

# CHAPTER 1 THE OVERVIEW OF CHALLENGES AND PRACTICES IN SOFTWARE DEVELOPMENT TEAMS

*Noorminshah A. Iahad and Masitah Ghazali*

## 1.1 OVERVIEW

Software systems today are at once ubiquitous and mission-critical. From health monitoring and public services to education, mobility, and entertainment. Software mediates value creation in society. With this centrality comes a pressing need to improve how software is specified, designed, built, validated, deployed, and maintained. Importantly, these activities are not only technical—they are deeply social and organisational. Teams must navigate ambiguity, coordinate expertise, and align with institutional goals under time and resource constraints. As Sommerville (2016) argues, engineering quality is inseparable from process quality and management discipline. Persistent challenges such as requirements volatility, communication breakdowns, testing bottlenecks, and post-release support continue to surface across contexts (Menezes et al., 2019).

This book brings together practise-informed chapters from developers, analysts, and academics who have delivered complex software in the Malaysian research university context and beyond. The compilation does not prescribe a single methodology. Instead, it

spotlights concrete practises that teams have used to overcome real constraints: Blending User Experience (UX) in conventional lifecycles, modelling goals with stakeholder roles, institutionalising daily stand-ups, leveraging GitLab at scale, structuring agile delivery in healthcare, improving user acceptance testing (UAT), and professionalising helpdesk ticketing to close the loop between operations and development. Throughout, we emphasise actionable heuristics and patterns that complement established bodies of knowledge.

Chapter 1 serves as the book’s roadmap. It situates the remaining chapters in the broader discourse of software engineering practise and management; synthesises recurrent challenges; presents a unifying lens to connect people, process, product, and platform concerns; and offers reading paths for different roles. The goal is to help readers grasp not merely what was done in each chapter, but why it worked, when to apply it, and how to adapt it in their own organisations.

## **1.2 HISTORICAL CONTEXT OF SOFTWARE ENGINEERING PRACTISES**

The journey of software engineering has been shaped by successive attempts to bring discipline and predictability to an inherently complex and creative endeavour. Since the 1968 NATO Conference on Software Engineering, often regarded as the field’s formal inception, the profession has evolved from viewing software as a product of individual craftsmanship to treating it as a managed, team-based process. Early paradigms emphasised structure and control: Methods such as Waterfall and V-Model formalised sequential phases of requirements, design, coding, testing, and maintenance. These approaches improved traceability but often failed to accommodate evolving requirements or stakeholder feedback.

By the 1990s, the community recognised the limitations of rigid life cycles. The rise of iterative and incremental models such as the Spiral Model, Rational Unified Process (RUP), and later Agile signalled a shift towards adaptability and collaboration. The Agile Manifesto (Beck et

al., 2001) further revolutionised thinking by prioritising individuals and interactions over processes and tools, and working software over comprehensive documentation. In parallel, the growth of open-source ecosystems and version control tools—CVS, Subversion, and later Git, reshaped how teams shared and reviewed code.

The 2010s brought DevOps, continuous delivery, and integrated toolchains that blurred boundaries between development and operations. In the same period, usability engineering and UX design entered mainstream software processes, enriching the discipline with human-centred perspectives. Today, software development in research universities, public agencies, and industry blends these influences: formal governance for accountability, agile practises for responsiveness, and human-centred design for usability. This historical evolution underscores a key idea: Effective software engineering is less about following a particular methodology than about adapting methods to purpose and context, a principle that anchors the chapters in this book.

### **1.3 METHODOLOGICAL LENS**

This book adopts a practise-based research orientation, drawing upon lived experiences of teams as a legitimate source of knowledge. In contrast to controlled experiments or purely theoretical analysis, practise-based research privileges situated insight, how professionals navigate ambiguity, negotiate trade-offs, and apply judgment under real constraints. The focus is not merely on what practises exist but on how and why they work in particular contexts.

Each chapter contributes to this collective understanding by documenting interventions that improved coordination, quality, or user satisfaction within Malaysian research universities and related organisations. The contributors comprise academics, developers, analysts, and support staff who operate at the intersection of technology, education, and governance. Their reflections combine empirical observation with tacit expertise, what Schön (1983) termed “reflection-in-action”.

