REGULAR ARTICLE

Systems Engineering Journal

Understanding Software Engineering Organizations Through Systems Thinking

Juliana Alves[1] | Henrique Alves[1,2] | André Costa Batista[1,3]

[1]Operations Research and Complex Systems Laboratory (ORCS Lab), Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

[2]Graduate Program in Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

[3]Department of Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

Correspondence
André Costa Batista, Department of Electrical Engineering, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil
Email: andre-costa@ufmg.br

Present address
Department of Electrical Engineering, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

Funding information
Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), Grant/Award Number: 001; Dean of Research of Universidade Federal de Minas Gerais (PRPq/UFMG), under Call 10/2024.

Software engineering organizations operate as complex adaptive systems where technical decisions, organizational structures, and market forces interact through feedback loops to produce emergent outcomes. Despite this inherent systemic nature, existing research predominantly examines isolated components rather than their dynamic interdependencies, limiting understanding of why common strategies succeed or fail. This research develops an agent-based model of software engineering organizations as complex adaptive systems, revealing critical resource thresholds determining survival. Through 240 simulations, we identify binary survival gates: minimum runway sufficient to bridge profitability gaps (200 months in the baseline parameterization), dual commercial capacity thresholds (≥6 staff, ≥30% composition), and quality-velocity regimes. The absolute cutoff values are model-dependent and should not be interpreted literally; the central finding is the existence of threshold effects and their implications for strategy. **Keywords** — Systems Thinking, Software Engineering Organizations, Agent-Based Modeling, Complex Adaptive Systems

Abbreviations: ABM, Agent-Based Modeling; TD, Technical Debt; SD, System Dynamics; IT, Information Technology; HR, Human Resources; PMF, Product-Market Fit

Nomenclature

State Variables

F	Product feature completeness
D	Technical debt accumulation
B	Bug count
M	Market share (%)
F_m	Product-market fit
A_{brand}	Brand awareness
A_{org}	Organizational alignment
C_{org}	Organizational conflict
Q_{prod}	Product quality (composite metric)
R	Financial runway (months)
$C_{i,j}$	Capacity j of agent i
V_{agg}	Aggregate development velocity
S_{cap}	Sales capacity
$L_{context}$	Learning context factor
S_{stress}	Stress factor

Parameters - Flow Rates

λ_{feat}	Feature development rate coefficient
λ_{debt}	Technical debt generation rate
λ_{bugs}	Bug introduction rate
λ_{acq}	Customer acquisition rate coefficient
μ_F	Feature deprecation rate
$\rho_{refactor}$	Refactoring effectiveness coefficient
ρ_{fix}	Bug fixing effectiveness coefficient

Parameters - Drag and Quality

δ_{debt}	Technical debt drag coefficient
α_{qual}	Quality investment allocation parameter
α_{HR}	Human Resources coordination capacity (normalized)
κ	Code quality factor
τ	Quality consciousness parameter

Parameters - Financial

γ_{rev}	Revenue conversion coefficient
β_{burn}	Per-employee monthly burn rate
$\beta_{overhead}$	Fixed operational overhead
$F_{funding}(t)$	Funding events at time t

| Parameters - Organizational Dynamics

η_{learn}	Base learning rate
η_{decay}	Capability degradation rate
τ_{base}	Baseline monthly turnover probability
ξ_{decay}	Organizational alignment decay rate
$\xi_{conflict}$	Conflict impact coefficient
χ_{base}	Baseline customer churn rate
χ_{bugs}	Bug-driven churn coefficient
χ_{misfit}	Market misfit churn coefficient

| Other Notation

n_{total}	Total organizational headcount
i	Agent index
j	Capacity type index
k	Management agent index
h	Human Resources agent index
t	Time (months)

1 | INTRODUCTION

Software engineering has evolved from a primarily technical discipline focused on code construction to a complex socio-technical endeavor that encompasses organizational dynamics, human factors, and strategic decision-making (Jr., 1995). Modern software companies face multifaceted challenges that extend beyond technical implementation, including team coordination, market adaptation, quality management, and resource allocation (Sommerville, 2016). The success or failure of software ventures increasingly depends on the interplay between technical decisions, organizational structure, and market forces, yet traditional approaches to understanding software engineering often focus on isolated components rather than their interactions (Boehm and Turner, 2003).

Viewing software engineering through a systems lens reveals it as a complex adaptive system where multiple agents—developers, managers, sales teams, and other stakeholders—interact to produce emergent outcomes (Miller and Page, 2007). In this perspective, organizational performance arises not from individual competencies alone but from the dynamic relationships, feedback loops, and non-linear interactions among system components. Technical decisions ripple through organizational structures, affecting team morale, product quality, and market positioning. Similarly, business strategies influence technical implementations, resource allocations, and development priorities. This systemic perspective challenges reductionist approaches that attempt to optimize individual components in isolation (Meadows, 2008).

Systems thinking provides a framework for understanding how complex systems behave over time, emphasizing circular causality, feedback mechanisms, and emergent properties (Senge, 1990; Sterman, 2000). Rather than seeking simple cause-and-effect relationships, systems thinking recognizes that behavior emerges from system structure—the pattern of relationships and information flows among system elements. In the context of software engineering, systems thinking enables practitioners and researchers to understand counterintuitive phenomena such as how increasing team size can decrease productivity (Brooks's Law), how focusing solely on speed can paradoxically slow

development through technical debt accumulation, or how optimizing individual team performance may undermine overall organizational effectiveness (Repenning, 2001). By making explicit the mental models that guide decision-making, systems thinking helps identify high-leverage interventions and anticipate unintended consequences (Doyle and Ford, 1998).

Despite the inherent systemic nature of software engineering and the availability of systems thinking methodologies, significant gaps remain in how these perspectives are applied to understand software organizations. Existing research predominantly focuses on either technical aspects (architecture, code quality, development practices) or organizational factors (team dynamics, management practices, culture) but rarely examines their co-evolution and mutual influences (Kruchten et al., 2012). Empirical studies of software teams often employ linear statistical models that fail to capture feedback effects, time delays, and non-linear relationships fundamental to organizational behavior (Madachy, 2008). Furthermore, while qualitative case studies provide rich insights into specific contexts, they offer limited ability to explore alternative scenarios, test strategic decisions prospectively, or identify general principles across different organizational configurations. Agent-based modeling (ABM) has emerged as a promising approach to bridge this gap, enabling researchers to formalize system structures, simulate dynamic behaviors, and conduct computational experiments (North and Macal, 2007; Wilensky and Rand, 2015), yet its application to software engineering organizations remains nascent.

This article addresses these gaps by developing an agent-based model of software engineering organizations that integrates technical, human, and market dimensions within a unified systems framework. Our specific contributions are threefold: First, we present a comprehensive agent-based model that represents key stakeholders (engineers, sales, marketing, human resources, and management) as autonomous agents whose interactions generate organizational outcomes including product evolution, market performance, and financial sustainability. Second, we conduct a systematic exploration of organizational dynamics through three computational experiments examining team composition strategies, quality-speed tradeoffs, and financial constraints. These experiments reveal non-obvious relationships between organizational decisions and outcomes, identifying conditions under which common strategies succeed or fail. Third, we provide a methodological blueprint for applying systems thinking and agent-based modeling to software engineering research, demonstrating how computational models can complement traditional empirical methods to generate theoretical insights and practical guidance. The novelty of our work lies in treating software engineering organizations as integrated systems where technical, organizational, and market subsystems co-evolve through feedback-driven processes, and in providing an executable model that can be extended, verified, and applied by other researchers and practitioners.

The remainder of this article is organized as follows: Section 2 reviews related work on systems thinking in software engineering, organizational modeling, and agent-based simulation approaches. Section 3 describes our methodological approach, including the agent-based model architecture, agent types and capabilities, feedback loop structures, and experimental design for three computational experiments. Section 4 reports results from experiments exploring team composition strategies, quality-speed tradeoffs, and financial constraints, identifying threshold effects and non-linear dynamics. Section 5 concludes with theoretical and practical implications, discusses model limitations including talent variance effects, and suggests directions for future research.

2 | RELATED WORK

Our work builds upon three interconnected research streams: systems thinking and feedback mechanisms in software engineering, agent-based modeling approaches to organizational dynamics, and empirical studies of team per-

formance. This section reviews key contributions in each area, identifies their limitations, and positions our work within this landscape.

2.1 | Systems Thinking and Feedback in Software Engineering

Lehman (1996) laid foundational work by proposing that software evolution constitutes a complex, multi-loop feedback system rather than a linear process. He distinguishes between S-type (fixed specifications) and E-type (embedded in real-world) software, arguing that the evolution of E-type systems is driven by feedback mechanisms that determine system dynamics and stability. The FEAST hypothesis posits that focusing exclusively on "forward path" innovations—new languages, tools, methodologies—fails to yield expected improvements because self-stabilizing feedback loops constrain overall process behavior. Empirical evidence from IBM OS/360 growth data reveals oscillatory patterns characteristic of feedback control systems. Building on this theoretical foundation, Lehman et al. Lehman and Ramil (1999) provide empirical validation by analyzing industrial systems, demonstrating that E-type systems exhibit self-stabilizing behavior and long-term deterministic trends. They identify regulatory feedback mechanisms where incremental growth exceeding statistical thresholds necessitates subsequent retrenchment, stabilizing long-term trends. These studies establish that effective software process improvement requires explicit modeling and tuning of global feedback loops rather than isolated technical innovations.

Franco et al. (2023) operationalize Lehman's feedback mechanisms specifically for Technical Debt (TD) management by constructing a causal map integrating software evolution laws with TD dynamics. They formulate interacting feedback loops—"Short-Term Focus" and "Haste Makes Waste"—explaining how shortcut-driven behaviors offer immediate throughput gains but accelerate quality violation accumulation (TD principal), degrading maintenance productivity and increasing change costs through interest accumulation. This work demonstrates that sustaining software evolvability requires explicitly managing trade-offs between functional growth and quality preservation, as these decisions are intrinsically interdependent through feedback structures governing operational lifespan.

Operationalizing these insights, Lee and Miller (2004) propose an integrated simulation framework that synergizes Systems Thinking with Critical Chain Project Management for multi-project environments. Their architecture models continuous feedback loops between Workload, Exhaustion, and Productivity, demonstrating trade-offs between extending project duration, hiring staff, or leveraging overtime. A key finding reveals an "Overwork duration threshold" beyond which boosting work rate leads to sharp declines in productivity due to accumulated exhaustion. This work illustrates how systems thinking enables managers to visualize systemic ripple effects of local decisions across project networks.

Chernoguz (2011) employs System Dynamics (SD) to investigate causal boundaries of Brooks's Law through ProjScout™, explicitly modeling veteran-rookie workforce dynamics. The study reveals limitations of previous SD models (specifically Madachy's) that fail to replicate paradoxical schedule delays under extreme value testing, often yielding linear production gains. By structurally separating veteran and rookie stocks and accounting for non-linear communication entropy and mentoring overhead, the model demonstrates that actual productivity decreases linearly with team size due to coordination burdens. Critically, simulations reveal Brooks's Law is not absolute—high mentoring investment can mitigate schedule slippage when veteran-rookie productivity differentials are large, suggesting context-dependent applicability requiring organizational-specific parameterization of communication costs and learning curves.

Kljajic and Farr (2010) investigate interdependencies between Information Systems, Systems Engineering, and Information Systems Development through the Systems Approach lens. They propose an anticipative framework integrating simulation into SE methodology to facilitate predictive decision support via feedback and anticipative

loops. While these works establish the theoretical importance of feedback mechanisms and provide specific modeling frameworks, they remain limited to particular contexts (multi-project management, process improvement) and do not integrate technical, organizational, and market dimensions within unified models.

2.2 | Agent-Based Modeling of Organizations and Teams

Crowder et al. (2012) developed a seminal ABM framework for simulating Integrated Product Teams by operationalizing socio-technical systems theory. Unlike purely theoretical models, their agents' behaviors—encompassing competency, motivation, trust, and availability—were derived from quantitative data across automotive and aerospace organizations. Simulation experiments reveal that while competency dominates work quality and effort, availability primarily determines completion time, highlighting a critical distinction often missed in organizational research. Furthermore, high technical competency can buffer negative performance impacts of low motivation. This empirically grounded framework demonstrates ABM's potential for risk-free optimization of team composition and workflows, bridging social psychological constructs with technical process modeling.

Extending ABM to cognitive diversity, Lapp et al. (2019) introduce KABOOM, the first agent-based model explicitly integrating cognitive style into team problem-solving simulations. By mapping Kirton Adaption-Innovation factors to simulated annealing parameters, they reveal complex interactions between cognitive composition, communication frequency, and task specialization. Homogeneous innovative teams require significantly higher communication frequencies than adaptive teams, while extreme specialization benefits mid-range styles but harms innovative teams. These findings demonstrate that optimal strategies are contingent upon problem characteristics and team composition.

Meluso et al. (2019) quantitatively link organizational miscommunication to complex system performance through CESIUM (ComplEx System Integrated Utilities Model), integrating network theory, design optimization, and sociolinguistics. They model miscommunication through varying estimate definitions (current vs. future projections), revealing that systems using current estimates statistically outperform those using future projections, with significant performance degradation when future-based communication dominates. Building on this foundation, Meluso et al. (2022) generalize CESIUM as a computational framework for theory-building in Complex Engineered Systems, demonstrating that convergence time scales with network size and macro-level performance depends fundamentally on objective function topology. This generative framework enables synthetic data generation and hypothesis testing regarding organizational and process variables.

While these ABM approaches provide sophisticated frameworks for team dynamics and cognitive diversity, they predominantly focus on engineering design teams and optimization processes, not capturing the full socio-technical complexity of software organizations including market forces, financial constraints, and multi-stakeholder interactions that characterize software ventures.

