Impact of Increased Vehicle Software Content on Automotive Product Development

by

Amy Elizabeth Rabatin

Bachelor of Science Mechanical Engineering
Marquette University

Submitted to the Department of Mechanical Engineering and the Sloan School of Management in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Mechanical Engineering
and
Master of Science in Management

in Conjunction with the Leaders for Manufacturing Program at the Massachusetts Institute of Technology
June 2002

© 2002 Massachusetts Institute of Technology
All rights reserved

Signature of Author

Sloan School of Management
Department of Mechanical Engineering
May 10, 2002

Certified by

John Sterman
Standish Professor of Management
Thesis Supervisor

Daniel Whitney
Senior Research Scientist, Center for Technology, Policy and Industrial Development
Thesis Supervisor

Accepted by

margaret Andrews
Executive Director of Master’s Program
Sloan School of Management

Accepted by

Ain Sonin
Chairman, Departmental Committee for Graduate Students
Department of Mechanical Engineering
Impact of Increased Vehicle Software Content on Automotive Product Development

by

Amy Elizabeth Rabatin

Submitted on May 10, 2002 to the Department of Mechanical Engineering and the Sloan School of Management in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Mechanical Engineering
and
Master of Science in Management

ABSTRACT

Software content in vehicles is rapidly increasing. Software development and integration, however, is currently not a core competency at a typical OEM. Software development differs from hardware development because of increased complexity, increased need for integration activities and more complex testing requirements. Therefore, the goal of the research is to develop insights and recommendations regarding the systemic issues of software implementation.

A combination of interviewing and causal loop diagramming was used to gain insight into the systemic issues. The recommendations made focus on the development process migrating from a subsystem focus to a system focus, on developing fundamental capabilities that support long-term productivity, and on aligning metrics and incentives with a system focused development process.

Thesis Supervisor: John Sterman
Title: Standish Professor of Management

Thesis Supervisor: Daniel Whitney
Title: Senior Research Scientist, Center for Technology, Policy and Industrial Development
ACKNOWLEDGEMENTS

First, I would like to thank my husband, Steve, for his limitless support throughout this project. No words can actually express my gratitude. I would also like to thank our parents for their constant love and support. I am also grateful for my sister, who kept me smiling through this process.

I wish to express my sincere appreciation to the people who assisted me with my research. I learned so much from the department and look forward to using the skills that I gained while there in my future assignments. Thanks must specifically be extended to Mike, Nick, Mark and Mark. Without their guidance and support, I would not have had the experience that I did.

Without the support of my sponsor company and specifically Tim, Donna, Sue, Lou, Maureen and Maryann, I am not sure that I would have had the confidence to take this adventure. Thank you for helping me through this journey.

I am especially grateful for the understanding and patience of my thesis advisors, Dan Whitney and John Sterman. Thank you for your gentle guidance and advice not to try to change the world...all at once at least.

Finally, I gratefully acknowledge the support and resources made available to me through the Leaders for Manufacturing Program. This experience challenged me in ways that I never imagined. This truly was an opportunity of a lifetime. Through the program, I have made friendships that will be forever cherished. I want to thank them for their intellectual and emotional support throughout my time here.
# TABLE OF CONTENTS

1. INTRODUCTION ............................................................................................................ 6  
   1.1. Background .................................................................................................. 6  
   1.2. Objective and Approach ........................................................................... 8  
   1.3. Thesis Organization ................................................................................. 9  

2. AUTOMOTIVE SOFTWARE CONTENT AND PRODUCT DEVELOPMENT PROCESS BACKGROUND ................................................................................................................. 10  
   2.1. Background and Industry Factors ................................................................. 10  
   2.2. Hardware Focused Product Development ................................................... 14  
   2.3. The “White Box” System ........................................................................... 17  
   2.4. Software Development Literature .............................................................. 18  
   2.5. Software as an “Architectural Innovation” ................................................... 21  

3. METHODS ................................................................................................................... 24  
   3.1. Interview Methodology ............................................................................. 24  
   3.2. System Dynamic Model Development .......................................................... 26  

4. THEMES FOUND IN INTERVIEW DATA ................................................................ 27  
   4.1. Common Themes from Interviews ............................................................... 27  
   4.2. Interesting Asides from Interviews ............................................................... 37  

5. DEVELOPMENT PROCESS DYNAMICS ................................................................ 39  
   5.1. Revised System Focused Development Process ........................................... 39  
   5.2. Other Major Dynamics .............................................................................. 50  

6. CONCLUSIONS ......................................................................................................... 61  

7. REFERENCES ............................................................................................................. 63  

8. APPENDICIES .......................................................................................................... 65  
   A. Tarek Abdel-Hamid’s Model ............................................................................ 65  
   B. Ford and Sterman – Precedence Relationships .............................................. 66
LIST OF FIGURES

Figure 2-1 Basic Four Phase Development Process Information Flow ......................15
Figure 5-1 Information Flow in Parallel Development Process ..............................40
Figure 5-2 Rework Dynamic Model .................................................................41
Figure 5-3 Revised Information Flow with Rework Loops ..................................44
Figure 5-4 System Focused Information Flow Basic Framework .........................45
Figure 5-5 Work Flow for One Iteration Through Development Process ..............48
Figure 5-6 Precedence Relationship for Series Process .........................................49
Figure 5-7 Basic Model Structure ...................................................................51
Figure 5-8 “Work Harder” Loop .......................................................................53
Figure 5-9 “No Time for Process” and “Never Ending Problems” Loops .............55
Figure 5-10 “Work Smarter”, “Reinvestment” and “Rework” Loops ....................57
Figure 5-11 Revised “Work Smarter” Loop .......................................................59
Appendix Figure B-1 Examples of External Concurrence Relationships ..........67
SECTION 1.0 INTRODUCTION

1.1 BACKGROUND

Software content in vehicles is rapidly increasing through new powertrain controls and telematics offerings, for example. Vehicle software is embedded in an Electronic Control Unit (ECU), which is a computer on the vehicle that enables the software related features and functions. An ECU dedicated to a door controls electric window and lock functionality; an engine ECU monitors and adjusts engine temperatures. Current vehicle software also enables voice-activated phones, headlights that switch off automatically, and sensors that watch for obstacles ahead. In the 2003 BMW 7-series, the vehicle software allows the driver to download e-mails and to adjust the suspension settings.

Software development, however, is currently not a core competency at a typical automotive original equipment manufacturer (OEM). Currently, the typical OEM provides functional/behavioral requirements for an ECU to suppliers with software competency. The suppliers then design and supply a “black box” ECU. Because the algorithms are products of the suppliers, the algorithms become the intellectual property of the suppliers.

Increasing software content in vehicles potentially provides the OEM with an increased value proposition. Increasing software content in a vehicle potentially reduces the cost of producing the vehicle. For example, piece cost could fall because hardware could be made less complex or replaced with software. New revenue could be generated via software by enabling option priced or subscription based customer features or upgrades. Therefore in light of these two examples, software in vehicles is both a process and a product. For an OEM to effectively compete in an
automotive industry where software has an increasingly dominant role, software must be viewed as both a strategic initiative and a technical initiative. How well the business and technical initiatives are addressed and implemented will ultimately determine the value of software for this OEM.

The OEM has three levels of options for involvement in the software process: 1) “black box”, where the OEM continues to write ECU functional specification as described previously, 2) “gray box”, where the OEM has a level of engagement greater than simply writing functional specifications, 3) “white box”, where the OEM has involvement, knowledge, and/or capability in all aspects of the software process.

Today, the thought process for getting software into a vehicle via a “white box” is in one of two schools. One school thinks the software architecture comes before determination of the software applications. The other school thinks determination of the software applications must be first, which will in turn drive the software architecture. In the author’s opinion these schools of thought are technically oriented and neither path will ultimately enable successful implementation of software on vehicles.

To successfully address software innovation as “white box”, the organization needs to develop new capabilities to allow it to work on software applications and develop architecture in parallel. The move requires an assessment of the existing silo organization in light of the requirements of a software initiative, in terms of product development process, supplier relationships, and
manufacturing capabilities, for example. Software related systemic capabilities will require new/different interaction/connections across silo boundaries.

1.2 OBJECTIVE AND APPROACH

Developing insights and recommendations regarding the systemic issues of software implementation in automotive product development is the primary goal of the research. The insights and recommendations will be structured around the following assumptions and issues.

