INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH AND INNOVATIVE STUDIES
ISSN: 2820-7157
www.ijsrisjournal.com

June 2026 Volume 5 Number 3 268-288

Received Date: 21 April 2026 Accepted Date: 12 May 2026 Published Date: 1 June 2026

# Modern methods of analysis, software design and emerging implementation tools: challenges, opportunities and prospects

**Blaise KAPALALA KAPENDA [1], Merveille BONTE KASSONGO [2], Dan LUKOKI TULANDA [3], Trésor MBUYAMBA MBUYAMBA [4]**

1. Department: Computer Science, Institut Supérieur Pédagogique de la Gombe, Province of Kinshasa, Democratic Republic of the Congo, blaisekapalala94@gmail.com , (+243) 811 459 397
2. Faculty: Science and Technology, Department: Mathematics, Statistics and Computer Science, National Pedagogical University, Province of Kinshasa, Democratic Republic of the Congo, merkasbonte@gmail.com , (+243) 812 180 450
3. Department of Computer Science, Gombe Higher Institute of Education, Kinshasa City Province, Democratic Republic of the Congo, danlukoki2@gmail.com , (+243) 823 841 178
4. Department of Computer Science, Institut Supérieur Pédagogique de la Gombe, Kinshasa City Province, Democratic Republic of the Congo, mbuyamba12@gmail.com , (+243) 851 916 057

**Abstract**

Software engineering is undergoing a phase of rapid transformation driven by the combined influence of agile methods, DevOps, cloud computing, microservices architectures and generative artificial intelligence. This article analyses the evolution of software analysis and design methods, compares traditional and modern approaches, examines emerging implementation tools and discusses the technical, human and economic challenges, with a particular focus on the contexts of developing countries and the DRC.

The findings point to four main conclusions:

1. Agile methods now dominate modern software projects thanks to their ability to adapt to changing requirements.
2. Microservices architectures improve scalability and resilience, at the cost of greater operational complexity.
3. Generative AI boosts developers' productivity (code generation, testing, documentation), but raises issues relating to governance, security and quality.

4. Low-code/no-code platforms accelerate application prototyping and delivery, whilst limiting customisation and portability.

By 2030, organisations that combine AI-assisted software engineering, automated DevOps, well-managed distributed architectures and continuous team training are expected to reap the most significant gains in terms of quality, time-to-market and maintainability.

**Keywords:** Software engineering; Software analysis; Software design; Agile methods; DevOps; Generative artificial intelligence; Microservices; Digital transformation.

## 1. Introduction

### 1.1 Background

Software has become the invisible infrastructure of the digital economy. Government bodies, businesses, universities and international organisations rely on information systems on a daily basis to manage their operations, communicate and make decisions. This growing dependence imposes high demands in terms of quality, security, availability and maintainability.

Software engineering emerged to meet these requirements by applying systematic, disciplined and measurable methods to the development, operation and maintenance of software. Today, the rise of the cloud, big data, DevOps and generative AI is profoundly transforming traditional design and development practices.

### 1.2 The Issue

Despite methodological and technological advances, organisations face several challenges:

- The increasing complexity of distributed systems;
- Rapidly changing business requirements;
- Cybersecurity and data protection;
- Quality, technical debt and maintainability;
- Integration of emerging technologies;
- A shortage of specialist skills in many developing countries.

It is therefore necessary to examine modern methods of analysis and design, as well as emerging tools that could improve the performance of the software development process.

### 1.3 General objective

To analyse modern methods of software analysis and design, as well as emerging implementation tools, in order to identify their challenges, opportunities and future prospects.

### 1.4 Specific objectives

1. To present the theoretical foundations of software analysis and design methods.
2. To examine the main modern implementation tools.
3. To identify the challenges associated with their adoption.
4. Assess their impact on software quality.
5. Propose future directions for software engineering.

### 1.5 Research questions

1. Which methods currently dominate software analysis and design?
2. Which emerging tools are transforming software development?
3. What technical, human and economic challenges accompany these technologies?
4. What opportunities do they open up for the future of software engineering?

## 2. Literature review

### 2.1 The concept of software

Software is a set of organised instructions enabling a computer system to perform specific tasks. These include, in particular:

- System software;
- Application software;
- Embedded software;
- Artificial intelligence software;
- Web and mobile applications.

### 2.2 Software analysis

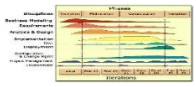
Software analysis aims to understand users' needs and define the expected functionalities. It includes requirements gathering, functional and non-functional analysis, process modelling and the preparation of specifications. A rigorous analysis reduces the risk of project failure by clarifying requirements from the outset.

### 2.3 Software Design

Design transforms requirements into technical architecture. It covers architectural design (components and interactions), detailed design (algorithms, databases, interfaces) and object-oriented design (classes, objects, relationships).

### 2.4 Evolution of development methods

**Table 1** — Different approaches to software development

| Approach | Principle | Strengths | Limitations |
|---|---|---|---|
| Waterfall  | Sequential phases | Simplicity, comprehensive documentation | Low flexibility in the face of change |
| V-model  | Validation/verification associated with the phases | Test traceability | Not well suited to changing requirements |
| Spiral  | Risk-centred iterations | Proactive risk management | Complexity of management |
| RUP  **RUP (Rational Unified Process)** | Object-oriented methodology based on UML | A structured framework rich in artefacts | High organisational cost |
| Agile methods (Scrum, XP, Kanban, Lean)  | Short iterations and collaboration | Adaptability, frequent deliveries | Requires strong team discipline |

## 3. Modern software analysis methods

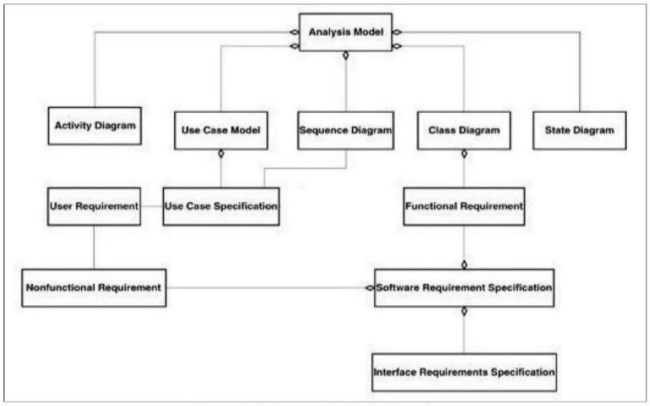



**Figure 1:** Software analysis model

### 3.1 Functional analysis

Functional analysis answers the question 'What must the system do?' without immediately imposing technical choices.

Common techniques

- Interviews and questionnaires;
- Process observation;
- Participatory workshops;
- Use cases;
- Flowcharts and BPMN.

Use cases facilitate dialogue between users and developers by describing the interactions between actors and the system.

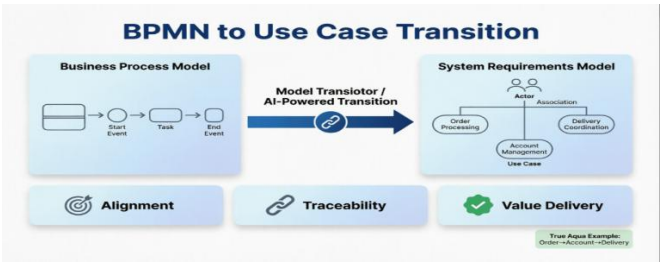



**Figure 2:** Functional analysis

### 3.2 Object-oriented analysis

The object-oriented approach models domain entities as objects with state (attributes) and behaviour (methods). The key concepts are:

- Classes: models describing similar objects;
- Objects: concrete instances;
- Inheritance: re-use and specialisation;
- Encapsulation: protection of data and invariants.

