Overview of Intelligent Systems & Operations Development

J. Pallix, G. Dorais and J. Penix
NASA Ames Research Center
Moffett Field, CA 94035

ABSTRACT

Approved for Public Release: distribution is unlimited

To achieve NASA's ambitious mission objectives for the future, aircraft and spacecraft will need intelligence to take the correct action in a variety of circumstances. Vehicle intelligence can be defined as the ability to "do the right thing" when faced with a complex decision-making situation. It will be necessary to implement integrated autonomous operations and low-level adaptive flight control technologies to direct actions that enhance the safety and success of complex missions despite component failures, degraded performance, operator errors, and environment uncertainty. This paper will describe the array of technologies required to meet these complex objectives. This includes the integration of high-level reasoning and autonomous capabilities with multiple subsystem controllers for robust performance. Future intelligent systems will use models of the system, its environment, and other intelligent agents with which it interacts. They will also require planners, reasoning engines, and adaptive controllers that can recommend or execute commands enabling the system to respond intelligently. The presentation will also address the development of highly dependable software, which is a key component to ensure the reliability of intelligent systems.

Introduction

NASA continues to push technology limits with its missions. Increased functional goals often result in increased implementation complexity, making both success and affordability that much harder to achieve. Moreover, it is accepted that not all risks can be mitigated during design: software and components will fail or degrade; operators will make mistakes; and operating environments are uncertain. In addition, the state of the system and its environment may dynamically increase control complexity or decrease reaction times such that traditional control means are inadequate. Development of critical intelligent system technologies that provide resiliency will enable future systems to adapt and recover from these unanticipated problems. Significant improvements in critical technologies, including those related to software engineering and verification, will be required to reduce both cost and risk in future NASA missions.

Mission Failures

In order to establish intelligent technology development priorities that address cost and risk reduction in future missions, many programs have carried out studies of recent mission failures. Common risk areas have been identified through classification of causes of mishaps from NASA, DoD and commercial accident reports. Sources of information included the actual accident case studies as well as failure analysis studies carried out by the Aerospace Corporation[1], Boeing[2], ARC[3], and others. Publications that detail the lessons learned from software related mission failures[4],[5] have also provided valuable insight into the requirements for developing the intelligent systems of the future.

The Aerospace Corp. examined nearly 4000 launches from 1957[1]. Based on the analysis results, this paper recommends enhancements for launch vehicles, including avionics redundancy, software and integrated system testing. In a smaller scale study on Mishap Cause Classification[3] carried out by NASA ARC, software function and propulsion and flight control subsystem failures ranked highest as initiators of mishaps. The mishap data presented in a Boeing report[2] indicate that loss of control in flight is the leading cause of fatal accidents and controlled flight into the terrain is the second leading cause in commercial airline accidents. Contributing factors in these accidents included software errors, component failures, lack of proper human-machine interactions (poor training), operator error on-board or on the ground (sometimes due to uninformed operator) and unanticipated operating environments. In fact, pilots have pointed out the demand for real-time, on-board integrated diagnostics that provide "answers not just clues" to the causes of multiple anomalous conditions occurring during flight. Nonintegrated caution warnings are not sufficient because pilots are responsible for cognitive integration that takes precious minutes and could mean the difference between life and death.

In general, software failures and lack of resiliency in operations and control systems have been identified as major causes of mission failures. Current technologies are not optimal for carrying out effective risk mitigation strategies as they lack significant capability to assess system condition or to validate system performance. System robustness, redundancy and capability for rapid recovery are currently inadequate. Also, because there is not adequate investment in software engineering science and technology research, this field is not keeping up with the demanding requirements of increasingly complex systems[5].