- The “white box” approach is the most viable path for the OEM to realize the value proposition. For the purpose of this thesis, it is assumed that the OEM desires to pursue a “white box” capability.
- Software development and system integration are not typical core competencies of the auto companies.
- Software development differs from hardware development because
  - Software complexity increases rapidly.
  - Integration across disciplines is more critical.
  - The system must be tested as a whole to identify different interaction behaviors that may not be visible until integration.
- Managing the software/system development process in the same way as the conventional development process will increase the effect of the above issues.

To approach these issues, the author conducted literature reviews and interviews and attended project meetings. A combination of scenario planning and causal loop diagramming (Sterman, 2000) was used to gain insight into the systemic issues of software implementation. Because the
data collected was qualitative rather than quantitative, causal loop diagramming was used to model the dynamics rather than formal mathematical modeling. The details of a potential software architecture and the details of the development process are outside the scope of this research.

1.3 THESIS ORGANIZATION

The thesis is organized as follows. Section 2 contains a deeper background on vehicle software content and explores hardware and software focused product development. Section 3 contains details about the interview methodology and causal loop diagram development. Section 4 presents main themes from the interview process. Section 5 contains discussion of the main themes and the organizational impact on the product development process. Section 6 provides concluding remarks and highlights areas for managerial focus.
SECTION 2.0 AUTOMOTIVE SOFTWARE CONTENT AND PRODUCT DEVELOPMENT PROCESS BACKGROUND

This section contains a brief summary of the history of software in vehicle applications. The uncertainties and influences, both internal and external, associated with vehicle software and applications presented here will facilitate discussion in later chapters. Section 2.2 describes the incorporation of software in a hardware based product development process. Section 2.3 describes “white box” capability, the assumed path for the OEM. Section 2.4 reviews some of the existing literature on software development. The literature reviewed in this section was chosen based on the unique perspectives that it offered on software development. Section 2.5 describes software as an “architectural” innovation.

2.1 BACKGROUND AND INDUSTRY FACTORS

Vehicles have contained software since the mid-1960’s, primarily for powertrain (engine and transmission) and radio applications. The anti-lock brake system (ABS) was the next system to incorporate software for control of a mechanical system. The first patent for a strictly mechanical anti-lock brake system was filed in the 1920’s, but this system proved unreliable. The Ford Motor Company revolutionized the use of the mechanical ABS system with the addition of software controls in their 1987 model year four wheel drive vehicles\(^1\).

Currently, vehicle electronics containing software account for approximately 19 percent of the cost of a mid-sized vehicle and are estimated to become 25-50 percent of the cost of a mid-size to luxury vehicle by 2005 (Berger, April 2002). Estimating and allocating the cost of vehicle software, however, is difficult because it is costly to initially produce but cheap to reproduce

\(^1\) This information was found on the Automotive Information Center website (see e.g. www.autosite.com).
(Shapiro and Varian, 1999). It also has no value on its own, but creates value by working as a system with the hardware. Therefore, the cost must be allocated, for example, by spreading the total up front development cost over the estimated production volume, which is not exact.

With the increased role of software in vehicles the reliability of software is paramount. The reliability challenges of vehicle software are much more intense than those for a traditional personal computer. Vehicle software and related electronics must be made impervious to severe environmental fluctuations, such as transient voltage, excessive heat and cold, electromagnetic interference, and mechanical thumps and bumps, an environment that a traditional PC would most likely not survive (Berger, April 2002).

Technology change/enhancement makes possible increased vehicle differentiation. Software in vehicles enables enhanced consumer applications, such as telematic applications and rear seat entertainment. Software enhances system applications, not directly experienced by the consumer/driver, for example powertrain applications and communication between subsystems. Some novel features enabled by vehicle software are (Hansen, July 2001):

- Adaptive cruise control (ACC). Radar sensors and sophisticated signal processing to see through fog, rain and snow, detect a car in the same lane ahead of the vehicle. ACC systems adjust the speed of the vehicle to keep a safe distance from the vehicle ahead. Currently, the system requires further development for performance at slow speeds and in stop-and-go traffic. So, very few vehicles currently have this function.

---

2 Telematics is an emerging market of automotive communications technology that combines wireless voice and data to provide location-specific security, information, productivity, and in-vehicle entertainment services to drivers and their passengers. (See e.g. www.motorla.com/telematics/faq)
- Airbag systems. Software enables vehicle occupant sensing, for example no occupant or a child, which could signal the need for modification of the deployment features of the airbag.

- Powertrain control systems. Software systems will enable features like electronic throttle control and cylinder deactivation, which will improve powertain performance, increase fuel efficiency and lower emissions.

Vehicle software may affect warranty costs and service experiences. A service center, equipped with the right technology, can use the on-board computers to provide technicians with more accurate diagnostic information, which will potentially reduce the cost of warranty for the OEM and increase the satisfaction of the customer, as the problem can be handled more quickly and accurately. More than 30 General Motors and Acura models on the road are equipped with remote diagnostic capabilities, enabled through the telematics system, that are triggered if the check engine light comes on or the engine knocks (Berger, April 2002).

The industry factors impacting the advancement of vehicle software are not limited to just the vehicle and the software content. In an effort to identify potential unique factors that may affect the pursuit of increased software content, the author, at the beginning of the research period, conducted a series of scenario envisioning exercises with a cross-functional team to hypothesize what the factors might be and to better understand the uncertainties and influences that might shape this space. Approximately forty members from the engineering community, research and development, marketing, and manufacturing made up the cross function team. A smaller planning team met for three, four-hour workshop planning sessions, using a scenario
development protocol. The entire team met for two, four-hour workshop sessions. A brief summary of the areas discussed follows.

- **Societal trends** – Will consumers view increased software positively or negatively? If consumer applications are pursued, will customers accept them and what role will the desire for privacy play? What are the trends in commuting? Will software applications enhance the commuting experience?

- **Consumer demand** – What measures will consumers use in their purchase or subscription decisions? Will consumers be willing to pay for offerings/services? What are consumers quality and reliability perceptions and demands? Will applications differentiate product among competition?

- **Environment and energy** – Can software help to reduce the weight of the vehicle by replacing mechanical systems with software based systems, thereby increasing fuel efficiency? Can replacing mechanical systems with software based systems reduce the effect of manufacturing on the environment?

- **Legislation** – Will the government institute FAA (Federal Aviation Administration) type regulations to ensure the safety and efficacy of the systems? Other regulation issues?

- **Technology advancement and standardization** – Electronic automotive applications tend to evolve more slowly than the consumer electronics industry. Does the automotive industry need to keep up with the consumer electronics industry?

- **Value proposition** – Is there money to made through software and software applications? If so, who will get the money and how much is there to be had?

- **Intellectual property** – Who will own the rights to the code/algorithms/software? How can infringements be enforced?
• **Supplier cooperation and relationships** – Do the suppliers want to write the software? Will they see it as business being taken away? Should the OEMs have industry partnerships? Will relationships help advance the OEM software competency more quickly?

• **Competencies** – What competencies are required to manage increased software content? If the OEM chooses to write code, do those competencies exist in the organization already? Do the competencies required to integrate systems with hardware content and increased software content exist within the OEM currently?

• **Supply chain coordination and system integration** – How will these functions be handled if the “black box” or “gray box” options are pursued?

Cost, quality, customer satisfaction and competitive advantage are major drivers of the unique issues described. Most importantly however, in the center of all of these uncertainties and influences is the product development cycle. In Clark and Fujimoto’s book, *Product Development Performance*, they state that product design and development will be a critical dimension affecting cost, quality, customer satisfaction and competitive advantage (Clark and Fujimoto, 1991). To fully understand that statement in terms of how software will support or challenge that statement one must look at the process as it is today and then review the statement in light of the dynamics of the software development process.

### 2.2 HARDWARE FOCUSED PRODUCT DEVELOPMENT

#### 2.2.1 Overview of hardware product development process
For simplicity and due to the proprietary nature of the development process, the current hardware based product development process of an OEM is viewed as a typical four-phase product development cycle. The information flow for the four-phase product development process is shown in Figure 2.1.

![Figure 2-1 Basic Four Phase Development Process Information Flow (Ford and Sterman, 1997)](image)

Depending on the development schedule followed, determination of the vehicle content typically occurs two to four years prior to the actual production of the vehicle. Presentation of the proposed vehicle content to the decision committee occurs and the four-phase process begins after approval. The vehicle development process continues through the phases primarily with a part level focus with verification and validation taking place at the integration builds. Accomplishment is measured primarily by adherence to schedule and budget. If, through the content determining phase however, a required or desired feature or significant modification, that increases the risk of missing program timing or budget, may be scheduled separately from the standard program timing.
2.2.2 Current methodology for getting software onto the vehicle

The current methodology for getting software into vehicles is through what the author terms the “black box” methodology. The term “black box” term refers to a system that may have specific inputs and will have specific outputs, but the workings inside the box that generate the desired outputs are not necessarily known by the OEM. A debate about the ownership of the content of the “black box” may occur, as the OEM specifies the desired output through the requirements, but a supplier writes the actual code inside.