This approach improves the maintainability and reusability of code in complex systems.

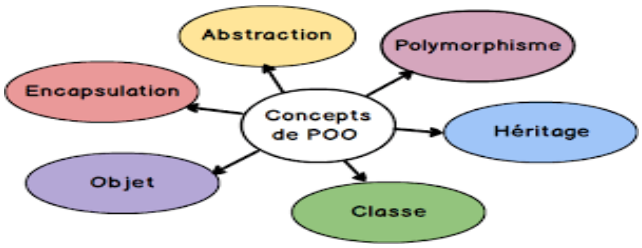



**Figure 3:** The object-oriented approach

### 3.3 Data-driven analysis

With the rise of Big Data, data has become a central element of software design. The main activities include:

- Conceptual, logical and physical data modelling;
- Predictive analytics;
- Data mining;
- Data governance and quality.

Modern systems often need to reconcile traditional transactional models with real-time analytical pipelines.

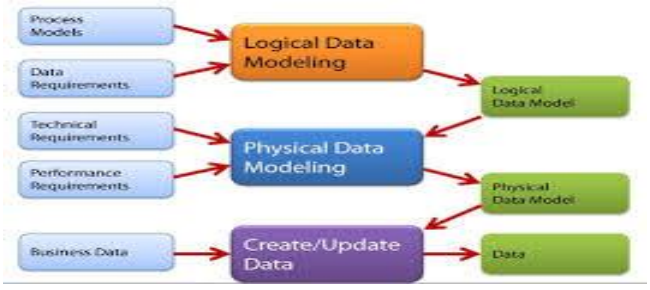



**Figure 4:** Data-driven analysis

### 3.4 Artificial Intelligence-assisted analysis

Artificial Intelligence is used to:

- Automatically analyse textual requirements;
- Detect inconsistencies or ambiguities;
- Predict project risks;
- Generate or supplement documentation.

These applications serve as decision-making aids: human validation remains essential, particularly for safety, compliance and business interpretation.

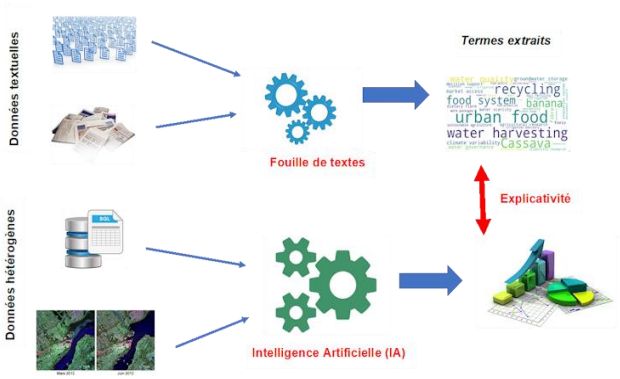



**Figure 5:** Analysis based on Artificial Intelligence

## 4. Software design methods

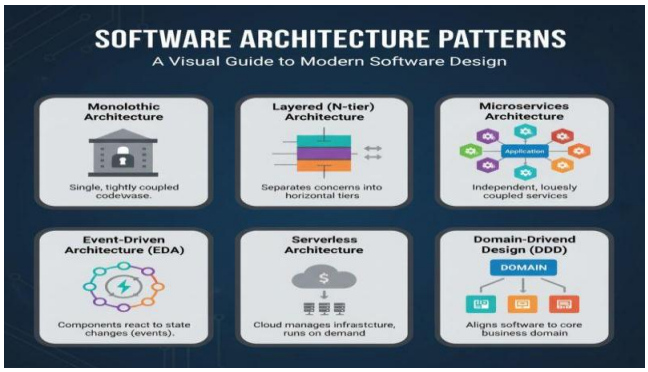



**Figure 6:** Software architecture

### 4.1 Structured design

Structured design breaks the system down into hierarchical modules. It remains useful for relatively stable, medium-sized applications, but is difficult to adapt when requirements change frequently.

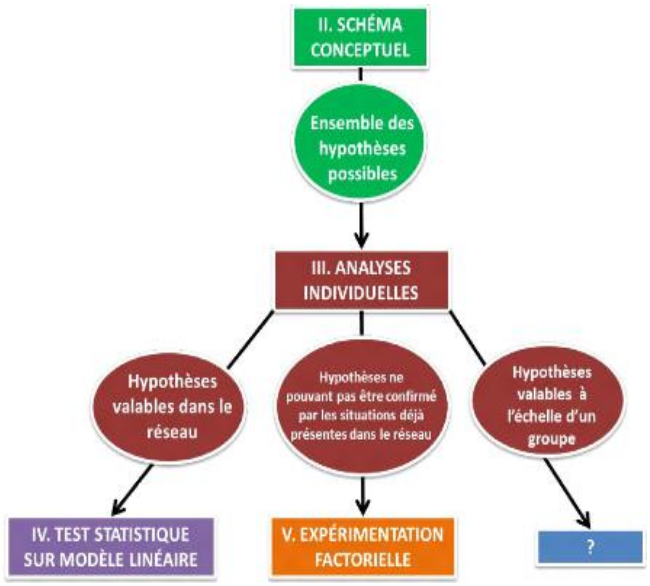



**Figure 7:** Structured design

### 4.2 Object-Oriented Design

Based on abstraction, encapsulation, inheritance and polymorphism, it promotes reusability, maintainability and scalability. It remains the dominant approach in modern enterprise applications.

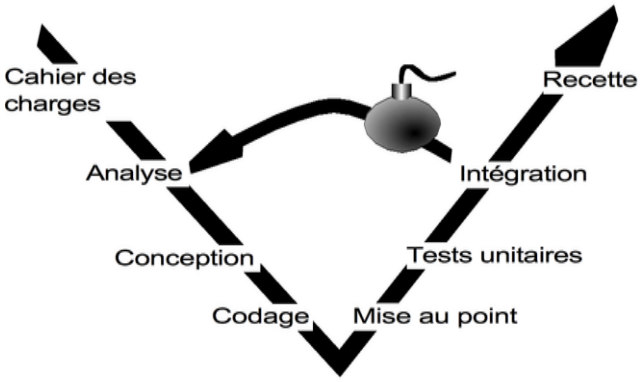



**Figure 8:** Object-Oriented Design

### 4.3 SOA (Service-Oriented Architecture)

SOA breaks applications down into services that communicate via standardised interfaces. It facilitates the integration of heterogeneous systems but can introduce complex orchestration layers.

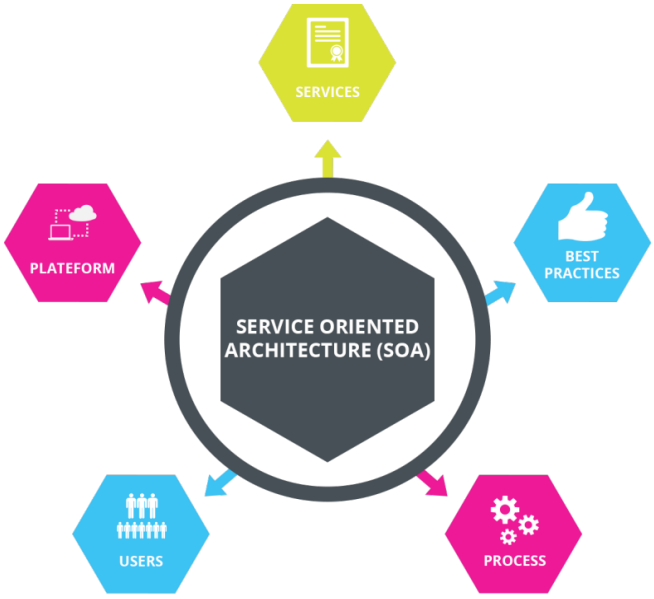



**Figure 9:** Service-Oriented Architecture

### 4.4 Microservices architecture

Microservices take the breakdown into autonomous services a step further. Characteristics

- Loose coupling;
- Independent deployment;
- High scalability;
- Increased resilience;
- Frequent alignment with product teams.

