Safety-critical Partitioned Software Architecture

A Partitioned Software Architecture for Robotic Spacecraft

Gregory Horvath
Ferner Cilloniz-Bicchi
Seung Chung
Dan Dvorak
Dave Hecox

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA

2010 Flight Software Workshop
December 9 2010

Copyright 2010 California Institute of Technology. Government Sponsorship acknowledged.

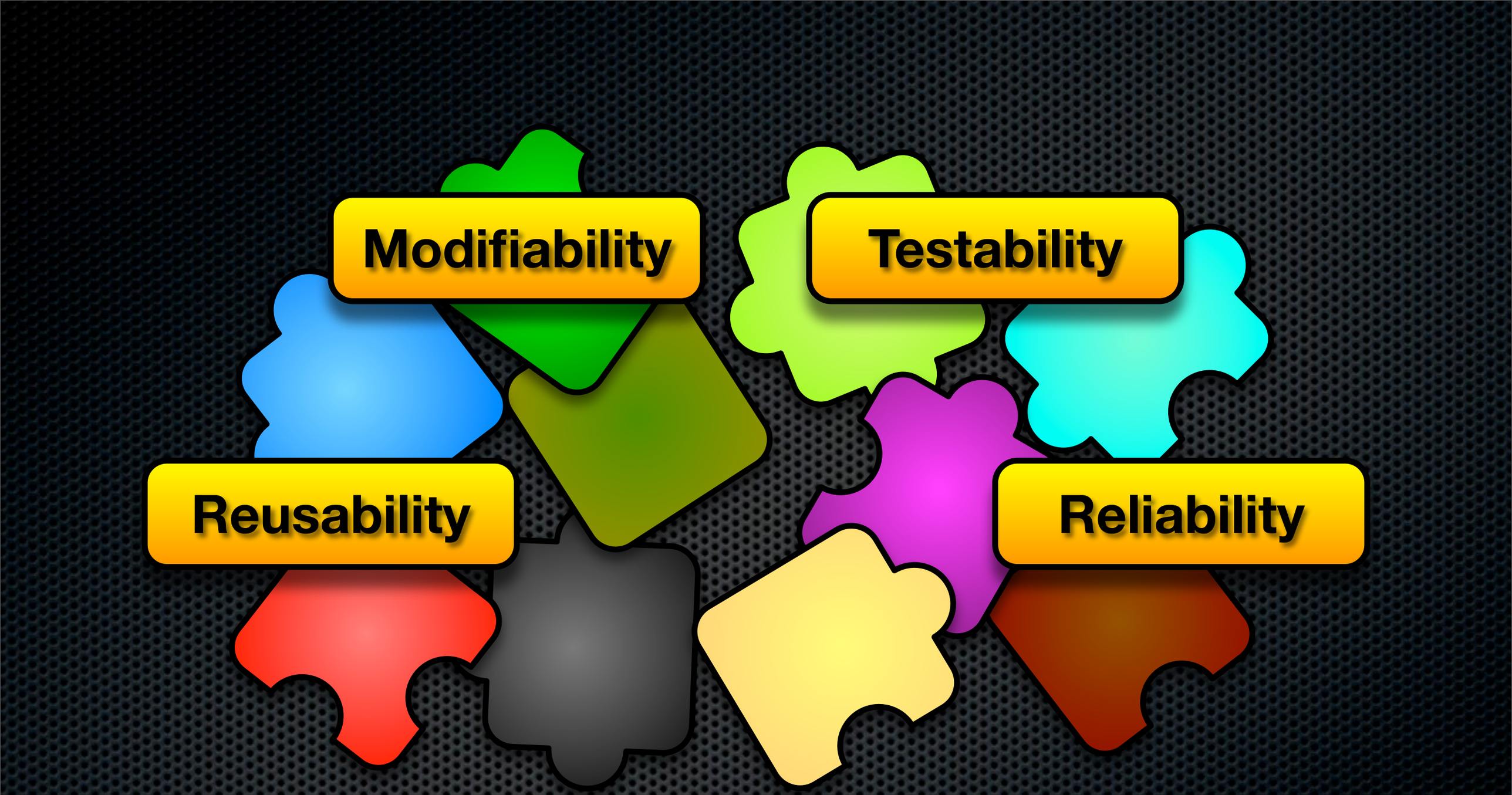
Motivations

- Partitioned OS offers many appealing benefits
  - Memory protection (space partitioning)
  - Execution guarantees (time partitioning)
  - Mature standard with many examples of successful application (ARINC 653)
  - Increased fault containment
- How to best leverage these benefits ***throughout the entire lifecycle*** while accommodating for and adjusting to new constraints imposed by the platform

Approach

- Identify relevant **quality attributes** of the resulting architecture
- Map quality attributes to lifecycle development activities
- Identify key architectural features to support the desired quality attributes
- Provide artifacts to guarantee adherence to architecture
  - Modeling and analysis tools
  - Framework software
  - Development processes and procedures

