rather than by suppressing deviations through aversive control. Errors and inadequacies, when treated as occasions for punishment, reduce participation and narrow the range of behavioral variation. In contrast, maintaining diversity of solutions allows multiple variations to be emitted, from which effective practices can be selected by their consequences.

Reinforcement can be arranged in both social and material forms. Public recognition during reviews, acknowledgment of effective problem solving in meetings, or granting greater autonomy after successful delivery function as positive reinforcement. For example, when a developer introduces a refactoring that reduces complexity and improves maintainability, immediate social approval and continued access to similar tasks increase the probability that such behavior will recur. Likewise, timely feedback confirming that an implemented feature meets user requirements reinforces adherence to quality and delivery standards. Further observable benefitial consequences should should be derived from the software developed. Then, a strong pairing with agile and the reinforcer keeps the occurence of the desired behavior.

To produce generalization and stable learning, behavior must be shaped through small steps. This can be done with successive approximations to the desired repertoire being consistently reinforced. Large behavioral changes rarely emerge at once; instead, effective practices are gradually selected when intermediate variations contact reinforcement. Maturity models offer a structured way to arrange these contingencies at the organizational level. As teams advance through higher levels of process maturity, effective practices are systematically reinforced. Frameworks such as CMMI, for example, define key process areas whose successful adoption can be followed by explicit consequences, including expanded decision-making authority, financial incentives, or formal recognition. When a desired response occurs within these environments, reinforcement must follow promptly; otherwise, the practice will not be selected and will fail to

stabilize as part of the team's cultural repertoire.

For this reason, approximations should be reinforced rather than suppressed. If a developer begins to involve the client—an action that previously did not occur—but fails to fully maintain simplicity or technical excellence, the appropriate response is reinforcement, not reprimand. Strengthening the newly established behavior increases the probability that further refinements will occur in subsequent iterations, allowing more complete adherence to agile principles to be shaped over time.

5 Conclusion

People change, markets shift, user preferences vary, projects are interrupted, and business decisions redirect priorities. These continuous environmental alterations select some implementations over others, but instability and pressure alone do not generate functional variability. Selection operates only on variation that is already present, and variability does not arise from coercion or urgency.

Indeed, variation emerges as a product of prior shaping, and its outcomes are selected by subsequent consequences (Skinner 1981). No effective response appears without a history of reinforcement, and the absence of a given repertoire cannot be attributed to those whose behavior has never been shaped or strengthened. When teams fail to produce robust solutions, the controlling variables lie in the training conditions, not in individual intention or effort.

Empowering developers therefore requires repeated contact with consequences through experimentation, testing, and incremental delivery, rather than punishment for reduced throughput or failed attempts. Restricting exposure to diverse conditions and relying on aversive control accelerates behavioral extinction (Sidman 1989). Innovation supplies behavioral variation, but only consequences that strengthen functional responses allow those variations to persist and contribute to long-term viability (Darwin 1859).

Agile environments counteract coercion when arranging frequent feedback, visible consequences, and reinforcement for functional progress. In the MatematecAR project, productivity and learning increased precisely because agile practices were adopted as a way to ensure continuous variation and selection. Developer efforts were socially recognized and reinforced by Professor Eduardo Colli, even when prototypes failed to meet expectations. This arrangement sustained exploration, prevented extinction of initiative, and transformed unsuccessful prototypes into sources of functional adaptation. Without such reinforcing contingencies, MatematecAR would not have functioned as a productive experimental environment, nor would its development process have remained genuinely agile.

References

Aleem, Capretz e Ahmed 2016 ALEEM, S.; CAPRETZ, L. F.; AHMED, F. Game development software engineering process life cycle: A systematic review. *Journal of Software Engineering Research and Development*, 2016.

Aleem, Capretz e Ahmed 2018 ALEEM, S.; CAPRETZ, L. F.; AHMED, F. Critical success factors to improve the game development process from a developers perspective. *arXiv*, 2018.

Azuma 1997 AZUMA, R. T. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, v. 6, n. 4, p. 355–385, 1997.

Beck et al. 2001 BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <https://agilemanifesto.org/>.

Billinghurst, Clark e Lee 2015 BILLINGHURST, M.; CLARK, A.; LEE, G. A. A survey of augmented reality. In: *Foundations and Trends in Human–Computer Interaction*. [S.l.: s.n.], 2015. v. 8, n. 2, p. 73–272.

Bloom et al. 2007 BLOOM, J. D. et al. Evolution favors protein mutational robustness in sufficiently large populations. *arXiv*, 2007.

Child in Time: Children as Liminal Agents in Upper Paleolithic Decorated Caves CHILD in Time: Children as Liminal Agents in Upper Paleolithic Decorated Caves.

Clark e Brennan 1991 CLARK, H. H.; BRENNAN, S. E. *Grounding in Communication*. Washington, DC: American Psychological Association, 1991.

Clune, Mouret e Lipson 2013 CLUNE, J.; MOURET, J.-B.; LIPSON, H. The evolutionary origins of modularity. *Proceedings of the Royal Society B*, v. 280, n. 1755, 2013.

Craig CRAIG, A. B. *Understanding Augmented Reality*. [S.l.: s.n.].

Cuperschmid, Ruschel e Freitas 2012 CUPERSCHMID, A. R. M.; RUSCHEL, R. C.; FREITAS, M. R. D. Technologies that support augmented reality applied in architecture and construction. *Cadernos*, n. 19, 2012.

Darwin 1859 DARWIN, C. *On the Origin of Species*. London: John Murray, 1859.

Developers 2025 DEVELOPERS, G. *ARCore API Reference*. 2025. <https://developers.google.com/ar/reference>.