On the other hand, they increase operational complexity: observability, networking, cross-service security, distributed data management and orchestration.

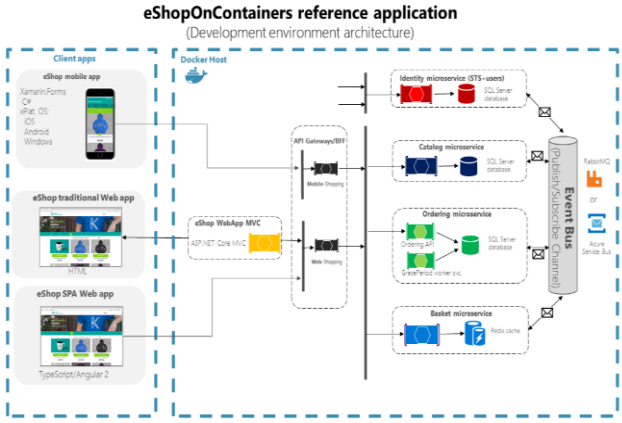



**Figure 10:** Microservices architecture

### 4.5 Component-based design

This approach prioritises the reuse of existing components in order to reduce development costs and lead times.

## 5. Modern and emerging implementation tools

### 5.1 Modelling tools

Tools such as StarUML, Visual Paradigm, Enterprise Architect and Draw.io facilitate the graphical representation of systems and communication between stakeholders.

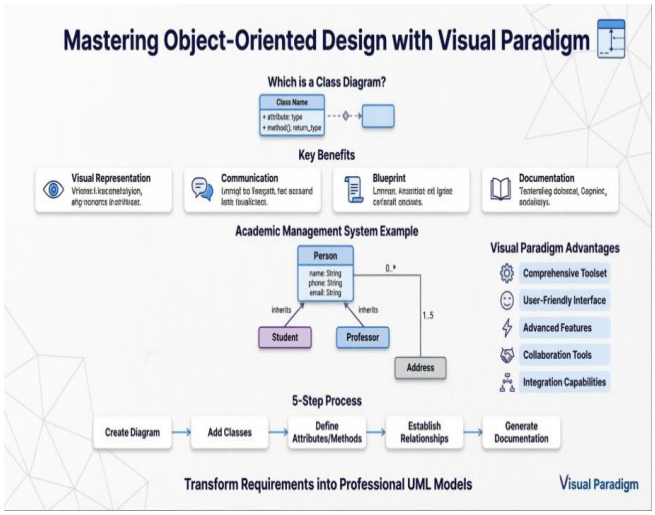



**Figure 11:** Various modelling tools

### 5.2 Integrated Development Environments (IDEs)

Modern IDEs (Visual Studio, IntelliJ IDEA, Eclipse, NetBeans and VS Code in common practice) offer intelligent code completion, refactoring, advanced debugging and test integration.


**Figure 12:** Integrated Development Environments (IDEs)

### 5.3 DevOps and automation

DevOps brings development and operations closer together through automation and continuous delivery.

Common tools

- Git, GitHub, GitLab;
- Jenkins and other CI/CD platforms;
- Docker;
- Kubernetes;
- Ansible and Infrastructure as Code tools.

The benefits observed include more frequent deployments, reduced time-to-production and improved operational reliability when correctly implemented.

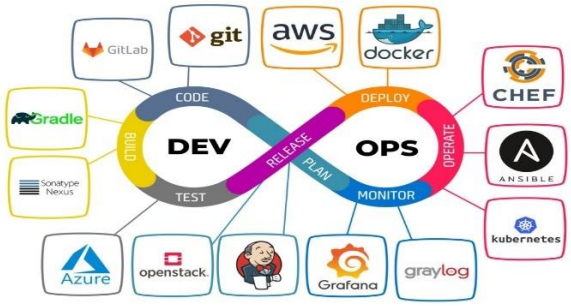



**Figure 13:** DevOps and automation components

### 5.4 Low-Code and No-Code Platforms

These platforms enable the rapid creation of applications with little or no programming.

**Table 2 —** Low-Code and No-Code Platforms

| Aspect | Advantages | Limitations |
|---|---|---|
| Development | Reduced development time | Customisation can sometimes be limited |
| Initial cost | Lower prototyping costs | Dependence on the supplier |
| Accessibility | Accessibility for non-developers | Governance and security to be managed |

### 5.5 Generative Artificial Intelligence for software development

Artificial Intelligence assistants are used for:

- Code generation;
- Test generation;
- Automatic documentation;
- Assistance with debugging and refactoring.

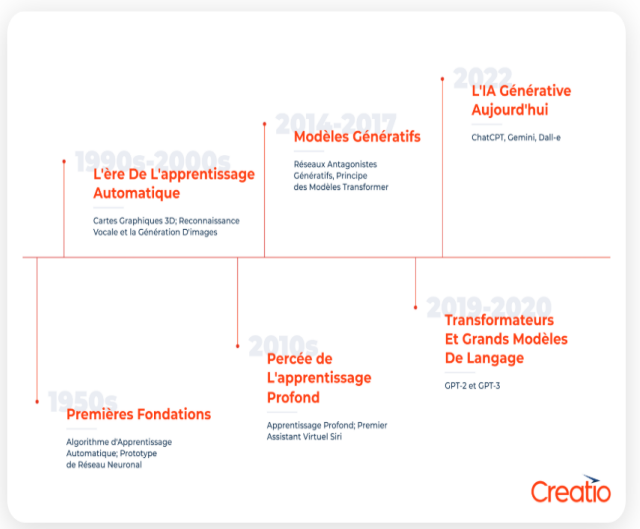



**Figure 14:** DevOps and automation

The main challenge is to transform these productivity gains into measurable quality improvements through code reviews, automated testing and appropriate security policies.

### 5.6 Software security and the DevSecOps approach

The previous sections have shown that modern methods — short iterations, automated deployment, decomposition into microservices, and the growing use of AI-driven code generation — accelerate software production. However, this acceleration shifts the attack surface: a defect introduced early in the process now propagates to production within minutes rather than over several weeks. Security can therefore no longer be treated as a final validation stage, but must become a cross-cutting life-cycle- , on a par with maintainability or reliability in the ISO/IEC 25010 quality model (ISO/IEC, 2023). This is precisely the aim of the DevSecOps approach, presented here as the logical extension of modern engineering practices rather than as a separate discipline.

#### 5.6.1 From ‘Security by Design’ to DevSecOps

The principle of ‘Security by Design’ posits that security properties must be specified and verified from the earliest stages of the lifecycle, rather than added as an afterthought. SWEBOK V4 (IEEE Computer Society, 2024) enshrines this integration by linking security to design and development activities, whilst the ISO/IEC/IEEE 12207 lifecycle standard provides the process framework for incorporating security checkpoints.

DevSecOps puts this principle into practice by extending the continuous integration and continuous deployment (CI/CD) pipeline described in section 5.3: in addition to quality and non-regression checks, automated security checks are performed whenever the code is modified. This approach of moving checks further upstream, commonly known as ‘shift-left’, aims to reduce the cost of rectifying vulnerabilities, a cost that rises significantly as the lifecycle progresses (Kim et al., 2023). DevSecOps does not eliminate security expertise; it industrialises it and makes it a continuous process.