Studies of past software/physical system failures help significantly to establish a baseline for predicting the problems that need to be addressed for future missions. The mishap studies have identified flight critical software as a high risk for current and future missions.[4] The discovery of errors in Space Shuttle flight control software is on-going and is among the highest risks that could potentially lead to loss of vehicle. Leveson states that, "since the shuttle started operation in 1980, 16 [crit 1] software errors have been discovered in the released software... these problems occurred despite NASA having one of the most thorough and sophisticated software development and verification processes in existence."[4] "Practical experience and empirical studies have shown that most safety-related software errors can be traced to the requirements and not to coding errors[4]". DOD reports concur with this perspective, stating that "great programmers will perfectly encode rotten requirements"[5]. Advanced software specification tools and V&V concepts will have significant impact on the success of future missions.

Recognizing that software does not work independently of the physical system in which it is embedded, lessons learned from software related failures[4,5] agree that research into software engineering science and technology is critical to the success of future missions which will continue to become more complex. "Safety is not a property of the software itself, but rather a combination of the software design and the environment in which the software is used: It is application-, environment – and system-specific".[4] DoD reports recommend that future systems focus on dynamic structures; a large number of tightly integrated and temporally distributed physical/information system components with reconfigurable interconnection. This will require significant investment in design theory technology. Embedded software is important due to the growing integration role of information technology across all vehicle platforms. Integration of physical and information sciences will allow design of software for achieving physical behavior and will allow software to absorb change in physical systems. Ultimately, the capability will exist to build and integrate physical systems dynamically from spatially distributed components which will not only affect the success of future missions but will ensure significant cost reduction in cross platform implementation.

Intelligent system technology requirements derived from the mishap studies, indicate that the most important physical component developments required to significantly reduce risk in future missions include:

1. Intelligent/autonomous flight control technologies
2. Intelligent/autonomous engine technologies
3. Advanced diagnostics and prognostics (includes development and implementation of sensor systems)
4. Advanced human-machine interfaces (includes communication technologies)

These studies also indicate that investment should be made in the development of software engineering tools and validation and verification concepts. Software engineering tools that would provide the highest return on investment are:

- Requirement Engineering Tools
- Safe Software Design Tools
- Verification and Testing Tools
- Tool Selection and Tuning methodologies
- Distributed/Collaborative Software Engineering Tools

Intelligent Systems & Operations for Risk Mitigation and Cost Reduction

The ultimate intelligent vehicle systems of the future will be able to carry out a system-level self-assessment and perform real-time adaptive control based on hazards encountered. The intelligent vehicle envisioned will be capable of planning and executing its mission, managing its health and scheduling its own repairs. Development and implementation of safe communications/interactions of the system with humans (on board or on the ground) is critical to the success of intelligent systems and operations.

Intelligent systems will require the development of integrated autonomous operations and low-level adaptive flight control technologies to direct actions that enhance the safety and success of complex missions despite component failures, degraded performance, operator errors, and environment uncertainty. Inherent in this endeavor is monitoring, management, and ultimately, control using a multi-level approach that progresses from components, to subsystems, to integrated systems. This includes integrating high-level reasoning and autonomous capabilities with multiple subsystem controllers for robust performance. Future intelligent systems will use models of the system, its environment, and other intelligent agents with which it interacts. They will also require planners, reasoning engines, and adaptive controllers that can recommend or execute commands that enable the system to respond intelligently. Automation carries serious human factors risks, however, including overreliance on automated fault diagnoses, information overload, and mode confusion. These risks can be retired via a human-centered approach to the design, development, and evaluation of advanced user interface concepts.

Figure 1 is a notional representation of the integration of intelligent technologies into vehicle hardware and operational systems. The key technology areas that must be integrated with the operational system are generically described as 1) Fault Detection, Isolation and Recovery (FDIR or sometimes thought of as IVHM), Human-Guided Operations, Intelligent Controls, Data Architectures, and Sensor Development. Figure 2 breaks out the technology area descriptions into more detail.