During the product definition phase, the supplier receives the “black box” requirements from the OEM. The development process continues on through the design phase to generate the hardware for that specific vehicle. Interface of the “black box” and the associated hardware occurs, maybe for the first time in the development process, at prototype testing. Typically, there are issues with the integration of the hardware and the software. Also, there may be a gap between the “black box” required and the “black box” delivered. The development team may attribute this gap, for example, to poorly written specifications, to a change in the content chosen earlier in the process, or to lack of supplier adherence to the specification. Additional issues, that are independent of or related to integration, may occur as the system, meaning the “black box” and associated parts, moves through the reliability/quality phase. All deviations from the specification and generated issues, whether internally generated or externally generated, lead to unscheduled rework in the development process, which challenges the ability of the process to meet cost, quality, timeliness and desired functionality targets. Deviations are a product
development risk regardless of software content, delayed integration and therefore delayed issue identification is the additional challenge with “black box” software.

In an ideal process, the deviations and issues generated in this cycle through the process would be avoided in the next cycle through the process by passing on the “lessons learned”. The transfer of the lessons however is dependent on many obstacles, for example, the time allotted for documentation of the lessons, the form the documentation takes and whether the next person reads them. An interesting dynamic, uncovered in the course of this research, was that the rate of repeated issues on the next development cycle of a similar “black box” is dependent on whether the supplier was a repeat supplier or not. Therefore, if the repeat contract goes to a new supplier, there is a potential for very little transfer of lessons learned.

2.3 THE “WHITE BOX” SYSTEM

In a “white box” system, the OEM would have involvement, knowledge, and/or capability in all aspects of the software development process. The OEM could continue having suppliers write code or the OEM could develop the applications, write the code in house, and then provide the information to a supplier to manufacture the box, for example. Regardless of the method chosen to use the “white box” capability, issues in the existing development process and organizational structure to support the development process must be addressed. Some advantages of pursuing “white box” capability are the control over the process enabling earlier product integration and the control of the intellectual property, both which increase the OEM’s probability of realizing the associated value. The dynamics of the “white box” system development process will be addressed in Section 5.
2.4 SOFTWARE DEVELOPMENT LITERATURE

2.4.1 The dynamics of software development

The growth seen in the computer hardware industry expands the demand for development for software applications. Issues that appear in fast growing hardware industries, for example, cost overruns, late deliveries, poor reliability and user dissatisfaction, in general, plague the fast growing software industry as well. Understanding the dynamics of software development will aid the understanding of the drivers of the cost overruns, late deliveries, poor reliability and user dissatisfaction issues and will help guide the decisions needed to manage the development process. The dynamics of software development was the basis of the Ph.D. research of Tarek Abdel-Hamid.

Tarek Abdel-Hamid’s work on the dynamics of software development aims to fill a void in research regarding managerial aspects of the software development process (Abdel-Hamid, 1991 and Kemerer, 1997). He developed a model, which is replicated in one form in Appendix A, capturing the dynamics of software development to gain insight into the affects of managerial actions and related dynamics on the success of the project. The goal was to develop a model that enhances the understanding of, provides insight into, and makes predictions about the process by which software development is managed. The critical management decisions that were addressed are:

- The implications of management actions to address a project that is behind schedule, such as revise completion date, keep completion date but hire more staff or increase overtime.
- The effect of quality assurance efforts on project completion times and total cost.
The effect of time distribution between project phases on the overall project.

The reasons for and implications of estimating productivity.

In his model, Abdel-Hamid divided the software development process into four subsystems: the human resource management subsystem, the software production subsystem, the control subsystem, and the planning subsystem. An important aspect to note in addressing the dynamics of the subsystems is that the variables addressed in the subsystems are not independent, but are interrelated in a very complex fashion. The descriptions of these subsystems are listed below.

- The human resource management subsystem. This subsystem captures the impact of the experience mix of the workforce, meaning the percentage of "rookies" and "old timers", on the productivity of the team. If the team is relatively young, for example, productivity may be lower due to the need for training. The perceived workforce level required to complete the project will directly impact the experience composition of the team, as more people are needed the hiring rate and therefore the "rookie" percentage may increase.

- The software production subsystem. This subsystem models the software development process. This subsystem captures the impact of the policies, decisions, and actions of the software development managers and the software development teams on the success/failure of software. The policies, decisions, and actions direct the error detection processes and quality assurance efforts, and, therefore, affect the maximum productivity achievable by the group.

- The control subsystem. This subsystem captures the perceived versus actual level of progress of the development team. Development teams experience difficulty when estimating progress due to the intangible nature of software. Therefore, forecasting
performance to the desired completion date is difficult until the system becomes more visible/tangible, which typically is later in the process. This ambiguity affects planning of resources, etcetera.

- The planning subsystem. This subsystem captures the dynamics of resource planning and timing. If a project is perceived as late with respect to the desired completion date more resources, for example, may be allocated to the project to avoid missing the date.

Although much of the research presented here is applicable to an OEM and increased vehicle software, Abdel-Hamid primarily models the dynamics for firms with established software capabilities, processes and activities and low complexity hardware integration. This however is not the case for the typical automotive OEM. The aggregation of the software production process and the exclusion of the initial content and requirements development stages inhibits the understanding of the effect of these dynamics on an organization developing software capabilities.

2.4.2 The product line methodology from the Software Engineering Institute at Carnegie Mellon

The product line methodology promotes software asset reuse to help reduce cost and reduce time to market. A “product line” is a group of products sharing a common, managed set of features. Organizations using a product line approach for software systems, therefore built from a common set of shared assets, are discovering an increase in market share, lower staffing requirements, significant cost savings, higher system reliability, greater customer satisfaction,

---

3 Information in this section was obtained from the Carnegie Mellon Software Engineering Institute website. (See e.g. www.sei.cmu.edu)
and shortened time to market. The product line methodology cannot be fully reviewed in this thesis, but is presented to highlight software development practices.

Systemic management of planned variations, while exploiting the commonalities across product lines is key to the management of this methodology. Systemic and strategic reuse allow for the exploitation of the commonalities through shared software architectures and components, test scenarios and models, and training and documentation. The focus of development, in the product line methodology, is not programming. Rather, the focus is evolution of the product lines. Reuse, in this environment, allows for less testing, increased reliability, and therefore, reduced development cost and time.

The product line methodology provides a very detailed plan for asset reuse in software development. The challenge with using this methodology for vehicle software is managing the hardware to maximize the use of the software assets. Developing practices to link the product line methodology with highly complex and changing hardware is an area for potential future research.

2.5 SOFTWARE AS AN ARCHITECTURAL INNOVATION

“Innovation is at once the creator and destroyer of industries and corporations” (Utterback, 1994, Introduction page xiv). Increased software content is an innovation for the automotive world. Radical or incremental are the traditionally categories of innovation. A radical innovation fundamentally changes the engineering skill set required and/or changes the scientific principles on which the product is based. Incremental innovations introduce relatively minor changes to
the existing product and typically utilize the existing knowledge of the firm. Automotive software innovation, however, does not fit into those categories when viewed as an alternative way to link components together. “Architectural Innovation” describes the notion of changing the way components are linked (Henderson and Clark, 1990). Addressing vehicle software as such an innovation is key to correctly addressing the innovation, as the distinction may provide some insight into the challenges associated with increased vehicle software content.

Automotive software is an innovation that does not necessarily change the components of a system, but rather changes the architecture of the system. The definition of “architecture”, per Henderson and Clark, is the method of integration of components or links between components. “Architectural knowledge”, therefore, is specialized knowledge about the ways components integrate into a product system or whole. Much of the component knowledge that the OEM possesses will be useful. However, current architectural knowledge may be an obstacle in addressing the innovation. Embedded in the firm is this type of knowledge, which affects the management and organization of the development process. Understanding what knowledge is useful and what is not and what knowledge needs to be acquired and applied, both in terms of software and the related hardware is essential to the mastery of this architectural innovation.