#### 5.6.2 Threat modelling

Before implementing security measures, it is first necessary to know what you are protecting yourself against. Threat modelling is a structured approach to identifying risks, carried out right from the design stage. The STRIDE method — an acronym standing for impersonation, data tampering, repudiation, information disclosure, denial of service and privilege escalation — provides a systematic and reproducible framework suited to engineering work.

When applied to the microservices-based e-commerce architecture discussed in section 9.1, this framework highlights risks that functional decomposition alone fails to reveal. The table below provides a concise and illustrative overview.

**Table 3 —** Threat modelling

| STRIDE Category | Example of a threat to the e-commerce architecture | Design countermeasure |
|---|---|---|
| Spoofing | A service calls the payment service whilst impersonating the order service | Mutual cross-service authentication (mTLS) |
| Tampering | Alteration of the amount of a transaction passing between services | Message signature and integrity |
| Repudiation | Lack of traceability for a sensitive operation | Centralised logging and time-stamping |
| Disclosure | Exposure of an internal API without access control | API gateway and the principle of least privilege |
| Denial of service | Service overload due to an abnormal volume of requests | Throttling and fault isolation |
| Elevation of privileges | A service token gains unauthorised rights | Fine-grained authorisation policies (zero trust) |

This approach illustrates the shift in modern methodologies: security is not an add-on, but a design constraint that guides architectural choices.

### 5.6.3 The DevSecOps pipeline: automation of controls

The practical integration of security relies on embedding automated controls within the CI/CD pipeline. Four families of tools, triggered with every code deployment, form the backbone of this approach:

- **Static analysis (SAST)** examines the source code for structural vulnerabilities (injections, incorrect error handling) without executing it;
- **Dynamic analysis (DAST)** probes the running application to detect exploitable flaws at runtime;
- **Software composition analysis (SCA)** identifies third-party dependencies and checks their versions against known vulnerability databases;
- **Secret detection** prevents the accidental leakage of access keys or passwords stored in the code repository.

These checks can condition the continuation of the pipeline via security gates: a critical vulnerability blocks deployment, much like the test coverage thresholds already common in continuous integration. The literature on the performance of development organisations shows that this automation, far from slowing down delivery, improves its stability when properly calibrated (Forsgren et al., 2018; Kim et al., 2023).

### 5.6.4 Security of microservices architectures

The decomposition into microservices, presented in section 4.4 as a factor in scalability and agility, multiplies the number of communication points and, consequently, the attack surface. Whereas a monolithic application exposes a single boundary, a distributed architecture must secure every inter-service flow. NIST devotes a series of dedicated recommendations to this issue (NIST SP 800-204 et seq.), the key principles of which are:

- **Mutual authentication (mtls)** and the systematic encryption of internal communications, often implemented using a service mesh;
- **The zero-trust model,** which abandons the assumption of a trusted internal network in favour of verifying every request;
- **Centralisation of access via an API gateway** that enforces authentication, authorisation and rate limiting;
- **Secure management of** distributed **secrets**, stored in dedicated vaults rather than within service configurations.

The security of a microservices architecture therefore does not stem from the robustness of each service taken in isolation, but from the systematic control of interactions between services.

### 5.6.5 Software supply chain security

The widespread use of third-party libraries, a hallmark of modern development, shifts a substantial portion of the risk to the supply chain: a compromised component upstream contaminates all the applications that depend on it. Attacks involving dependency confusion or the injection of malicious code into popular packages illustrate the critical nature of this issue.

The methodological approach is based on three key elements. The software bill of materials (SBOM) provides a comprehensive and verifiable inventory of an application's components, which is a prerequisite for any rapid response in the event of a published vulnerability. Verification of the provenance and integrity of artefacts, formalised by standards such as SLSA (OpenSSF), ensures that the deployed binary does indeed correspond to the audited code. Finally, the integration of these controls into the pipeline itself is the subject of specific recommendations from NIST (NIST SP 800-204D, 2024), which complement the more general framework of the Secure Software Development Framework (NIST SP 800-218). These measures extend the security requirements defined at the design stage throughout the production chain.

### 5.6.6 Security of code generated by artificial intelligence

The rise of code-generation assistants, analysed in the previous sections as a major driver of productivity, introduces a category of risks that has so far received little attention in the French-language literature (Fu et al., 2024). Three areas of concern are worth highlighting.

Firstly, models may suggest non-existent or misdirected dependencies: a package name 'hallucinated' by the model may be registered by an attacker, opening up a supply chain attack vector. Secondly, the generated code frequently replicates vulnerable patterns present in its training data, without the developer realising the flaw. Finally, the practice of submitting code or configuration elements to an external assistant exposes organisations to leaks of sensitive data.

These risks do not invalidate the use of generative AI, but they do require it to be properly managed: systematic human review of suggestions, subjecting generated code to the same SAST and SCA checks as manually written code, and clear policies governing the use of assistants. The security of AI-generated code is therefore a natural extension of the DevSecOps approach rather than a separate issue (Fu et al., 2024; Hou et al., 2024).

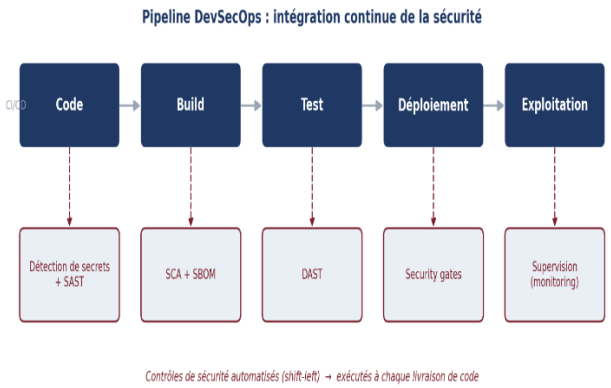



**Figure 15:** DevSecOps pipeline and security checkpoints

## 6. Research methodology

The study adopts a descriptive, analytical and prospective approach, based on a recent literature review and a qualitative comparison of methods and tools.

Sources used

- Scientific publications and reference works;
- Industry reports;
- Case studies;
- Technical documentation from major platforms.


**Figure 16:** Research methodology

Analytical techniques

- Comparative analysis;
- Thematic analysis;
- Triangulation between academic literature and industry feedback.

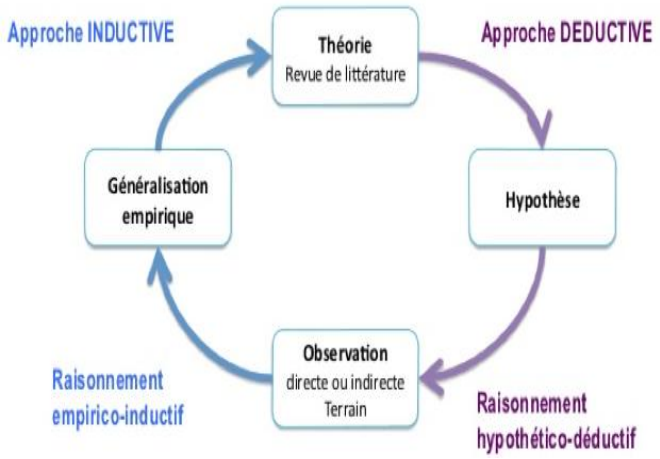



**Figure 17:** Approach

### 6.1 Empirical component: survey on the maturity of practices in the DRC

In order to go beyond a purely documentary analysis and ground the study in the Congolese context, this study is supplemented by an exploratory survey conducted amongst software development and information systems professionals working in the Democratic Republic of the Congo. The aim is to measure the degree of adoption of modern methods (agile approaches, DevOps, microservices, application security) and to identify barriers specific to the local context.