To maintain rigour, each case study was structured around three guiding questions:

- (1) Challenge – What systemic or recurring issues were encountered?
- (2) Intervention – What process, tool, or organisational change was introduced?
- (3) Outcome – What benefits or lessons were observed, and what limitations remain?

This framing enables cumulative learning across chapters. It also positions the book as both a scholarly and practical contribution, bridging the often-cited gap between research and real-world practise.

## **1.4 OVERCOMING THE CHALLENGES**

A useful way to view software work is through two complementary perspectives:

- (1) Process perspective  
Are we doing the right activities, in the right order, with the right stakeholders? Are requirements elicited with context, prioritised transparently, and refined into models and user stories that guide design and testing? Are risks surfaced early and managed continuously?
- (2) Product perspective  
Does the evolving system meet its specifications and stakeholder intent? Is it testable, observable in production, compliant, and maintainable? Are we validating fitness-for-use with representative users, not just verifying conformance to requirements?

Teams that succeed integrate these perspectives. Menezes et al. (2019) and Saiedian (2002), suggest that success factors cluster around four levers: (1) Engaged stakeholders and empowered teams, (2) Incremental value delivery with feedback, (3) Disciplined engineering (definition of done, reviews, continuous integration (CI), test automation), and (4) Visible risk management. Shift-left practises, bringing requirements clarification, quality engineering, and security

earlier into the lifecycle, reduce cost of change (Rani et al., 2023). Meanwhile, research on requirements automation shows growing tool support for analysis, traceability, and consistency checking (Umar & Lano, 2024).

However, practises only work when adapted to local realities. The cases in this book occur within public-sector governance, academic calendars, heterogeneous infrastructure, and evolving policies. They show how teams customise agile rituals, low-fidelity prototyping, GitLab pipelines, and ticketing workflows to their environments, illustrating a pragmatic, context-sensitive agility rather than method orthodoxy.

## **1.5 SIGNIFICANCE OF THIS BOOK**

This book is written for practitioners (developers, testers, UX, product owners, project managers), leaders in information technology (IT) organisations, and students learning to bridge theory and practice. Its significance is threefold:

- (1) Practise-grounded synthesis – Each chapter distils what practitioners tried, why, what failed, and what finally worked.
- (2) Transferable checklists – We extract ‘ready-to-apply’ steps and anti-patterns to accelerate adoption in other teams.
- (3) Educational value – Cases complement coursework on software engineering management, requirements, Human-computer Interaction and User Experience (HCI/UX), and DevOps.

By foregrounding lived experience, the volume complements reference texts (Sommerville, 2016) with situated know-how. It offers a realistic narrative of improvement under constraints, rather than idealised textbook scenarios.

Beyond its pedagogical and professional relevance, this volume aligns with Malaysia’s broader agenda for digital transformation in higher education. As universities modernise their information systems and student services, in-house software development teams face the dual challenge of technical excellence and institutional compliance.

Documenting successful practises in requirements management, agile rituals, DevOps, and user-centred design thus becomes an act of capacity building, strengthening institutional memory and promoting professional standards across campuses.

The cases in this book exemplify how public universities can operate as innovation labs: Experimenting with methods, tools, and governance models while serving diverse internal users. In doing so, they contribute to national digital resilience, ensuring that local systems evolve sustainably, without overdependence on external vendors. This local, context-sensitive innovation has implications beyond academia; it informs public-sector digitalisation, IT governance, and software quality assurance more broadly.

## **1.6 SOFTWARE DEVELOPMENT TEAMS IN THE MODERN ERA**

Modern teams are cross-functional and increasingly distributed. They coordinate across time zones, institutional units, and vendors; they combine domain knowledge with engineering, UX, data, and operations. Tools such as GitLab provide a single platform spanning work planning, version control, continuous integration/continuous delivery (CI/CD), code review, and incident response. Yet tools are not panaceas. Teams must still establish working agreements, definitions of ready and done, branching strategies, and escalation paths. Management must align incentives and remove friction. The chapters in Part 1 exemplify this blend of culture and tooling.