Quality Attributes

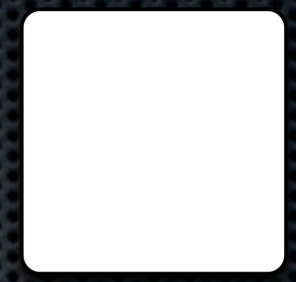


Quality Attributes

Quality Attributes

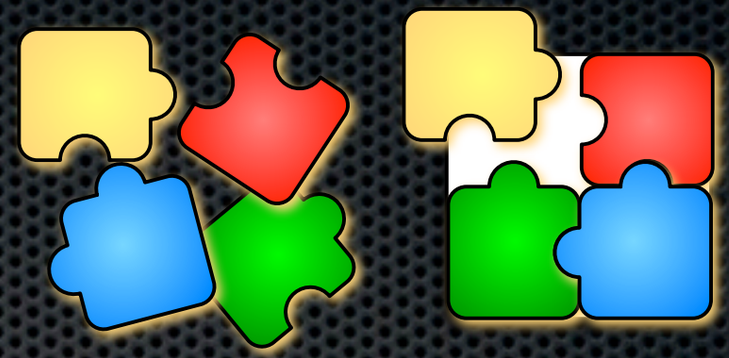
- Quality attributes, or QAs, are *non-functional* requirements on a system.
  - QAs can capture run-time or static properties, business requirements, etc.
- Partitioned software architectures are uniquely able to support and enhance the following QAs
  - Reusability
  - Modifiability
  - Testability
  - Reliability

QA: Reusability



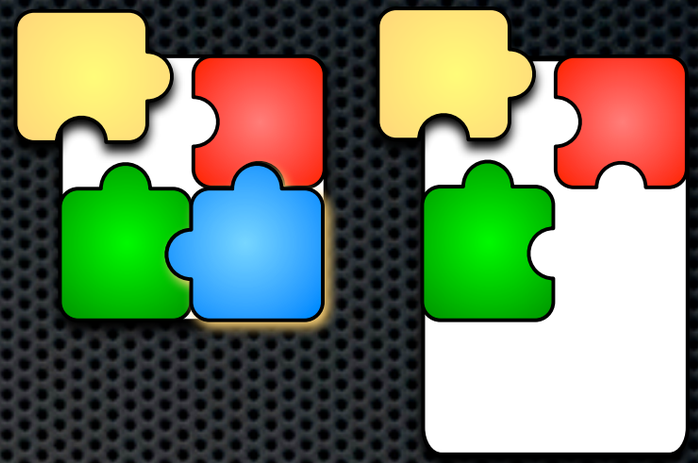
- Improve **Reusability** through well defined interface specification
  - Design modularly with well defined interface specifications
  - Design with ‘plug-n-play’ paradigm
- ARINC 653 Ports and Channels
  - Port data specification
  - Queueing Port size specification
  - Sampling Port timing specification
  - Port read/write timing specification
  - Port connecting Channel specification

QA: Reusability



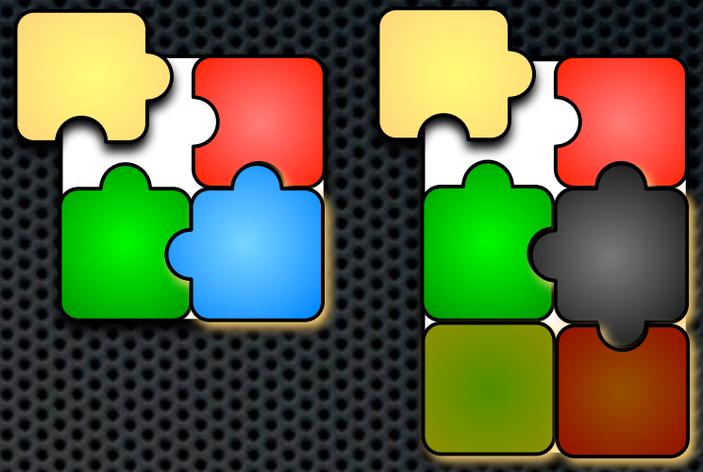
- Improve **Reusability** through well defined interface specification
  - Design modularly with well defined interface specifications
  - Design with ‘plug-n-play’ paradigm
- ARINC 653 Ports and Channels
  - Port data specification
  - Queueing Port size specification
  - Sampling Port timing specification
  - Port read/write timing specification
  - Port connecting Channel specification

QA: Modifiability



- Improve **Modifiability** through function composition
  - Change desired/required functionalities
  - Change the grouping of functionalities based on the concerns (e.g. safety, timing, etc.)
- ARINC 653 Partitions and Processes
  - Regroup Processes into the appropriate Partitions
  - Update the Ports of the Partitions
  - Update the Channels that connect the Ports of the Partitions