One way the effect of this innovation could play out is in the following scenario. In the hardware focused product development environment, communication between engineers working on different vehicle hardware components may not occur until a communication event is scheduled, for example, a formal review or an actual integration build. The effect of delayed communication or the failure to communicate may manifest itself in hard contact between parts
that must be resolved, for example. Software innovation may, however, change the function of
or link between their components. Their parts for example may interface via the same controller
or their parts may need to share input and output data, use the same bus or compete for
processing time on the ECU. Delay or failure to communicate and coordinate may increase the
effect of the architectural innovation on this scenario. For example, Roth and Kleiner, in their
book Car Launch, document a situation that could similarly play out in the vehicle software
scenario (Roth and Kleiner, 2000). The situation called the “tragedy of the power supply”
resulted from each component group trying to maximize the performance of their electrical
component. During system integration, the team discovered that the combined system required
an additional 12 amps capacity. Providing an additional 12 amps was impossible because a lack
of room and lack of time did not allow for the addition of a second battery. If communication
and coordination occurred earlier this issue may have been avoided. System integration that
does not occur early in the process for the software related components, may result in a particular
production intent control unit not having the required memory to perform all of the functionality
assigned to it or a controller not receiving the correct data to perform a function because another
unit is not providing the data.

The effect of the architectural innovation and the benefits to the OEM of “white box” capability
drive the need for the development process to become more system focused. A system focused
development process and the related organizational dynamics will be discussed in the following
sections, in light of the background and issues presented in this section.
SECTION 3.0       METHODS

This section contains a description of the interviewing method and the causal loop diagram model development.

3.1 INTERVIEW METHODOLOGY

The goal of the interview process and selection was to gain a cross-functional perspective on the perception of systemic issues related to increased vehicle software content. Twenty individuals were interviewed, ranging from executives to front line engineers to supplier representatives. The subjects spanned the organization both horizontally and vertically and could have responsibility for or be affected by increased vehicle software. Individuals interviewed from the OEM were from the following organizations: Product Development and Engineering, Manufacturing Engineering, Purchasing, Research and Development, Planning and Portfolio, and the Powertrain Division.

Individuals were first contacted through a phone call or personal meeting to explain the nature and use of the interview. The interview time was then set and the questions to guide the discussion were sent via e-mail to the interviewee. The author used a semi-structured interview method, meaning the initial set of questions was to start the discussion and the interview was allowed to deviate from it during the conversation. The questions were tailored depending on the experience and function of the individual. Where appropriate, the author used “feature based” questioning, where an actual cross-engineering silo feature enabled through software helped guide the interviewee through the thought process of how a cross-functional feature would be
handled today in their organization and in the product development process, and what would be the future steps and contacts that would better enable the implementation.

The basic set of interview questions are as follows:

- In the development process, how does software integration for vehicle applications happen today (if applicable)?

- What challenges does the interviewee foresee due to increased software content? For example, if necessary:
  - Reusability
  - Organizational capabilities
  - System complexity and development process capability

- Verification and validation of software and the system
  - Can both be done through simulation?
  - Issues/challenges with either one?

- Do you know how rework affects achieving performance targets? If so, what are the major sources of rework with respect to current software usage? How will these change with more integration?

- What are the major sources of potential schedule slippage...is software one of them?

- How will further integration of components/features/functions, due to software, change your (your groups) current activities?
  - Staff size/Competencies
  - Task Definitions
  - Current metrics/Future Metrics
  - Communication activities
  - Interrelated task coordination

- How late in the development process can you introduce software? What conditions would have to exist for delayed differentiation? Issues with delayed differentiation?

The interview length ranged from 1 hour to 3 hours depending on the time and interest of the interviewee. The interviews were audio taped with permission of the interviewee. Only one interviewee would not allow taping of the interview. The author also typed notes while interviewing. The interviews were transcribed from the tape verbatim. The raw data was
reviewed and grouped into main themes to enhance the model development. To preserve the anonymity of the interviewee only the generic job title is used to place quotes and references.

3.2 SYSTEM DYNAMICS MODEL DEVELOPMENT

A functioning System Dynamics model was not appropriate for this research, as very little quantitative data were available to calibrate a mathematical model (Forrester, 1961). Instead causal loop diagrams were created to capture the interaction of the variables associated with the increased vehicle software development process. The causal loop diagrams were developed based on models in existing research and were supplemented and supported with qualitative interview data (Burchill and Fine, 1997).

The rectangles in the model represent stock or state variables, which increase or decrease over time based on the integration of the inflows less the outflows. The pipes with valves represent the flows into or out of the stock variables. The arrows in the model indicate the direction of causality. A “+” sign at an arrowhead indicates that an increase (decrease) in a variable on the tail of the arrow will result in a corresponding increase (decrease) in the variable at the arrowhead. A “-” sign at an arrowhead indicates that an increase (decrease) in a variable on the tail of an arrow will result in a decrease (increase) in the variable at the arrowhead.
SECTION 4.0 THEMES FOUND IN THE INTERVIEW DATA

In this section, the results of the interview data are outlined and briefly summarized. Due to the open-ended nature of the interviews, most interviewees provided significant discussions about topics not directly addressed in the initial set of interview questions. Therefore, the common themes found are presented in section 4.1 and noteworthy additional conversation themes are presented in section 4.2.

4.1 COMMON THEMES FROM FOCUSED INTERVIEW QUESTIONS

4.1.1 Definition of increased vehicle software

Eighteen of the respondents agreed that increasing vehicle software content was necessary. When asked what the focus of the increased software content should be, the answers varied and were somewhat predictable based on their job function or background. Responses varied from use software only to create subscription or revenue based features to the view that software applications should be “completely transparent” to the customer, meaning the customer would not directly interface with the software enabled features and functions. Responses also varied from “software should be used to enhance hardware functionality” to “software should be used to replace hardware parts”. No one interviewed stated that the corporation should not be investigating increasing vehicle software content.

When asked about whether software development should be in-sourced or out-sourced, the respondents had different opinions. 90% of the executives interviewed argued for in-sourcing because future revenues may be linked to the ability to produce software and it may become the
differentiating factor in vehicles. So, if ‘the corporation is to stay competitive in this arena the
capabilities must reside in house’. One executive responded

“There should be no debate about if we are going to pursue this in house…only debates
about how.”

The engineers argued for in-sourcing because:

“Lack of control over the intellectual property that is represented in software is stopping
us [the engineers] from further integration”.

“…maybe the supplier of [X] screws up and you have to change your supplier at the last
minute and you can’t sell a certain type vehicle without [X]. The whole ripple effect
happens…you would like to believe that if you specify everything up front, four years
later at the end you would have it all. That is not the way it works for a lot of reasons.
One of them is that we don’t control the IP [intellectual property]. Without owning some
of that information we may not be able to respond quickly to the need for a new
supplier.”

“You will find the value of engineering controls that are flexible, which allow you to
solve problems later in the development cycle, leads to this desire to use more and more
software because software is more flexible.”

The argument for out-sourcing, by the remaining executives and an engineer, was that the
corporation is not a software company, nor should it become one. If out-sourcing was the
opinion, the respondent was asked what role the corporation would play if vehicle software
content increased. The respondents agreed that the corporation should handle the integration and
the management of the technology, which would also be true if the technology was in-sourced.
Therefore, the development process should be evaluated to address the integration needs and the
capabilities of the organization should at least be evaluated to address the management of technology needs.

4.1.2 Vehicle focus versus part focus

The majority of the interviewees responded that the current development process is not ideal for increased vehicle software content. Currently features and problems are not addressed at a vehicle level, “we [the vehicle development community] look right to the subsystem”. By design of the development process, features and problems are addressed by the subsystem or even at the part level. A Product Development Manager stated

“We are designed to think in ‘buckets’, for example steering, body, and suspension. Because we work in buckets, easy [easily solved] things don’t get done because one group does not know that the other group has the data that they need. Software needs to be designed at the vehicle level, because integration is paramount to the success of the software and hardware. If we are not thinking and working at the system [vehicle] level, we will not be able to manage the development.”

An executive stated when asked about “bucket” thinking

“We have too many cooks in too many kitchens…we have lots of pockets of excellence [activities].”

The interviewees that interface primarily with the development process stated that the software and hardware development processes must move in parallel and be thought of systemically. A parallel process, it is thought, will help change the paradigm that “software is free” and that it “really doesn’t take much time”. Both concepts of “software is free” and “doesn’t take much time” seem to revolve around an evolution concept. An engineer and a director, respectively, remarked
“The software is free thought gets complicated when it applies to an evolutionary application. If it is a new feature the value and therefore a cost seems pretty obvious, but when it is an adaptation the thought that it [the software] is free is still out there.”

“Software is inherently faster to develop. People understand when you say we don’t have that feature yet. It will take x amount of time because you would have to design, make some prototypes and test them, etc…that takes time. But just because it [the software feature] can prototype more quickly, the [development] time is not zero.”