A recurring theme is information radiators; artifacts that make work and quality visible: User stories tied to personas, goal graphs with stakeholder roles, sprint boards, CI pipelines, UAT dashboards, and helpdesk analytics. Visibility enables shared situational awareness, a prerequisite for collective ownership and faster learning loops.

## 1.7 KEY TERMS AND A SHARED LENSE

To connect the chapters, we adopt a simple lens: People–Process–Product–Platform (P4) as a mental model for interventions:

- (1) People – Roles, skills, communication patterns, leadership, and stakeholder engagement.
- (2) Process – Ways of working: Requirements, design, development, testing, release, support.
- (3) Product – The system’s evolving quality: Usability, reliability, security, compliance, maintainability.
- (4) Platform – The toolchain and infrastructure: GitLab, pipelines, environments, ticketing, monitoring.

Across the chapters, improvements succeed when at least two P4 dimensions are addressed in concert (e.g., daily stand-ups change People and Process; GitLab pipelines affect Process and Platform; UX prototyping shapes People, Product, and Process).

## 1.8 CHALLENGES MAP ACROSS THE BOOK

Table 1.1 maps recurring challenges to the chapters that address them and highlights the primary P4 dimensions involved.

**Table 1.1** Recurring challenges

| Challenge                                   | Addressed In | Primary Practises                                                 | P4 Focus                 |
|---------------------------------------------|--------------|-------------------------------------------------------------------|--------------------------|
| Blending UX in conventional lifecycles      | Chapter 2    | Personas, Design Thinking, Low-fi prototyping, Agile UX           | People, Product, Process |
| Clarifying stakeholder goals and trade-offs | Chapter 3    | Role-based goal graphs, confidence factors, prioritisation matrix | People, Process          |
| Coordinating large codebases and teams      | Chapter 4    | GitLab branching, CI/CD, runners, protected branches              | Process, Platform        |

continue

**Table 1.1** (Continued)

| Challenge                                        | Addressed In | Primary Practises                                           | P4 Focus                 |
|--------------------------------------------------|--------------|-------------------------------------------------------------|--------------------------|
| Keeping alignment in day-to-day work             | Chapter 5    | Daily stand-ups, sprint reviews, retrospectives             | People, Process          |
| Agile delivery in regulated domains (healthcare) | Chapter 6    | Backlogs, PM tools, iterative releases, stakeholder reviews | Process, Product         |
| Strengthening User Acceptance Testing (UAT)      | Chapter 7    | UAT planning, roles, metrics, feedback loops                | People, Product, Process |
| Closing the loop with operations/support         | Chapter 8    | Helpdesk ticketing, SLAs, categorisation, analytics         | Process, Platform        |

While each chapter offers distinct insights, several unifying themes emerge across the collection.

(1) Visibility and transparency

Effective teams make work visible. Whether through sprint boards, GitLab dashboards, or UAT logs, information radiators transform tacit activity into shared awareness. This visibility builds trust, enables early problem detection, and fosters joint ownership of outcomes.

(2) Continuous learning

Improvement is treated as iterative, not episodic. Teams hold retrospectives, gather feedback, and refine both their products and their processes. The act of reflection becomes part of the workflow rather than a post-mortem exercise.

(3) Contextual adaptation

Every team reinterprets global methods to fit local constraints. Agile principles are preserved, but rituals and artefacts are tailored. This reflects an understanding that frameworks are scaffolds, not prescriptions. The chapters reveal that success often lies not in full compliance with a method, but in intelligent adaptation.

(4) Human-centred collaboration

Across chapters, the most persistent challenges are social rather than technical: Communication gaps, role ambiguity, and motivational drift. Teams that thrive invest in clarity of roles, psychological safety,

and cross-functional empathy. The P4framework provides a language for diagnosing these interactions and balancing them holistically.

(5) Institutional learning

Finally, improvement does not end with a single project. Teams document lessons, share checklists, and influence policy. In this sense, each case study represents a node in an evolving institutional learning network, where knowledge is accumulated, refined, and reused.

Together, these themes depict software engineering as a socio-technical ecosystem, not merely a technical craft. They remind readers that sustainable improvement depends on *learning systems*, not just individual competence.

