
by Dmytro Nikulin, Lead Software Engineer;
reviewed by Orkhan Gasimov, Technology Director

Detecting Architectural Gaps with Automation

Today's rapidly changing environment requires agile, adaptable software architecture that can evolve and support the success of the business in a high-tech world.

Software architecture is the technical foundation of product development and as such, its readiness for change is a key business consideration. Uncontrolled architecture drift and erosions may lead to product failure, even when it is already in a production state or has been live and successful for a while.

For example, architecture drift can lead to the loss of initially expected extensibility and new customers may fail to use a product if the number of users increases. Such drifts might be identified during an architecture review.

Typically, this review is a complex process. It takes a great deal of time for experienced architects to assess the software architecture using largely manual processes. Manual checks may not be as efficient, and many architectural issues might not be discovered due to the influence of human factors.

In this paper, we'll examine methods for detecting architecture drift and erosion, the business impacts of it, and potential automated solutions with use case scenarios.

Business Impacts of Architecture Drift and Erosion

Architecture drift or erosion can prevent software from meeting its product and architectural requirements, which could negatively affect the product's success.

For example, the product may end up at risk of failure to operate. Or, regulations may disallow the sale of the product altogether.

The product may also have performance issues that impact its success in the marketplace and prevent adoption.

Architecture Drift and Erosion Detection

Architectural drift refers to the phenomenon where a software system gradually deviates from its intended architectural design over time.

This drift can occur due to various factors such as ad hoc changes, evolving requirements, lack of documentation, development team turnover, or a lack of

adherence to architectural guidelines. An example of drift in architecture may include microservices with an API that does not adhere to the initial software architecture due to issues with implementation.

Architecture erosion, on the other hand, is the process by which the architecture of a software system degrades or deteriorates over time.

It typically occurs due to incremental changes, patches, bug fixes, and enhancements made to the system without considering the long-term architectural impact.

Architecture drift and erosion can lead to a number of problems within a software system.

These include reduced maintainability, increased complexity, decreased performance, decreased reliability, and difficulties in extending or modifying the system in the future.

Software architecture reviews can prevent these issues, stopping erosion and drift into areas not suggested by best practices and architecture governance.

Here are a few common types of manual architecture review:

Dependency analysis

Analyzing the dependencies between software components can help identify instances where components violate the intended architectural boundaries. Tools can examine dependencies at the code level, module level, or even at the architectural level.

Metrics analysis

Utilizing software metrics, such as [cyclomatic complexity](#), coupling, cohesion, and code duplication can provide insights into the system architecture's overall health. Deviations from acceptable thresholds or changes in metric values over time can indicate potential drift or erosion.

Change impact analysis

Whenever a change is introduced to the system, conducting a change impact analysis can help understand its implications on the architecture. This analysis assesses how the change affects different architectural components and whether it aligns with the architectural principles and constraints.

Peer reviews and code inspections

Regular code reviews and inspections involving architects and experienced developers can help spot architectural issues, drift, or erosion. A fresh set of eyes can identify inconsistencies and suggest corrective actions.

Recommended Reading:



Continuous Testing: How to Measure & Improve Code Quality

The Challenges of Manual Software Architecture Review

Proactive architecture reviews help to identify and prevent architecture drift or erosion. The main goals of such reviews are:

- Validating the architecture's capabilities to support current and future business goals.
- Checking the architecture's ability to meet non-functional requirements.
- Detecting design mistakes as early as possible.
- Identifying potential technical risks to the project.

Even if the evolving architecture is reviewed periodically, implementation changes might still go in a direction not expected by the initial software architecture.

Manual architecture review, while beneficial in detecting software architecture drift and erosion, can come with several challenges, including:

Subjectivity

Manual architecture review heavily relies on the expertise and judgment of the reviewers. Different reviewers may have varying perspectives, experiences, and interpretations of architectural guidelines, making the assessment subjective.

This subjectivity can lead to inconsistencies in identifying and addressing architecture drift and erosion.

Time and resource-intensive

Manual architecture review can be a time-consuming and resource-intensive process, especially for large and complex software systems.

Reviewers must invest significant effort in understanding the system's architecture, studying relevant documentation, and analyzing the codebase. As a result, the review process may take a considerable amount of time and may not be feasible for frequent or continuous monitoring.

Limited scope

Reviewers typically have limited visibility into the entire system's codebase and runtime behavior. They may only be able to review a subset of the system or focus on specific components or modules.

This limited scope can lead to potential blind spots, where architecture drift and erosion may occur in unreviewed areas.

Lack of complete documentation

Manual architecture review heavily relies on accurate and up-to-date documentation to understand the intended architecture.

However, software systems often lack comprehensive documentation, especially when they have evolved over time or when documentation maintenance is neglected.

In such cases, reviewers may face challenges in assessing deviations from the original architectural design.

Human error and oversight

Reviewers, like any human, can make errors or overlook certain aspects during the review process.

They may miss subtle signs of architecture drift or erosion, fail to consider all relevant factors, or misinterpret certain code constructs.

These human errors can result in false negatives or false positives in identifying architectural issues.