2.3 | Organizational Dynamics and Empirical Studies

Blackburn et al. (2006) provide empirical validation of Brooks's Law through analysis of 117 software projects using Three-Stage Least Squares estimation. They reveal that while software complexity (logical complexity and interface count) drives team size inflation, larger teams significantly diminish productivity measured as function points per man-month. Critically, complexity affects productivity indirectly through team size rather than directly, demonstrating non-linear relationships characteristic of systemic feedback. The authors conclude that productivity optimization requires managing interdependencies and reducing logical complexity rather than workforce expansion, validating

the counterintuitive dynamics central to systems thinking approaches.

Complementing this quantitative validation, Hindiyeh et al. (2023) conducted a systematic review of 53 articles on engineering team dynamics evolution, providing the first comprehensive bibliometric analysis of this nascent field. Research is heavily skewed toward Information Technology (IT) (34% of studies), predominantly using qualitative or mixed methodologies. The dominant theme (79% of corpus) investigates impacts of specific factors—skillsets, collaboration, personality—on team performance. While collaboration consistently correlates positively with performance, skillset impacts are mixed. The review highlights the field's limited maturity and empirical diversity, identifying needs for research beyond IT contexts and more rigorous statistical approaches.

Petkov et al. (2008) provides theoretical grounding by reconceptualizing organizations as information processing systems constrained by human channel capacity and noise, applying Shannon's Information Theory. The framework explains hierarchies as bandwidth management mechanisms and redundancy as mathematical necessity for error correction—a perspective aligned with communication overhead dynamics central to Brooks's Law and team coordination challenges.

Meta-frameworks contribute organizing perspectives for modeling team and process dynamics. Fan and Yen (2004) taxonomize teamwork modeling approaches across agent-only and mixed human-agent teams, while Wynn and Clarkson (2018) categorize design process models by scope and type. Both reveal that no single framework captures the full complexity of collaborative work, highlighting persistent gaps between theoretical formulations and industrial implementation alongside needs for context-specific model selection.

2.4 | Synthesis and Positioning

Systematic reviews by Sandria-Flores et al. (2024) and Negahban and Yilmaz (2014) reveal broader trends in ABM applications. Sandria-Flores' review of collaborative software demonstrates ABM's role in simulating group dynamics and decision-making processes, facilitating paradigm shifts from individual-centric to group-centered design. However, Negahban and Yilmaz identify critical limitations: prevalence of abstract theoretical models lacking rigorous empirical grounding, and need for robust verification techniques before ABM can mature into reliable decision-support tools.

Zali et al. (2014) systematically review System Dynamics applications in entrepreneurship research (2000-2014), revealing that while 65% of studies include simulations, only 5% extend to designing alternative strategies. The review identifies critical gaps including the need to model internal transformation processes of nascent ventures and dynamic impacts of technological and market changes. Studies concentrate on micro (41%) and meso (38%) levels, with macro-level environmental factors underrepresented. These findings complement Negahban's critique of ABM, suggesting that computational modeling in both SD and ABM traditions remains underutilized for strategic decision support in venture contexts.

The reviewed literature establishes three key foundations: (1) software engineering processes involve complex feedback mechanisms that constrain improvement efforts (Lehman, 1996; Lehman and Ramil, 1999); (2) agent-based modeling effectively captures team dynamics, cognitive diversity, and organizational structures (Crowder et al., 2012; Lapp et al., 2019; Meluso et al., 2022); (3) organizational performance emerges from interactions between technical work, human factors, and structural constraints (Hindiyeh et al., 2023; Petkov et al., 2008).

However, significant gaps remain. Existing ABM frameworks predominantly address engineering design teams or specific organizational processes in isolation, rarely integrating technical development, organizational dynamics, and market forces within unified models. Empirical studies identify important factors affecting team performance but employ linear statistical models failing to capture feedback effects, time delays, and non-linear relationships. Systems

thinking frameworks remain largely theoretical or applied to narrow contexts like multi-project management, without comprehensive operationalization across software venture lifecycles.

Our work addresses these gaps by developing an integrated agent-based model of software organizations that unifies technical (product quality, development velocity), organizational (team composition, management focus), and market (customer acquisition, competitive positioning) dimensions within a feedback-driven framework. Unlike prior ABM studies focusing on optimization or design convergence, our model captures the full venture lifecycle including financial sustainability, strategic pivots, and resource allocation under uncertainty. We extend Lehman's systems thinking foundations and Crowder's empirically-grounded ABM approach by representing diverse stakeholder types (engineers, sales, marketing, human resources, management) whose interactions generate emergent organizational outcomes. Through systematic computational experiments, we identify non-obvious relationships between organizational decisions and performance, revealing conditions under which common strategies succeed or fail—advancing both theoretical understanding and practical guidance for software engineering organizations.

3 | METHODOLOGY

Our methodological approach integrates agent-based modeling with systems thinking to represent software engineering organizations as complex adaptive systems. This section describes the model architecture, agent types and behaviors, feedback mechanisms, and experimental design.

3.1 | Model Architecture and Design Philosophy

We developed an agent-based model implemented in Python using the Mesa framework (Wilensky and Rand, 2015), enabling discrete-event simulation of organizational dynamics over time. Each simulation step represents one month of organizational operation, chosen to balance computational efficiency with meaningful strategic decision intervals. The model architecture follows a modular design separating three interconnected subsystems: (1) the *Technical Subsystem* governing product evolution through feature development, code quality, technical debt, and defects; (2) the *Market Subsystem* capturing customer acquisition, retention, brand awareness, and competitive positioning; and (3) the *Organizational Subsystem* representing team dynamics, alignment, conflicts, and resource allocation.

Unlike traditional equation-based models that impose top-down behavioral assumptions, our ABM approach allows organizational outcomes to emerge from bottom-up interactions among heterogeneous agents operating under local decision rules (North and Macal, 2007). This generative methodology enables exploration of non-obvious dynamics, path dependencies, and threshold effects that characterize real software ventures but resist analytical tractability.

3.2 | Agent Types and Capabilities

The model represents five distinct stakeholder types, each with specialized capabilities that contribute to organizational performance. Table 1 summarizes agent types and their primary attributes.

Each agent's capabilities are initialized randomly within specified ranges, representing the heterogeneity of real organizational talent. Capabilities evolve over time through learning mechanisms modulated by product state, organizational alignment, and strategic focus, implementing experience accumulation and skill degradation under adverse conditions. Capability ranges are bounded at [30, 100] rather than [0, 100] to reflect hiring thresholds—individuals below minimum competency (30) would not pass selection processes or would impose net negative value through

TABLE 1 Agent types, capabilities, and organizational roles

Agent Type	Capabilities	Primary Functions
Engineer	Development Capacity (30-100), Explanation Capacity (30-100)	Feature implementation, bug fixing, refactoring, technical debt management
Sales	Selling Capacity (30-100), Understanding Capacity (30-100)	Customer acquisition, market feedback, revenue generation
Marketing	Improvement Ideas Capacity (30-100), Publicize Capacity (30-100)	Brand building, lead generation, market insight
Human Resources	Coordination Capacity (30-100), Talent Development (30-100), Conflict Resolution (30-100), Hiring Quality (30-100)	Team alignment, turnover reduction, skill development, recruitment
Management	Management Capacity (30-100), Strategic Vision (30-100), Crisis Management (30-100)	Strategic focus, resource allocation, crisis response

coordination overhead exceeding contribution. Initial capability assignments follow uniform distributions across this range—while this simplifies analysis, it abstracts from extreme talent heterogeneity where a small number of high-impact individuals (e.g., "10x engineers" or exceptional founders) could dominate learning, execution, and market feedback. Section 5 discusses implications of this modeling choice for interpreting results.

Engineers drive product evolution through two primary capacities: *Development Capacity* determines the rate of feature implementation and technical work output, while *Explanation Capacity* represents code quality consciousness and communication effectiveness. Engineers with high explanation capacity produce cleaner code, generate less technical debt, and introduce fewer defects. The aggregate engineering team output scales with team size but is subject to coordination overhead and technical debt drag.

Sales Agents possess *Selling Capacity* (conversion effectiveness) and *Understanding Capacity* (market insight generation). Sales teams not only acquire customers but also contribute market feedback that improves product-market fit, creating a bidirectional coupling between commercial and technical activities. Sales effectiveness depends multiplicatively on product quality, brand awareness, and lead quality, implementing dependencies among organizational functions.

Marketing Agents employ *Improvement Ideas Capacity* to generate market insights and *Publicize Capacity* to build brand awareness and generate qualified leads. Marketing effort exhibits diminishing returns as brand awareness saturates, and lead quality depends on the interaction between marketing competence and actual product quality—sophisticated marketing cannot compensate for fundamentally flawed products.

Human Resources (HR) Agents provide organizational infrastructure through four specialized capacities: *Coordination Capacity* improves organizational alignment; *Talent Development* accelerates skill growth across teams; *Conflict Resolution* reduces organizational conflict; and *Hiring Quality* determines the capability distribution of new recruits. HR effectiveness constitutes a critical but often underestimated leverage point, as strong HR functions reduce turnover, accelerate learning, and buffer against organizational dysfunction.

Management Agents set strategic direction through *Management Capacity* (baseline coordination), *Strategic Vision* (long-term planning), and *Crisis Management* (emergency response). Management agents assess organizational

state each period and adjust strategic focus among five modes: engineering-focused (maximize development), sales-focused (maximize acquisition), marketing-focused (build brand), quality-focused (reduce debt/bugs), or balanced (distribute effort). This dynamic focus mechanism models managerial attention allocation under competing demands.

3.3 | Product State Variables

The software product evolves through ten state variables representing technical, market, and organizational dimensions (Table 2).

TABLE 2 Product state variables and their dynamics

Variable	Range	Description
Feature Completeness	$[0, \infty)$	Accumulated functionality scope
Code Quality	$[0, 100]$	Internal code maintainability
Technical Debt	$[0, \infty)$	Accumulated design shortcuts
Bug Count	$[0, \infty)$	Known defects inventory
Market Fit	$[0, 100]$	Product-market need alignment
Market Share	$[0, 100]$	Percentage of market captured
Brand Awareness	$[0, 100]$	Market recognition level
Lead Quality	$[0, 100]$	Sales pipeline qualification
Organizational Alignment	$[0, 100]$	Strategic coherence across teams
Organizational Conflict	$[0, 100]$	Internal friction level

These variables evolve according to differential equations driven by agent actions and modulated by feedback mechanisms. Feature completeness increases with development effort but is penalized by technical debt and bug drag, implementing the widely observed phenomenon that shortcuts accelerate initial development but compound into long-term velocity loss (Franco et al., 2023).

Technical debt accumulates proportionally to development work modulated by a quality focus factor:

$$q_f = \min(C_{exp}/100, 0.8) \tag{1}$$

where C_{exp} is average explanation capacity. The multiplier $(1 - q_f)$ determines what fraction of development effort generates debt—teams with low explanation capacity ($C_{exp} \approx 30$) generate debt at rate $0.7\times$ development effort, while high-capacity teams ($C_{exp} \geq 80$) generate debt at only $0.2\times$ effort. This formulation ensures that code quality consciousness meaningfully reduces debt accumulation rather than the unrealistic scenario where all teams converge to identical debt generation rates.

Technical debt is reduced through explicit refactoring investment (proportion $\alpha_{refactor}$ of engineering capacity) and quality-focused work (proportion $\alpha_{quality}$), with reduction rates governed by configurable parameters (`refactor_reduces_debt`, `quality_reduces_debt`). Bug count increases with development velocity under the same quality focus modulation and decreases through quality-focused engineering effort. Critically, both technical debt and bugs

impose exponential drag on development velocity:

$$V_{effective} = V_{nominal} \cdot \frac{1}{1 + (D/50)^2} \cdot \frac{1}{1 + (B/20)^{1.5}} \tag{2}$$

where D is debt and B is bug count. This non-linear penalty ensures that high debt ($D > 100$) or critical bug counts ($B > 30$) severely impair productivity, creating incentives for quality investment.

Market share grows through sales effort modulated by product quality, brand awareness, and lead quality, exhibiting saturation dynamics as the remaining addressable market shrinks. Customer churn operates continuously, driven by bugs, market misfit, and a baseline attrition rate. Organizational alignment increases through management and HR coordination but decays naturally due to entropy, requiring continuous maintenance. Detailed mathematical specifications for all state evolution equations are provided in Appendix 6.

3.4 | Feedback Loop Structure

The model implements eight primary feedback loops (four reinforcing, four balancing) that govern system behavior. Figure 1 presents the causal loop diagram integrating these mechanisms. The appendix 7 provides detailed mathematical formulations for each loop.

Loop R1 "Short-Term Focus (Reinforcing)" implements the positive feedback between development effort, feature accumulation, market conquest, and revenue generation. As development produces features, market share increases (assuming positive product-market fit), generating revenue that funds additional development. This loop drives growth when product quality remains high but can accelerate organizational dysfunction when coupled with Loop B1.

Loop B1 "Technical Debt Brake (Balancing)" represents the classical quality-velocity tradeoff articulated by Lehman (1996) and operationalized for technical debt by Franco et al. (2023). Development effort prioritizing speed over quality generates technical debt, which imposes a drag coefficient on development velocity, reducing the rate of feature production. This negative feedback loop creates long-term path dependencies where early shortcuts compound into persistent productivity loss.

Loop R2 "Bug Accumulation (Reinforcing)" captures the catastrophic potential of defect cascades. When bug count exceeds critical thresholds, engineering effort increasingly diverts from feature development to firefighting, producing more bugs through hasty fixes, further increasing firefighting demand. This vicious cycle can trigger organizational crisis if not arrested through quality-focused intervention.

Loop B2 "Market Saturation (Balancing)" implements diminishing returns to market conquest. As market share increases, the remaining addressable market shrinks, reducing acquisition efficiency through a $(1 - M)^2$ saturation function. This prevents unrealistic exponential market capture and introduces strategic tension between growth and profitability.

Loop R3 "Brand-Sales Synergy (Reinforcing)" connects marketing effort, brand awareness, lead quality, sales effectiveness, and market share, creating compounding returns to coordinated commercial activity. Strong brands generate higher-quality leads, improving sales conversion, which funds further brand investment.

