

Technical Approach in Reducing GUI Development Cycle Time for Adopting the Connectivity Solutions in Automotive Infotainment Systems

Kasiraja Thangapandian
Visteon Technical and Services Centre

Kumaran Bharatheedasan
Visteon Automotive Systems India Ltd.

Binoy Melatt Vythakkatt
Visteon Technical and Services Centre

ABSTRACT

With ongoing integration of various systems in the Vehicle, the usage of display unit is increasing day-by day. The necessity to access data, stored on a remote system, via a human machine interface (HMI) is growing and also the need for developing a Graphical User Interface (GUI) in an efficient manner. The customer/user request to view or browse the system via display is becoming more complex and it is very annoying for the customer/user to wait for a system reaction on his input to the local system for a longer time.

With the above limitations we would need to develop the GUI for quick turnaround with high quality.

In this paper we are about to discuss on the Model View Controller (MVC) architecture, platform based approach, modular approach based on Core - Server for GUI development, auto code generation, behavioral model simulation, reusable packages and various tools that are used in development of GUI.

The selection process of GUI tools is now turning to be vital during the development phase. The proposal of GUI development process can allow for Graphics to be developed by the OEM/Studio and can be developed in multiple cadences to allow for parallel development with the customer reducing the need for upfront design, by performing various GUI simulation and visualization of other product features (e.g. Fuel computation, Gauges information etc.,) to ensure end result is what the customer desires. Various graphics rendering techniques are being discussed in order to have better performance during the cycle time reduction tradeoffs.

CITATION: Thangapandian, K., Bharatheedasan, K., and Vythakkatt, B., "Technical Approach in Reducing GUI Development Cycle Time for Adopting the Connectivity Solutions in Automotive Infotainment Systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 7(2):2014, doi:10.4271/2014-01-0262.

INTRODUCTION

Definition of GUI: In computing, Graphical User Interface (GUI) sometimes pronounced ('gooey') is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.

The GUI development model is continually evolving to cater for increasingly complex graphical demands.

Various benchmarks used in GUI application development were Frame per Seconds, rendering cycle time, memory & CPU consumption and startup time.

The graphic assets which could in GUI development are Bitmaps, Fonts, Vector Graphics, 2D Animations, 3D Models (Mesh, Cameras, Lights, Materials, and Animation)

Primary factors that drive cycle time include:

Amount of functionality to be delivered - The “size” can be measured in function points, story points, use cases, lines of code or whatever. Any discussion of cycle time improvement is meaningless absent the “how much” (size) dimension - it is trivially obvious that we can deliver faster if we deliver less.

The delivered quality requirements - A meaningful measurement of cycle time improvement needs to consider some measure of delivered quality, such as defects per size or defect containment effectiveness.

Time on task - This is the extent to which the project team is actually devoting time to the project of interest. Rarely is it 100 percent. Most teams will spend a certain amount of time fixing defects related to a prior release, working on other parallel projects, attending training, taking vacations, etc. Interruptions are a fact of life and have a profound, but usually unmeasured, impact on both cycle time and productivity. Every interruption requires a “context switch,” i.e., absorbing the particulars related to the cause of the interruption, such as understanding the specifics of a prior release defect report, investigating potential causes, determining potential solutions, implementing and testing fixes. Some studies suggest that the context switching of the software developer can absorb up to 50 percent of total effort when interruptions are frequent, as often they are. Hence, one key to improving cycle time is reducing interruptions that require context switching.

Defect insertion rate - This is largely a function of the experience level of the assigned personnel, the complexity of the problem being addressed and the reasonableness of the established schedule. As all industry cost-estimating models clearly demonstrate, excessive schedule pressure leads to significantly higher defect insertion rates, and hence higher rework.

Appraisal methods and timing - The type and amount of appraisal (reviews, inspections and testing) done at each phases of the development process (requirements, design, and construction) have a profound influence on the effort required to find and fix defects. Failure to do sufficient early appraisal leads to significantly increased effort in appraisal (defect finding) and rework (defect fixing). More effort spent on finding and fixing defects invariably leads to longer cycle times.

Hand-offs and queues - Lean thinking has clearly shown that that both hand-offs and batch queues contribute to increases in cycle time. Smaller batches (functional scope) and end-to-end team responsibility contribute positively to reduced cycle time.

Cycle Time Reduction Possibilities

It remains important to address and measure all of the factors that contribute to longer cycle times. Agile methods can reduce, but do not eliminate, root causes of cycle time problems. Lean Six Sigma drives to root-cause solutions.

Certainly cycle time is an important Y (outcome metric), but so is efficiency, which might be defined as cost (effort) per unit of functionality delivered (size). Ultimately, cycle time and efficiency are intimately connected. Optimizing one without regard to the other is certainly false economy. If shorter cycle time leads to lower throughput (cumulative functionality delivered), the gain can easily be an illusion. In some circumstances that can occur as a consequence of testing overhead associated with subsequent iterations, especially when there are many inter-dependencies with other systems. Even when cycle time is the primary metric of effectiveness (the primary CTQ, or critical-to-quality element), productivity will always be an important secondary metric. No matter how it is sliced, efficiency will always constrain cycle time.

