OPERATIONAL EXCELLENCE NEWSLETTER

When to Stop Treating Every Part Like a Unique Build Job.

Moving from bespoke programming to standardized volumetric envelopes in high-mix environments.

Automated Quality Control · High-Mix Manufacturing · Operational Efficiency · Process Automation

> The newsletter for quality engineers who prefer scalable process architecture over bespoke programming solutions.

The Problem: Building Custom Code for Every Single Part Variation

I’ve spent enough time on production floors to know what a mounting headache looks like. It usually starts with "the next big thing"—a new part, or perhaps just a minor variation of an existing FRU that needs its own unique identifier in the system. Suddenly, the engineering team is buried under a mountain of custom code. They are spending days, sometimes weeks, tweaking robot paths and sensor triggers to accommodate every slight change in geometry, every fillet radius, and every tiny deviation in surface finish for the next part on the line.

This isn't just an "engineering hurdle." It’s a drain on your most valuable resources. When you write unique code for every minor variation, you aren't building a scalable process; you are building a series of bespoke workarounds. You are essentially treating every new piece as if it were a one-off prototype rather than part of a repeatable manufacturing flow.

The result is a brittle system where the "knowledge" required to run the line resides in complex, tangled lines of code that only the person who wrote them understands. If the robot hits a snag or the sensor drifts, you can’t just tweak it; you have to go back into the weeds of the custom logic to figure out why it's failing for *this specific part*. It creates a situation where your automation is actually working against your ability to scale. You are trading long-term stability for short-term "fixes," and eventually, that debt comes due in the form of downtime and frustrated operators who have no idea how to adjust the machine when things go slightly off-script.

What is Actually Happening on Repeat Orders?

In many high-mix environments, there is a gap between what we think our engineers are doing and what they are actually achieving. We call this **The Programming Trap**. It’s easy to look at a complex, custom-coded path as a sign of "high precision," but often, it's just an elaborate way of avoiding the hard work of standardizing requirements.

The Comfortable Rationalization	The Underlying Reality
"We need unique code because every part has a different shape."	We are using complex code to compensate for poorly defined inspection zones.
"Custom paths ensure we hit every critical feature perfectly."	Custom paths create "hidden" complexity that makes the system harder to maintain.
"It takes time, but it’s necessary for high-precision accuracy."	It is an avoidance of standardizing the *volume* in favor of chasing every fillet.
"The software requires specific coordinates for each SKU."	The hardware doesn't care about the shape; it only needs to know where to look.

When we fall into this trap, we are spending our time solving problems that don't actually exist on the floor. If a part is 90% similar to the last one, and the inspection points are in the same general area, there is no reason for the robot to "know" exactly where every curve of the new part sits. The fact that we spend hours programming those details into the path means we aren't trusting our process; we’re trying to out-engineer a lack of standardization.

Why Are We Treating Every Job Like a Prototype Build?

There is a specific brand of "hero" culture in engineering where solving a complex problem—even if it’s the wrong kind of problem—is rewarded. An engineer who spends forty hours perfecting a custom path for a new part might feel like they are doing something high-value. In reality, they are just digging a deeper hole.

We treat every job like a prototype because it is mentally easier to write more code than it is to define simpler requirements. It’s much faster to tell the robot exactly where to go based on a CAD model of a

specific part than it is to sit down and map out a common inspection zone that works for an entire family of products. We are letting "bespoke craftsmanship" creep into our automation logic.

On the shop floor, this manifests as a lack of consistency. When you have dozens of different programs running because every part was treated like its own unique project, your ability to troubleshoot evaporates. If something goes wrong on Tuesday, it might be caused by a sensor issue; if it happens on Wednesday, it’s probably just that the custom code for "Part B" is slightly different than the code for "Part A." You can't find the common cause because you haven't allowed there to be any common process. We need to move away from seeking perfection in the path and start demanding reliability in the system.

The Path Forward: Standardizing the Volume, Not the Part

The solution isn't more sophisticated code; it’s simpler logic. Instead of trying to program a robot or a scanner to "know" every unique contour of a part, we move toward **Standardized Volumetric Envelopes**.

In plain terms, this means using a "bounding box." We define the physical space where a family of parts exists and we tell the system to scan that volume. If you have five different models of a housing, they might all have slightly different dimensions, but they all occupy roughly the same amount of space in the fixture. Rather than programming a path that hugs every curve of each individual model, we define a "zone" or an envelope.

By standardizing the volume, we decouple the inspection logic from the specific geometry of the part. The robot doesn't need to know what it’s looking at; it just needs to know where the area of interest is. This moves us away from "bespoke programming" and toward a "set-it-and-forget-it" approach for entire product families. When you change parts within that family, the inspection logic stays exactly the same because the volume hasn't changed. You aren't just making it easier for the engineers; you are making the system more robust against drift and much easier to maintain on a daily basis.

Four Steps to Implementing Envelope Scanning Logic

Moving to an envelope-based system isn't about jumping straight into complex software; it’s about disciplined planning before any code is written.

1. **Define the Part Family Boundaries:** Stop looking at parts as individuals. Group your products into families based on their shared footprints and inspection requirements. If a group of items shares 80% of its physical footprint, they belong in one "envelope" logic block.
2. **Establish Worst-Case Feature Locations:** Instead of aiming for the center of every feature, identify the outermost boundaries where any possible feature could exist within that family. This creates your "buffer zone." If a feature is slightly larger or shifted on Model B compared to Model A, it still falls inside this pre-defined box.
3. **Program a Simplified Scan Path through the Envelope:** Once you have your bounding boxes, write one path for each envelope. The robot should move at a steady pace through these

defined volumes. You are no longer "hunting" for features; you are scanning a territory. This simplifies the logic and makes it much easier to calibrate.

4. **Validate Coverage against Required Inspection Points:** Finally, run a check. Ensure that every critical inspection point—the ones your quality standards demand—falls within the simplified scan path. If they do, any variation in part shape is irrelevant to the scanner's success.

Practical Takeaways for Next Week's Line Walk

You don’t have to overhaul your entire automation suite by Monday morning, but you can start changing how you approach these problems today. Here are three things to do on your next walk:

- **Audit Your Programming Time:** Ask the team who manages your automated inspection systems to show you a log or provide an estimate of time spent on "path setup" versus "execution." If they are spending more than 20% of their time tweaking paths for minor part variations, you have identified a prime area for simplification.
- **Talk About Cuboid Logic:** Sit down with your automation specialists and move the conversation away from "precision pathing" toward "cuboid logic." Ask them what it would take to implement bounding boxes for your most frequent high-mix parts. Challenge them to find ways to reduce the number of unique programs required per product family.
- **Map Your Product Families:** Take a physical look at your fixtures and inventory. Start grouping items into families based on their "footprint" rather than their SKU numbers. Create a simple chart that shows which products share similar inspection zones. This will be the foundation for moving away from bespoke code and toward a scalable, robust system.

Call to Action

What is the most complex, part-specific routine you currently run? Share it with a colleague who might benefit from this perspective. Send us an issue recommendation to newsletter@sixsigmakaizen.com

Newsletter replies: newsletter@sixsigmakaizen.com · X: @kaizen_6sigma

Stay curious, stay improving,

David Rodgers
SixSigmaKaizen.com

References: Quality Digest: Simplified Scan Trajectories in High-Mix Manufacturing Eliminate Complexity