Loop B3 "Quality Investment Tradeoff (Balancing)" governs the allocation of engineering capacity between feature development and quality improvement (refactoring, bug fixing). Increased quality investment reduces development velocity in the short term but improves it in the long term by reducing debt drag, creating strategic tension between immediate and delayed gratification.

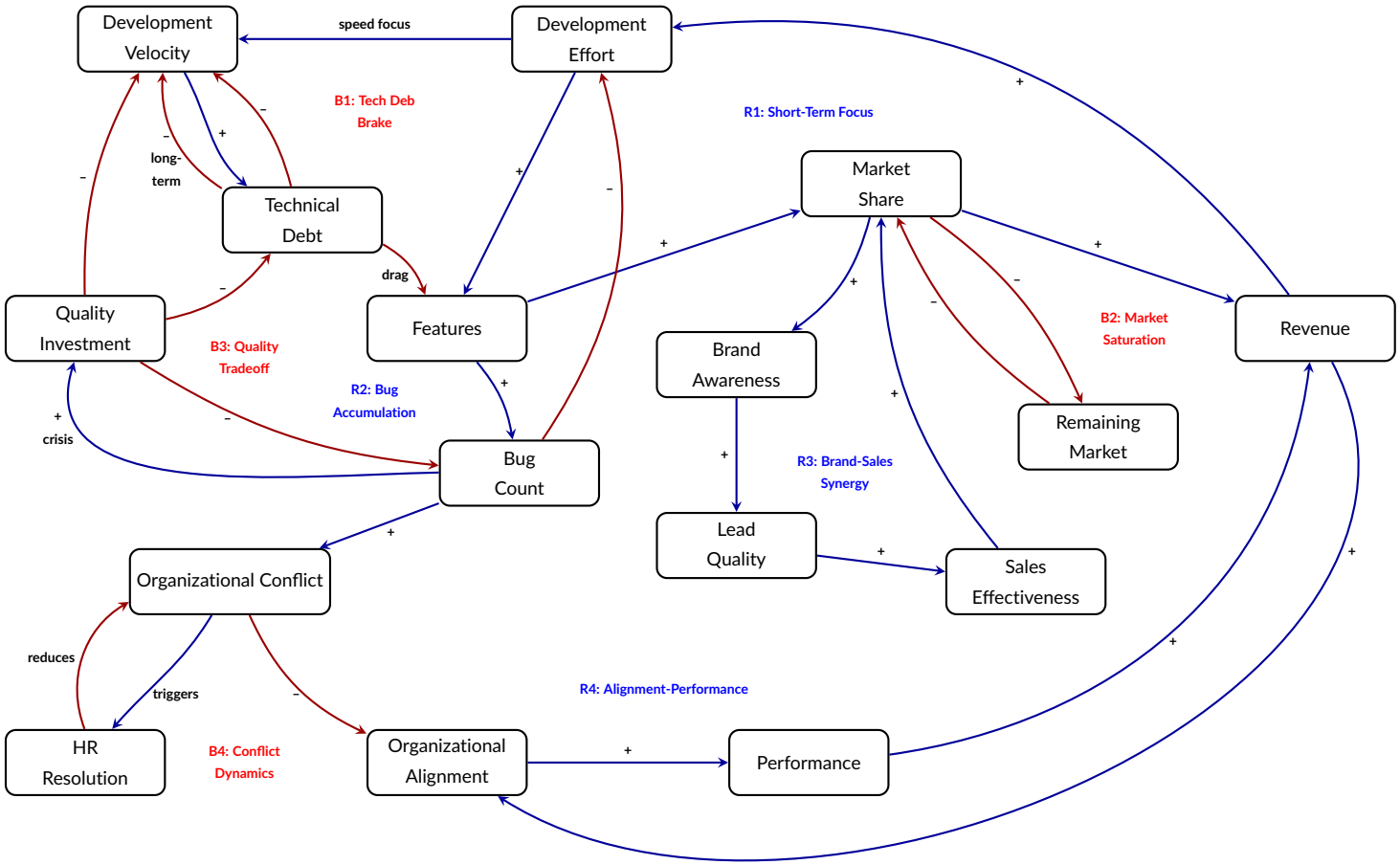

FIGURE 1 Eight primary feedback loops governing organizational dynamics. Reinforcing loops (R1-R4, blue) amplify growth or decline; balancing loops (B1-B4, red) provide constraints and self-regulation. Loop R1 drives growth through feature-market-revenue coupling. B1 implements technical debt drag on velocity. R2 captures bug cascades when defects exceed resolution capacity. B2 prevents unrealistic market saturation. R3 creates brand-sales synergy. B3 governs quality-velocity tradeoffs. R4 couples alignment with performance. B4 represents conflict generation and resolution dynamics.

Loop R4 “Alignment-Performance Spiral (Reinforcing)” couples organizational alignment with performance outcomes. High alignment improves coordination efficiency, enhancing product quality and market performance, which generates resources for management and HR investment, further improving alignment. Conversely, misaligned organizations waste effort on internal friction, degrading performance and reducing resources for alignment activities.

Loop B4 “Conflict Dynamics (Balancing)” represents conflict generation and resolution. Misalignment, bug crises, and resource constraints generate organizational conflict, which HR conflict resolution mechanisms attempt to reduce. However, if conflict generation exceeds resolution capacity, conflict accumulates, increasing turnover and further degrading organizational capability.

3.5 | Strategic Decision Mechanisms

Management agents assess organizational state periodically and execute strategic interventions based on predefined rules representing managerial heuristics:

- Technical Debt Crisis Response: When technical debt exceeds 150 (75% of failure threshold), shift strategy to quality-focused mode, allocating increased engineering capacity to refactoring ($\alpha_{features\_vs\_debt} \downarrow 0.3$), prioritizing debt reduction over new features.
- Bug Crisis Response: When bug count exceeds 30 (30% of failure threshold), trigger crisis management protocol, diverting majority engineering effort to bug resolution ($\alpha_{dev\_vs\_quality} \downarrow 0.4$), allocating 60% capacity to quality work.
- Market Fit Recovery: When market fit falls below 40, increase marketing and sales effort to generate market insights and adjust product direction.
- Growth Investment: When cash runway exceeds 50 months and average engineering capacity exceeds 75, invest in talent acquisition to scale development capacity.
- Pivot Execution: When market share remains below threshold after extended period despite adequate product development, execute strategic pivot—retain team but reset product attributes with learning bonuses based on accumulated team experience.
- Emergency Layoffs: When cash runway falls below critical threshold (5 months), reduce headcount to extend survival, prioritizing retention of engineering talent.

These decision rules implement bounded rationality—management responds to observable symptoms using plausible heuristics rather than optimizing over full state space. This design choice reflects empirical observations that real managers operate under cognitive constraints and information asymmetries (Petkov et al., 2008).

3.6 | Financial Dynamics and Resource Constraints

The model incorporates explicit financial constraints through cash runway dynamics, following established approaches to modeling venture survival under resource scarcity (Zali et al., 2014):

$$\frac{dR}{dt} = \gamma_{rev} \cdot M \cdot Q_{prod} + F_{funding}(t) - (\beta_{burn} \cdot n_{total} + \beta_{overhead}) \tag{3}$$

where R represents months of runway, γ_{rev} is the revenue conversion coefficient, M is market share, Q_{prod} is

product quality (composite of market fit, features, and bug-free operation), $F_{funding}(t)$ represents discrete funding events, β_{burn} is per-employee burn rate, n_{total} is total headcount, and $\beta_{overhead}$ represents fixed operational costs.

Revenue generation depends multiplicatively on market share and product quality, with quality penalized by both bugs and technical debt:

$$Revenue = \gamma_{rev} \cdot M \cdot Q_{mktfit} \cdot \frac{1}{1 + (B/30)^2} \cdot \frac{1}{1 + (D/80)^{1.5}} \tag{4}$$

This implements the constraint that customer acquisition without product-market fit yields unsustainable growth (Lehman, 1996), and that quality problems directly reduce revenue through customer churn and sales friction. Burn rate scales linearly with headcount plus fixed overhead, consistent with empirical observations of software venture cost structures (Madachy, 2008).

The model implements three terminal failure conditions: (1) financial exhaustion when runway depletes ($R \leq 0$) without securing emergency funding; (2) technical death spiral when technical debt exceeds 200, representing codebase unmaintainability where refactoring costs exceed rewrite costs; and (3) product collapse when bug count exceeds 100, representing unusable product requiring complete rearchitecture. These latter two conditions operationalize the observation that ventures can fail from technical causes even when adequately capitalized (Franco et al., 2023).

Funding rounds occur when runway falls below 20 months, providing capital injection proportional to product quality and market traction. This implements investor behavior where funding availability depends on demonstrated progress (Zali et al., 2014). The model tracks funding rounds and pivots as strategic resources—organizations have limited opportunities to reset direction before exhausting investor patience.

3.7 | Team Evolution and Turnover Mechanisms

Agent capabilities evolve dynamically through learning and degradation mechanisms. Each period, agent capacities adjust according to:

$$\Delta C_{i,j} = \eta_{learn} \cdot L_{context} \cdot (100 - C_{i,j}) - \eta_{decay} \cdot S_{stress} \cdot C_{i,j} \tag{5}$$

where $C_{i,j}$ is capacity j of agent i, η_{learn} is the base learning rate, $L_{context}$ represents learning-conducive conditions (high organizational alignment, strong HR talent development, strategic focus on relevant function), η_{decay} is the degradation rate, and S_{stress} represents stress factors (high bug count, organizational conflict, misalignment).

Turnover occurs probabilistically each period, with base rate modulated by HR effectiveness and organizational conflict:

$$P_{turnover} = \tau_{base} \cdot (1.5 - \alpha_{HR}) \cdot (1 + 0.5 \cdot C_{org}/100) \tag{6}$$

where τ_{base} is the baseline monthly turnover probability (2%), α_{HR} is normalized HR coordination capacity, and C_{org} is organizational conflict level. Departed agents are replaced through hiring processes where new recruit capabilities depend on HR hiring quality, implementing the constraint that dysfunctional organizations struggle to attract top talent.

3.8 | Simulation Algorithm

Algorithm 1 formalizes the core simulation loop executed each monthly timestep. The algorithm integrates agent capacity evolution, role-specific actions, state updates, strategic decisions, financial dynamics, and team turnover within a unified framework that ensures proper causal ordering.

Algorithm 1 Monthly Organizational Simulation Step

Input: State S_t, Agents $\mathcal{A}$, Parameters Θ
Output: State S_{t+1}
for each agent $a \in \mathcal{A}$ **do**
    Update capacities: $\Delta C_a \leftarrow \eta_{learn} L_{context}(100 - C_a) - \eta_{decay} S_{stress} C_a$
end for
Compute aggregate outputs by role:
    Engineers: $\Delta F, \Delta D, \Delta B$ from development velocity V_{agg} with debt drag
    Sales: ΔM_{acq} from sales capacity, product quality, brand, market saturation
    Marketing: ΔA_{brand}, lead quality from marketing capacity
    HR: $\Delta A_{org,HR}$, conflict resolution from coordination capacity
    Management: $\Delta A_{org,mgmt}$ from management capacity
Update product & organizational state: $F, D, B, M, A_{brand}, A_{org}, C_{org}$
if crisis detected (debt, bugs, or market misfit) **then**
    Adjust strategy: increase $\alpha_{quality}$ or reallocate effort
end if
if pivot conditions met **then**
    Execute pivot (reset product, retain team)
end if
Compute finances: $Revenue \leftarrow \gamma_{rev} MQ_{prod}$; $Burn \leftarrow \beta_{burn}|\mathcal{A}| + \beta_{overhead}$
Update runway: $R \leftarrow R + Revenue - Burn + F_{funding}(R, Q_{prod}, M)$
for each agent a **do**
    if random() $< P_{turnover}(HR_{eff}, C_{org})$ **then**
        Replace a with new agent
    end if
end for
if $R \leq 0$ **then return** Failure
else if $t > t_{max}$ **then return** Success
end if
return S_{t+1}

The algorithm implements seven phases: (1) agent capacity evolution through learning and degradation; (2) role-specific actions producing technical, market, and organizational changes; (3) state integration with feedback dynamics; (4) strategic interventions based on crisis thresholds; (5) financial computation with revenue, burn, and funding; (6) probabilistic turnover and hiring; (7) termination evaluation. This ensures causal ordering where capabilities evolve before actions, actions precede state updates, and financial constraints operate as ultimate survival filters.

3.9 | Experimental Design

We conducted three systematic experiments to explore organizational dynamics under varying conditions (Table 3). Each experiment manipulates specific parameters or initial conditions while holding others constant, enabling isolation of causal mechanisms.

TABLE 3 Experimental scenarios and research questions

Experiment	Manipulation	Research Question
E1: Unbalanced Teams	Engineer-heavy vs. sales-heavy vs. balanced	Can technical excellence compensate for weak commercialization?
E2: Quality vs. Speed	Vary $\alpha_{quality}$ strategy	When does quality investment pay off?
E3: Initial Capital	Vary starting cash runway	Does extended runway improve outcomes or enable waste?

Each experiment runs 20 replications with different random seeds to account for stochastic variation. Simulations extend for 200 months (16+ years) or until venture termination. We collect time-series data on all state variables and analyze both trajectory patterns and terminal outcomes.

4 | RESULTS

We present results from three computational experiments designed to explore organizational dynamics under varying configurations. Each experiment runs 20 replications across parameter values to account for stochastic variation, with simulations extending 200 months or until venture termination. This section reports findings for each experiment, identifying non-linear relationships, threshold effects, and counterintuitive dynamics that emerge from feedback loop interactions. All model code and experimental configurations are available in the project repository for reproducibility: `www.github.com/andre-batista/useotst`

4.1 | Experiment 1: Unbalanced Team Configurations

Experiment 1 tests extreme team compositions spanning tech-heavy (80% engineers), sales-heavy (55% sales), marketing-heavy (44% marketing), balanced, and minimal (12 total) configurations (Table 4). Each runs 20 replications across headcounts from 12 to 27.