QA: Modifiability

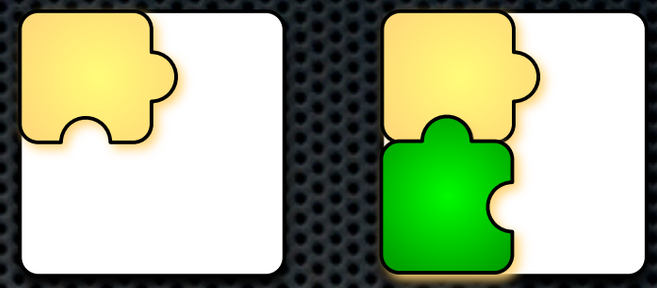


- Improve **Modifiability** through function composition
  - Change desired/required functionalities
  - Change the grouping of functionalities based on the concerns (e.g. safety, timing, etc.)
- ARINC 653 Partitions and Processes
  - Regroup Processes into the appropriate Partitions
  - Update the Ports of the Partitions
  - Update the Channels that connect the Ports of the Partitions

QA: Testability

- Improve **Testability** through simplified integration and test
  - Test each component in isolation
  - Test components at varying levels of integration
  - Test critical components at a higher certification level
- ARINC 653 Partitions
  - Test each Partition in isolation
  - Test the interactions of Partitions
  - Test critical Partitions at a higher certification level

QA: Testability

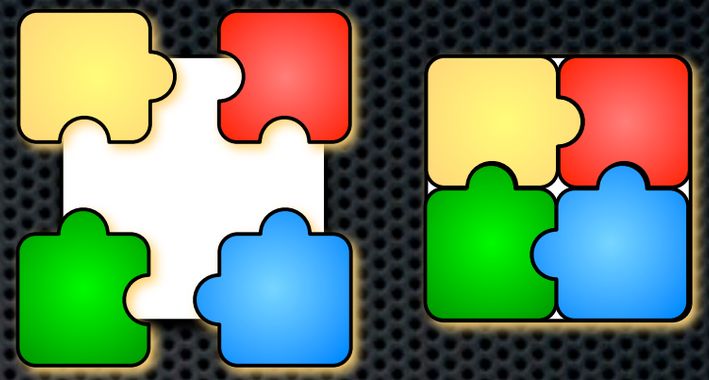


- Improve **Testability** through simplified integration and test
  - Test each component in isolation
  - Test components at varying levels of integration
  - Test critical components at a higher certification level
- ARINC 653 Partitions
  - Test each Partition in isolation
  - Test the interactions of Partitions
  - Test critical Partitions at a higher certification level

QA: Reliability

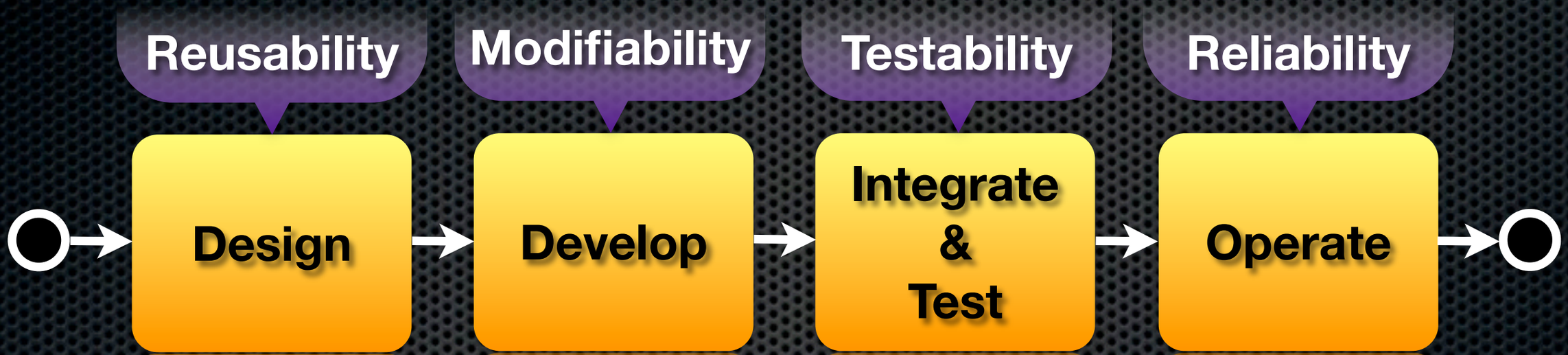
- Improve Reliability through isolation
  - Isolate independent functionalities
  - Isolate critical functionalities/Isolate less mature functionalities
  - Isolate the faults of a function
- ARINC 653 Partitions
  - Isolate the memory that can be physically used by a Partition
  - Isolate the time that can be taken up by a Partition
  - Overall improvement to fault isolation through time and space partitioning