## **1.9 EXTENDED SYNOPSES OF CHAPTERS**

The Chapter 2 – Embracing and Blending UX Approach in Conventional Software Development. This chapter chronicles two initiatives; UTMSmart and Monitoring Online Kids Activity (MOOKA), showing how UX research can be introduced incrementally into existing lifecycles. Authors discuss integrating user experience (UX) methodologies in projects like MOOKA and the UTM Smart mobile application. For UTMSmart, the team moved from requirement lists to user stories grounded in personas, supported by internal mock-ups and usability evaluations before major releases. The practise matured into recurring workshops and journey reviews that made user value visible. In MOOKA, the authors apply design thinking: empathising with teachers and parents, ideating through “How Might We” (HMW) exercises, and validating flows using paper prototypes. Key takeaways: (1) Low-fidelity prototyping accelerates learning, (2) UX reduces rework by clarifying value early, and (3) Small wins compound into culture change. The chapter offers a practical pattern for teams wanting UX benefits without halting ongoing delivery.

Chapter 3 – Role-Based Goal Modelling with Requirements Classification Matrix. Authors examine role-based goal modelling, emphasising the importance of clear roles in requirements categorisation.

Here the authors extend goal modelling by explicitly attaching stakeholder roles to goals and by rating sub-goals with confidence factors: ASSUME, ACHIEVE, MANDATE, REFINE. Feasibility and adequacy propagate through the goal graph, creating a visual map of where to proceed, exercise caution, or do not proceed. A labour-management case illustrates how this method informs prioritisation and stakeholder conversations. For teams, the technique functions as a lightweight risk register for requirements, aligning expectations and surfacing uncertainty early.

Chapter 4 – An Experience of System Developers Using GitLab. This chapter documents the introduction of GitLab as an end-to-end collaboration platform. Authors reflect on using Gitlab to manage large-scale projects. The proposed three-branch framework (A for developer integration, B for development deployments, and C for production releases) clarifies roles and reduces conflicts. The team addresses common pain points: Merge conflicts when developers forget to pull, single-admin bottlenecks, protected branches, GitLab Runners for parallel CI, and Docker for consistent environments. Lessons emphasise that tool adoption must be accompanied by working agreements (branching, reviews, rollbacks) and role clarity.

Chapter 5 – It’s Not Just Stand-up, It’s Daily Stand-up Meeting! Daily stand-ups, when treated as a serious planning ritual rather than a status recital, become the heartbeat of agile delivery. Authors highlight how agile daily stand-up meetings have transformed their development practises. The chapter reframes the stand-up as a coordination micro-meeting: What we’re aiming to finish by next stand-up, what is blocked, who needs to pair or swarm, and what quality gates apply. Coupled with sprint reviews and retrospectives, stand-ups create fast feedback and shared accountability—particularly important for distributed teams.

Chapter 6 – Applying System Agile Development with Project Management Tools for Healthcare System: An Experience Report. Healthcare imposes high stakes and compliance constraints. Authors share their experience applying systematic agile development in healthcare. The authors report how agile delivery and PM tools help balance iterative development with stakeholder reviews and traceability. The case highlights backlog hygiene, definition of ready/done, and the benefits of time-boxed

releases for managing expectations. The chapter is valuable for teams operating in regulated domains that must demonstrate due diligence while staying responsive.

Chapter 7 – Challenges and Suggestions of User Acceptance Testing Improvement: A Research University Case Study. UAT often becomes a late-stage bottleneck. Authors discuss challenges and recommendations for user acceptance testing in a research university setting. This chapter surfaces organisational causes; unclear roles, insufficient test data, and limited user availability, and proposes remedies: Early UAT planning, scenario-based test design, decision logs, and metrics such as defect leakage and UAT cycle time. The message: Shift-left UAT by engaging users early and continuously, so acceptance becomes a confirmation rather than a surprise.