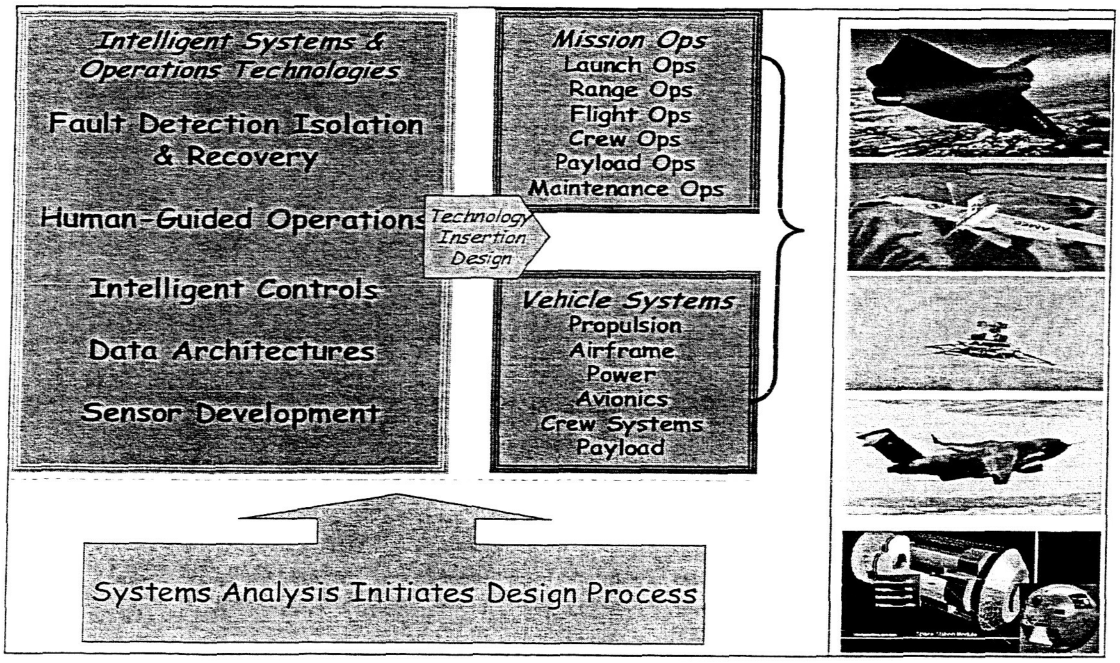

Figure 1. Notional representation of the integration of intelligent system technologies with vehicle operational systems