QA: Reliability



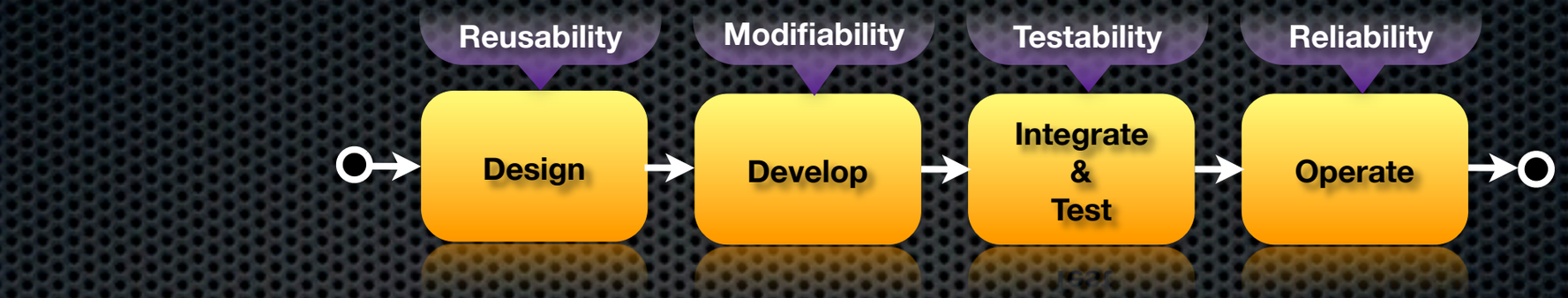
- Improve Reliability through isolation
  - Isolate independent functionalities
  - Isolate critical functionalities/Isolate less mature functionalities
  - Isolate the faults of a function
- ARINC 653 Partitions
  - Isolate the memory that can be physically used by a Partition
  - Isolate the time that can be taken up by a Partition
  - Overall improvement to fault isolation through time and space partitioning

Mapping Quality Attributes to Lifecycle



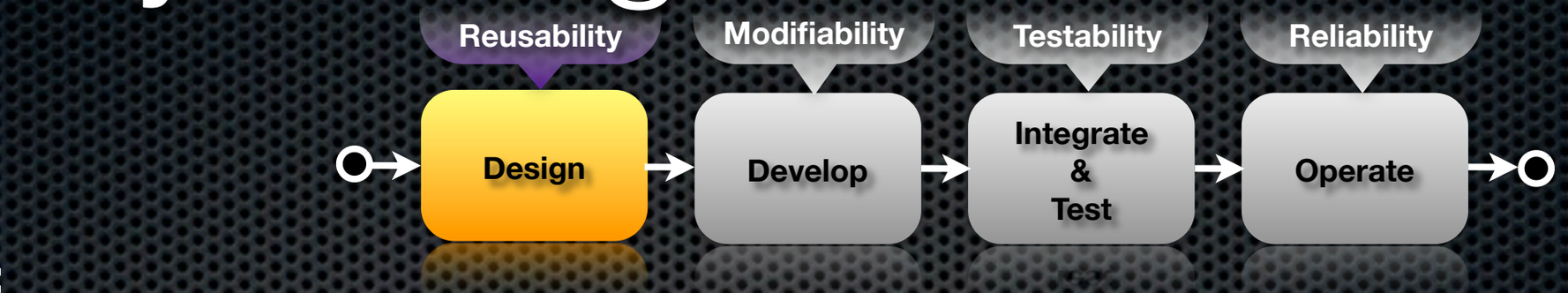
Mapping Quality Attributes to Lifecycle

Quality Attributes in Practice



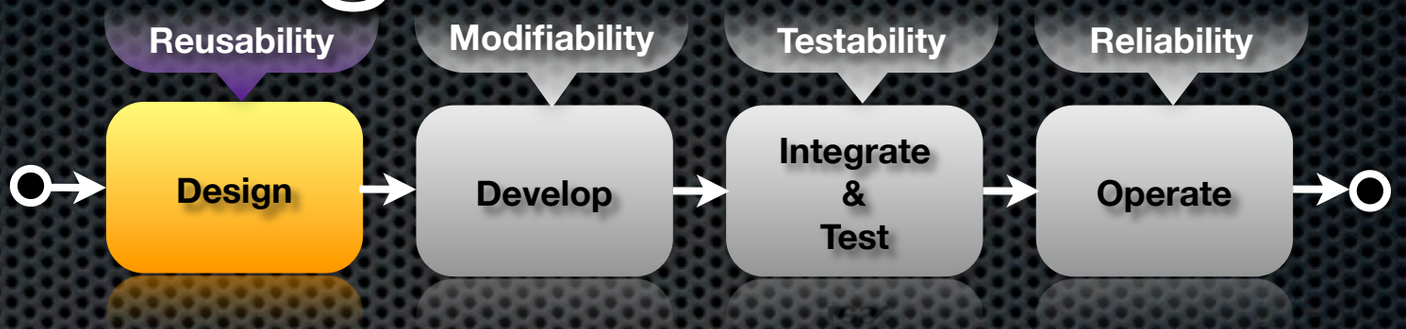
- We've defined the qualities that we'd like our systems to embody. Now you may ask:
- **How do we use this information to build robust systems?**
  - How do these QAs help us build reliable systems?
  - How are these QAs expressed during design and construction?
- We'll also touch on supporting elements that can help to ensure adherence to these QAs