Results reveal dramatic bifurcation (Table 5): Tech-Heavy achieves 0% survival (failing at 14.75 months), Minimal achieves 35% survival (failing at 19.85 months), while commercial-capable configurations (Sales-Heavy, Balanced, Marketing-Heavy) achieve 100% survival.

A dual-threshold structure governs viability. First, commercial proportion threshold: Tech-Heavy (8% commercial) fails universally from revenue incapacity despite 20 engineers. Second, absolute capacity threshold: Minimal (5 commercial staff, 42% proportion) experiences 65% failure despite adequate proportion—insufficient absolute headcount triggers quality-driven collapse (debt=24.21, bugs=6.86). Survival requires both $\geq$30% commercial proportion AND $\geq$6 absolute commercial staff.

TABLE 4 Experiment 1: Unbalanced team configurations (20 replications each)

Configuration	Eng	Sales	Mkt	HR	Mgmt	Total	Strategy
Tech-Heavy	20	2	1	1	1	25	Product excellence
Sales-Heavy	5	15	5	1	1	27	Market conquest
Balanced	10	10	5	1	1	27	Equilibrium
Marketing-Heavy	8	5	12	1	1	27	Brand building
Minimal	5	3	2	1	1	12	Lean startup

TABLE 5 Experiment 1: Aggregate performance by configuration

Configuration	Survival	Time (mo)	Mkt Share	Revenue	Tech Debt	Bugs
Tech-Heavy	0%	14.75 ± 1.07	0.00	3.43 ± 0.20	21.23 ± 0.94	8.54 ± 0.49
Minimal	35%	19.85 ± 2.97	1.63 ± 2.30	29.04 ± 34.47	24.21 ± 2.62	6.86 ± 5.29
Marketing-Heavy	100%	—	7.52 ± 0.88	120.30 ± 18.11	21.01 ± 0.92	0.00
Balanced	100%	—	12.80 ± 1.10	234.20 ± 26.10	20.86 ± 0.57	0.00
Sales-Heavy	100%	—	17.70 ± 1.58	341.41 ± 37.96	21.22 ± 0.86	0.00

Among survivors, Sales-Heavy dominates with 17.70% market share and $341.41 revenue—46% higher than Balanced despite identical headcount (Figure 2). Marketing-Heavy underperforms substantially (7.52% share, $120.30 revenue), revealing that conversion capacity dominates demand generation.

Quality metrics bifurcate sharply (Figure 3). Survivors converge to uniform quality (features 116.9-118.0, debt ≈21.0, bugs 0.0), eliminating technical differentiation—commercial capacity determines outcomes. Failed configurations degrade severely: Tech-Heavy achieves higher code quality (70.6) than survivors (60.0) yet fails catastrophically, demonstrating that local code cleanliness proves irrelevant without commercial viability.

Time-series analysis (Figure 4) reveals distinct failure mechanisms. Tech-Heavy experiences financial collapse at 14.75 months from pure commercial incapacity (2 salespeople insufficient for revenue generation). Minimal demonstrates delayed quality-driven collapse at 19.85 months: 5 engineers prove insufficient under resource pressure, triggering debt accumulation (Loop B1) and bug cascades (Loop R2) that accelerate financial depletion—the first clear quality-velocity tradeoff engagement observed.

Strategic implications establish minimum viable capacity: ≥8 engineers, ≥6 commercial staff, ≥15 total headcount. Below these thresholds, organizations enter stochastic failure zones. Sales capacity provides greater leverage than marketing (135% market share advantage, 184% revenue advantage). Among survivors, engineering headcount variation produces indistinguishable quality outcomes, suggesting engineering sufficiency threshold beyond which additional capacity should redirect to commercial functions.

The Minimal configuration (12 employees) exhibits significant efficiency, achieving 64% of Sales-Heavy's market share on a per-employee basis. With total revenue of $97.40 across 12 employees, Minimal generates $8.12 per employee—higher than Marketing-Heavy ($5.65) and approaching Balanced ($11.57). Figure 5 quantifies this efficiency advantage.

Figure 6 demonstrates that organizational alignment and conflict exhibit minimal differentiation across configurations. Commercial-capable survivors maintain 100.0 alignment and zero conflict. Tech-Heavy maintains 97.84% alignment (only 2.16% degradation) and zero conflict before failure. Minimal averages 99.55% alignment (0.45%

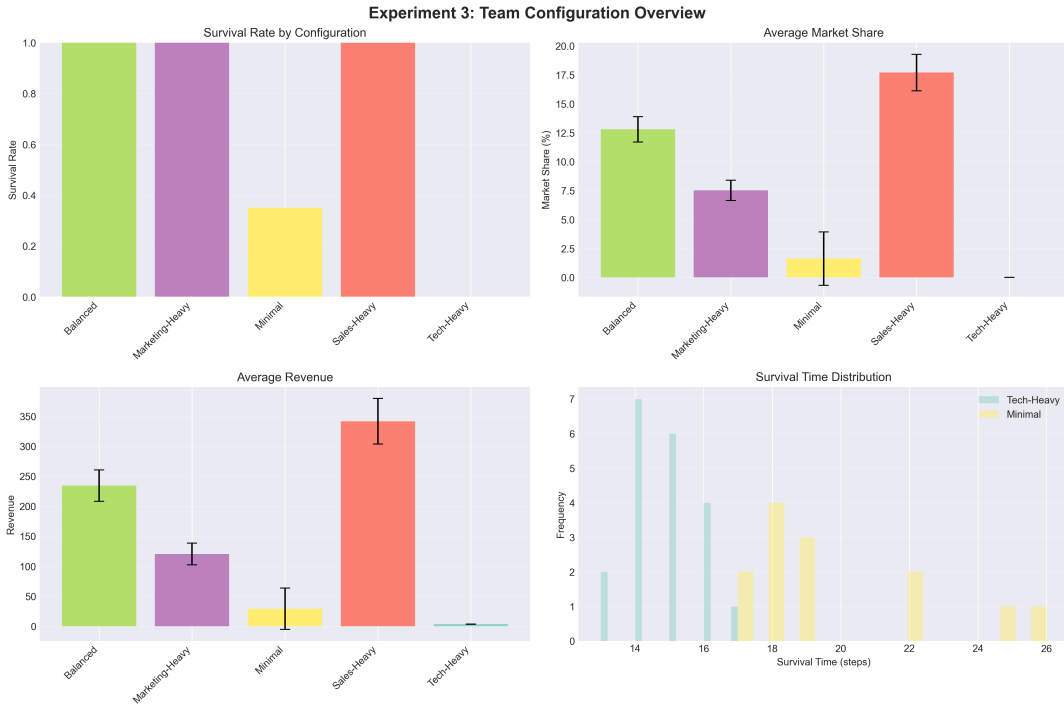

FIGURE 2 Experiment 1: Team configuration performance overview. Top-left: Survival bifurcates dramatically. Top-right: Market share varies 10.9× (0-17.70%), with Sales-Heavy leading survivors. Bottom-left: Revenue exhibits 11.8× variation, stratified by commercial capacity. Bottom-right: Survival time distributions show Tech-Heavy modal failure at 14-15 months.

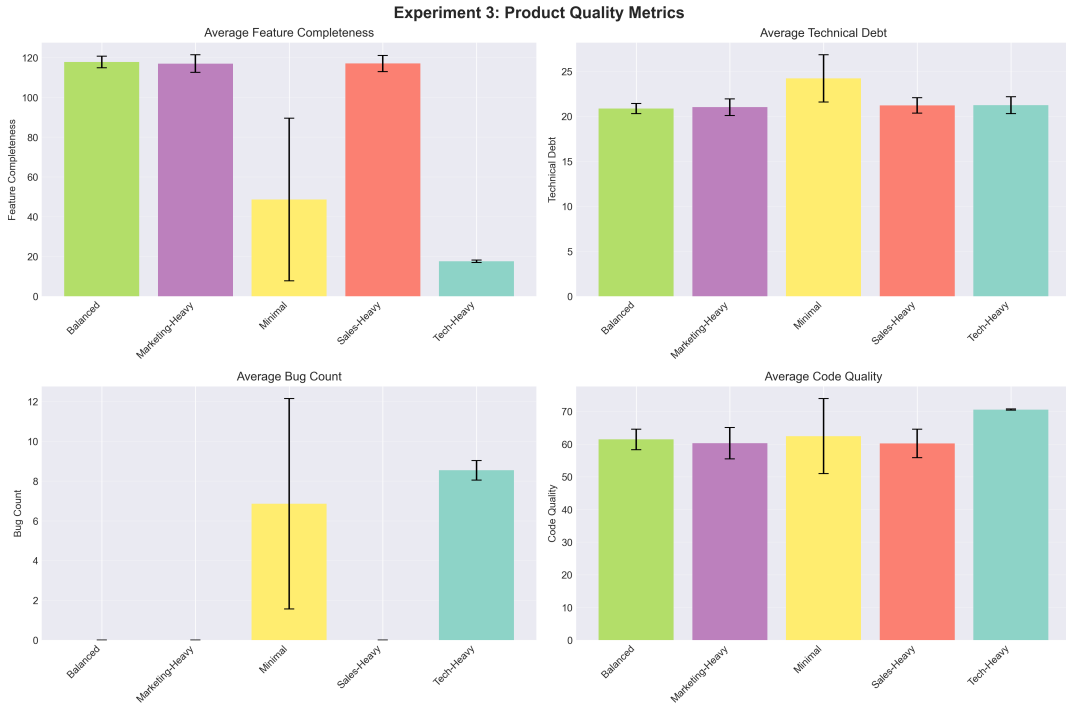

FIGURE 3 Experiment 1: Product quality metrics reveal failure-driven degradation. Top-left: Features stratify. Top-right: Debt bifurcates. Bottom-left: Bugs separate failures from survivors. Bottom-right: Code quality clusters.

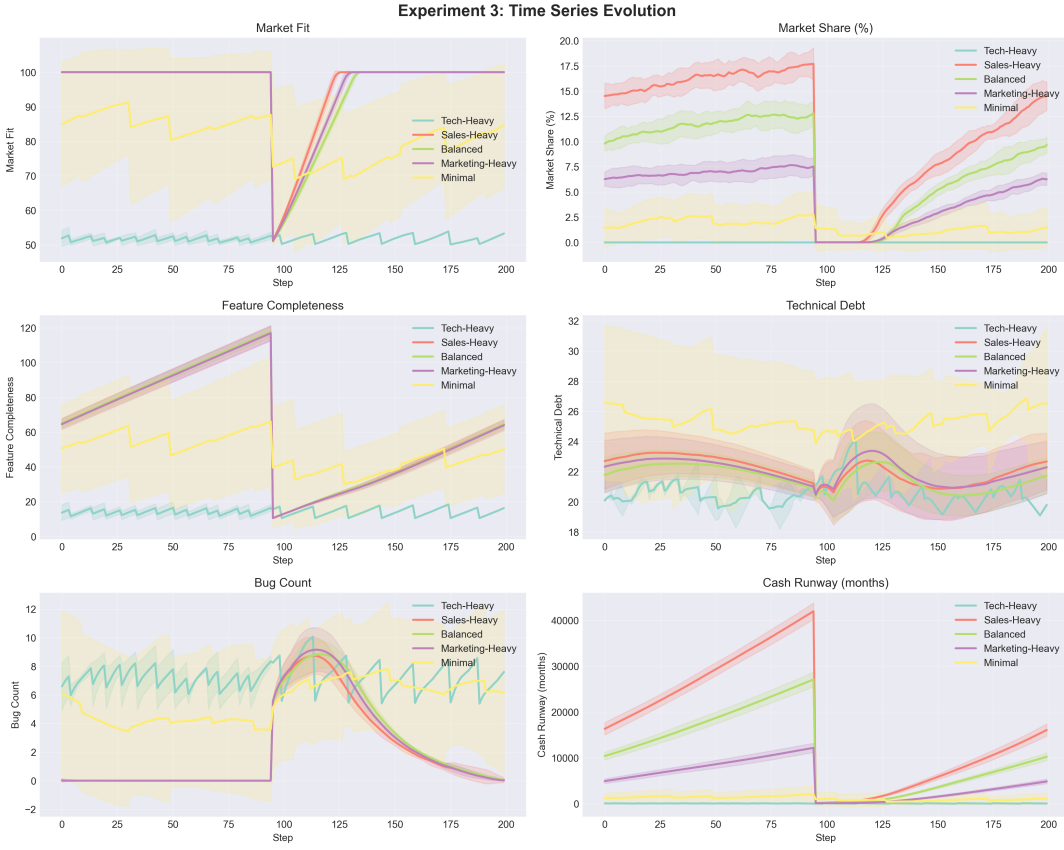

FIGURE 4 Experiment 1: Time-series evolution reveals dual failure modes. Top-left: Market fit bifurcates sharply. Top-right: Market share shows exponential divergence—Sales. Middle-left: Feature completeness stratifies dramatically. Middle-right: Technical debt exhibits transient spikes for survivors. Bottom-left: Bug count spikes catastrophically for failures. Bottom-right: Cash runway shows exponential accumulation for survivors.