Scalability and consistency

Manual architecture review becomes increasingly challenging as the size and complexity of the software system grow. Reviewers need to ensure consistency in their assessments across different components and versions of the system.

Maintaining a consistent review process and applying architectural guidelines uniformly can be difficult, especially in organizations with multiple development teams or distributed projects.

To overcome these challenges, organizations often combine manual architecture review with automated analysis techniques and tools that provide objective assessments, perform code analysis at scale, and identify potential drift and erosion patterns.

Automated analysis is used to analyze source code, binaries, and runtime behavior, to identify inconsistencies and deviations from the intended architecture. These tools often compare the current state of the system against the architectural models or specifications.

[Continuous integration and testing](#) incorporates architectural checks and tests into the continuous integration and deployment pipeline to help identify violations or deviations early on. Automated tests can assess system adherence to the expected constraints after each change.

An automated approach helps mitigate the challenges of manual review and enhances architecture drift and erosion prevention efforts efficacy.

Important Considerations for Evaluating Automated Architecture Review Solutions

Architecture is a broad area, and the proposed solution might cover various levels and aspects within those levels to verify the architecture and identify possible gaps.

These important considerations may be found in each of the following levels:

- Compliance
- Architecture
- Data
- Security
- DevOps
- Scalability
- Quality

Let's take a look at each area and examine the business value of potential checks for architectural aspects of the general verification process.

This is not limited to the described scope and might be significantly expanded based on real product needs.

Compliance level

Compliance level checks show how the architecture meets regulatory requirements. If regulatory requirements are not met, it will not be possible to sell the product at all. These checks might include:

- Data privacy standards (such as PII, GDPR, etc.)
- Data sovereignty checks

Architecture level

Architecture level checks show how the architecture covers architecture requirements. If not met, production can go into a risk state up to failing to operate. These checks may include:

- Logical and component structure
- CDN requirements
- Workflow/integration architecture
- Data/BigData architecture
- SOA/Microservices architecture
- API structure rules

- Data access rules
- Traceability compliance
- OLAP/OLTP processing
- Transaction/concurrency model rules
- Software configuration (secret and non-secret)
- Different user access roles checks
- Multi-tenancy structure rules
- Scaling and disaster recovery models

Data level

Data-level checks show how well the architecture manages the data. If a check fails, issues such as data inconsistency or data duplications can affect the quality of the data or cause performance issues. These checks include:

- Governance rules
- Data Loss Prevention checks
- Data use/reuse policy
- Single source of truth policy
- Archive and purge policy
- Extract/Transform/Load rules
- Logging and indexing rules

Security level

Security-level checks show how well the architecture is secured. If a check fails, the resulting privacy data leakages can cause reputational risks. These checks include:

- Identity and Access management rules
- Key management policy
- API/Endpoints security rules
- Encryption/hashing policy
- DB encryption policy

DevOps level

DevOps checks assess the architecture's readiness to go to market. Failure can delay release or impact product quality. These checks include:

- Environment topology/rules
- Cluster and Containers structure rules
- Services discovering policy
- Scaling code approach to verify readiness to scale
- Metrics/Monitoring policy
- Time zone basis and updates to verify that solution supports work in different time zones and supports update of time zones

Use Case Scenarios

Extract software architecture from code to assess architecture

Architectural assessments are a common part of the advisory phase. Architecture information is extracted from documents, code, configuration files, and other sources, then assessed to identify gaps and improve the existing architecture. Extraction may occur when the current solution is facing issues, such as poor performance or inability to scale to handle a growing number of users. In such cases, the solution is run on demand.

Detect software architecture drift and erosion on regular basis

The chance of uncontrolled drift or erosion is high during construction and initial implementation, typically due to the speed and volume of changes being made. Regularly detecting architecture drift at the initial stage and later is strongly recommended to prevent deviation from the target architecture.

This can be implemented as one of the steps in the CI/CD (SCA) process, an essential part of every fast-growing product. CI/CD processes connect the development and production environments and help deliver the product with frequent updates in smaller chunks, thereby improving time-to-market.

Testing new software architecture

It is not uncommon to find demand to update the architecture to support new requirements. An example of this might be a request to scale a solution to onboard many more new customers within an application.

In this case, the architecture is changed by a business request. Usually, for such cases, architecture changes are developed separately from the main workstream, and the changes are verified on a separate stream before being pushed to the main one.

Conclusion


Architecture management is essential, and it's important to understand the challenges associated with manual reviews.

Analyze the use case scenarios herein and assess their relevance to your own software projects. Explore ways to enhance your own practices to prevent architecture drift and erosion.

You might consider implementing similar approaches, such as extracting software architecture from code, regularly detecting architecture drift, or testing new software architecture in a separate stream.

Those interested in exploring automated analysis techniques and tools to enhance architecture review processes can research and evaluate the solutions available. Pay special attention to how these tools address the challenges we've explored here.

Still have questions? [See how GlobalLogic's Architecture Practice helps businesses like yours](#) modernize legacy systems and transform to meet the demands of a digital future.



CHOOSE TO
DELIGHT YOUR CUSTOMERS
SURPRISE COMPETITORS
TRANSFORM YOUR BUSINESS
BE EXCEPTIONAL

GlobalLogic
A Hitachi Group Company