Reusability during Design



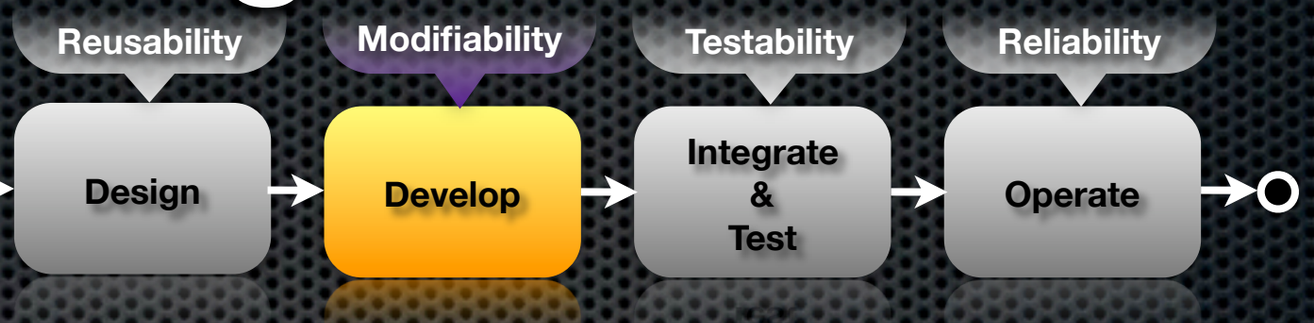
- **Common Problems:**
  Design decision to create new or reuse software components
  - **Create reusable software components:**
    Typically minimal or no consideration is made in creating a reusable software components.
  - **Reuse existing software components:**
    Typically the difficulty of reusing existing software components is largely underestimated.
- **Concerns:**
  Difficulty of creating reusable software and Uncertainties associated with reusing software
  - **Interoperability uncertainty:**
    Software components are typically designed with their own specific interface and behavior specifications that are generally poorly documented. Thus, the interoperability of the existing software components with the new components is uncertain.
  - **Cost & Schedule uncertainty:**
    Due to the uncertainty of the interface and behavior specifications of the existing software components, the cost and the schedule associated with interfacing with the existing software component is difficult to predict.

Reusability during Design



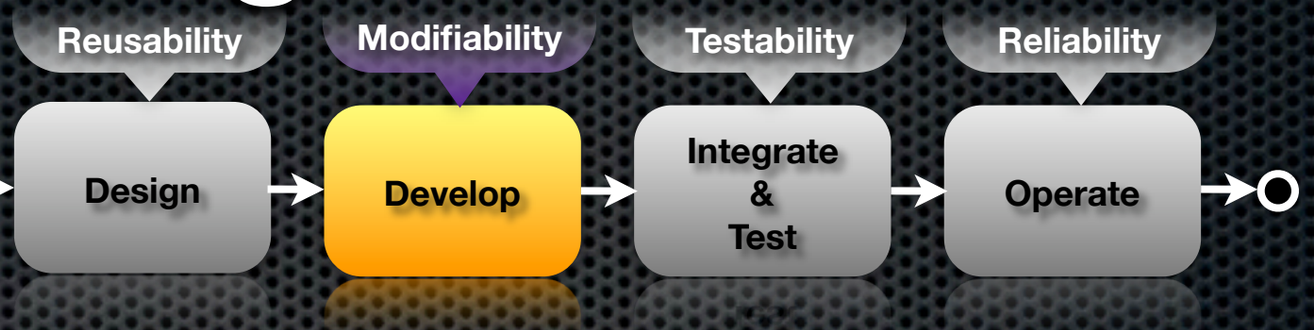
- **For Partitioned OS:**
  - **Create reusable software components:**
    New software component is built in a partition, with specific ports and timing requirements.
  - **Reuse existing software components:**
    This software component can be reused as was designed as long as the port and the timing requirements are satisfied.
- **Benefits provided by Partitioned OS:**
  - **Mitigate interoperability uncertainty:**
    Partition interoperability is well defined by the port and timing specifications.
    - Inter-partition communication occurs over statically-defined channels
    - Strong interface specification minimizes the possibility of runtime errors introduced by improper communication among various tasks within the system.
    - Statically defined communication channels and data types allow for data flow and schedulability analysis before application is ever executed.
  - **Mitigate cost & schedule uncertainty:**
    Well defined interface and timing specifications lead to more predictable reuse costing and scheduling.
    - **Required time allocation:**
      During the design time, the time allocations can be scheduled statically.
    - **Required I/O ports:**
      Design the port and channel configuration and memory allocations statically.