If the systems are developed in parallel, the system will be able to be tested early and often and the visibility of the cost and timing associated with software will be better understood and therefore better optimized. Software development is not just “control-S [save], control-P [print]”. Another interviewee responded

“We tend to talk about software as an end unto itself. Integration [of the software and hardware] is one of the biggest issues of the entire system, so these processes cannot be separate.”

The metrics and incentives associated with the current development process tend to be “short-term” according to an executive interviewed. Currently, performance measurements for individual vehicle programs are adherence to cost and timing targets that optimize that individual vehicle program. Subsystem incentives are similar, but optimize for the subsystem and not necessarily the vehicle. An engineer recommended delivery conformance as a metric for a system-focused development process. He thought that conformance measurement should be based on delivering a “system” that works as originally specified on time. To support this, one respondent stated
“How happy are you going to be if I say here is about 80% of the content you wanted? You want what you asked for. With more and more integrated features and functions, the coordination efforts are going to be more complex. To achieve 80% of the desired content of a more complex system will take a lot of effort and people need to be given the incentive to do it.”

The other metric that was cited by most respondents to help encourage a system/vehicle focus for the development process was percent reuse, i.e. “how much changed”. Reuse will be discussed later, but the main thought was if the tools are put in place to enable it, people should be held accountable for it. A passionate advocate stated

“Quite honestly, the measurement that we will have if we don’t have software reuse is failure. We won’t have the resources to keep up if we don’t reuse. The reuse incentive needs to be measured at the vehicle line level, not at the software engineer level to drive the accountability from the top.”

Reuse does increase cost and time savings, but it can limit the amount of innovation that occurs if too strictly enforced. Assuming that a library of code exists, deviations from the reusable code require management. A manager stated

“You want to stay on the mainstream with the applications, but you can get to the point in a program that you will have to spin off a development team. You have to tell the development team, these are the only resources you are getting. If you don’t manage it you will have parallel development teams all doing things their own way, because there is no incentive to share/reuse or to do anything in a common way.”

An executive, not as convinced about the merits of reuse, remarked

“We should have a library of reusable modules with certain rules, but it should not dictate what we offer our customers.”
4.1.3 Capabilities

All interviewees, regardless of the exact path to follow, meaning either in-sourcing or outsourcing of software, agreed that an OEM must have some software related capabilities. The respondents for in-sourcing defined capabilities as the ability to design software at the vehicle level, this would include specification writing, code writing, code verification and integration skills, for example.

Capabilities in terms of verification and validation seemed to be the largest area of required development for an OEM increasing their software related capabilities. The time and the investment required to develop these capabilities will be great. An engineer familiar with verification through simulation said

“What is important for simulation for verification is accuracy and the ability to emulate the real world…that takes a lot of investment of time and money. The complexity of the software must be managed to allow for simulation and testing. If you change something you have to come up with ways to make sure that it is the only thing that you changed. You need to be able to do that before you do validation on a real vehicle. It is too expensive any other way.”

Another engineer said

“People may think they know what is in their software. But until you have the system together you can’t know all of the linkages. With verification through simulation you can prototype these different scenarios because software allows for it.”

The consensus for validation is that it should continue to be performed on actual vehicles.
The engineering and development team managers interviewed responded that the vehicle integration role needs to be strengthened, regardless of the role the OEM takes in developing their software. They agreed that the corporation does not have the necessary amount of integration capability yet. Developing interface requirements for the different systems in the vehicle was seen as a necessary first step to integration. The interface requirements were thought to start to set the “rules for how systems can and should interface”. One engineer reflected

“The real pockets of software expertise and integration are probably in the OEM spin off companies [i.e. Delphi and Visteon].”

“Integration is so important because the systems are so complex and at times not very visible. You can make a mistake and something gets changed. More importantly is not just that I did not mean to change it, but there was a linkage and we did not know that this piece of data was actually being used by this other subsystem as well.”

4.1.4 Rework

All interviewees responded that rework is an issue, with or without software. When asked about how rework affects their work or the work upstream or downstream of their tasks, the response was generally that it is difficult to track and communicate. It is well known that rework may cause a schedule delay. The interviews revealed that to avoid the delay the decision may be made to just increase the risk tolerance going into a test or extreme lengths may be taken to satisfy requirements, with resulting increases in cost.

One major cause of rework cited by most interviewees is lack of decisions in the vehicle content decision stage. “People don’t know what they want, so we do too much guessing” was a phrase commonly stated. Many times due to timing constraints, designers and suppliers need to get
work started based upon interpretation of the upfront vehicle content decisions that drive the feature and function requirements. Due to the interpretation of requirements, the required rigor may not be put into the specifications or if it is put in the interpretation could be incorrect. All of which leads to increased rework.

Software related rework is difficult to quantify. A software engineer interviewed stated

“There is an iterative nature to developing software. You cannot fully understand the requirements until you put it on the bench. But you cannot put it on the bench until you understand the requirements. You have to resolve that somehow and that inherently creates rework…but which end did the rework come from?”

Time pressure is a source of rework. Suppliers today are supposed to deliver embedded software that is to be validated prior to delivery, but there tends to be, according to the suppliers interviewed, an “unrealistic expectation” for time to design a complex module. An engineer, having just come from a meeting with a software supplier stated

“They [the supplier] tell us they don’t have the feature yet and we are okay with that. But when they say it is going to take 9 months to develop, we are not okay with that. We need to get the development time reduced.”

The pressure to complete the module increases and the supplier commits to it. This time commitment limits the time available for validation prior to delivery, so the validation and subsequent rework are forced later in the system. When asked about the affect of rework on the eventual system validation, an engineer with software experience outside of the OEM environment replied

“Some day we will talk about “software fatigue”! I am kidding, but I actually think that like a metal fatigues when you stress it over and over…it is not the construction of the
software that stresses it, it is the amount that you change it. There is a certain point that
so many people have touched the code that you can’t remember/tell what its original
goals were. That is my definition of “software fatigue”. The analogy to the mechanical
world is that once you bend it enough it will break. Bending here is not the final
function…how many changes per unit time have you tolerated in this area is.”

The impact of rework on the development process is explored in section 5.0.

4.1.5 Reuse

90% of the interviewees and many literature sources tout software reuse as a key to successfully
managing the inevitable increased vehicle software content. However, “the enablers are not in
place to promote and facilitate reuse at this time”. The degree of reuse of software and hardware
components may be dictated by the financial impact that the reuse will have. For example, one
manager cited a story of attempting to reuse a hardware module across vehicles that had similar
requirements, because of the “so called economies of scale and similarities in the software
requirements”. What was found, however was that the piece cost substantially increased with
respect to the previous part cost because the “reused” module had 30 extra pins that were not in
use. According to a purchasing representative, the financial model that was used to support the
reuse most likely calculated a piece cost reduction due to increased volume for the new module,
but did not take into consideration the piece cost of the old module. Basing reuse decisions
strictly on tangible financial data does not take into consideration intangible effects, such as
increased first time quality due to previous testing. Accounting for those intangibles is a difficult
task. One engineer stated
“A goal for reuse of software is because we would have already developed it and we know that it will work in a system, therefore we don’t have to test as much. That will save us a lot of time...we really understand that.”

The consensus is that the development of a library of software features and functions based on a common architecture would be best. The idea behind the library is that at the beginning of a program, the options and features for the vehicle would be determined and based off of a library of standard components. There will be times that being common will not be ideal but the goal would be to minimize those times. Therefore, requirements that are common and uniform will help to enable reuse. Some dynamics related to software reuse are discussed in section 5.0.

4.1.6 Intellectual Property (IP)

The opinion, primarily among the engineers more involved with the “black box” software development, is confusion about whether or not the corporation can regard the software in the “black boxes” as the corporation’s intellectual property. As demonstrated in previous sections, however, the interviewees believe that the OEM should hold the rights to certain software used in the vehicle. One technical director stated

“If we don’t control the intellectual property of our vehicles, we will never be able to realize the synergies between all of the separate pieces.”

According to most interviewees, including all of the executives interviewed, the criteria of pursuing IP should be “does it offer competitive advantage?” Software related IP is very difficult to enforce, however. Therefore if you own the IP, engaging in a claim is the only way to you can detect if there has been an infringement.
4.2 INTERESTING ASIDES FROM INTERVIEWS

4.2.1 Purchasing

Currently the software related costs are typically not listed as a line item on a break down of a supplier quote. Software is not free, however. As cited previously, the percent of vehicle cost due to software is approximately 19% for a mid-sized car. According to the supplier representatives interviewed, the OEMs are requesting that software become a line item on the piece cost break down, which as previously mentioned is difficult to determine but necessary.