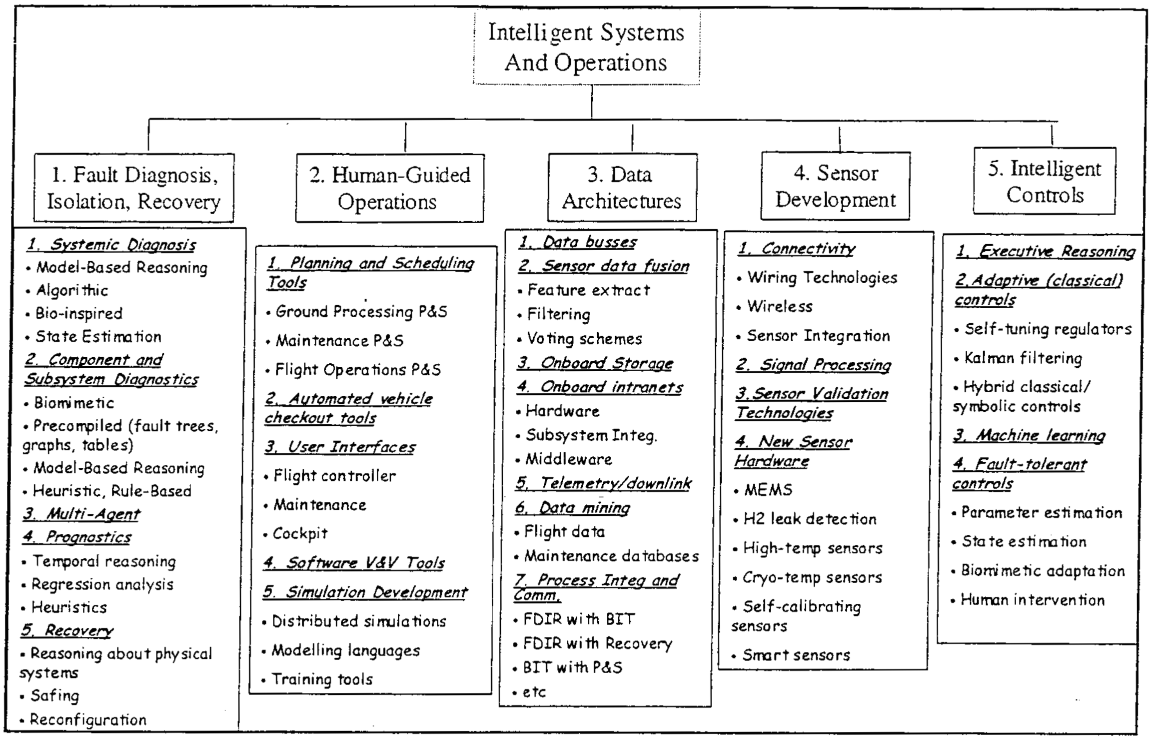

Figure 2. Detailed breakdown of Intelligent Systems Technologies

IS&O Developments for Major Subsystems

The subsystem technology areas of highest priority for development and integration within the IS&O paradigm include intelligent flight and propulsion control and the human-machine interfaces required to safely operate and maintain the selected autonomous/intelligent system lifecycle test platform. System intelligence is currently confined to limited, unintegrated health assessment, low level autonomy and software systems that are designed and verified using relatively ad-hoc practices. The national IS&O effort will eventually bring today's low level autonomy technologies and ad hoc software engineering processes to a future vehicle capable of operating indefinitely without human intervention. Major technology investment areas that will enable IS&O include the following:

- Intelligent and Adaptive Flight Control Research will endeavor to develop neural network technologies that can automatically compensate for damaged or malfunctioning aircraft. Such next-generation neural flight controllers will be capable of automatically compensating for a broad spectrum of damage or failures, controlling remote or autonomous vehicles, and reducing costs associated with flight control law development (this avionics technology will be demonstrated on the C-17 in FY04). Autonomous Propulsion Control Research will deliver technologies for autonomous accomplishment of propulsion system "monitoring", "diagnosing" and "adapting" functions which are critical for enabling highly or fully autonomous vehicle operation. The three main elements to this research are: 1) Self-Diagnostic and Self-Prognostic Technology, 2) Self-Optimizing Engine Control, and 3) Integrated Controls and Diagnostics.
- The Human-Machine Interface Research will attempt to retire human factors risks associated with incorporating advanced technologies into system control and health management via a human-centered approach to the design, development, and evaluation of advanced user interface concepts.

Note: The latter research will produce crosscutting technologies applicable to all vehicle programs whether manned or unmanned. The complex and dynamic nature of aerospace vehicle systems raises the distinct possibility of a systems malfunction during flight. Today, the primary responsibility for fault management lies with the crew or ground personnel. In the next generation of vehicles, intelligent systems technologies are expected to automate much of the fault management process and enable "at-a-distance" real-time fault management for remotely operated or autonomous vehicles. Incorporating these advanced technologies into the fault management process carries human factors risks, including overreliance on automation, information overload, poorly designed user interfaces, and mode confusion: insufficient understanding of automated activity and/or insufficient awareness of the effects of automated actions on systems functioning.

Integration of IS&O Technologies (system software framework)

We define an intelligent-autonomous system as an integrated set of components that achieves specified goals without requiring human intervention to make decisions that would otherwise require human intelligence to make properly. We define an intelligent agent as an intelligent-autonomous system, or network of such systems, that senses its environment, reasons based on internal state (stored information that may include sensory data, goals, procedures, constraints, models, etc...), and acts by changing its internal state and the state of its environment in order to achieve its goals without violating its constraints. By changing its internal state, an agent may "learn" such that its performance is a function of its experience. That is, while it acts, it learns from its environment in order to improve the effectiveness of its actions with time. However, learning is not a necessary agent capability and can generally be "turned off" as required. Intelligent agents that integrate technologies outlined above may have the following desired capabilities for resilient vehicle control:

- Maintain stability or recover in the presence of vehicle and environmental uncertainties and changes.
- Reconfigure control system to compensate for damage/failure to control effectors.
- Gracefully degrade performance, while maintaining functionality to the greatest extent possible, if unable to fully recover from damage/failure.
- Optimize achievable control performance through integration of motion control, power, propulsion, and structural subsystems.