Modifiability during Development



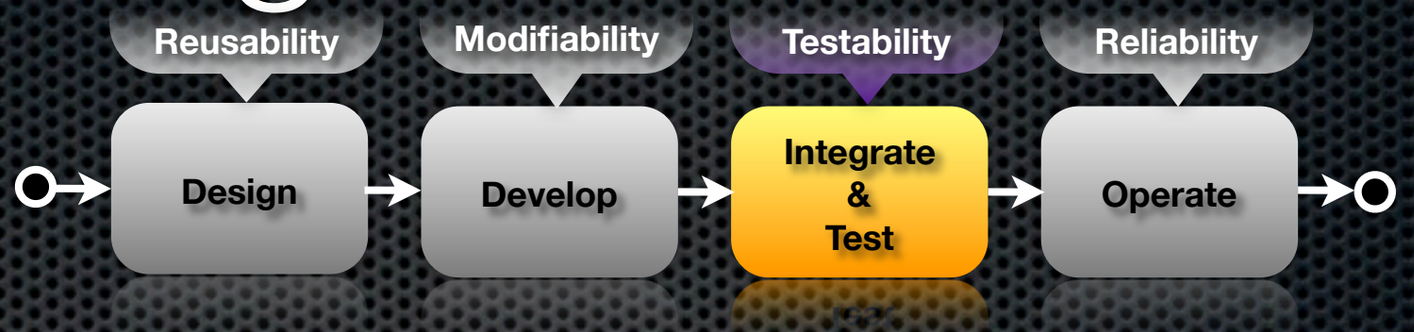
- **Common Problems:**
  Changes in the requirements, design, schedule, budget, etc.
  - **Addition or Removal of features:**
    Tightly coupled applications with brittle or weak interfaces make modification of functionality a tedious and error-prone process
  - **Redistribution/refactoring of features:**
    This is a unique challenge associated with developing partitioned applications. Namely, how do we handle the scenario where we decide that our partitioning must be modified?
- **Concerns:**
  Uncertainties due to design and development changes
  - **Integrity uncertainty:**
    Adding or removing features can affect other parts of the software and determining the extent of the effects can be challenging.
  - **Cost & Schedule uncertainty:**
    With the uncertainty in the effect of the changes, the development cost and schedule also becomes uncertain.

Modifiability during Development



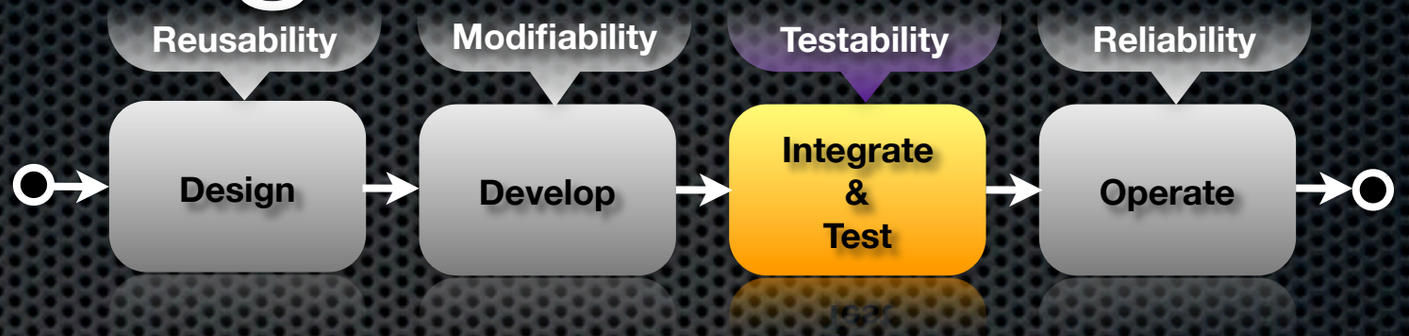
- **For Partitioned OS:**
  - **Addition or Removal of features:**
    Add and Remove features by adding or removing partitions or processes.
  - **Redistribution of features:**
    Redistribute features by splitting or combining partitions.
- **Benefits provided by Partitioned OS:**
  - **Mitigate integrity uncertainty:**
    Making changes to the relevant partitions in isolation does not affect other partitions.
  - **Mitigate cost and schedule uncertainty:**
    Isolating the changes and knowing how the interfaces must change accordingly leads to predictable cost and schedule.
    - **Number of new partitions to be added:**
      Change memory and time allocation. For removed partitions, simply remove the allocated memory and time.
    - **Number of ports and channels to change:**
      Change the port and channel configuration and memory allocations.

Testability during I&T



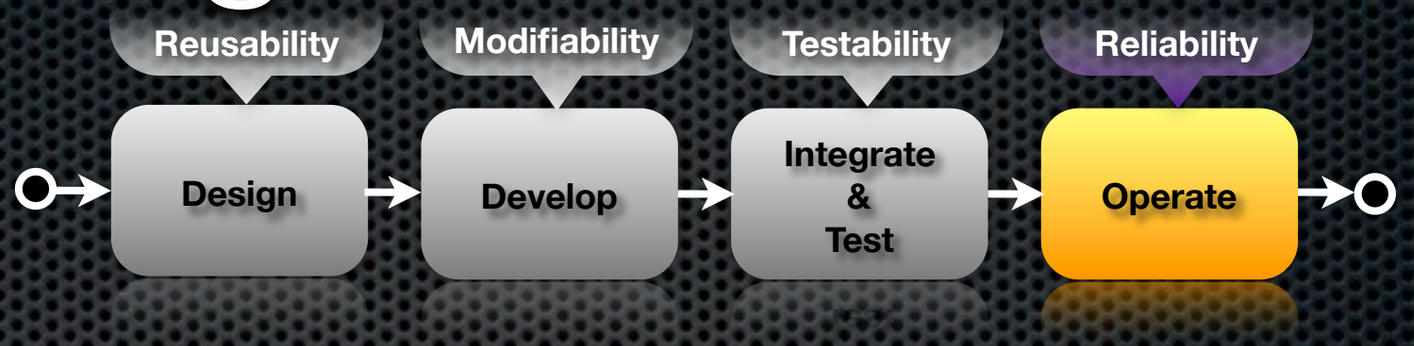
- **Common Problems:**
  Rolling up many features into one large delivery
  - **'Big bang' integration:**
    Many problems are caught during this integration step, and such problems may lead to late feature changes.
  - **Testing fully integrated deliveries:**
    Debugging a fully integrated software can be exponentially more difficult. Though very difficult, such integrated test is required since one part of the code can inadvertently affect other parts of the code.
- **Concerns:**
  Uncertainties due to the integration process and integrated software complexity
  - **Testing completeness uncertainty:**
    For fully integrated software, identifying all important test cases is impossible.
  - **Cost and Schedule uncertainty:**
    Much of the cost and schedule overrun occur during integration and testing phase.