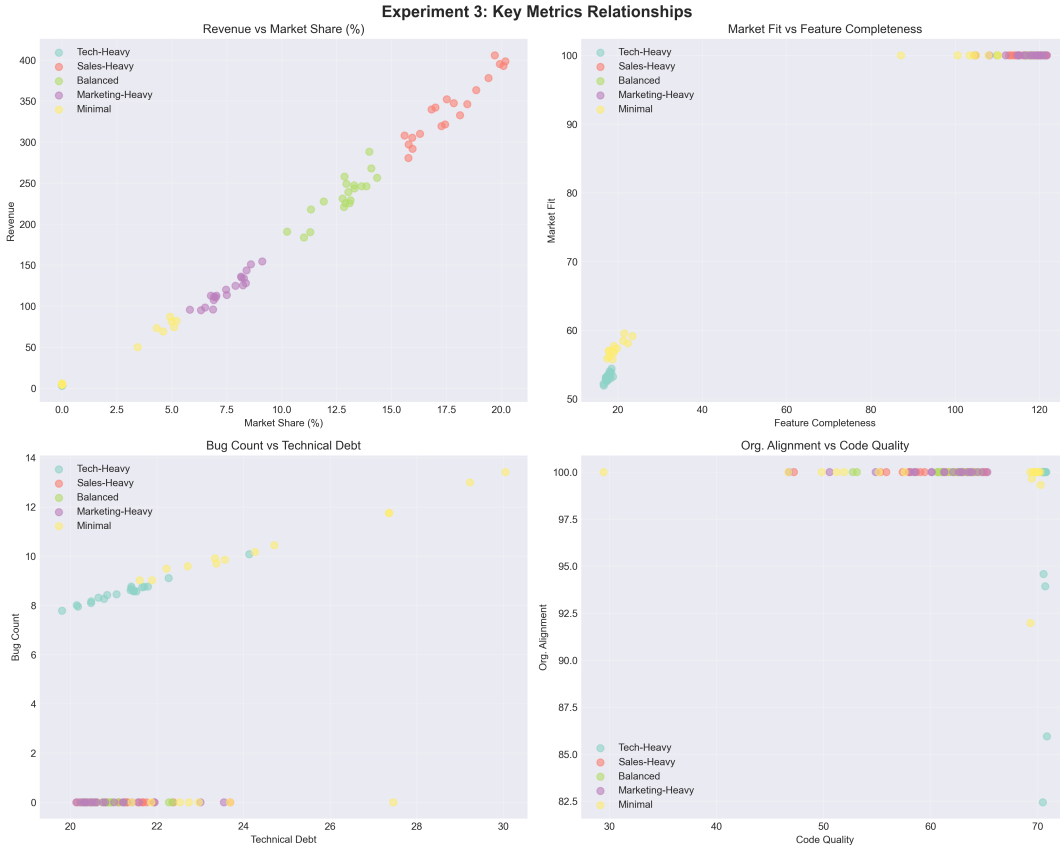

FIGURE 5 Experiment 1: Comparative metrics across configurations. Top-left: Revenue vs. market share. Top-right: Market fit reaches ceiling (100%) for all survivors. Bottom-left: Technical debt and bugs remain zero for all survivors. Bottom-right: Organizational alignment.

degradation) and zero conflict despite 65% failure rate.

This uniformly high organizational health despite catastrophic failures confirms that financial constraints (insufficient revenue) and quality degradation (bugs, debt accumulation) dominate survival outcomes under current model parameterization. Organizational health variables (alignment, conflict, morale) operate as moderators rather than direct failure drivers—teams remain aligned and conflict-free while failing financially or quality-wise.

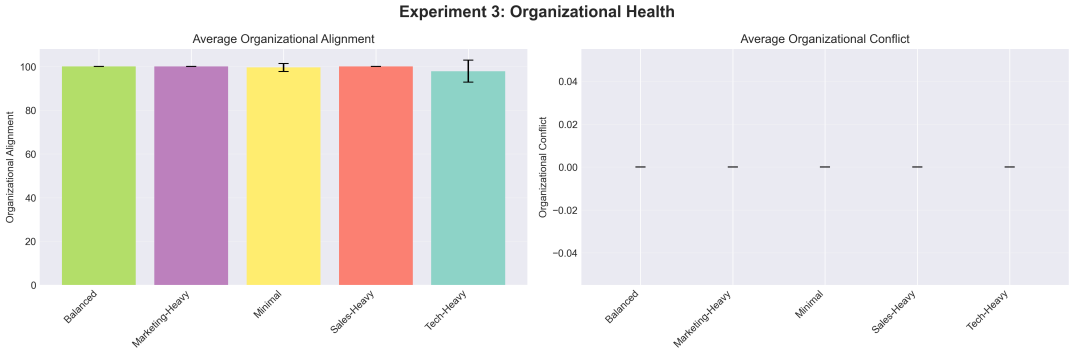

FIGURE 6 Experiment 3: Organizational health metrics show minimal differentiation. Left: Organizational alignment remains uniformly high—survivors 100.0, Minimal 99.55, Tech-Heavy 97.84—indicating organizational dysfunction does not explain failures. Right: Organizational conflict remains precisely zero for all configurations throughout simulations, confirming conflict plays no role in observed failures.

4.2 | Experiment 2: Quality vs. Speed Development Strategies

Experiment 2 tests quality-velocity tradeoffs (Lehman, 1996; Franco et al., 2023) by comparing five development strategies (Table 6): Technical Debt Spiral (fastest, 0.025 features/step, 0.012 debt generation), Move Fast Break Things, Balanced Pragmatic, Quality First, and Extreme Quality (slowest, 0.005 features/step, 0.001 debt generation). All configurations maintain identical teams (10 engineers, 5 sales, 3 marketing, 1 HR, 1 management) and capital (50 months), isolating strategy effects.

TABLE 6 Experiment 2: Development strategy parameterizations (20 replications each)

Strategy	dev_to_features	dev_to_debt	dev_to_bugs	Philosophy
Technical Debt Spiral	0.025	0.012	0.006	Extreme speed, unsustainable
Move Fast Break Things	0.020	0.008	0.004	High speed, moderate discipline
Balanced Pragmatic	0.012	0.005	0.002	Pragmatic balance
Quality First	0.008	0.002	0.001	Quality prioritized
Extreme Quality	0.005	0.001	0.0005	Maximum quality, minimum speed

Results reveal survival bifurcation: Technical Debt Spiral achieves 0% survival (collapsing at 27.75 ± 5.82 months with debt=68.14, bugs=37.42), while four other strategies achieve 100% survival (Tables 7, 8). This demonstrates Loops B1 (Technical Debt Brake) and R2 (Bug Accumulation) engaging destructively when debt generation (0.012)

exceeds refactoring capacity.

Quality First outperforms Move Fast Break Things despite 2.4× slower velocity: 139.89 vs 57.91 features (2.4×), $142.20 vs $41.07 revenue (246% higher), sustainability 0.998 vs 0.269 (3.7×). Move Fast accumulates debt=110.20 imposing 71% revenue penalty via Loop B1 velocity drag, while Quality First maintains zero debt enabling sustained productivity (Figures 7, 8, 9).

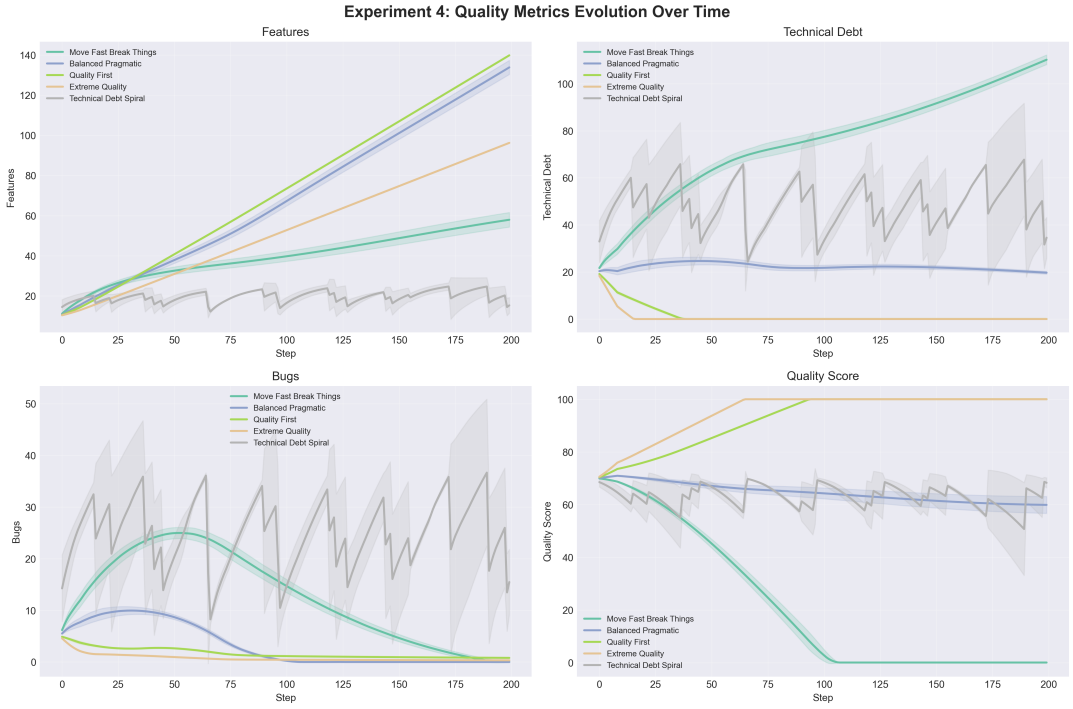

FIGURE 7 Experiment 2: Quality metrics evolution demonstrates functional quality degradation mechanisms. Top-left: Features stratify by velocity. Top-right: Technical debt. Bottom-left: Bug count. Bottom-right: Code quality.

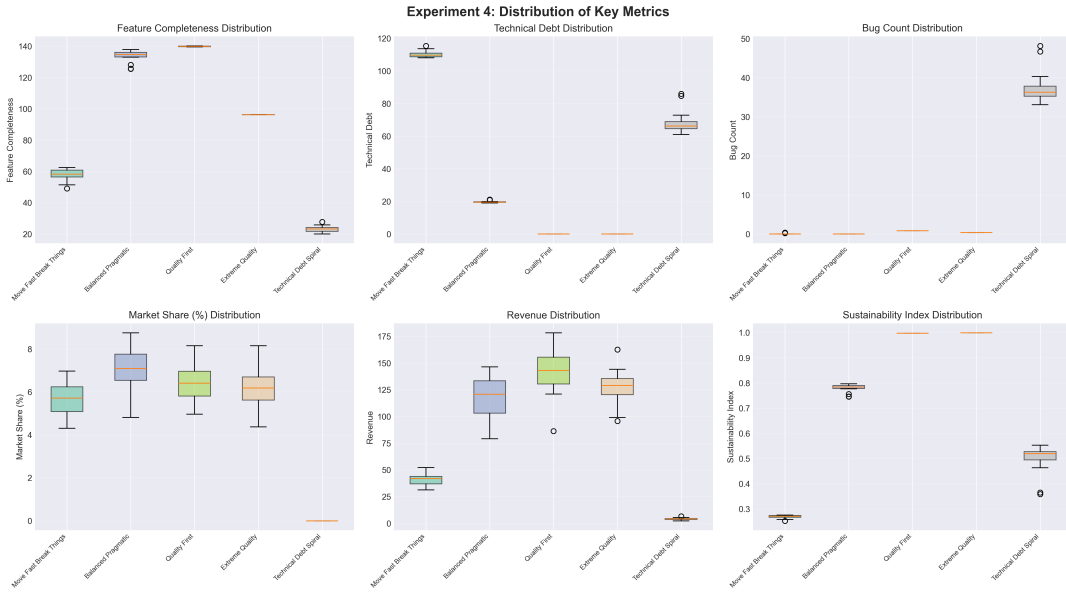

FIGURE 8 Experiment 2: Performance distributions reveal quality-velocity tradeoff emergence. Top-left: Feature completeness. Top-center: Technical debt distributions. Top-right: Bug count. Bottom-left: Market share correlates with sustainability. Bottom-center: Revenue follows quality maintenance. Bottom-right: Sustainability index.

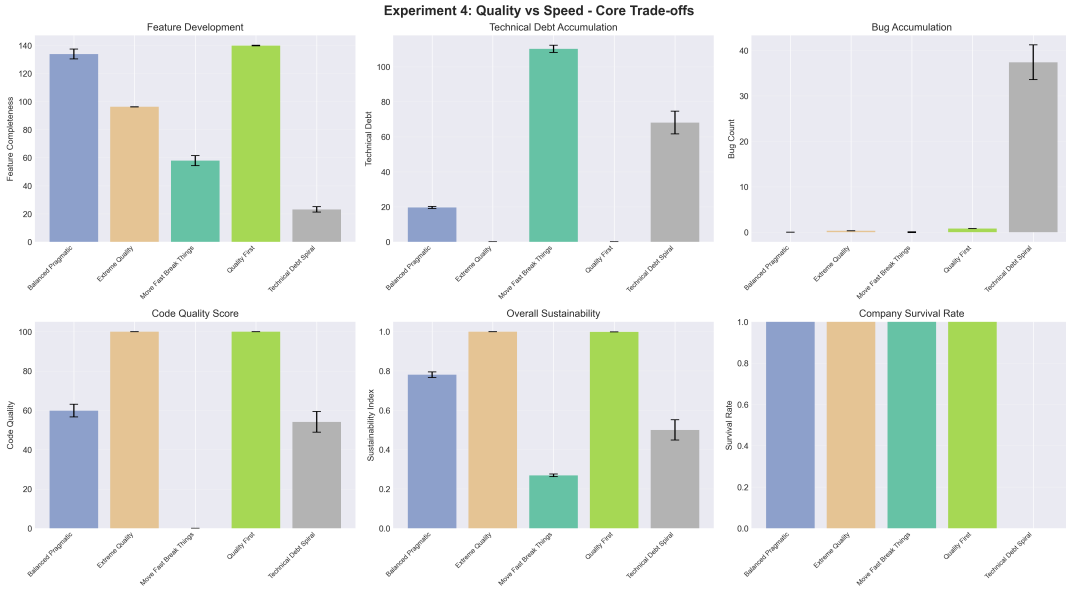

FIGURE 9 Experiment 2: Core tradeoff analysis demonstrates functional quality feedback loops. Top-left: Features stratify (quality 140, Move Fast 58, Debt Spiral 23), illustrating Loop B1. Top-center: Debt accumulates (Debt Spiral 68.14, Move Fast 110.20, Balanced 19.70, quality 0). Top-right: Bugs cascade for Debt Spiral (37.42), controlled for survivors. Bottom-left: Code quality bifurcates (quality 100, Balanced 60, Move Fast 0, Debt Spiral 54). Bottom-center: Sustainability stratifies (0.999-0.998 vs 0.780 vs 0.269 vs 0.500). Bottom-right: Survival bifurcates (100% sustainable vs 0% Debt Spiral).