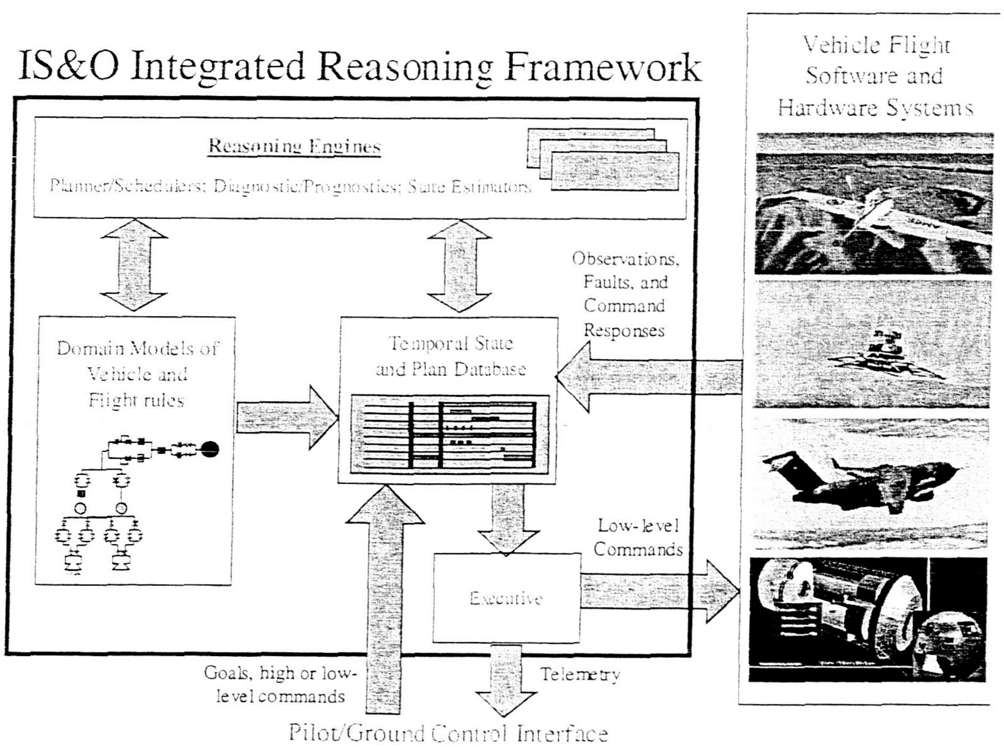

Figure 3. An Integrated Reasoning Framework for Autonomous Vehicle Control

Figure 3 depicts a general purpose integrated reasoning framework for an intelligent agent controlling a vehicle. Goals or commands are received from a crew member, remote operator, or another system and are entered into the temporal state and plan database where they are sequenced and decomposed by various reasoning engines into primitive commands as needed. Sensory information and subsystem feedback from previous commands, which may be fused or used by reasoning engines to determine states that are not directly sensed, e.g., a diagnosis, are also entered into the database. The reasoning engines and the state and plan database use the models, rules, etc..., to insure that any command sequence in the database does not violate a constraint. As a result, command sequences will adapt to changes in the system and environment. The executive sends the primitive commands to the specified control system at the appropriate time and generates the requested level of telemetry to be broadcast. An example of an intelligent agent using this framework to autonomously control a vehicle is the Remote Agent used to control the ion-engine propelled Deep Space One spacecraft in 1999[6].