Testability during I&T



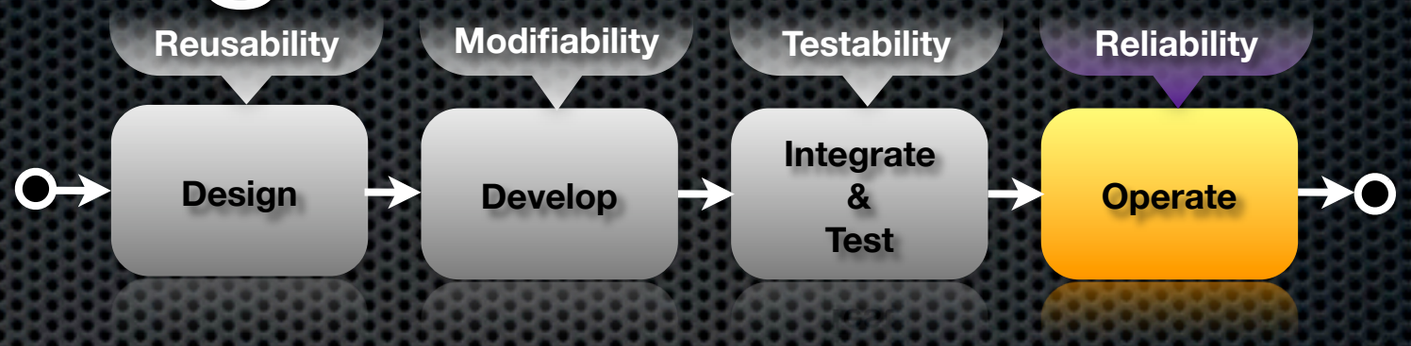
- **For Partitioned OS:**
  - **'Big bang' integration:**
    Integrate partitions as they become available.
  - **Testing fully integrated deliveries:**
    Test partially integrated partitions based on their interface specification and other stub partitions.
- **Benefits provided by Partitioned OS:**
  - **Mitigate testing completeness uncertainty:**
    Identifying the important test cases becomes much easier.
    - Test cases for each independent partitions
    - Test cases based on the coupling of the partitions, i.e. ports and channels are the only potential coupling among partitions.
  - **Mitigate cost and schedule uncertainty:**
    The complexity of partition coupling can help predict the cost and schedule associated with integration and testing.
    - **Number of independent groups of partitions:**
      Test each independent groups of partitions in isolation.
    - **Number of connections among interdependent partitions:**
      The number of required test cases increase at least exponentially with the number of connections.

Reliability during Operation



- **Common Problems:**
  Faults occurring during operations
  - **Fault isolation:**
    A fault in one software component may corrupt the memory region of other software components and steal processing times from others.
  - **Fault recovery:**
    Fault must be handled in a timely manner to assure mission success.
- **Concerns:**
  Lack of robust fault isolation and recover may risk the safety and reduce productivity
  - **Fault propagation:**
    Fault must be isolated and prevent it from causing a chain reaction of faults.
  - **Lack of robustness:**
    Fault recover software may also be affected by other faulty software components.
  - **Reduced productivity and functionality:**
    Less mature, but more advance algorithms are not allowed on board due to potential risk on critical components.