TABLE 7 Experiment 2: Performance by development strategy (survivors only, mean ± std)

Strategy	Survival	Velocity	Features	Debt	Bugs	Market Share	Revenue
Quality First	100%	0.70 ± 0.001	139.89 ± 0.20	0.00 ± 0.00	0.80 ± 0.001	6.44% ± 0.90	$142.20 ± 21.12
Balanced Pragmatic	100%	0.67 ± 0.018	133.88 ± 3.50	19.70 ± 0.60	0.00 ± 0.00	7.16% ± 0.96	$118.80 ± 18.44
Extreme Quality	100%	0.48 ± 0.0002	96.24 ± 0.04	0.00 ± 0.00	0.34 ± 0.0001	6.21% ± 0.86	$126.69 ± 16.32
Move Fast Break Things	100%	0.29 ± 0.018	57.91 ± 3.63	110.20 ± 2.03	0.03 ± 0.09	5.66% ± 0.80	$41.07 ± 4.90
Technical Debt Spiral	0%	0.00	23.16 ± 1.90	68.14 ± 6.49	37.42 ± 3.83	0.00%	$4.00 ± 0.98

TABLE 8 Experiment 2: Quality metrics stratification (survivors at month 200, failures at collapse)

Strategy	Technical Debt	Bug Count	Code Quality	Sustainability
Quality First	0.00 ± 0.00	0.80 ± 0.001	100.00 ± 0.00	0.998 ± 0.000002
Extreme Quality	0.00 ± 0.00	0.34 ± 0.0001	100.00 ± 0.00	0.999 ± 0.0000002
Balanced Pragmatic	19.70 ± 0.60	0.00 ± 0.00	59.88 ± 3.20	0.780 ± 0.015
Move Fast Break Things	110.20 ± 2.03	0.03 ± 0.09	0.00 ± 0.00	0.269 ± 0.006
Failed Configurations (at collapse time)				
Technical Debt Spiral	68.14 ± 6.49	37.42 ± 3.83	54.12 ± 5.27	0.500 ± 0.052

Three mechanisms explain quality feedback activation. First, exponential debt drag $\delta_{debt}(D) = \exp(-0.05 \cdot D)$ imposes 96-99.6% velocity penalties at debt=68-110. Second, bug cascade thresholds trigger at bugs>20, generating secondary defects. Third, diminishing refactoring returns $(0.1/(1 + D/20))$ prevent debt recovery when generation exceeds capacity.

Finally, the strategic implications are: sustained velocity beats initial velocity (Quality First delivers 2.4× more features over 200 months), quality investment exhibits increasing returns (zero-debt strategies maintain constant productivity while debt strategies decay exponentially), moderate debt (19.70) proves sustainable while extreme debt (68.14-110.20) constrains or collapses performance, and revenue correlates with sustainability not velocity (Figure 10).

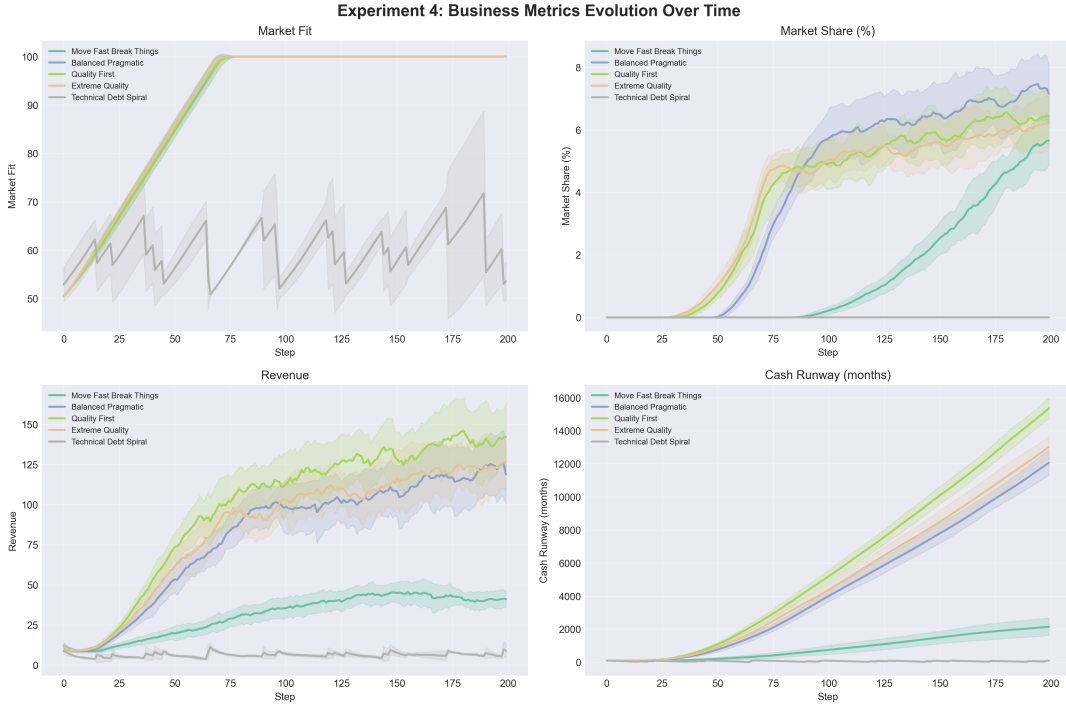

FIGURE 10 Experiment 2: Business metrics evolution shows quality-driven bifurcation. Top-left: Market fit diverges (quality 100 by month 75, Debt Spiral collapses month 28). Top-right: Market share stratifies (Quality/Balanced 6-7%, Move Fast 5-6%, Debt Spiral 0%). Bottom-left: Revenue bifurcates (quality $120-145, Move Fast $40, Debt Spiral $4). Bottom-right: Cash runway grows for survivors (12,000-15,000 months), collapses for Debt Spiral (-2,840 months).

4.3 | Experiment 3: Initial Capital and Financial Constraints

Experiment 3 tests seven capital levels from 30 months (bootstrapped) through 300 months (well-funded) across 140 simulation runs with identical teams (10 engineers, 5 sales, 3 marketing, 1 HR, 1 management), isolating financial constraint effects on survival.

Results reveal a sharp survival threshold at 200 months: configurations below experience 100% mortality (120/120

failures), while Series B (200 months) achieves 55% survival and Well Funded (300 months) achieves 100% (Table 9). Increasing capital from 150 to 200 months shifts survival from 0% to 55%, demonstrating financial constraints as hard survival determinants.

TABLE 9 Experiment 3: Survival outcomes by initial capital level

Capital Level	Initial Runway	Survival Rate	Mean Survival Time	Survivors
Below Survival Threshold (100% Mortality)				
Bootstrapped	30 months	0%	3.7 ± 0.57 months	0/20
Seed Stage	50 months	0%	5.9 ± 0.72 months	0/20
Series A Low	75 months	0%	8.1 ± 0.91 months	0/20
Series A Standard	100 months	0%	10.2 ± 1.11 months	0/20
Series A High	150 months	0%	16.05 ± 1.85 months	0/20
At or Above Survival Threshold				
Series B	200 months	55%	120.2 ± 90.54 months	11/20
Well Funded	300 months	100%	200 months (all)	20/20

Table **??** presents performance metrics across capital levels, revealing uniform commercial outcomes among organizations that achieved PMF before capital depletion. Market share varies only 8% across capital levels (6.87-7.44%), revenue varies 9% ($111-$121), and technical debt remains uniformly at 21.32 ± 0.58 (coefficient of variation 2.7%). This uniformity demonstrates that capital abundance does not improve product-market performance, feature development velocity, or quality maintenance. Organizations that survive long enough to achieve PMF converge to identical commercial outcomes regardless of initial funding.

TABLE 10 Experiment 3: Performance metrics by capital level (at failure or month 200)

Capital Level	Market Share	Revenue	Features	Debt	Capital Eff.
Bootstrapped	7.44% ± 1.11	$121.37 ± 20.97	112.86 ± 4.37	21.45 ± 1.00	0.248 ± 0.037
Seed Stage	7.04% ± 1.09	$120.07 ± 17.21	113.79 ± 1.94	21.15 ± 0.39	0.141 ± 0.022
Series A Low	6.92% ± 1.20	$116.17 ± 18.85	111.96 ± 4.14	21.61 ± 1.05	0.092 ± 0.016
Series A Standard	6.91% ± 0.91	$119.22 ± 17.79	113.08 ± 1.63	21.27 ± 0.30	0.069 ± 0.009
Series A High	6.91% ± 1.23	$111.28 ± 20.89	112.17 ± 4.49	21.53 ± 1.06	0.046 ± 0.008
Series B	7.00% ± 1.01	$116.51 ± 17.78	113.61 ± 1.84	21.17 ± 0.37	0.015 ± 0.017
Well Funded	6.87% ± 0.90	$116.84 ± 17.01	113.56 ± 2.85	21.23 ± 0.58	0.000 ± 0.000

Critically, all 140 runs achieved product-market fit within 15.29 ± 1.36 months regardless of capital level (100% PMF rate). Performance metrics prove capital-insensitive (Table 10): market share varies 8% (6.87-7.44%), revenue varies 9% ($111-$121), technical debt remains uniform (21.32 ± 0.58), features identical (112-114). Organizations surviving to PMF converge to identical outcomes regardless of initial funding (Figures 11, 12).

Capital efficiency degrades exponentially with funding abundance: Bootstrapped achieves 0.248 market share per capital month, Well Funded achieves 0.000—representing 100% efficiency loss with 10× capital increase (Figure 13). Additional capital merely extends runway without accelerating PMF, improving quality, or enhancing commercial out-

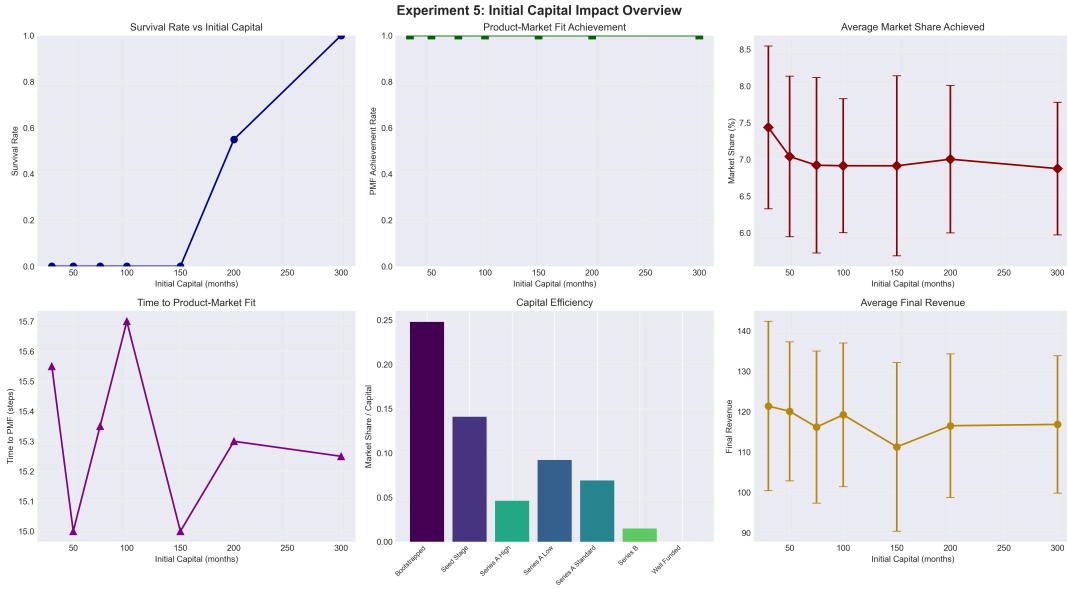

FIGURE 11 Experiment 3: Initial capital impact overview. Top-left: Survival rate with sharp threshold at 200 months. Top-center: PMF achievement uniformly 100% across all levels. Top-right: Market share uniform at 6.9-7.4%, independent of capital. Bottom-left: Time to PMF consistent at 15.29 ± 1.36 months. Bottom-center: Capital efficiency decreases exponentially . Bottom-right: Revenue uniform at $111-121, independent of funding.

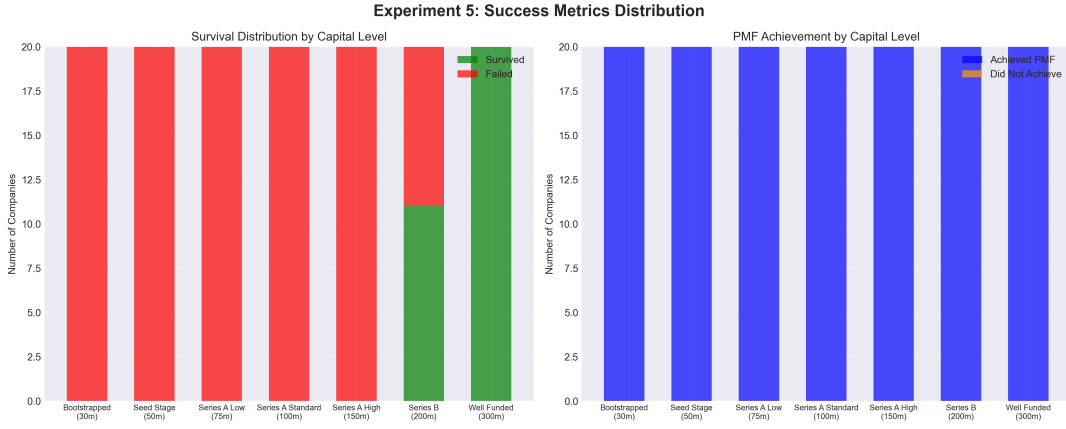

FIGURE 12 Experiment 3: Metric distributions by capital level. Top-left: Market share uniform at 6-8%. Top-center: Revenue overlap ($100-145). Top-right: Features consistent (111-114). Bottom-left: Debt uniform (21.3 ± 0.6). Bottom-center: Efficiency decreases exponentially ($0.248 \rightarrow 0.0$). Bottom-right: Survival time linear with capital (3.7-16.05 months).

comes.