The survey is based on a self-administered questionnaire, distributed online, structured into five thematic sections: organisational profile, development methods, tools and DevOps, security practices, and perceived obstacles. The majority of items are measured on a five-point Likert scale, supplemented by a few closed-ended questions and one open-ended question. The sample, which is non-probabilistic (convenience sample), was drawn from the authors' professional networks and aims to include between thirty and fifty respondents. Given this sampling method, the results should be interpreted as exploratory trends rather than as statistically representative measures of the sector as a whole. Data analysis utilises descriptive statistics (adoption rates by practice, averages) and some simple cross-tabulations, notably between the observed level of maturity and the size of the organisation. Participation is anonymous and subject to the respondents' prior consent.

## 7. Results and analysis

The results point to the dominance of agile approaches in new projects, the growing use of microservices for highly scalable platforms, and the rapid adoption of AI assistants in development and testing activities. Low-code/no-code platforms are particularly effective at accelerating prototyping and low-complexity business applications.

### 7.1 Estimated impact of modern practices on the delivery cycle

**Table 4 —** Estimated impact on the delivery cycle

| Practice | Typical observed impact | Conditions for success |
|---|---|---|
| CI/CD (DevOps) | Reduced delivery time, increased deployment frequency | Automated testing, environment management |
| Microservices | Improved scalability and independent deployment | Observability, API governance, SRE |
| Generative AI | Increased productivity in coding, testing and documentation | Human Review, Safety and Quality Policies |
| Low-code / no-code | Faster prototyping and simple applications | Governance, known customisation limitations |

### 7.2 Assessment of modern practices against the ISO/IEC 25010:2023 quality model

To assess the impact of modern methods on software quality, this section uses the product quality model from the ISO/IEC 25010 standard, in its second revised edition of 2023, as an analytical framework. This revision has updated the model, which now comprises nine characteristics instead of eight: 'safety' has been added as a separate characteristic, whilst 'usability' and 'portability' have been renamed *'interactivity'* and *'flexibility'* respectively, with the latter incorporating *'scalability'* as a sub-characteristic. The analysis compares the five families of practices examined in this article (agile approaches, DevOps/CI-CD, microservices architecture, generative AI, low-code/no-code platforms) against these nine characteristics, in order to identify favourable effects, areas for caution and contextual effects.

Key: (+) generally favourable effect; (−) area for concern or unfavourable effect; (≈) contextual, neutral or ambivalent effect.

**Table 5 —** Summary of the assessment of modern practices against the ISO/IEC 25010:2023 quality model

| Characteristic (ISO/IEC 25010:2023) | Agile | DevOps / CI-CD | Microservices | Generative AI | Low-code / No-code |
|---|---|---|---|---|---|
| Functional fit | + | ≈ | ≈ | ≈ | + |
| Performance efficiency | ≈ | + | + | + | ≈ |
| Compatibility | ≈ | + | + | ≈ | ≈ |
| Ability to interact | + | ≈ | ≈ | ≈ | + |
| Reliability | + | + | ≈ | - | ≈ |
| Safety | ≈ | + | - | - | - |
| Maintainability | + | ≈ | ≈ | ≈ | - |
| Flexibility | + | + | + | ≈ | - |
| Safety | ≈ | ≈ | ≈ | - | ≈ |

Reading this matrix prompts several comments.

- **Agile approaches** primarily improve functional fit and maintainability: short feedback loops align the product with actual needs, and incremental development encourages continuous refactoring (Forsgren et al., 2018). Their impact on security remains context-dependent, as delivery pressures may cause security considerations to take a back seat.

- **DevOps and continuous integration** present the most consistent profile: the automation of testing and deployments enhances reliability and maintainability, containerisation improves compatibility and flexibility, and the integration of security controls (DevSecOps, see section 5.6) makes it a powerful driver of security (Bildirici & Codal, 2023; Kim et al., 2023). This is the practice that delivers the most robust quality gains.

- **The microservices architecture** significantly promotes efficiency, compatibility and flexibility through the independent deployment and scalability of services (Fowler & Lewis, 2024). However, its impact on reliability is mixed — fault isolation is offset by the complexity of the distributed system — and security remains a key concern due to the increased attack surface (NIST SP 800-204).

- **Generative AI** maximises development efficiency, but shifts the focus of concern to reliability, security and safety: untested code, vulnerable designs and increased risks in critical contexts when suggestions are not reviewed (Fu et al., 2024; Hou et al., 2024). Its contribution to quality therefore depends heavily on the human and automated controls put in place.

- **Low-code/no-code platforms** accelerate the delivery of standard requirements and offer ready-to-use interfaces, enhancing functional suitability and usability. However, they have a negative impact on maintainability, flexibility and security, due to vendor lock-in and limited control over the underlying code.

Ultimately, no single practice uniformly improves all quality characteristics: each involves trade-offs. DevOps stands out for its balanced profile of benefits, whilst generative AI and microservices, despite their benefits, shift the risk towards security and reliability. This finding reinforces the need for a controlled adoption rather than an indiscriminate roll-out of modern tools.

### 7.3 Exploratory overview of practices in the DRC (n = 11)

To illustrate the Congolese context, a preliminary exploratory study was conducted in accordance with the protocol described in section 6. Eleven professionals responded, the majority of whom work in start-ups and small organisations (seven teams of one to three people) that are relatively new (all organisations have been developing software for less than five years). Given this small sample size, the results are presented in absolute terms and should be interpreted as indicative trends that cannot be generalised to the sector as a whole; they aim to identify avenues that further research, based on a larger sample, may consolidate.

*Adoption of practices.* Taking into account respondents who stated they ‘somewhat’ or ‘strongly’ agreed (a score of 4 or 5 out of 5), the most widely adopted practices are cloud hosting, version control and security by design (eight out of eleven respondents for each). Conversely, the automation practices

that place the greatest demands on infrastructure appear to be the least widespread: only two respondents report having a CI/CD pipeline, and just one uses containerisation. Aggregated by domain, the average scores place application security (3.7 out of 5) slightly ahead of DevOps tools (3.4) and agile methods (3.2); this ranking, which may be explained by the composition of the sample, remains to be confirmed.

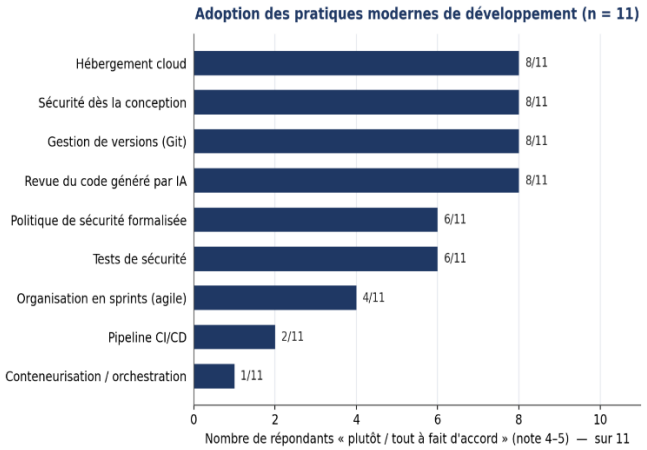



**Figure 18:** Adoption of modern development practices (n = 11)

*Perceived barriers.* Limited budget is by far the most commonly cited obstacle (seven out of eleven respondents), ahead of connectivity and infrastructure (three respondents); the other barriers — skills, governance and resistance to change — were each mentioned only once.