Benefits provided by this framework include:

- Plans can be dynamically updated at execution time. This allows the agent to respond to change in the system or environment, e.g., reconfiguring itself due to the failure of a component, and achieve the system goals or a subset of them.
- Domain knowledge is separated from the program code. This allows the vehicle, mission, and environment to change without having to change code and revalidate it. Moreover, depending the on domain knowledge language, it may be provably verifiable rather than requiring exhaustive tests. This is particularly valuable since for many intelligent systems, there are far too many execution paths through the code to ever exhaustively test each one by executing it. As a result the cost of reimplementing this agent on another vehicle or in another domain can be significantly reduced since all the coded components can be reused without change.
- Sensing is unified so that decisions are made based on a consistent state of the system and its environment. Otherwise, for example one reasoning engine may continue a command sequence based on the belief that a switch is on because it previously commanded it on, whereas another reasoning engine may infer that the switch is off and make decisions accordingly.
- Commanding is unified so that conflicts are resolved prior to execution. An agent may have several reasoning engines and be flexible so that it can dynamically generate plans to achieve complex goals. However, there is value in being able to enter in one location, a constraint, e.g., Switch A and B shall never be on at the same time, so that regardless of the reasoning engines used, the agent will never command switches A and B to be on at the same time.
- The framework is modular allowing components to be exchanged without redesigning the system. This is of particular value since it facilitates the integration of specific intelligent system technologies without requiring it to meet all the requirements of the vehicle that it will be used to control.
- The framework is amenable to being distributed over multiple computers and control multiple vehicles. In order to achieve low latencies between sensing and acting, particularly when this may involve computationally intensive activities such as planning and image recognition, it is helpful to distribute the computational load over multiple processors in a straight forward manner. Moreover, the framework can be used to control multiple vehicles in a manner similar to how it would control a single vehicle with multiple subsystems or in other manners more similar to how people might achieve a goal by negotiation.

Figure 5, depicts a generic architecture for an intelligent flight control system (IFCS) that could be integrated into the system level framework described above. There may be multiple subsystem architectures to deal with on any platform. Note that the IFCS architecture shown in the figure is generic in the sense that it can be reused on many vehicle platforms by simply changing the reference model and the model used for parameter estimation. This type of architecture will significantly reduce the cost of control system development for future vehicle systems.

As described above, the ability to validate an implementation of this framework is dependent on the modeling language(s) used. Moreover, the more expressive a language is and hence, the more easily a wide variety of systems, physics, tasks, and constraints can be described, the more difficult it is to validate that language. Validation is discussed further in the next section.