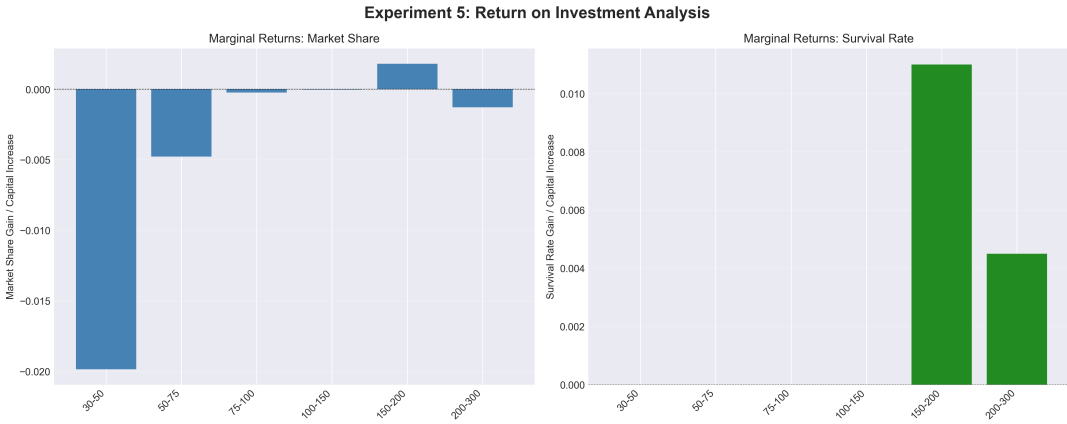

FIGURE 13 Experiment 3: Return on investment analysis. Left: Market share gain decreases (0.248→0.141→0.069→0.000), marginal returns negative above 150 months. Right: Survival step function (0% at 30-150 months, 55% at 200 months, 100% at 300 months).

Failure mechanism analysis reveals profitability gap dynamics: all organizations achieve PMF at month 15 but require months 18-22 to reach cash-flow positivity, creating a 5-7 month profitability gap. Undercapitalized organizations (<200 months) exhaust cash during this gap despite achieving PMF and generating revenue, collapsing at predictable intervals (Bootstrapped: 3.7 months, Series A High: 16.05 months) with linear correlation to initial capital ($r = 0.997$). The 200-month threshold represents minimum capital to bridge PMF achievement (15 months) + profitability gap (5-7 months) + safety buffer (Figure 14).

Strategic implications: capital operates as binary survival gate (necessary but not sufficient), PMF achievement proves capital-independent (100% success regardless of funding), performance remains uniform across 10× capital range (contradicting capital-performance assumptions), and optimal capitalization targets minimum viable runway sufficient to bridge profitability gaps rather than maximum availability, supporting lean startup capital discipline over growth-at-all-costs strategies. Note that the specific 200-month threshold reflects model parameterization limitations (Section 5) and should be interpreted qualitatively rather than prescriptively.

5 | CONCLUSION

This research developed an agent-based model of software engineering organizations as complex adaptive systems and conducted theoretical exploration through 240 simulation runs across three experiments. Four primary contributions emerge: (1) identification of critical resource thresholds functioning as binary survival gates—capital sufficient to bridge profitability gaps, and dual commercial capacity thresholds (≥6 staff at ≥30% composition)—below which organizations experience systematic failure; (2) demonstration of selective quality degradation, where extreme velocity strategies (Technical Debt Spiral) trigger catastrophic collapse while moderate approaches avoid debt spirals; (3) demonstration that resource thresholds dominate strategic optimization—capital abundance yields no performance improvement among survivors while inadequacy guarantees failure; and (4) methodological demonstration revealing emergent phenomena potentially obscured in empirical datasets by survivorship bias. Importantly, reported numeric

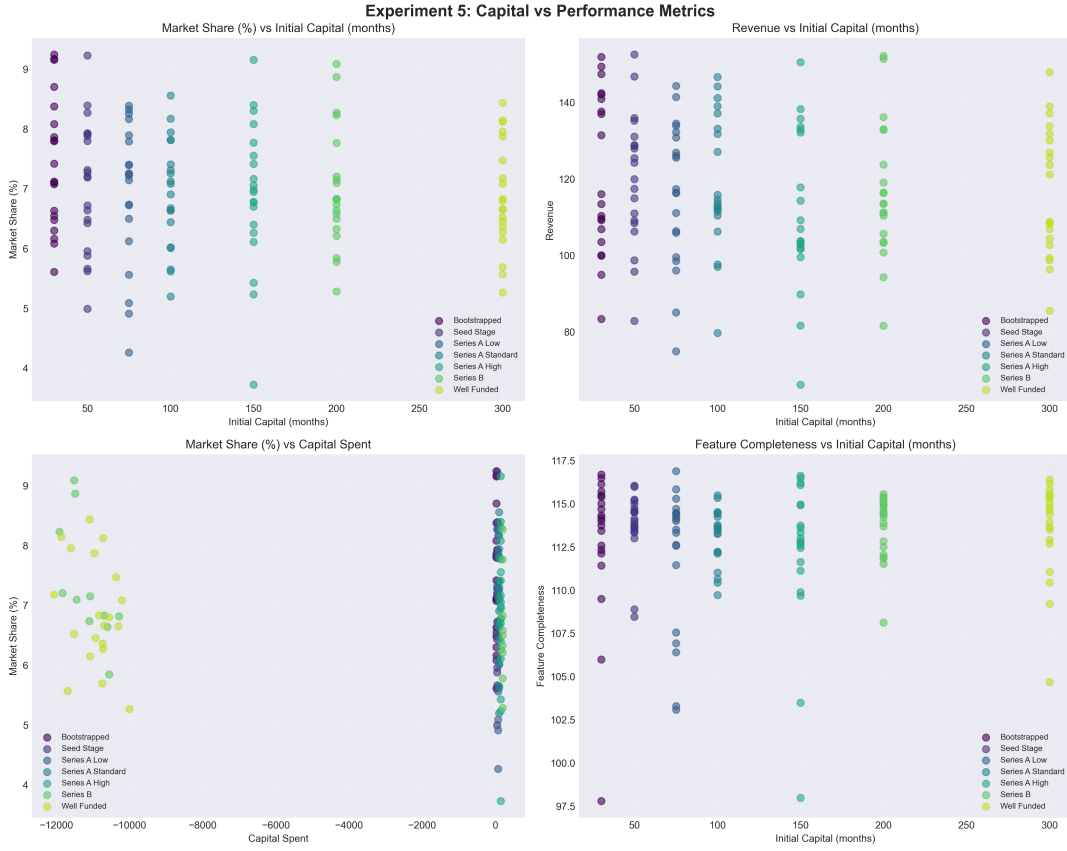

FIGURE 14 Experiment 3: Capital versus performance metrics. Top-left: Market share independent of capital. Top-right: Revenue independent of capital. Bottom-left: Market share independent of spending. Bottom-right: Features independent of capital.

cutoffs (e.g., 200 months, $\geq$6, $\geq$30%) are model-dependent and should be read as qualitative indicators of threshold structure, not literal prescriptions.

For practitioners, resource adequacy precedes strategic optimization. In the model, surviving organizations require runway sufficient to bridge the profitability gap between PMF achievement and cash-flow positivity (noting that the baseline 200-month value reflects conservative revenue parameters requiring empirical calibration), dual commercial capacity thresholds ($\geq$6 staff at $\geq$30% headcount), and moderate quality strategies avoiding Technical Debt Spiral extremes. Quality-focused approaches paradoxically deliver superior long-term velocity through sustained productivity. Growth-stage companies should recognize exponential capital efficiency degradation with funding abundance and structure balanced teams. Investors must evaluate portfolio companies against resource adequacy for their specific market dynamics and assess quality management practices identifying catastrophic failure risks.

Theoretically, this research advances understanding of software organizations as complex adaptive systems where outcomes emerge from feedback loop structures. Selective activation of quality degradation mechanisms illustrates systems thinking principles—organizations maintaining moderate quality parameters operate in stable equilibria, while extreme strategies cross threshold boundaries entering unstable regimes where reinforcing feedback loops accelerate collapse. This explains organizational trajectory heterogeneity and reconciles conflicting literature by specifying boundary conditions under which quality-speed tradeoffs engage destructively.

Limitations warrant acknowledgment. Model simplifications and parameter calibration from literature estimates introduce uncertainty—results represent theoretical explorations identifying plausible mechanisms rather than predictive forecasts. Several outcomes exhibit near-perfect convergence and low variance (organizational alignment $\approx$ 100%, conflict $\approx$ 0, PMF 100% in Experiment 3) reflecting baseline parameterization: strong stabilizing feedbacks, weak conflict/turnover processes, bounded capability ranges, and limited exogenous shocks. Consequently, organizational variables remain in non-binding ranges, and survival is dominated by financial and commercial gates (runway and sales capacity) rather than organizational dysfunction. Exploring regimes with weaker HR/leadership or higher dysfunction is necessary to evaluate when organizational breakdown becomes the dominant failure mode.

The 200-month capital threshold (Experiment 3) is unrealistic for venture-backed startups, reflecting conservative revenue scaling that underestimates post-PMF growth. The threshold indicates the qualitative pattern that inadequate runway causes failure during profitability gaps, not a quantitative prescription—recalibration would likely yield 24-36 months. Additionally, uniform talent distributions [30, 100] abstract effects where exceptional individuals ("10x engineers") might alter survival thresholds through accelerated learning (Loop R4), debt reduction (Loop B1), or market validation (Loop R1); testing heavy-tailed distributions (Pareto/lognormal) to evaluate "superstar" effects is a natural extension. Future work should validate revenue parameters, explore talent distribution effects (uniform, bimodal, heavy-tailed), and incorporate competitive dynamics and organizational learning. The central lesson remains: with inadequate resources, no strategy prevents failure; with adequate resources, diverse strategies may succeed.

Conflict of interest

The authors declare that they have no conflicts of interest to report regarding the publication of this paper.

6 | STATE EVOLUTION EQUATIONS

This appendix presents the formal mathematical specifications for state variable evolution. All equations are implemented in discrete-time simulation with monthly timesteps.

6.1 | Technical Subsystem Dynamics

Feature Completeness:

$$\frac{dF}{dt} = \lambda_{feat} \sum_i C_{dev,i} \cdot (1 - \delta_{debt}(D)) - \mu_F \cdot F \tag{7}$$

where the debt drag function implements saturation dynamics:

$$\delta_{debt}(D) = \frac{D}{D + 200} \tag{8}$$

and $\mu_F \approx 0$ (feature deprecation negligible).

Quality Focus Factor: Engineering teams exhibit heterogeneous quality consciousness. To prevent the pathological case where $\overline{C_{exp}} \approx \overline{C_{dev}}$ (leading to $\tau \approx 1$ and $(1 - \tau) \approx 0$, which artificially eliminates debt/bug generation), we normalize quality focus as:

$$\tau = \min\left(\frac{\overline{C_{exp}}}{100}, 0.8\right) \tag{9}$$

This ensures $(1 - \tau) \in [0.2, 1.0]$, creating debt accumulation dynamics even when teams have high explanation capacity.

Technical Debt Evolution:

$$\frac{dD}{dt} = \lambda_{debt} \cdot V_{agg} \cdot (1 - \tau) - \rho_{refactor} \cdot W_{refactor} \tag{10}$$

where V_{agg} is aggregate development velocity, $W_{refactor}$ is refactoring work allocation, and $\rho_{refactor} = 0.04$ represents refactoring effectiveness.

Exponential Velocity Penalty: Technical debt imposes a non-linear drag on development velocity:

$$\delta_{debt}(D) = \frac{1}{1 + (D/50)^2} \tag{11}$$

This exponential penalty ensures that accumulated debt ($D > 100$) drastically reduces productivity, creating the feedback dynamics of Loop R1. When $D > 200$, the system enters technical bankruptcy (game over condition).

Code Quality Evolution:

$$\frac{dQ_c}{dt} = \gamma_{refactor} \cdot W_{refactor} - \gamma_{decay} \cdot D \tag{12}$$

with $\gamma_{refactor} = 0.01$ and $\gamma_{decay} = 0.01$, implementing bidirectional coupling between quality and debt.

Bug Count Evolution:

$$\frac{dB}{dt} = \lambda_{bugs} \cdot V_{agg} \cdot \kappa(Q_c) \cdot (1 - \tau) - \rho_{fix} \cdot W_{bugfix} \tag{13}$$

where the quality modulation function is:

$$\kappa(Q_c) = \frac{100 - Q_c}{100} \tag{14}$$

and ρ_{fix} represents bug fixing efficiency.

Bug Velocity Penalty: Bugs impose exponential drag on development velocity:

$$\delta_{bugs}(B) = \frac{1}{1 + (B/20)^{1.5}} \tag{15}$$

Beyond critical thresholds ($B > 100$), the system becomes unmaintainable, triggering game over.

6.2 | Resource Allocation Strategy

Engineering capacity is allocated according to strategic parameters α_{dev} and $\alpha_{features}$:

$$W_{dev} = C_{available} \cdot \alpha_{dev} \tag{16}$$

$$W_{quality} = C_{available} \cdot (1 - \alpha_{dev}) \tag{17}$$

$$W_{features} = W_{dev} \cdot \alpha_{features} \tag{18}$$

$$W_{refactor} = W_{dev} \cdot (1 - \alpha_{features}) \tag{19}$$

$$W_{bugfix} = W_{quality} \tag{20}$$

where $C_{available}$ is total available engineering capacity.

6.3 | Market Subsystem Dynamics

Product Quality Composite:

$$Q_{prod} = \frac{F_m}{100} \cdot \frac{F}{F + 50} \cdot \left(1 - \frac{B}{B + 50}\right) \tag{21}$$

combining market fit, feature completeness (with saturation), and bug-free operation.

