Practical Application of New Technologies

POC (Proof of Concept)

Automatic Release System (Semantic Release)
- https://github.com/LuxoriaSoft/auto-release

This solution is used in our Luxoria solution, repo here : https://github.com/LuxoriaSoft/Luxoria

Luxoria’s auto-release workflow uses the semantic-release engine in a GitHub Action. Every push to develop triggers a PRERELASE, when the same commits reach main, semantic-release converts that tag into the corresponding final version and publishes the full GitHub Release. The approach mirrors the “official” pattern promoted across many open-source projects, and is explicitly called out by AWS as a best practice for CI/CD pipelines and by npm for trustworthy package distribution.

Pros :
- Zero-click versioning : Eliminates manual bumping and tagging; every merge ships with the right SemVer
- Predictable prereleases : Stakeholders can try develop snapshots without destabilising production
- Instant changelogs : Release notes are generated from commit history, keeping docs in sync with code

Pre-Alpha for LuxStudio
- https://github.com/LuxoriaSoft/Luxoria.WebApp-PA

POC issued from our Pre-Alpha, based on Next.JS with Mongoose as ORM and MongoDB as Database.
We've chose Next.Js for its SSR feature which advantages us for SEO referencement.
Another advantage it is that the static pages or static sections can be compiled in a way to reduce the latency when a page is requested by the client.
We've compared the performance / advantages of Next.JS in the benchmarks (See them on the Regular Technology Watch part)

Pros :
- SEO-friendly : Fully rendered HTML lets Google and other crawlers index content without requiring headless browsers or deferred hydration tricks.
- Unified rendering choices : Next.js lets us mix SSR, SSG, ISR and client-side rendering per route, so we can fine-tune for latency or freshness without switching frameworks
- Schema-driven data layer : Mongoose adds declarative schemas, validation and hooks, making domain logic clearer and reducing boilerplate.

Cons :
- Cold-start overhead : SSR pages incur server compute on every request; under heavy traffic this can raise costs unless edge caching or ISR is tuned carefully.

- Mongoose abstraction : While convenient, Mongoose adds an extra layer that can limit use of advanced MongoDB features or require verbose work-arounds for edge-case queries.

Luxoria App Pre-Alpha
- https://github.com/LuxoriaSoft/Luxoria.Desktop

POC issued from our Pre-Alpha version, based on Microsoft UWP Framwork, official & recommanded framework until Windows 11 came out.
Provides an modular GUI, able to design as we wish, at the year 2025, this framwork will be deprecied, replacing by the new Microsoft WinUI3 framwork. (Official & Recommended framework at the time of Windows 11)

This POC provides a basic overview of Luxoria, from importing an asset, editing it and publishing using LuxStudio POC.

Pros :
- Native look, zero bloat : UWP delivers Fluent-design controls, hardware-accelerated rendering, and direct WinRT API access without extra runtime layers.

Cons :
- Deprecated framework : While still supported, UWP is no longer the focus of new Windows features; key components are already deprecated, signalling an eventual sunset.

Luxoria App WinUI3 POC
- https://github.com/LuxoriaSoft/winui3-app

Before starting our Luxoria's Alpha version, we needed to compare all possible solution, we wanted an official Microsoft solution, to be closer than Windows.
We've chosen the new official way to create an application on Windows 10/11.
This POC is the creation of a small WinUI3 application.
Based used in our Solution at :
https://github.com/LuxoriaSoft/Luxoria/tree/main/Luxoria.App

UWM will be deprecated on the 14th of October 2025.

Pros :
- Official, forward-looking framework : WinUI 3 is Microsoft’s “primary UI technology for new Windows apps,” decoupled from OS releases and updated via NuGet.
- Long-term security : Building on WinUI 3 avoids the 2025 end-of-life cliff facing Windows 10 and many UWP-based Office/Mobile apps.
- Rich windowing & theming : The new AppWindow API simplifies custom title bars, task-bar icons and multi-monitor layouts—features that were clumsy or impossible in UWP.