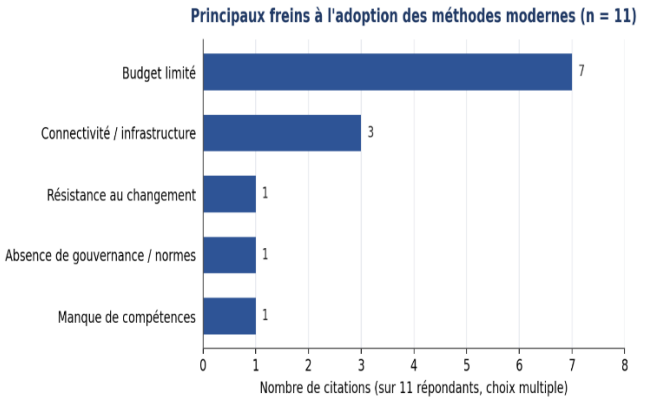



**Figure 19:** Main barriers to the adoption of modern methods (n = 11)

The open-ended responses reinforce this finding and add a dimension specific to the local context. Several respondents attributed progress to a question of resources — one of them felt that 'a good budget and a well-equipped team are needed to achieve the objectives' — whilst others pointed to restrictions on access to digital services resulting from regional regulation. Finally, security was spontaneously cited as an area requiring strengthening.

*Summary*. This initial overview suggests a sector where fundamental practices and security awareness are relatively well established, but where the adoption of advanced engineering practices remains hampered less by a lack of knowledge than by material and budgetary constraints. These observations are consistent with the analysis of the challenges specific to the DRC set out in section 9.5.

## 8. Discussion

The adoption of emerging technologies often improves the speed of development and the scalability of systems. However, the benefits are not automatic. Three points emerge from recent literature

1. ***DevOps and cloud-native***: the most robust gains are seen when automation (CI/CD, IaC, observability) is coupled with organisational changes, not merely the addition of tools;

2. ***Microservices***: these promote scalability and resilience, but increase the need for monitoring, cross-service security and distributed data management;

3. ***Generative Artificial Intelligence***: individual productivity may improve, but quality depends heavily on safeguards (code reviews, testing, dependency management, compliance).

4. **Security**: the benefits of modern methods only materialise if security is embedded throughout the entire cycle (DevSecOps) rather than added at the end; in particular, microservices architecture and AI-generated code shift the risk towards security and the supply chain.

In resource-constrained environments, the priority should often be to standardise practices (Git, CI/CD, automated testing, clear architecture) before adopting highly complex distributed architectures.

This finding is consistent with the exploratory review conducted in the DRC (section 7.3), where the adoption of advanced practices appears to be held back more by budgetary and infrastructure constraints than by a lack of awareness.

## 9. Current challenges facing modern software analysis and design methods

The rapid evolution of digital technologies has profoundly transformed software engineering practices. However, despite the advances brought about by agile methods, microservices architectures, DevOps, cloud computing and artificial intelligence, several challenges remain and directly influence the success of IT projects. These challenges can be grouped into four main categories: technical, human, economic and organisational.

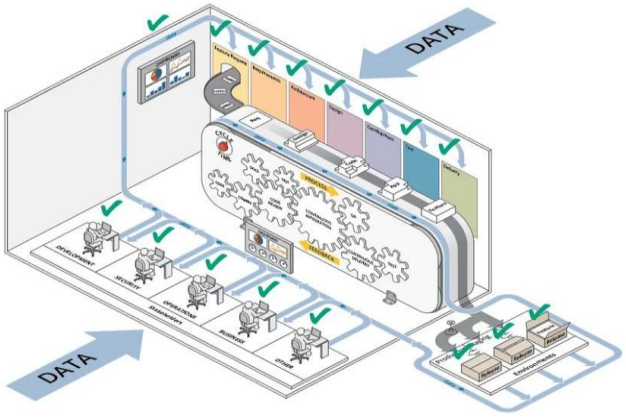



**Figure 20:** Current challenges

### 9.1 Technical challenges

#### a) Increasing complexity of IT systems

Modern applications have become extremely complex. They often incorporate multiple technologies, databases, web services, APIs and cloud infrastructures.

This complexity leads to:

- Design difficulties;
- An increased risk of errors;
- More costly maintenance;
- Greater difficulty in managing dependencies between components.

For example, a modern e-commerce application may consist of dozens of microservices communicating constantly via the internet.

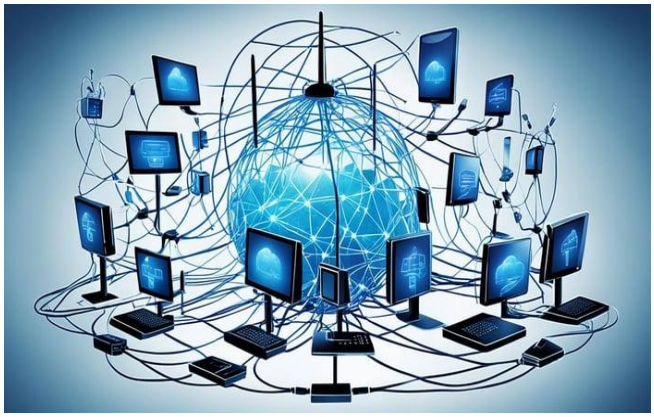


**Figure 21:** IT systems – Increasing complexity

#### b) Cybersecurity

Security is now one of the greatest challenges facing software engineering. The main threats are:

- Hacking;
- Ransomware attacks;
- Data theft;
- SQL injection;
- DDoS attacks;
- API vulnerabilities.

Designers must now incorporate security from the earliest stages of development**,** in line with the **'Security by Design'** approach **(**see section 5.6).


**Figure 22:** Security challenges

**c) Technical debt**

Technical debt refers to the accumulation of quick-fix or temporary solutions adopted to speed up development.

Its consequences are:

- Difficulty in maintenance;
- Increased future costs;
- A decline in software quality;
- Increased risk of failure.

Companies sometimes spend up to 40 per cent of their IT budgets on managing this debt.


**Figure 23: Accumulation of technical debt**

**d) Interoperability**

Organisations often use systems from different suppliers. Interoperability aims to enable:

- Data exchange;
- Communication between applications;
- The integration of existing systems.

The lack of common standards is a major obstacle to this integration.


**Figure 24:** Levels of interoperability

**e) Big Data management**

The explosion in data volume presents new challenges:

- Massive storage;
- Real-time processing;
- Data security;
- Data governance;
- Infrastructure performance.

Traditional architectures are often insufficient to handle these large volumes effectively.

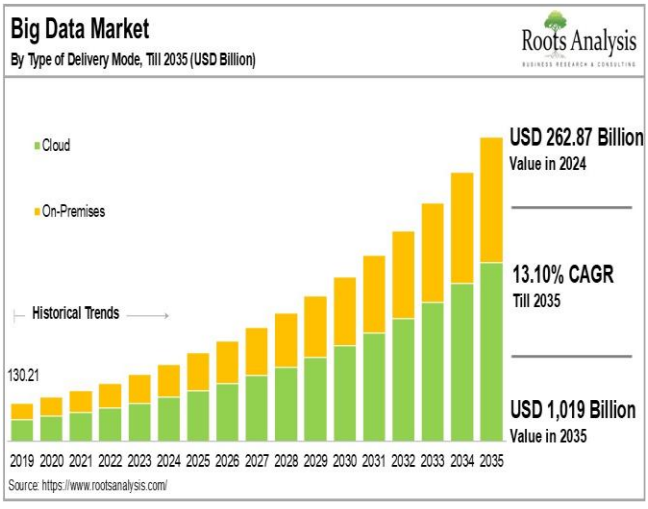



**Figure 25:** Challenges posed by the explosion in data volumes

### 9.2 Human challenges

**a) Resistance to change**

The adoption of new methods or technologies frequently meets with resistance from users and technical teams. The causes include:

- Fear of the unknown;
- Work habits;
- Fear of job loss;
- Difficulty adapting.

This resistance can significantly slow down digital transformation projects.