4.2.2 Standards

Standards can reduce the cost of change. Standardization could lower the cost of features and functions. The interviewees, including the suppliers interviewed, believe that standards need to be driven through the OEMs. The suppliers are competitors and may be resistant to agreeing on an industry standard. The supplier representatives remarked that there needs to be a culture change if automotive standards for vehicle software are to come about, as “custom” is a very prevalent mentality among OEMs. A development manager at a supplier stated

“If we all develop them together, standards will help to ensure the safety and reliability of vehicle software and will help the consumer feel more comfortable with it.”

4.2.3 Timing of software integration

The opinion of the interviewees that have software and programming experience is software can be done “pretty darn late”, mainly because it takes less time than hardware to develop. An engineer when asked to expand on “pretty darn late” stated

“It just comes down to risk, which is very dependent on your ability to verify and validate through simulation. With software the risk for the customer is that they will have to go to
a dealership to have their controller re-flashed. At OEM X, who has a smaller more
discriminating customer base, they send an apologetic letter and ask the owner to come in
for service because they would like to make some adjustments. But that is really thinking
only about the software...the hardware has to be compatible and that has a longer lead
time.”

One interviewee stated that his desired time to integrate software would be the day before
production started, given that the capabilities are in place to have the software debugged and
ready to go. In his opinion, this would allow for the latest and most up to date versions of the
software to be loaded into the vehicle and could alleviate some of the issues with indecision at
the content development phase. The “realistic” consensus, however, was that parallel
development is necessary, as there is “always a dependency on hardware being able to do what
you need or be at the necessary level”. Delaying the introduction of software will increase the
risk that the software and hardware will not be a compatible system, especially when the
feature/function is new.
SECTION 5.0 DEVELOPMENT PROCESS DYNAMICS

In this section the dynamics related to the major interview themes are discussed. In section 5.1, the flow of information for a system-focused process is diagramed as well as the necessary iteration process. Also in this section, the dynamics and impact of rework are discussed as well as the precedence relationships in the system focused development process. In section 5.2, other major dynamics exacerbated due to increased software content are described by building off the concept of rework and process problems.

5.1 REVISED SYSTEM FOCUSED DEVELOPMENT PROCESS

5.1.1 Revised information flow

As referenced in Section 2.5, software is an architectural innovation for the automotive world. If the product development process is not changed in light of this innovation, the process will continue to give more of what it gives now. The first aspect to address is the flow of information, as this will affect the ultimate structure of the organization. The product development process has a flow of information, just like the manufacturing process has a flow of parts. How software will impact this flow of information is a fundamental question. Based on the interview data collected, the respondents stressed that software would increase the need for a parallel development process. The proposed information flow diagram, based on the interview data, is presented in Figure 5-1. When compared to the current flow of information presented in Figure 2-1, the necessary information flow paths increase in complexity.
In the system engineering phase vehicle content is selected and initial specifications are written. The process then moves through software design and hardware design based on the same information generated in the system engineering phase. Now, however instead of integration communication occurring at the end of the process, communication and coordination occurs throughout the entire process. Communication that occurs throughout the process could take the form of code reviews, system reviews and simulation testing, for example, assuming the capabilities are in place. Using the proposed information flow in Figure 5-1 with “black box” development would prove difficult, as the coordination efforts would increase in complexity due to dependency on outside suppliers.

A system focused development process, because of the concurrent structure, increases the dynamic complexity of the process. A system approach, such as the one proposed, is not followed today for many reasons beyond just the complexity. For example, the people involved
in the development process are currently so over scheduled that there is no time to go out beyond the assigned tasks to potentially mitigate downstream issues or prevent the upstream issues from happening again. Also, the organizational structure may not facilitate the necessary communication, because the required subsystem integration tasks occur at the end of the process.

5.1.2 Rework

Within development phases, there is always the possibility of generating rework. The inherent complexity of the system focused process leads to an increased amount of rework, when compared to a simple process, that has to be addressed by the resources in the system. The causes of rework, which are similar to those in a component focused process, occur in the development process in many ways, for example poorly written specifications, changing customer requirements, or lack of quality or completeness in work received from an up stream phase. Figure 5-2 below is a diagram of the typical rework cycle.

![Figure 5-2 Rework Dynamic Model](Sterman, 2000)
Figure 5-2 diagrams a single phase in the development cycle. There are certain tasks to be completed named *Tasks to Do* within the development cycle and the rate at which those tasks are completed is called *Tasks Being Done*. The rate of task completion is dependent on the number of *People* in the labor force and the *Productivity* level of the work force. Tasks will either be completed correctly or incorrectly, which is dependent on the *Quality* of the work. Tasks completed correctly go into the *Tasks Done Correctly* stock and are available for the next phase of the development process. Tasks not completed correctly go into the stock of *Undiscovered Rework*. The tasks in the *Undiscovered Rework* stock are assumed to be completed correctly and may be fed to downstream tasks through the information flows presented in Figure 5-1. Potentially the *Undiscovered Rework* could move all of the way through the development process, which is one reason that the information flow paths in Figure 5-1 must be supported by communication events, for example. The discovery of these incorrectly completed tasks may eventually happen by downstream phases, verification or validation testing or other quality assurance efforts. One interviewee responded when asked about rework discovery lessons learned said the following:

"I think we should do more verification. I think we are willing at times to compromise that [verification] because we can. We still know that we are going to take the whole system through a validation run. Depending on the timing charts, we are going to have validation fleets where we are running product around the country. Therefore, from a system level we will find and correct bugs that a driver could find. Therefore, in principle we have tested the whole system."

Based on this comment, the discovery of rework would be pushed later in the development process, which was demonstrated in Section 2.2.
Another source of rework that happens later in the process is *Decision Changes* to decisions previously made. These decision changes could be due to revised customer needs or revised budget constraints, for example.

The stock of *Known Rework* must be rectified at some point before the development cycle is complete. There are not many options for emptying the stock of *Known Rework*. Emptying the stock may take more time, money and resources than are available. As stated previously, the metrics that become incentives for performance are execution to time and budget targets. So, one option is to move forward knowing that the risk of error is greater in subsequent tasks. Another option is to “de-content” the vehicle, meaning to take out or disable the problematic feature or function. The least desirable option, due to the potential financial impact on the OEM, is to delay the program to allow the rework stock to be depleted.

Figure 5-3 is a diagram of the revised information flow with the rework loop in each phase of the process. This figure represents one pass through the phases of the system focused development process. Multiple passes through the phases after system engineering are addressed in later sections. Managing a multiple phase development process and addressing the impact of rework is a very complex problem. Each individual phase will have its own rework loop, but the undiscovered rework may get passed into the downstream phases through the work product, via the information flow path, from the upstream phase. A challenge in reducing the frequency of rework is the ability to attribute the issues to the hardware components or the software components. Therefore, a major benefit of the systemic development process is the verification of the system requirements will be performed earlier in the process, which will potentially reduce
the cost of change by minimizing the impact/down flow of the rework. Minimizing the flow of rework between phases is possible if integration and verification steps that follow the information flow arrows are incorporated into the activities of the development process. As stated in the interview summary, sophisticated simulation tools will be required to enable this earlier required verification.

Figure 5-3 Information Flow with Rework Loops
(Adaptation of Pugh-Roberts Associates Project Dynamics)
(Lyneis, 2001)

5.1.3 System Focused Information Flow Basic Framework

Figure 5-4 depicts a single iteration in the development process. To enable desired efficiencies such as component reuse and incorporation of lessons learned within current or future vehicle
programs, feedback needs to occur to upstream processes. The arrows moving from right to left depict that information feedback required to flow to the phases upstream. The feedback time between the phases should be as short as possible to create more timely awareness of defects and issues and to move them upstream where the associated correction cost is typically less. The challenge then becomes designing the communication system to facilitate the stepwise integration of the iterations through the multi-phase process. The success of the information flow will depend on breaking the integrated project down into pieces, which will be addressed through the development of a precedence relationship diagram in section 5.1.4.

Figure 5-4 System Focused Information Flow Basic Framework Adaptation of Pugh-Roberts Associates Project Dynamics Model (Lyneis, 2001)
Due to the increased complexity in the information flow paths, the process to communicate back upstream the task itself or the impact of the rework will potentially be more difficult due to the sheer number of people involved. It may also become more difficult to document the revisions and improvements that were made. If the process goes out of control, meaning there is too much rework, there may not be enough time and people/resources to complete the tasks at hand, as well as complete the rework necessary to produce a quality product. The delays in the system for information and tasks to flow downstream or upstream will increase and the system could potentially go into firefighting mode.