Cons :

- Designer friction – Visual Studio’s XAML Designer has limited support for WinUI 3, so iterative UI work relies on live-reload or manual rebuilds.
- Only for Windows, not compatible on MacOSX / Linux

Luxoria Modular Architecture (Modules) + CI
- https://github.com/LuxoriaSoft/modular-arch

This POC has been used to design an software architecture for our solution.
Luxoria’s Modular Architecture + CI proof-of-concept marries an event-driven core with a micro-kernel & plug-in pattern: the product ships a lean host that fires domain events, while every extra capability lives in an external DLL “Module.” Modules subscribe to events through an internal bus, load at runtime via .NET’s AssemblyLoadContext, and publish their own events in turn. This keeps the core tiny, lets third-parties drop in new features without recompiling the platform, and—because the repo travels through a layered GitHub-Actions pipeline—ensures each module is built, tested, versioned before it ever lands in production.

Pros :
- Parallel, community-led innovation : Teams (or the wider community) can develop modules independently, on their own cadence, then drop them into production when ready.
- Consistent DevOps story : A single monorepo but per-module pipelines keep build times short, cache dependencies efficiently, and align with micro-service CI guidance.

Cons :
- Runtime safety risks : Malicious or badly written DLLs can crash the host or leak resources if not sandboxed with reflection-only permissions or process isolation.

CommitLint CI/CD POC
- https://github.com/LuxoriaSoft/commitlint-cicd

This repository shows how Luxoria wires commitlint into every layer of its automation so that only Conventional-Commit-compliant messages reach shared branches. A lightweight Husky commit-msg hook catches problems on the developer’s machine; a GitHub Action re-checks every push or pull-request, guaranteeing that commits created outside local hooks are still validated.

Pros :
- Consistent commit history : Standardised messages make git log, git blame, and release notes far more readable.
- Works with Segmentic Release : Have a standardised way to write the CHANGELOG

Cons :
- Bypass via --no-verify : Local hooks can be skipped; without the CI layer that would undermine enforcement.

Brisque DLL/C++ Implementation

- https://github.com/LuxoriaSoft/brisque_impl
- .NET WRAPPER : https://github.com/LuxoriaSoft/brisque_impl_netlib

Luxoria’s Brisque DLL/C++ Implementation and its .NET wrapper (Luxoria.Algorithm.BrisqueScore) bring the widely-studied BRISQUE no-reference image-quality metric into a form that native Windows apps and managed .NET code can call in a single line, giving every part of the Luxoria tool-chain a fast, local way to grade the perceptual quality of assets before they enter a pipeline.

On BrisqueScore(imagePath), the DLL converts the image to grayscale, builds Mean Subtracted Contrast-Normalised (MSCN) coefficients, extracts 36 natural-scene-statistic features, and feeds them to the LIBSVM model to obtain a score typically between 0 (great) and 100 (poor)

Builds target x86 / x64 / ARM64 artefacts so the same DLL can ship inside desktop, tablet or Surface-Pro X installers.

Currently on board of Luxoria's LuxFilter : https://github.com/LuxoriaSoft/Luxoria/blob/main/Modules/LuxFilter/LuxFilter.Algorithms/Algorithms/ImageQuality/ResolutionAlgo.cs

This implemention has also been released to the Open World source under a Nuget : https://www.nuget.org/packages/Luxoria.Algorithm.BrisqueScore (+2700 downlaods as of 22/06/2025)

Pros :
- True no-reference IQA : BRISQUE needs no “gold” image, making it ideal for user-generated content pipelines.
- Native-speed scoring : Pure C++ with OpenCV SIMD optimisations beats Python/PyPI
- Multi-arch support : Shipping x96, x64 and ARM64 binaries up-front avoids later rebuilds

Scene Dimensionality Analysis
- https://github.com/LuxoriaSoft/scene_dimensionality_analysis
- https://github.com/LuxoriaSoft/scene_dimensionality_analysis/blob/main/scene_dimensionality.ipynb

This POC tests whether the intrinsic dimensionality of real-world images can be estimated quickly enough to drive automatic “scene complexity” decisions inside Luxoria’s pipeline. It loads the Stanford Background dataset (715 outdoor photographs with pixel-wise labels) and runs several intrinsic-dimension estimators—plain PCA eigen-spectrum cuts, correlation-dimension and a recent tangent-space method—then benchmarks how strongly those numeric scores track human judgments of “busy vs. simple” scenes.