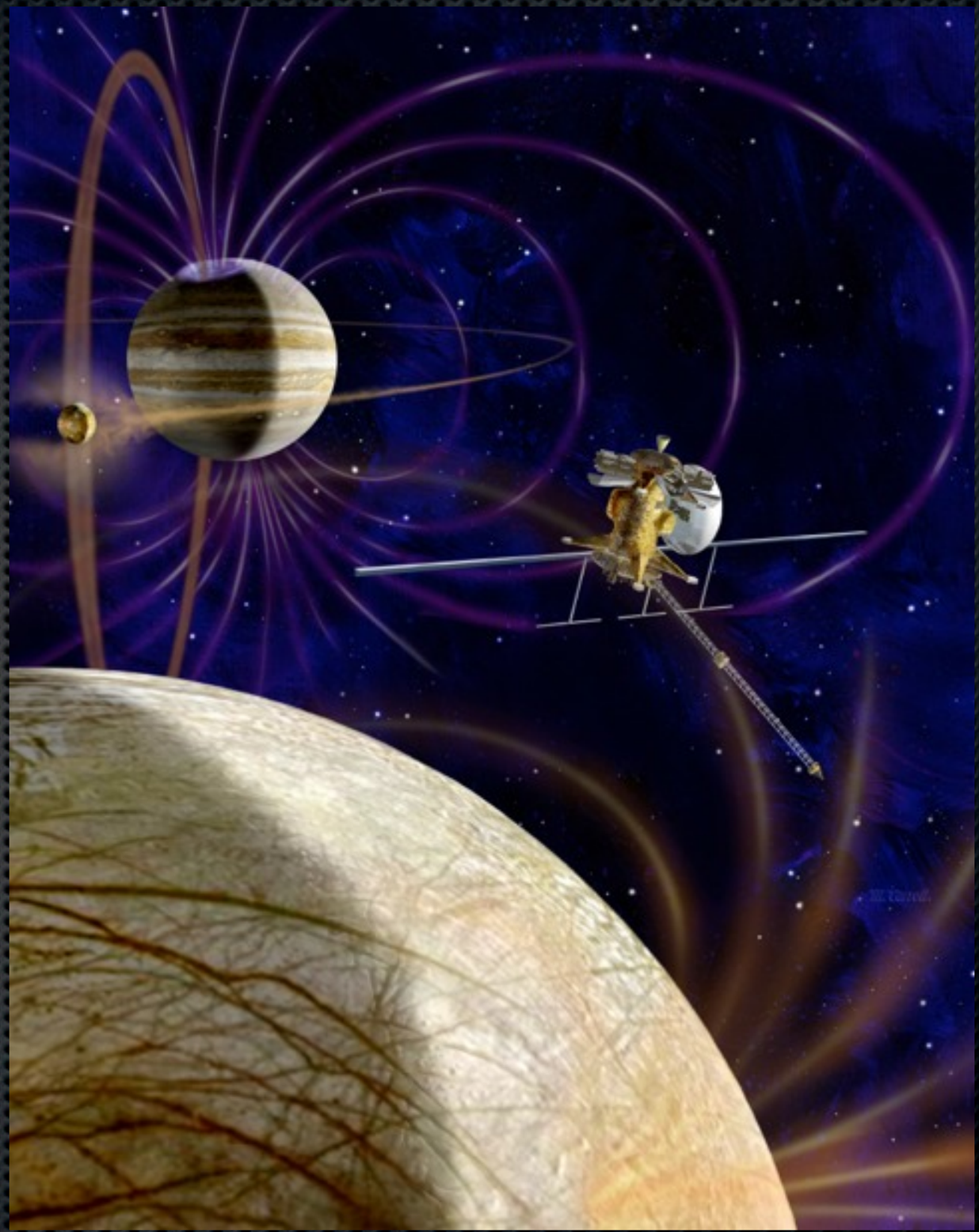
Reliability during Operation



- **For Partitioned OS:**
  - **Fault isolation:**
    Only way for faults to propagate from one partition to another is through the ports and channels. The memory and processing time of a partition is not affected by another faulty partition.
  - **Fault recovery:**
    Built-in health monitoring enables more robust software health management techniques.

- **Benefits provided by Partitioned OS:**
  - **Mitigate fault propagation:**
    Only way for faults to propagate from one partition to another is through the ports and channels.
    - Decreased susceptibility to problems arising in non-critical tasks (i.e. SEUs...)
    - The core functionality is protected against latent errors in less mature application code.
  - **Improve robustness:**
    Eases implementation of the ‘simplified backup’ strategy for achieving redundancy within the application.
  - **Improve productivity and functionality:**
    Integration of advanced on-board algorithms with reduced concern for the safety of the system.

What's Left?



This is all great, but...

- There is still work to be done before projects can effectively deploy software systems based on a partitioned OS
  - Process and Procedures
    - Software classification: need to unambiguously define the classification level of each function within the flight software application
    - Roles and responsibilities: Who is responsible for maintaining the system configuration? Who allocates memory and CPU time to each module?
  - Tools
    - System modeling and specification
      - Graphical system design, configuration file generation, some basic code generation
      - Data flow and schedulability analysis + simulation
    - System Characterization
      - Performance analysis (CPU/memory utilization, data throughput)
  - Framework Software
    - Promote reusability through useful abstractions
    - Codify useful design patterns for partitioned applications to ensure adherence to underlying architecture

Looking Further

- This work has only considered changes to the software system design
- Even greater robustness can be achieved by considering alternative avionics architectures
  - ARINC 659 (Backplane Data Bus for IMA)
  - Time-Triggered Bus Architectures
  - Passenger airframe style redundancy and voting (7E7)
  - …
- Future system designs should consider avionics trades within this space for greater benefits
- Related topic: stepping stone to multicore
  - How does the partitioned application paradigm map to multicore applications?
  - If the similarity is more than just conceptual, we can carry over many of the analysis techniques, tools, and processes for partitioned applications as we migrate to multicore solutions