Lead Quality:

$$L_q = \text{clip}\left(\frac{\overline{C_{pub}} + \overline{C_{ideas}}}{2} \cdot Q_{prod}, 0, 100\right) \tag{22}$$

where $\overline{C_{pub}}$ and $\overline{C_{ideas}}$ are mean marketing capacities.

Brand Awareness Growth:

$$\frac{dA_{brand}}{dt} = \omega_{mkt} \cdot M_{eff} \cdot \left(1 - \frac{A_{brand}}{100}\right) - \mu_{brand} \cdot A_{brand} \tag{23}$$

where M_{eff} is effective marketing effort and decay μ_{brand} represents brand erosion.

Market Share Dynamics:

$$\frac{dM}{dt} = \lambda_{acq} \cdot S_{cap} \cdot Q_{prod} \cdot A_{brand} \cdot (1 - M/100)^2 - \chi_{total} \cdot M \tag{24}$$

where total churn is:

$$\chi_{total} = \chi_{base} + \chi_{bugs} \cdot \frac{B}{B + 50} + \chi_{misfit} \cdot \left(1 - \frac{F_m}{100}\right) \tag{25}$$

implementing multi-factor attrition.

6.4 | Organizational Subsystem Dynamics

Organizational Alignment:

$$\frac{dA_{org}}{dt} = \sum_k C_{mgmt,k} + \sum_h C_{coord,h} - \xi_{decay} \cdot A_{org} - \xi_{conflict} \cdot C_{org} \tag{26}$$

with half-life approximately 15 months without active management ($\xi_{decay} \approx 0.047$).

Conflict Generation:

$$\frac{dC_{org}}{dt} = \gamma_{misalign} \cdot \left(1 - \frac{A_{org}}{100}\right) + \gamma_{pressure} \cdot \frac{B}{B + 50} - \rho_{HR} \cdot \sum_h C_{resolve,h} \tag{27}$$

where $\gamma_{misalign} = 0.3$ and $\gamma_{pressure} = 0.2$ represent conflict generation rates, and ρ_{HR} is HR resolution effectiveness.

6.5 | Financial Dynamics

Revenue Generation:

$$R_{revenue} = \gamma_{rev} \cdot Q_{prod} \cdot S_{power} \cdot M \tag{28}$$

where sales power is:

$$S_{power} = \overline{C_{sell}} \cdot n_{sales} \tag{29}$$

with $\overline{C_{sell}}$ mean selling capacity and n_{sales} sales team size.

Cash Runway Evolution:

$$\frac{dR_{cash}}{dt} = R_{revenue} + F_{funding}(t) - (\beta_{burn} \cdot n_{total} + \beta_{overhead}) \tag{30}$$

where runway depletion ($R_{cash} \leq 0$) triggers crisis protocols or venture termination.

6.6 | Quality Penalties on Revenue

Product quality defects directly impact revenue generation through exponential penalty functions:

$$\text{bug\_penalty} = \frac{1}{1 + (B/30)^2} \tag{31}$$

$$\text{debt\_penalty} = \frac{1}{1 + (D/80)^{1.5}} \tag{32}$$

These penalties multiply with $R_{revenue}$, ensuring that technical shortcuts eventually erode market performance.

6.7 | Game Over Conditions

The model implements three terminal conditions:

1. Financial Bankruptcy: $R_{cash} \leq 0$ (cash runway exhausted)
2. Technical Bankruptcy: $D > 200$ (technical debt exceeds maintainability threshold)
3. Quality Crisis: $B > 100$ (bug count exceeds customer tolerance)

These conditions formalize the multi-dimensional failure modes observed in software ventures.

6.8 | Capacity Evolution Mechanisms

Agent capabilities evolve through learning and degradation processes:

$$\Delta C_{i,j} = \eta_{learn} \cdot L_{context} \cdot (100 - C_{i,j}) - \eta_{decay} \cdot S_{stress} \cdot C_{i,j} \tag{33}$$

where learning context incorporates:

$$L_{context} = L_{base} \cdot \left(1 + \frac{A_{org}}{100}\right) \cdot \left(1 + \frac{TD_{boost}}{100}\right) \cdot F_{focus} \tag{34}$$

with TD_{boost} representing HR talent development contribution and F_{focus} strategic focus alignment.

Stress factors include:

$$S_{stress} = S_{base} \cdot \left(1 + \frac{C_{org}}{100}\right) \cdot \left(1 + \frac{B}{B + 50}\right) \tag{35}$$

implementing degradation under conflict and quality crises.

Turnover Dynamics:

$$P_{turnover} = \tau_{base} \cdot \left(1.5 - \frac{\overline{C_{HR}}}{100}\right) \cdot \left(1 + 0.5 \cdot \frac{C_{org}}{100}\right) \tag{36}$$

where $\tau_{base} = 0.02$ (2% monthly baseline) and $\overline{C_{HR}}$ represents normalized HR coordination capacity.

7 | STATE VARIABLE SPECIFICATIONS

This appendix provides detailed mathematical specifications for each of the six primary state variables and their governing equations.

7.1 | Product Features (F)

Accumulated functionality representing product scope. Growth is non-linear due to technical debt's velocity penalty ($\delta_{debt} \cdot D$), creating path-dependent development trajectories where early shortcuts compound into long-term velocity loss.

Equation:

$$\frac{dF}{dt} = \lambda_{feat} \sum_i C_{dev,i} \cdot (1 - \delta_{debt} \cdot D) - \mu_F \cdot F \tag{37}$$

where $\mu_F \approx 0$ (negligible deprecation), λ_{feat} is the feature development rate coefficient, $C_{dev,i}$ is development capacity of engineer i, and δ_{debt} is the technical debt drag coefficient.

7.2 | Technical Debt (D)

Accumulated design compromises and shortcuts. Represents debt principal (D), while the debt drag coefficient (δ_{debt}) operationalizes interest as an ongoing productivity tax on feature development.

Equation:

$$\frac{dD}{dt} = \lambda_{debt} \cdot V_{agg} \cdot (1 - \tau) - \rho_{refactor} \cdot W_{refactor} \tag{38}$$

with exponential penalty $\delta_{debt}(D) = 1/(1 + (D/50)^2)$, implementing Loops R1/B1. Here λ_{debt} is the technical debt generation rate, V_{agg} is aggregate development velocity, τ is quality consciousness parameter, $\rho_{refactor}$ is refactoring effectiveness coefficient, and $W_{refactor}$ is refactoring work allocation.

7.3 | Bug Count (B)

Known defects awaiting resolution. Exhibits threshold behavior—beyond critical levels ($B > 30$), bugs cascade through system interdependencies faster than resolution capacity, triggering Loop R2 (Bug Accumulation) if not arrested by crisis management.

Equation:

$$\frac{dB}{dt} = \lambda_{bugs} \cdot V_{agg} \cdot \kappa \cdot (1 - \tau) - \rho_{fix} \cdot W_{bugfix} \tag{39}$$

with penalty $\delta_{bugs}(B) = 1/(1 + (B/20)^{1.5})$, implementing Loop R2. Here λ_{bugs} is the bug introduction rate, κ is code quality factor, ρ_{fix} is bug fixing effectiveness coefficient, and W_{bugfix} is bug fixing work allocation.

7.4 | Market Share (M)

Captured market percentage ($M \in [0, 100]$). Saturation effects create diminishing returns: $(1 - M)^2$ models increasing difficulty of capturing remaining market. Multi-factor churn (base, bug-driven, misfit-driven) implements Loop B2 (Market Feedback). **Equation:**

$$\frac{dM}{dt} = \lambda_{acq} S_{cap} Q_{prod} A_{brand} (1 - M)^2 - (\chi_{base} + \chi_{bugs} B + \chi_{misfit}(100 - F_m)) M \tag{40}$$

where λ_{acq} is customer acquisition rate coefficient, S_{cap} is sales capacity, Q_{prod} is product quality, A_{brand} is brand awareness, χ_{base} is baseline customer churn rate, χ_{bugs} is bug-driven churn coefficient, χ_{misfit} is market misfit churn coefficient, and F_m is product-market fit. The saturation constraint $(1 - M)^2$ implements Loops R3/B2.

7.5 | Organizational Alignment (A_{org})

Degree of strategic coherence across teams ($A_{org} \in [0, 100]$). Acts as a capacity multiplier—misaligned organizations waste effort on internal friction rather than value creation. Natural entropy (ξ_{decay}) requires continuous management and HR coordination to sustain, implementing Loop B4 dynamics.

Equation:

$$\frac{dA_{org}}{dt} = \sum_k C_{mgmt,k} + \sum_h C_{coord,h} - (\xi_{decay} A_{org} + \xi_{conflict} C_{org}) \tag{41}$$

where $C_{mgmt,k}$ is management capacity of manager k, $C_{coord,h}$ is coordination capacity of HR agent h, ξ_{decay} is organizational alignment decay rate (half-life $\approx$ 15 months without active management), $\xi_{conflict}$ is conflict impact coefficient, and C_{org} is organizational conflict level.

7.6 | Financial Runway (R)

Available capital measured in months of operation (R). Revenue inflow depends on market share and product quality; burn rate scales linearly with headcount plus fixed overhead. Depletion triggers crisis responses (layoffs, pivots) or venture failure, implementing critical constraint on organizational dynamics.

Equation:

$$\frac{dR}{dt} = \gamma_{rev} M Q_{prod} + F_{funding}(t) - (\beta_{burn} n_{total} + \beta_{overhead}) \tag{42}$$

where γ_{rev} is revenue conversion coefficient, $F_{funding}(t)$ represents funding events at time t, β_{burn} is per-employee monthly burn rate, n_{total} is total organizational headcount, and $\beta_{overhead}$ is fixed operational overhead. Depletion ($R \leq 0$) triggers game over or crisis response.

References

Joseph Blackburn, Michael A Lapre, and Luk N Van Wassenhove. Brooks' law revisited: improving software productivity by managing complexity. *Available at SSRN 922768*, 2006.

Barry W. Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, Boston, MA, 2003.

David G Chernoguz. The system dynamics of brooks' law in team production. *Simulation*, 87(11):947–975, 2011.

Richard M Crowder, Mark A Robinson, Helen PN Hughes, and Yee-Wai Sim. The development of an agent-based modeling framework for simulating engineering team work. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(6):1425–1439, 2012.

Jim Doyle and David Ford. Mental models concepts for system dynamics research. *System Dynamics Review*, 14(1):3–29, 1998.

Xiaocong Fan and John Yen. Modeling and simulating human teamwork behaviors using intelligent agents. *Physics of life reviews*, 1(3):173–201, 2004.

Eduardo Ferreira Franco, Kechi Hirama, Stefano Armenia, and Joaquim Rocha dos Santos. A systems interpretation of the software evolution laws and their impact on technical debt management and software maintainability. *Software Quality Journal*, 31(1):179–209, 2023.

Ramy I Hindiyeh, William K Ocloo, and Jennifer A Cross. Systematic review of research trends in engineering team performance. *Engineering Management Journal*, 35(1):4–28, 2023.

Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA, anniversary edition edition, 1995.

Miroljub Kljajic and John V Farr. The role of systems engineering in the development of information systems. In *Strategic Information Systems: Concepts, Methodologies, Tools, and Applications*, pages 369–381. IGI Global Scientific Publishing, 2010.

Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012.

Samuel Lapp, Kathryn Jablokow, and Christopher McComb. Kaboom: an agent-based model for simulating cognitive style in team problem solving. *Design Science*, 5:e13, 2019.

Bengee Lee and James Miller. Multi-project software engineering analysis using systems thinking. *Software Process: Improvement and Practice*, 9(3):173–214, 2004.

Meir M Lehman. Feedback in the software evolution process. *Information and Software technology*, 38(11):681–686, 1996.

Meir M Lehman and Juan F Ramil. The impact of feedback in the global software process. *Journal of Systems and Software*, 46 (2-3):123–134, 1999.

Raymond J. Madachy. *Software Process Dynamics*. Wiley-IEEE Press, Hoboken, NJ, 2008.

Donella H. Meadows. *Thinking in Systems: A Primer*. Chelsea Green Publishing, White River Junction, VT, 2008.

John Meluso, Jesse Austin-Breneman, and Lynette Shaw. An agent-based model of miscommunication in complex system engineering organizations. *IEEE Systems Journal*, 14(3):3463–3474, 2019.

John Meluso, Jesse Austin-Breneman, James P Bagrow, and Laurent Hébert-Dufresne. A review and framework for modeling complex engineered system development processes. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(12): 7679–7691, 2022.

John H. Miller and Scott E. Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton University Press, Princeton, NJ, 2007.

Ashkan Negahban and Levent Yilmaz. Agent-based simulation applications in marketing research: an integrated review. *Journal of Simulation*, 8(2):129–142, 2014.

Michael J. North and Charles M. Macal. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, New York, NY, 2007.

Doncho Petkov, Denis Edgar-Nevill, Raymond Madachy, and Rory O'Connor. Information systems, software engineering, and systems thinking: Challenges and opportunities. *International Journal of Information Technologies and Systems Approach (IJITSA)*, 1(1):62–78, 2008.

Nelson P. Repenning. Understanding fire fighting in new product development. *Journal of Product Innovation Management*, 18 (5):285–300, 2001.

Daniel Sandria-Flores, Carmen Mezura-Godoy, and Edgard Benítez-Guerrero. Agent-based modeling and simulation for collaborative software: A systematic literature review. In *2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 117–126. IEEE, 2024.

Peter M. Senge. *The Fifth Discipline: The Art and Practice of the Learning Organization*. Doubleday, New York, NY, 1990.

Ian Sommerville. *Software Engineering*. Pearson, Boston, MA, 10th edition, 2016.

John D. Sterman. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. McGraw-Hill, Boston, MA, 2000.

Uri Wilensky and William Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. MIT Press, Cambridge, MA, 2015.

David C Wynn and P John Clarkson. Process models in design and development. *Research in engineering design*, 29(2):161–202, 2018.

Mohammad Reza Zali, Mina Najafian, and Amir Mohammad Colabi. System dynamics modeling in entrepreneurship research: A review of the literature. *International Journal of Supply and Operations Management*, 1(3):347, 2014.