Draghi et al. 2022 DRAGHI, J. A. et al. Exploring the expanse between theoretical questions and evolvability. *Philosophical Transactions of the Royal Society B*, v. 377, n. 1856, 2022.

DualSense Wireless Controller DUALSENSE Wireless Controller. Disponível em: <https://www.playstation.com/pt-br/accessories/dualsense-wireless-controller/>.

Eco ECO, U. *Tratado Geral de Semiótica*. [S.l.: s.n.].

Fowler 2001 FOWLER, M. *Agile Software Development: Principles, Patterns, and Practices*. [S.l.]: Addison-Wesley, 2001.

Friedlander et al. 2013 FRIEDLANDER, T. et al. Mutation rules and the evolution of sparseness and modularity in biological systems. *arXiv*, 2013.

Gamma et al. 1994 GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley, 1994.

Glenn 1988 GLENN, S. S. Contingencies and metacontingencies: Toward a synthesis of behavior analysis and cultural materialism. *The Behavior Analyst*, v. 11, n. 2, p. 161–179, 1988.

Hansen 2003 HANSEN, T. F. Is modularity necessary for evolvability? *Evolution*, v. 57, n. 8, p. 1523–1530, 2003.

Houle e Rossoni 2022 HOULE, D.; ROSSONI, D. M. Complexity, evolvability, and the process of adaptation. *Annual Review of Ecology, Evolution, and Systematics*, v. 53, p. 137–159, 2022.

Kock 2005 KOCK, N. Media richness or media naturalness? *IEEE Transactions on Professional Communication*, v. 48, n. 2, p. 117–130, 2005.

Malthus 1798 MALTHUS, T. R. *An Essay on the Principle of Population*. London: J. Johnson, 1798.

Masel 2010 MASEL, J. Robustness and evolvability. *Trends in Ecology Evolution*, v. 25, n. 9, p. 406–414, 2010.

McColgan et al. 2024 MCCOLGAN, et al. Understanding developmental system drift. *Philosophical Transactions of the Royal Society B*, v. 379, n. 1903, 2024.

Mehrabian 1972 MEHRABIAN, A. *Silent Messages*. Belmont, CA: Wadsworth, 1972.

Melo e Marroig 2016 MELO, D.; MARROIG, G. Modularity: Genes, development and evolution. *Philosophical Transactions of the Royal Society B*, v. 371, n. 1688, 2016.

Michael 1982 MICHAEL, J. Distinguishing between discriminative and motivating functions of stimuli. *Journal of the Experimental Analysis of Behavior*, v. 37, n. 1, p. 149–155, 1982.

Milgram 1994 MILGRAM, P. A taxonomy of mixed reality visual displays. In: . [S.l.: s.n.], 1994.

Nietzsche 1873 NIETZSCHE, F. *On Truth and Lie in an Extra-Moral Sense*. 1873.

Paulk 2001 PAULK, M. C. *A History of the Capability Maturity Model for Software*. [S.l.]: SEI, Carnegie Mellon University, 2001.

Pélabon et al. 2025 PéLABON, C. et al. Evolvability: Progress and key questions. *BioScience*, v. 75, n. 1, p. 1–14, 2025.

Rolland, Hua e Gao 1994 ROLLAND, J. P.; HUA, H.; GAO, G. A calibration method for head-mounted displays. In: IEEE. *Proceedings of Virtual Reality Annual International Symposium*. [S.l.], 1994. p. 246–255.

Schmandt-Besserat SCHMANDT-BESSERAT, D. *How Writing Came About*. [S.l.: s.n.].

Schubert, Qin e Cremers 2021 SCHUBERT, D.; QIN, T.; CREMERS, D. Visual–inertial tracking for mobile augmented reality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Schwaber SCHWABER, K. *SCRUM Development Process*.

Sidman 1971 SIDMAN, M. Equivalence relations and behavior: A research story. 1971.

Sidman 1971 SIDMAN, M. Reading and auditory-visual equivalences. 1971.

Sidman 1989 SIDMAN, M. *Coercion and Its Fallout*. [S.l.]: Authors Cooperative, 1989.

Skinner 1932 SKINNER, B. F. On the rate of formation of a conditioned reflex. *Journal of General Psychology*, v. 7, n. 1, p. 22–33, 1932.

Skinner 1938 SKINNER, B. F. *The Behavior of Organisms: An Experimental Analysis*. New York: Appleton-Century-Crofts, 1938.

Skinner 1953 SKINNER, B. F. *Science and Human Behavior*. [S.l.]: Macmillan, 1953.

Skinner 1969 SKINNER, B. F. *Contingencies of Reinforcement: A Theoretical Analysis*. New York: Appleton-Century-Crofts, 1969.

Skinner 1974 SKINNER, B. F. *About Behaviorism*. New York: Alfred A. Knopf, 1974.

Skinner 1981 SKINNER, B. F. Selection by consequences. *Science*, v. 213, n. 4507, p. 501–504, 1981.

Skinner 1984 SKINNER, B. F. The evolution of behavior. *Journal of the Experimental Analysis of Behavior*, v. 41, n. 2, p. 217–221, 1984.

Skinner 1986 SKINNER, B. F. The evolution of verbal behavior. *Journal of the Experimental Analysis of Behavior*, v. 45, n. 1, p. 115–122, 1986.

Steed 2016 STEED, A. Understanding and measuring latency in virtual reality systems. *Presence: Teleoperators and Virtual Environments*, v. 25, n. 2, p. 123–136, 2016.

Wen, Schwerdtfeger e Klinker 2013 WEN, M.; SCHWERDTFEGER, B.; KLINKER, G. Mobile augmented reality: A systematic review of techniques, challenges, and applications. In: IEEE. *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality*. [S.l.], 2013. p. 1–10.