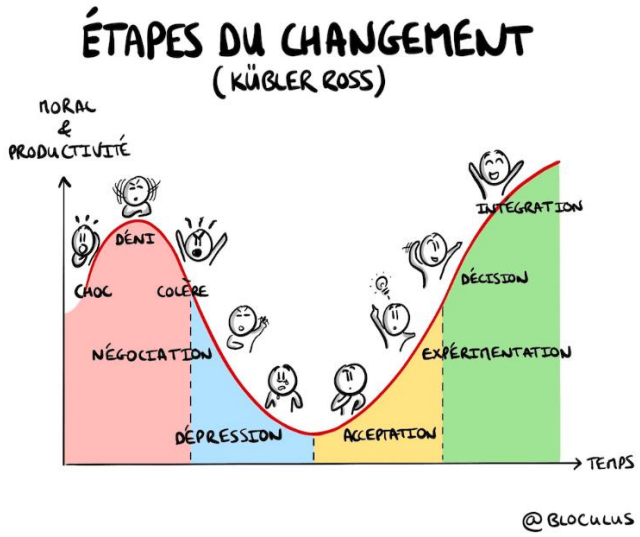



**Figure 26:** Challenges relating to resistance to change

**b) Shortage of specialist skills**

The global market is facing a significant shortage of specialists in several fields:

- Artificial Intelligence;
- Cybersecurity;
- Data Science;
- Cloud computing;
- DevOps;
- Software Architecture.

This shortage is driving up recruitment costs and slowing down innovation.

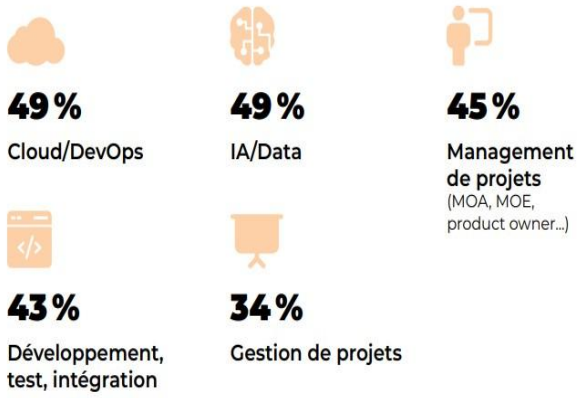



**Figure 27:** Global challenges in specialist skills

**c) Continuing professional development**

Technologies are evolving rapidly. Professionals must constantly update their knowledge in order to master:

- New programming languages;
- New platforms;
- New methods;
- New development tools.

Continuing professional development is therefore becoming a strategic necessity.


**Figure 28:** Continuing professional development

**d) Multidisciplinary collaboration**

Modern projects involve:

- Developers;
- Analysts;
- Architects;
- Business experts;
- Systems administrators;
- Data scientists.

Coordination between these different roles is an ongoing challenge.


**Figure 29:** Coordination of multidisciplinary collaboration

**9.3 Economic challenges**

**a) Cost of cloud infrastructure**

The cloud offers numerous advantages but also entails significant expenditure:

- Hosting;
- Storage;
- Bandwidth;
- Managed services;
- Security.

Poor resource management can lead to a rapid increase in operating costs.

**b) Software licence procurement**

Some business solutions require:

- Annual licences;
- Subscriptions;
- Maintenance costs.

These costs can be a barrier for small businesses.

**c) Return on investment (ROI)**

Assessing return on investment remains complex. Organisations need to measure:

- Productivity gains;
- Cost reductions;
- Quality improvements;
- User satisfaction.

The lack of reliable indicators can make it difficult to justify investment in technology.

**d) Funding for innovation**

Research and development require significant financial resources. Organisations must invest in:

- Laboratories;
- Training;
- Prototypes;
- Technological trials.


**Figure 30:** Economic challenges

### 9.4 Organisational challenges

#### a) Governance of information systems

Companies must establish clear rules regarding:

- Data management;
- Responsibilities;
- Development standards;
- Validation processes.

Inadequate governance increases operational risks.

#### b) Management of distributed projects

With remote working and international teams, projects are often spread across several geographical locations.

The main challenges are:

- Communication;
- Coordination;
- Time management;
- Cultural differences.

#### c) Regulatory compliance

Software must comply with various regulations:

- Data protection;
- Cybersecurity;
- Industry standards;
- Government requirements.

Failure to comply with these obligations may result in significant penalties.


**Figure 31:** Organisational challenges

### 9.5 Challenges specific to the Democratic Republic of the Congo (DRC)

The DRC faces several specific constraints in the field of software engineering:

#### a) Limited internet connectivity

- Poor network coverage in certain regions;
- Insufficient internet bandwidth;
- High cost of internet access.

#### b) Shortage of specialists

The number of qualified experts remains limited in several advanced digital fields.

#### c) Inadequate technological infrastructure

- Limited data centres;
- IT equipment that is sometimes obsolete;
- Low uptake of local cloud services.

**d) Low levels of research funding**

Universities and research centres often have limited resources to:

- Carry out innovative projects;
- Acquire modern equipment;
- Participate in international programmes.

**e) Digital transformation still in its early stages**

Many government departments and businesses are still in the process of transitioning to modern digital systems.

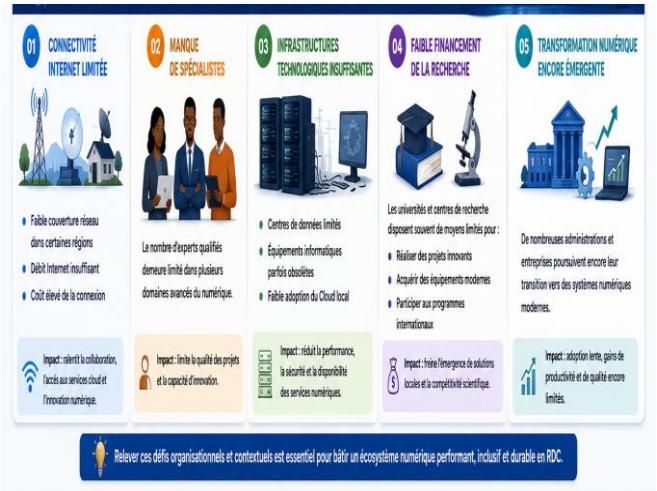



**Figure 32:** Specific challenges in the Democratic Republic of the Congo

**10. Future opportunities**

**10.1 AI-assisted software engineering**

Intelligent assistants are expected to become permanent collaborators with developers for code generation, testing, documentation and requirements analysis.

**10.2 Partially autonomous development**

Development pipelines capable of automatically generating, testing and deploying certain classes of applications are already emerging, under enhanced human supervision.

**10.3 Quantum computing**

Quantum computing could open up new possibilities for optimisation, simulation and certain types of big data processing, although its practical impact on everyday software engineering remains limited in the short term.

**10.4 Sustainable software engineering**

The energy consumption of digital infrastructure is becoming a key architectural consideration. Future systems will need to incorporate environmental performance and energy efficiency objectives.

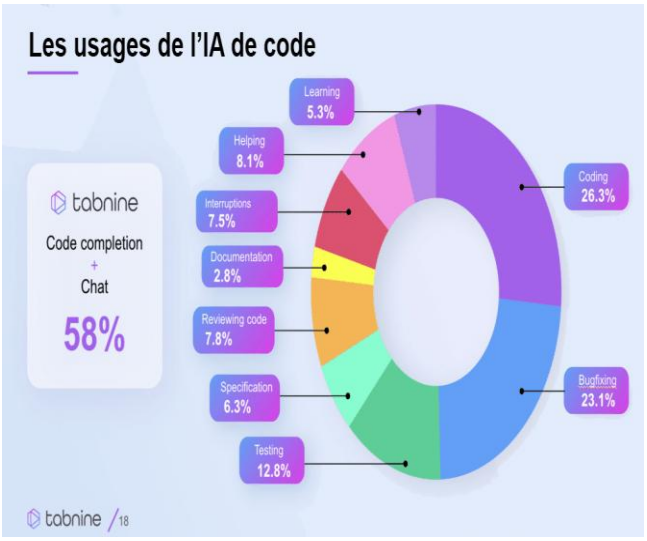



**Figure 33:** Artificial Intelligence in a future projection of software development

**10.5 Smart Cities and Massive IoT**

Smart cities will require architectures capable of integrating millions of connected devices, with stringent requirements in terms of security, availability and real-time processing.