Firefighting describes the unplanned allocation of resources, for example time, people, and/or test equipment, to fix problems discovered late in the development process because a quality product must reach the market (Repenning, 2001). Another option to firefighting, as previously mentioned, is to cancel an option or de-content vehicle, for example. Lack of attention to the up-front tasks, in this case the system engineering phase, due to resource constraints for example, creates the catalyst for future firefighting activities as short-term gains can impact long-run performance. To get out of the cycle of firefighting there must be a willingness to support additional resources until the process is stable. In terms of the system focused development process, management must be aware of the need for more time and resources to initially implement a new process and way of interacting. If the new processes and ways of interacting, for example, are not accompanied by a reduced workload, their introduction is likely to lead to more firefighting and a further decline in process capability (Repenning, 2001).
To facilitate information flow in the system focused development process and further avoid firefighting, system review meetings, not just code review meetings, will become extremely important to help general communication, communicate issues and concerns, identify errors, identify potential interaction issues and help with general iteration, as the participants may have different concerns and priorities due to varied roles. Therefore, the review meetings will have a direct impact on the goals of producing a robust product at the right time, cost and quality.

According to Kazman and Bass of the Software Engineering Institute at Carnegie Mellon University, system review meetings that take into consideration the business goals as well as the technical goals will provide the greatest benefit to the stakeholders and directly affect the attainment of the respective goals (Kazman and Bass, 2002). To help with the quality of the review meetings the related organizational issues should be acknowledged. To acknowledge the organizational issues, Kazman and Bass suggest the following; that people with the system knowledge be present, that there is a stated purpose and goal with clear expectations, that there must be buy in for the review from all parties to ensure that the review has value, and that social behavior is managed, for example no finger pointing (Kazman and Bass, 2002).

5.1.4 Work Flow Cycle of System Focused Development Process

The iterative nature of development process becomes more complicated with the increased flow of information and required integration. Figure 5-5 illustrates a possible map of work flow through the system. This process would be for one series of hardware and related software. Multiple vehicle systems, for example chassis and body, would be moving through the iterations, before the complete integration vehicles, in parallel to promote early integration and verification.
efforts. Also, multiple cycles would be occurring simultaneously for a vehicle and different model year vehicles.

![Figure 5-5 Work Flow for One Iteration Through Development Process](image)

To better understand how the flow of work would actually go through the system, the work of Ford and Sterman (1997) was referenced for use of precedence relationships. See Appendix B for further description of the research. Precedence relationships describe how a development
process behaves over time. The diagram in Figure 5-6 shows the relationship between the percent of downstream work completed or available for completion as a function of the percent of work upstream completed and released. Articulating the precedence relationship diagram will help in management of the system focused development cycle, as the need and allocation of resources may be better understood and targeted.

The precedence relationship diagram in Figure 5-6 was determined with the interviewees during the interview process and reflects their vision of how the system focused development process should proceed.

![Figure 5-6 Precedence Relationship for Series Process](image)

In segment 1 of the graph, the downstream work cannot begin until the upstream work is complete to a certain percentage. When that percent complete is reached, the downstream tasks can progress quite quickly as the interdependency of tasks decreases, which is represented by
segment 2 of the graph. The decrease in task interdependence means that dependence of downstream tasks, on upstream tasks, to move forward with work is not as great as in segment 1. The arrows in segment 3 represent the time when more iteration will occur due to the near final system integration. The effect of these iterations depends on the degree of concurrence of tasks within that portion of the development cycle. If the tasks are highly concurrent, the amount of work that will be obsolete due to the iteration will be greater than the amount for lower concurrence. The increase in vehicle software will most likely increase the concurrency between tasks, therefore will increase the effect of the iteration. In segment 4, again as in segment 1, the downstream tasks are highly dependent on the upstream tasks for further movement in the process, as the system is approaching the final production stages. Again, the role of the system review meetings will be paramount to managing the work flow according to the established precedence relationships.

5.2 OTHER MAJOR DYNAMICS

To successfully execute a system focused development process the related organizational dynamics need to be taken into consideration. As an OEM senior executive stated, “The biggest risk in an OEM becoming more involved in vehicle software is the organizational risk”, meaning the risk associated with building the capabilities and difficulty in revising the way people do work. The organizational dynamics were assessed using the variables highlighted through the interview process. To address some of these organizational affects, the system focused development process is referred to in the aggregate and is generically labeled in the model as Process. Therefore, the variables and dynamics discussed in this section will not be for any particular phase of the process or iteration, but will be generalized for the system as a whole.
The foundation of the analysis is based on the work of Repenning and Sterman (2000) on process improvement dynamics. The model was modified to capture the dynamics of the specific scenario of increased vehicle software content. Figure 5-7 contains the basic model structure.

The stock *Process Problems* represents any deficiency in the system focused development process, for example communication issues, process steps performed out of sequence, documentation issues, unclear specifications, or inadequate up front decision making. The stock of *Process Problems* drains by implementing improvement activities such as more effective communication events as previously described, incentives related to documentation efforts or
enhanced specification guidelines. Decreasing the stock of *Process Problems* will ultimately increase the *Net Process Throughput* because the rate of *Tasks Done Incorrectly* will decrease.

The inefficiencies caused by the *Process Problems* potentially increase the number of *Tasks Done Incorrectly*, which will in turn increase the *Rework* stock. The draining of the *Rework* stock depends on the rate of *Tasks Reworked*. The *Net Process Throughput* is the *Tasks Reworked* plus the *Gross Process Throughput* minus the *Tasks Done Incorrectly*. The *Desired Throughput* is defined as the rate of work needed to accomplish the time schedule set forth by a standard timing chart or by the managers. The *Throughput Gap* is the difference between the *Net Process Throughput* and the *Desired Throughput*. Based on the research of Reppening and Sterman (2000), the type of efforts used to close the *Throughput Gap* will dictate if the fix is long term or short term and will create some interesting organizational dynamics.

5.2.1 Short-term Productivity to Close the Gap

The *Throughput Gap* created by the difference between the *Desired Performance* and the *Net Process Throughput* typically creates a reaction from management or the people that have to explain the gap to increase the amount of time people spend working. Increasing the *Pressure to Do Work* occurs, for example through mandatory overtime, revised incentives potentially including penalties, more frequent project reviews and/or project reviews with more senior people. The balancing loop, B1 Work Harder, in Figure 5-8 serves to maintain balance between the *Desired Throughput* and the *Net Process Throughput* by increasing the *Pressure to Do Work*, which increases the *Gross Process Throughput*. The increase in the *Gross Process Throughput*
increases the *Net Process Throughput* and thus decreases the *Throughput Gap*, at least for a while.

![Diagram](image)

**Figure 5-8 “Work Harder” Loop**  
*(Adaptation from Repenning and Sterman, 2000)*

The idea of closing the gap is not unique to the scenario of increased vehicle software content, as it is well documented in general product development literature. When the development process is subsystem or part focused, pressure to do work can be very targeted and focused. What is unique about the situation with increased vehicle software content is the reasons for and methods to increase the pressure to do work may be different in light of the system focused development process. The reason for pressure may be due to the intangible nature of software reflected in the measurement problems of cost, quality and productivity (Jacobson, 1984). Determining progress in software development is elusive as well, as the process is non-linear in nature. People tend to
overstate the level of project completion and understate the remaining resource requirements. The phrase “I am 90% done” is commonly used and commonly misunderstood. The 10% remaining may take more time to correct than the 90% completed before it. The impact of the Pressure to Do Work depends on the competencies of the individuals in the system focused development process, the success of the revised coordination activities and the management of intangibles, like the “that is not how we used to do it” mentality. An executive interviewee stated

“When something is new, there is a period of everyone saying no, and then later on people are ready to deal with it, but then you’re behind the competition.”

The lack of required competencies, the difficulty in coordinating integration activities or the threat of competition, for example, especially in times of pressure may increase the risk of shortcutting the process. Again the shortcuts in the process for a system focused development process are not much different from those in a subsystem focused development process. Figure 5-9 shows the balancing feedback (B2) and a reinforcing feedback (R1) created by Changes to Process. Changes to the process may be a reaction to the lack of understanding of the precedence relationships previously described or from entering the firefighting mode as a result of increased pressure to do work.
The balancing loop, B2 “No Time for Process” will again serve to maintain balance between Desired Throughput and Net Process Throughput. In other words, the larger the gap to close the more need for changes to the process, which will in turn serve to close the gap. Reinforcing loop, R1 “Never Ending Problems”, shows that as Changes to Process increase the Problem Introduction rate increases, which increases the stock of Process Problems. Because Process Problems increase, the Tasks Done Incorrectly increase, which decreases the Net Process Throughput and therefore increases the Throughput Gap. The increasing gap will reinforce the behavior of ad hoc changes to the process.