Reusable Software Components

Reusable software components are the best in industry to increase the productivity by reducing the development cycle time, the idea is “develop once and deploy it many”. Intelligent use of object oriented approach to create reusable software objects will reduce the developers writing the software code many times and also reduces the chances of introducing new errors to the software.

In GUI, efficient use of Model View Controller (MVC) architecture leads to develop many reusable software components by deploying a clean separation between the data, logic and the visual part. Reuse can be applied from a complete system to a well-defined function based on the requirement.

The advantages of reusable software components are increasing the reliability of the software as it is an already proven component or module/function.

Lean Software Development

To cut a long story short, the answer to why reducing cycle time matters is that a short cycle-time is the only thing that can truly differentiate between a great development team and a mediocre one. A team that has reduced its cycle-time to say, about a day, has mastered many things in the process.

In other words, it takes a lot of things for a team to be able to deliver software in a super-short cycle, but it is those very things that become its biggest advantages. Another factor is that the reduction in development time and possibility of engaging in to multiple projects in shorter durations motivates the people and also increases the knowledge span among the team.

The reducing cycle-time simply becomes getting a feature into the hands of a customer as fast as possible and identifying and eliminating waste in the development process are considered.

Identifying the complexity in the conventional approach and evolving to an alternative solution is a difficult stuff. This has to go through a process of creating a Center Of Excellence for feature development by utilizing the multi skilled resource in GUI development cycle.

Factors Considered for Benchmarking

The following factors were considered to differentiate the conventional approach of development with the proposed GUI development methodology.

- Development time
- Number of features delivered in a limited time.
- Quality of delivered projects or features
- Number of defects
- Efficient use of process and tools
- User Interface
- Intuitiveness

GUI - Current Stage of Art

The GUI development model is continually evolving to cater for increasingly complex graphical demands. GUI should unlock the features and functionality which brings the application or even device to the center of user attraction. For this, the User Interface development needs to be rapid, seamless and should be easily portable to any new platforms and hardware's within a short time.

Below are some of the challenges faced during GUI development in automotive software environment

- Frequently Customer requirement are updated, which increase development cost
- Long developmental time required to develop features which involves high cost
- 3D Rendering and achieving Photo realistic rendering quality is difficult
- Multiple rendering sources: Multiple cameras, video source and blending all to a single display output
- Handling different user input systems

Conventional Approach

The conventional approach used in developing GUI applications are requiring high amount of hand coding, not able to simulate the developed applications in the developing platform, Software rendered graphics, Hardware dependent software, No clear separation between graphics, logics and data etc.

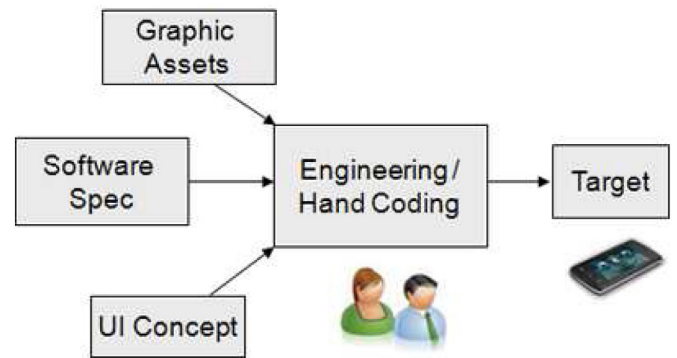


Figure1. Conventional GUI development

Constraints and Limitations of Conventional Approach

The main difficulty will be the dependency on hardware to test the Graphical User Interface, each change need to be tested on the final hardware which may be less efficient than testing in the developing platform itself.

In this approach it is difficult to upgrade the software version and the system is more prone to have manual errors being injected. Each time when the hardware is upgraded the software also has to be reworked significantly.

Software rendered graphics and lack of having a standard API interfaces leads the programmer's life difficult and the performance of the display output will be lagging as there is no efficient hardware accelerated rendering.

Proposed Methodology and its Benefits

A series of changes are applied to the conventional approach to become a successful and fast developing GUI which are based on the different assessment and benchmarking procedures identified.

Major changes are done to improve the developmental cycle time by completely describing a Model, View and Controller architecture and deploying a clear separation of view and the logic part. This will be discussed in detail in the later part of the document.

Reducing hand coding by deploying graphic tools for visual development is another major step, which actually helps to create the UI's visually and WYSIWYG way and finally generate the code for the target hardware. This helps to reduce a lot of coding time required to do complex graphic algorithms and programming. Also reduces the manual injected errors occurs during hand coding. This will be described in detail in the later part of the document.