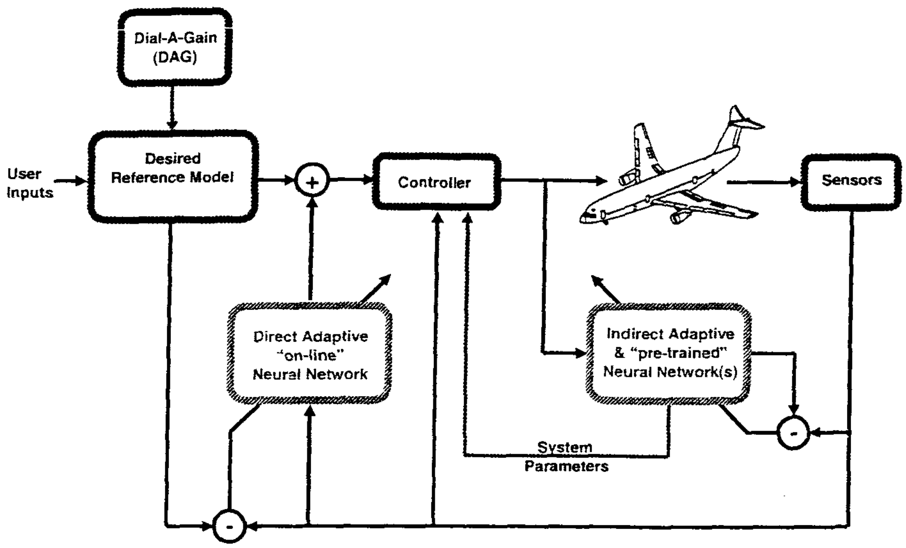

Figure 4. Generic Architecture for an Intelligent Flight Control System

Software Engineering and Validation

Software is playing an increasing role in NASA's mission critical systems. It is often instrumental in assuring that systems meet critical mission requirements and maintain safe operations. However, this increased dependence on software introduces new risks into mission and system design, which are not always understood or properly managed. Misunderstandings regarding the exact nature and criticality of software requirements and, consequently, inappropriate software development practices played a lead role in the failure of both the Mars Climate Orbiter and the Mars Polar Lander, demonstrating that these risks are real. It is clear that to enable mission success, software risks must be identified, and appropriate measures must be taken to eliminate or mitigate these risks. However, we currently lack cost-effective methods for managing the risk of using software to control complex systems in practice. Figure 5 shows the growth in software complexity for NASA missions between 1977 and the present. Next generation space science missions are increasingly relying on highly complex software to achieve greater levels of autonomy and robustness. As mentioned previously, managing the risk of the ever-increasing complexity of intelligent control and operations software will require significant investment in software engineering science and technology.

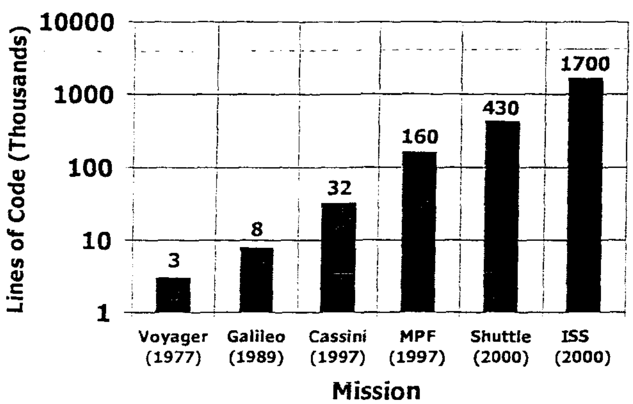

Figure 5. Increase in lines of code used in NASA missions since 1977.

The development of dependable software requires software engineering activities to respond to changes in critical requirements that effect system risks and have the capability to adapt based on feedback from development and operations to support process improvement. Current software engineering practices are largely ad-hoc resulting in brittle practices with single points of failure, allowing critical errors to enter products and escape detection. Design of more resilient software engineering practices will require 1) empirically validated criteria for measuring the dependability of software products, 2) a clear understanding of how software development methods impact software dependability, and 3) cost-effective tools to enable these methods to be used in practice.

Empirical evaluation of proposed solutions for software dependability (both methods and tools) has been lacking because of the difficulty for individual researchers to perform experiments at realistic scale. This has hampered both the science and the technology transfer/infusion for software dependability solutions. A simultaneous effort should be made to identify dependability attributes of software artifacts, engineering practices, and operational environments and to identify measures for these attributes. With these measures, the causal relationships between technical decisions and dependability outcomes can be measured and predicted. Efforts are necessary to create notations to support the description of software artifacts and dependability. There is already a significant effort underway in the software engineering community, to define and prove engineering techniques, tools, design principles, practices and processes to support affordable creation of dependable systems and to disseminate the processes and practices in educational programs. If these systems are not made more affordable and easier to implement, the risk of future missions will increase.

The current best practices in software development include a range of techniques for requirements and design modeling, software process management and software testing. These techniques alone are not effective for providing high-levels of confidence that software will behave safely - that intended or unintended software behavior will not lead to critical system failures. At each phase of software development, teams must apply and reformulate requirements and understand complex interactions between requirements: system requirements determine software requirements, software requirements influence software architecture, architectural constraints impact algorithm selection and implementation, and test cases are derived from requirements. This problem is compounded when software development for several interacting sub-systems is performed by different organizations. Without methods and tools to support the consistent specification and management of these requirements across life-cycle phases and organizations, there is high risk of introducing critical errors into the software.