**Figure 34:** Smart Cities and Massive IoT

## 11. Recommendations

— **For universities**

- Modernise curricula (Agile, DevOps, cloud-native, security, AI);
- Strengthen practical laboratories and industrial projects;
- Develop skills in distributed architecture and observability.

— **For businesses**

- Invest in continuing professional development;
- Gradually adopt CI/CD, automated testing and IAC;
- Start with pilot projects before undertaking large-scale microservices migrations;
- Regulate the use of generative AI through quality and security policies.

— **For public authorities**

- Improve digital infrastructure and connectivity;
- Support applied research in software engineering;
- Foster university–industry partnerships;
- Develop governance frameworks for AI and data.

## 12. General conclusion

Software analysis and design methods remain the cornerstone of modern systems development, but their implementation is being profoundly reshaped by Agile, DevOps, the cloud, microservices and generative AI. Sequential approaches still have a place in certain highly regulated contexts, whilst organisations facing changing requirements favour iterative methods and automated pipelines. Beyond this overview, the article offers an analysis of the impact of these practices on software quality in light of the ISO/IEC 25010:2023 standard, an explicit integration of security throughout the lifecycle (DevSecOps), as well as an initial field overview from the DRC.

Generative AI represents a significant game-changer: it accelerates coding, testing and documentation, but does not replace software architecture, business validation, or security and compliance responsibilities. Productivity gains only become sustainable when integrated into a controlled engineering pipeline: code reviews, automated testing, observability and dependency governance.

For developing countries and the DRC, the strategic priority lies less in immediately adopting the most sophisticated architectures than in strengthening the fundamentals: connectivity, training, version management, CI/CD automation, software quality and security. A gradual approach — standardising practices, building capabilities, then targeted architectural modernisation — generally offers a better cost-risk ratio than a sudden transformation. Future research would benefit from consolidating the field insights presented here across a broader sample and from exploring the security dimension in greater depth within these resource-constrained contexts.

Looking ahead, software engineering will evolve towards increasingly AI-assisted development environments, highly automated pipelines and massively observable distributed systems. Organisations capable of combining technological innovation, governance and skills development will enjoy a sustainable competitive advantage in the digital economy of the coming years.

## References


1. ACM Computing Surveys. (2024). *Artificial Intelligence in Software Development: Trends and Challenges.*
2. Amazon Web Services (AWS). (2024). *Well-Architected Framework.*
3. Bhatt, A. D. (2024). DevOps and Continuous Delivery: Enhancing Agility in Software Engineering. *International Journal of Advanced Research in Computer Science & Technology*, *7*(6).
4. Bildirici, F., & Codal, K. S. (2023). *DevOps and Agile Methods Integrated Software Configuration Management Experience.* arXiv.
5. CNCF. (2024). *Cloud Native Survey Report 2024.*
6. Escalona, M. J. (2023). Advanced Healthcare AI and Machine Learning with Agile DevOps, Blockchain Security and Cloud-Native Automation. *IJRPETM*, *6*(5), 9362–9371.

7. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps.* IT Revolution Press.

8. Fowler, M. (2023). *Refactoring: Improving the Design of Existing Code* (3rd ed.). Addison-Wesley.

9. Fowler, M., & Lewis, J. (2024). *Microservices Patterns and Best Practices.* Addison-Wesley.

10. Fu, M., Pasuksmit, J., & Tantithamthavorn, C. (2024). *AI for DevSecOps: A Landscape and Future Opportunities*. arXiv.

11. Future Generation Computer Systems. (2024). *ChatOps for Microservice Systems: A Low-Code Approach Using Service Composition and Large Language Models.*

12. Gartner. (2024). *Top Strategic Technology Trends for 2024.*

13. Google Cloud. (2024). *State of DevOps Report 2024.*

14. Grande, R., Vizcaíno, A., & García, F. O. (2023). Is it Worth Adopting DevOps Practices in Global Software Engineering? Possible Challenges and Benefits. *Computer Standards & Interfaces, 88,* 103767.

15. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology, 33*(8). https://doi.org/10.1145/3695988

16. IBM Research. (2024). *Artificial Intelligence for Software Development and Maintenance.*

17. IEEE Computer Society. (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK V4).*

18. IEEE Software Magazine. (2023). *Special Issue on AI-Assisted Software Engineering.*

19. Information and Software Technology. (2024). *Microservice-Based Projects in the Agile World: A Structured Interview.*

20. ISO/IEC. (2023). *ISO/IEC 25010: Systems and Software Quality Requirements and Evaluation (SQuaRE).*

21. ISO/IEC/IEEE. (2024). *ISO/IEC/IEEE 12207: Software Life Cycle Processes.* International Organisation for Standardisation.

22. Journal of Systems and Software. (2024). *An Empirical Investigation into the Competences and Roles of Practitioners in Microservices-Based Architectures.*

23. Kim, G., Humble, J., Debois, P., & Willis, J. (2023). *The DevOps Handbook* (2nd ed.). IT Revolution Press.

24. Kokol, P. (2024). The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature. *Information, 15*(6), 354. https://doi.org/10.3390/info15060354

25. Martínez-López, J. A., García, F., & Ruiz, F. (2024). Contributions of Enterprise Architecture to Software Engineering: A Systematic Literature Review. *Journal of Software: Evolution and Process, 36*(4), e2572.

26. McKinsey & Company. (2023). *The Economic Potential of Generative AI: The Next Productivity Frontier.*

27. Microsoft Research. (2024). *GitHub Copilot and Developer Productivity Report.*

28. Mohottige, T. I., Polyvyanyy, A., Buyya, R., Fidge, C., & Barros, A. (2024). *Microservices-Based Software Systems Reengineering: State-of-the-Art and Future Directions.* arXiv.

29. NIST. (2019). *Security Strategies for Microservices-based Application Systems* (SP 800-204). National Institute of Standards and Technology.

30. NIST. (2022). *Secure Software Development Framework (SSDF) Version 1.1* (SP 800-218). National Institute of Standards and Technology.

31. NIST. (2024). *Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines* (SP 800-204D). National Institute of Standards and Technology.

32. OpenAI. (2023). *GPT-4 Technical Report.*

33. OpenSSF. (n.d.). *Supply-chain Levels for Software Artifacts (SLSA).* Open Source Security Foundation.

34. OWASP Foundation. (n.d.). *OWASP Top 10 & OWASP SAMM (Software Assurance Maturity Model).*

35. Pressman, R. S., & Maxim, B. R. (2024). *Software Engineering: A Practitioner's Approach* (10th ed.). McGraw-Hill.

36. Red Hat. (2024). *The State of Kubernetes and Cloud-Native Development.*

37. Sauvola, J., Tarkoma, S., Klemettinen, M., Riekki, J., & Doermann, D. (2024). The Future of Software Development with Generative AI. *Automated Software Engineering*, *31*(26). https://doi.org/10.1007/s10515-024-00426-z

38. Sommerville, I. (2023). *Software Engineering* (11th ed.). Pearson Education.

39. Söylemez, M., Tekinerdogan, B., & Tarhan, A. K. (2024). Microservice Reference Architecture Design: A Multi-Case Study. *Software: Practice and Experience*, *54*(1), 58–84.