Chapter 8 – Helpdesk Ticketing Approach: Motivating Effective Supports to System Development Services. Support is part of the product. The helpdesk ticketing chapter shows how categorisation, Service Level Agreements (SLAs), and analytics convert ad-hoc requests into a managed flow of work. Authors describe leveraging helpdesk ticketing for system development support. When connected with development (e.g., linking tickets to issues and releases), the helpdesk becomes a learning engine: Surfacing patterns, informing prioritisation, and closing feedback loops with users.

## **1.10 CONTEXT: THE MALAYSIAN RESEARCH UNIVERSITY**

A distinctive feature of the cases is their setting in a research university ecosystem. Teams must integrate student and staff needs, academic cycles, national reporting (e.g., research assessment), and enterprise systems. Budgets and staffing ebb and flow; yet service expectations are high. The chapters demonstrate how to move forward with incremental modernisation; Adopting agile rituals, improving requirements quality, instrumenting toolchains, while stewarding legacy systems and processes.

### **1.11 FUTURE RESEARCH DIRECTIONS**

The practise cases compiled in this book open several avenues for further inquiry. Comparative studies could examine how similar practises unfold in different institutional settings: Public versus private universities, or academia versus government agencies. Longitudinal research may also reveal how agile adoption and DevOps maturity evolve over time within the same organisation, identifying enablers, and barriers to sustained improvement.

A promising direction lies in bridging software engineering with HCI and ethics. As systems become increasingly intelligent and data-driven, developers must address usability, transparency, and fairness as first-class design criteria. This requires not only technical tooling but also interdisciplinary collaboration among software engineers, designers, and social scientists.

Additionally, future work could explore the educational dimension: How best to train students and early-career professionals in reflective software practise. Embedding these lessons in curricula would help close the gap between classroom learning and professional realities, an ongoing challenge in engineering education.

### **1.12 THE ROAD AHEAD**

Looking forward, three developments will shape software teamwork:

- (1) Increasing automation in the lifecycle. Requirement’s analysis, test generation, and deployment checks will see more artificial intelligent (AI) support (Umar & Lano, 2024).
- (2) Product-orientation of teams. Persistent, cross-functional teams owning outcomes, not just projects, will become the norm.
- (3) Socio-technical resilience. Security, privacy, accessibility, and sustainability will be treated as first-class requirements from the outset.

The through-line in this book is disciplined adaptability. Practises must be explicit, measured, and continuously improved; but they must

also be humane and context-aware. We invite readers to adopt, adapt, and extend the patterns presented here.

And as artificial intelligence continues to permeate the software lifecycle—from automated requirement analysis to AI-assisted code generation—new opportunities and risks emerge. Tools like Copilot, ChatGPT, and automated testing agents accelerate productivity but also demand new ethical literacy—ensuring that generated artefacts remain accountable, secure, and inclusive. Teams must cultivate the capacity to supervise the automation, not surrender to it.

Equally, sustainability is becoming an inseparable part of software quality. Energy-efficient coding, cloud resource optimisation, and lifecycle management now sit alongside maintainability and security as core concerns. Green software engineering encourages teams to quantify and reduce the carbon impact of their systems, aligning technical excellence with environmental stewardship.

The future of software work will thus hinge on sociotechnical resilience: The ability of teams to respond to uncertainty with transparency, adaptability, and care. As this book illustrates, progress in software engineering is not simply about adopting tools or frameworks, but about nurturing reflective, collaborative, and ethically grounded practitioners.

## REFERENCES

- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. Agile Alliance.
- Menezes, J., Gusmão, C. & Moura, H. (2019). Risk factors in software development projects: a systematic literature review. *Software Quality Journal*, 27, 1149–1174 (2019). <https://doi.org/10.1007/s11219-018-9427-5>

- Rani, V. S., Babu, D. A. R., Deepthi, K. & Reddy, V. R. (2023). *Shift-left testing in DevOps: A study of benefits, challenges, and best practises* [Conference session]. 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), 1675–1680.
- Saiedian, H. (2002). Best practises in software engineering. *Journal of Systems and Software*, 61(3), 171–172.
- Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic Books.
- Sommerville, I. (2016). *Software engineering* (10th Edition). Pearson Education Limited.
- Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: A systematic literature review. *Requirements Engineering*. <https://doi.org/10.1007/s00766-023-00411-0>