Research will address the cost-effectiveness of software development techniques by tools that enable more effective and efficient software engineering practices within NASA. To provide this capability, tools will have the following properties:

- Semantically Well-Founded - tools accurately predict software behavior during all phases of the life-cycle
- Human Centered - tools are designed to improve the way that people and organizations work
- Information Rich - tools actively support the creation, application and maintenance of plans, models, knowledge and data

The main limitation of existing software engineering technology is that achieving high levels of dependability is very labor and resource intensive. Testing requires significant resources, but still does not provide adequate assurance for large complex systems. Commonly used graphical modeling notations, such as the Unified Modeling Language, do not have well-defined semantics and therefore support only limited automated analysis. Formal methods based on mathematical models can support extensive high-fidelity and automated analysis capabilities across the software lifecycle. However, these techniques are very expensive to use because they require extensive domain knowledge and familiarity with the mathematical modeling notations. The tools themselves often have serious limitations such as only providing support for limited modeling languages which do not capture the complexities of real software; methods which are idealistic and cannot be integrated into a software engineering process of realistic complexity; and stand-alone prototypes, which can only be used effectively in close collaboration with the tool developers.

Future research will support developments that address these limitations. Tools must be constructed so they are compatible with commercial modeling languages and tools being used within NASA projects. To be cost effective, they should support mixed-fidelity modeling and generate models from existing models and source code when possible. The tools developed should be designed to be human-centered and to be flexible regarding the software engineering processes that can be supported. They must also be intuitive enough to be picked up and used with a reasonable amount of training by the average NASA software engineer or contracted software engineer. They should also not overly burden the Faster-Better-Cheaper–but–still-Safe budgets and schedules levied on programs in trying to apply these new techniques.

Many current tools also lack support for collaboration among development teams. This is a critical issue for NASA when many software projects are managed and performed by multi-disciplinary teams spanning several NASA and contracting organizations. The recent significant advances in collaboration technology are beginning to be brought to bear on this technology. However, to really make an impact on the cost of software development, the application of collaborative technology in software engineering must go beyond the simple use of these technologies to support basic communication tasks. Investigations are underway to develop software architectures for task-aware applications which provide customizable support for the different roles that stakeholders play in collaborative tasks, such as requirements elicitation and design reviews.

Perhaps more important than making individual methods and tools more cost effective, it is necessary to improve the way that tools are used together in development processes. We are currently lacking technology to supports the cost vs. risk and benefits trade-offs that occur during software engineering process design and tool selection. These tools will help us understand and model the relative cost-effectiveness of various techniques (algorithms, designs, processes, and tools) for avoiding and detecting classes of system failures. There are currently a number of investigations of the use of several technologies to address this problem, including risk-management and decision support technology, software cost and reliability modeling, and organizational simulation tools.

Summary

To achieve the ambitious objectives for deployment of intelligence into future systems it will be necessary to implement integrated autonomous operations and low-level adaptive flight control technologies to direct actions that enhance the safety and success of complex missions despite component failures, degraded performance, operator errors, and environment uncertainty. This includes the integration of high-level reasoning and autonomous capabilities with multiple subsystem controllers for robust performance. Future intelligent systems will use models of the system, its environment, and other intelligent agents with which it interacts. They will also require planners, reasoning engines, and adaptive controllers that can recommend or execute commands enabling the system to respond intelligently. Key to the successful integration and implementation of these technologies is the development of highly dependable software.

Acknowledgements:
The authors would like to thank, J. Kaneshige, B. Glass, M. Shirley, and T. Panontin, M. Gaunce for valuable discussions that were incorporated into this overview.

---

[1] Chang/Aerospace study (AIAA Journal of Spacecraft & Rockets)

[2] Statistical Summary of Commercial Jet Airplane Accidents (Worldwide Operations 1959-2000); StatSum@PSS.Boeing.com

[3] ARC MCC ECS L1 final report, Pantonin et al. (to be released)

[4] Safeware: System Safety & Computers (Leveson)

[5] DoD SW Engineering Science and Technology Summit Report

[6] http://nmp-techval-reports.jpl.nasa.gov/DS1/Remote_Integrated_Report.pdf