Cons :

- Scale-dependent results : Image resolution and colour-space choices shift PCA variance ratios; scores aren't directly comparable across datasets unless normalised.
- Noise sensitivity : Correlation-dimension explodes with JPEG artefacts or sensor noise, so pre-filtering is mandatory.
- Limited theoretical range : For fully textured non-Lambertian scenes, the upper bound is effectively unbounded, so high scores tell us little beyond “complex.”
- Dataset bias : Stanford Background focuses on outdoor scenes; indoor studio shots or synthetic renders might show very different dimensionality curves.

Image Segmentation (FNC / U-NET Model)
- https://github.com/LuxoriaSoft/imageseg

The POC packages two reference architectures—Fully Convolutional Network (FCN-8s) and the canonical U-Net—plus an ONNX-Runtime inference script, all tuned on the MIT ADE20K scene-parsing dataset (150 classes). The goal was to determine whether Luxoria can run fast, per-pixel segmentation offline (desktop) and online (web service) with one shared model file, then pick the better performer as a building-block for future 3-D scene tagging and background-removal features.

Pros :
- Mature architectures : FCN and U-Net still dominate segmentation leaderboards and have thousands of citations, making bugs and tuning tips easy to find.
- Shared ONNX graph : One file serves WinUI-3, ASP.NET Core, and Python micro-services alike, thanks to ONNX Runtime's cross-language API.

Cons :
- Class-imbalance sensitivity : ADE20K's long-tail classes (e.g., “microwave”) see IoU < 0.30 without focal-loss or re-weighting tweaks
- Data-set bias
- Bad performance in term of Accuracy / Precision

Image Scoring (OpenAI CLIP)
- https://github.com/LuxoriaSoft/image_scoring

This POC wraps an OpenAI CLIP encoder in a thin Python/ONNX pipeline so that any service can assign a semantic “relevance” score to an image against one or more text prompts

Pros :
- True zero-shot flexibility : CLIP can rate virtually any concept (“vintage aesthetic”, “contains text”) without retraining, just by changing the prompt set
- Cross-stack reuse : A single ONNX graph serves Python, .NET, and C++ callers via ONNX Runtime bindings, matching Luxoria's multi-platform footprint.

Cons :
- Bias & robustness gaps : CLIP over-weights large, salient objects and inherits data biases; scores can mis-rank diverse or abstract imagery

GrabCut Implementation (using OpenCV)
- https://github.com/LuxoriaSoft/grabcut-impl-cpp-oc410

The POC exposes the OpenCV's GrabCut pipeline modelling of foreground/background colours

Pros :
- High-quality edges with minimal input : A single rectangle often suffices to isolate the subject cleanly
- Offline / CPU-only : No GPU or Internet access is needed, so the same binary runs in build agents, kiosks, or on battery-powered tablets.
- Well-studied & documented : Many tutorials and reviews make debugging or customising the code straightforward

Cons :
- Strong dependence on a good initial guess – Poor rectangles or inaccurate scribbles lead the GMM to lock onto the wrong colours, requiring several manual corrections.

Automatic Bounding Box (Subject Discovery)
- https://github.com/LuxoriaSoft/obb-extractor

This POC wraps modern YOLOv8 medium and large detectors inside a slim Python/ONNX pipeline that can spot the one-or-several “main subjects” of any image and return their bounding boxes (or ready-cropped cut-outs) in real time. By combining YOLO’s high-accuracy multi-class predictions with a size-and-centroid heuristic, the extractor reliably isolates the most salient object(s) for downstream tasks such as auto-zoom, smart thumbnails or dataset labelling.

Pros :
- State-of-the-art accuracy : YOLOv8-l scores >53 mAP on COCO yet still runs <10 ms per frame on a mid-range GPU.
- Real-time on CPU : ONNX optimisation plus optional DeepSparse delivers 20–50 fps without a GPU, making the POC viable for serverless or kiosk deployments.

Cons :
- Dataset bias : COCO’s 80 classes miss niche objects, so rare subjects may be ignored unless the model is fine-tuned.