Figure 5-9 “No Time for Process” and “Never Ending Problems” Loops (Adaptation from Repenning and Sterman, 2000)
Ad hoc changes in the process are seen through increased risk tolerance when entering the system testing phase because time for coordination is limited or the work from upstream was not delivered to schedule. The degree of documentation, either for the current process or to enable reuse, may be sacrificed to accommodate the pressure.

Increasing the number of people resources on the project is not necessarily the answer to increasing Net Process Throughput, either. The size of a typical software development project is measured in man-months (Brooks, 1995). The interesting result that Brooks noticed is that doubling the number of people on a project does not necessarily double the productivity of the team. His assertion is that men and months are not interchangeable. Increasing the number of people on a project increases the cost of communication, therefore according to Brooks’ Law, adding more people to a pressured project can make it later. Graicuna’s formula states that as the number of people on a project increases arithmetically, the amount of time spent communicating increases geometrically (Reifer, 1981). If these formulas hold true, these efforts would only increase the Throughput Gap.

Stopping the “Work Harder” cycle, and thus the “No Time for Improvement” cycle, is very difficult because of the time delay between recognizing the short term gain in productivity and paying the price for incorrect tasks. As Repenning and Sterman (2000) documented:

“A software engineer who forgoes documentation in favor of completing a project on time incurs few immediate costs; only later, when she returns to fix bugs discovered in testing does she feel the full impact of a decision made weeks or months earlier.”

5.2.2 Long Term Capability to Close the Gap
A fundamental way to close the *Throughput Gap* is through the balancing loop, B3 “Work Smarter”, in Figure 5-10 (Repenning and Sterman, 2001). Fundamentally closing the gap for a development process with high software content occurs through investing the time, resources and money into developing capability activities such as verification and validation through simulation and system integration tools. By investing in such capability and supporting its use throughout the development process, the rate at which *Problem Correction* occurs would increase, as the root causes of the *Process Problems* would be eliminated. The *Tasks Done Incorrectly* rate would decrease due to the increased draining of the *Process Problems* stock, thus helping to close the throughput gap through long-term productivity gains.

![Figure 5-10 “Work Smarter”, “Reinvestment” and “Rework” Loops (Adaptation from Repenning and Sterman, 2000)](image-url)
Capabilities such as verification via simulation are required to increase process performance. Rework cannot be completely eliminated, therefore, the goal should be to limit its cost. Investment in capabilities such as verification and system integration tools must have the time and financial support required for their development. These capabilities will allow for early evaluation of the system and will enhance the system capability to handle changes that occur throughout the project lifecycle.

Realizing the gains from investing in capability is a challenge, as there is a time delay between the actual investment and reaping the benefits of the investment, which can increase the dominance of loop B2 where the increase in productivity due to shortcutting is immediately visible. Another challenge is to balance time spent on developing capabilities and time spent draining the rework stock. Obviously, there are a limited number of resources and the incorrect tasks must eventually be rectified. The adage “use it or lose it” applies because resources focused on rework will not develop/sustain capabilities to promote/maintain long-term productivity gains (reference loops B4 and R2). This is also true for the “black box” scenario, as capabilities for software related productivity gains reside at the suppliers.

5.2.3 Reuse Dynamics

Reuse is another method proposed for long-term productivity gains in the system focused development cycle. Strategic reuse of software algorithms and associated hardware may help to minimize the impact of rework, resource allocation and short-term productivity focus. Reuse requires the development of a library of reusable components. The development of this library, however, is subject to similar dynamics as the other long-term productivity gain activities.
The ability to have and maintain this library depends on the management of the time spent documenting the components and the quality/discipline in the documentation, as seen in the revised loop B3, "Work Smarter". Again, if the short-term gains of the "Work Harder" and the "No Time to Improve" loops are not managed, the **Throughput Gap** will not decrease because the code library will offer no benefit as a long-term reduction in the **Tasks Done Incorrectly**. Therefore, reuse must become an integral part of the system focused development process. The resources developing the code/component library may also be separate from the resources dedicated to the actual process to further avoid the pressure to get work done. This concept presents a new set of dynamics that will not be explored in this paper.

---

**Figure 5-11 “Work Smarter” Loop**  
(Adaptation from Repenning and Sterman, 2000)
Another element driving the revised “Work Smarter” loop is the rate of Code Obsolescence or lifecycle, which will dictate the draining/expiration of the code/components in the library. One way to help decrease the rate of obsolescence is through the creation of standards, even though the creation standards can be a slow process. According to Paul Hanson, publisher of The Hanson Report on Automotive Electronics, standards are key to fully taking advantage of the promise of automotive electronics (Hansen, October 2001). Standards can also help increase reliability of the system, decrease costs from both the design and manufacturing perspectives, and decrease time to market by decreasing the development time, for example. The use of standards does not have to limit innovation or the uniqueness of the automobile, as proprietary elements may be added to the applications. The parts and communication systems must also be available to create an entire system and standards will help suppliers get involved to ensure the availability of parts.
SECTION 6.0 CONCLUSIONS

Issues facing an established automotive OEM incorporating an architectural innovation into the product development cycle and keep to cost, quality and timing targets were explored. Through this exploration the following insights were highlighted:

- Currently the development process has a part-level design focus with verification and validation at the integration builds at the end of the process.
- Impact of work quality on downstream work is not well understood.
- Software will not necessarily change the role of vehicle components, meaning a heat sensor is still a heat sensor. Software may change the links between the components, meaning the data from the heat sensor may be needed by a new component.
- System level design and verification must occur early in the process.
- If rework related iterations are not managed, the system may reach a point where the product development cycle will not be able to be completed on time, within budget and with acceptable quality.
- Strategic reuse of software algorithms and associated hardware can also help to minimize the impact of rework and resource allocation.

Most importantly, however, the results of this investigation are applicable to the development process as it stands today without increasing the software content. The time to market for new and improved vehicles is getting faster and faster. One way to decrease the time required is to improve the product development process. Therefore, the architects and managers of the development process should be aware of the following, which will help to mitigate the issues presented in the introduction:
Changes to the development process due to increased vehicle software are not just revisions/additions of tasks. The changes are fundamental revisions in the way the information needs to flow, which should be reflected in the organizational structure. By just focusing on short-term productivity (i.e. getting the job done) and not focusing on improving the fundamental capabilities of the organization that will increase long-term productivity, the system may enter a vicious cycle where more and more time is spent working harder and less and less time is spent working smarter.

The organization will need to tolerate a worse before better situation in terms of productivity and investment, if long-term productivity capabilities are desired.

Metrics and incentives must be integrated with the development process. Just as the development process is system focused, metrics should also be system focused to support the development process. Management must expand cost, quality and timing dimensions to include elements of reuse and risk.
SECTION 7.0 REFERENCES


Appendix B: Ford and Sterman work on precedence relationships

Ford and Sterman (1997) define dynamic concurrence relationships as the relationship between internal, within development phase, or external, between development phase, available work constraints and iterative flows of work. The work of Ford and Sterman (1997) will be the primary reference for this appendix. Only between phase relationships will be outlined in this appendix. Again, the intention of this appendix is to provide a brief overview of the concepts.

“An External Process Concurrence relationship describes the amount of work that can be done in a downstream phase based on the percent of work released by an upstream phase.” The relationships can be non-linear and can represent changes in the degree of concurrence between the phases as upstream work progresses. Appendix Figure B-1 contains examples of external process concurrence relationships.

Appendix Figure B-1a portrays a relationship where the phases of work have no relationship. The parallel process captured in Appendix Figure B-1b shows that the downstream work may begin when 0% of the upstream work is complete. Essentially, the phases are independent. Appendix Figure B-1c is the opposite, where 100% of the upstream work must be completed prior to beginning the downstream work. These phases are considered dependent. Appendix Figure B-1d captures a delayed start where 100% of the downstream work is available for completion upon a certain percentage of upstream completion. The lockstep process portrayed in Appendix Figure B-1e shows that after x% is completed upstream x% will be available for completion downstream. Finally, Appendix Figure B-1f captures a process that has a continuous non-linear relationship. Therefore the relationship between upstream and downstream available work constraints varies, which is probably more typical of a real world situation.
Appendix Figure B-1 Examples of External Concurrence Relationships