Truly Curious: Examining the Rover Compute Element and the Mars Science Laboratory Software Development Process

David Keck

Abstract

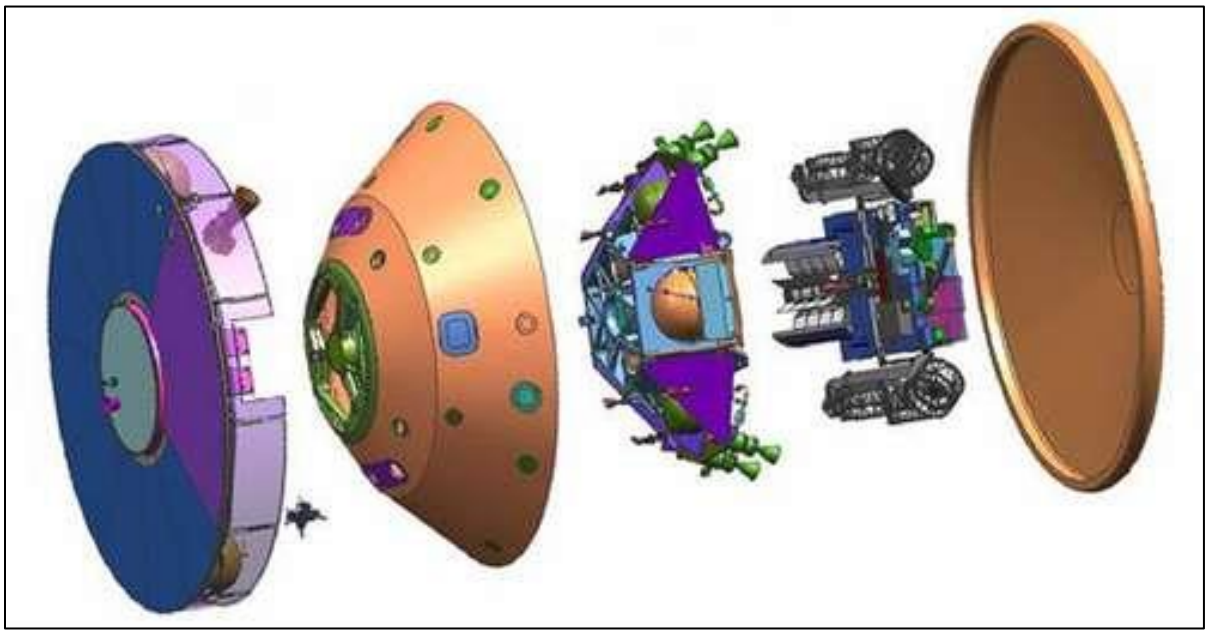
The Mars Science Laboratory (MSL), which contains the Curiosity rover, was launched on November 26, 2011. This mission successfully landed Curiosity in Gale Crater on August 6, 2012 (NASA/JPL 2013). Over the course of this mission, the spacecraft acted as three separate configurations. These include the cruise stage, entry, descent, and landing (EDL), and the surface operations or rover stage (Weiss 2012). The software that controls all of these stages was created by a team of about 35 computer scientists (Holzmann 2013) with an additional software testing team of 10+ people (Havelund 2009). The programs were coded in the C programming language and total 5 million lines of code (Cichy 2010). This paper will examine the creation and testing of this software, as well as the rover compute element (RCE) which the software operates on.

Keywords: NASA, Curiosity, Mars, robust software engineering, autonomous software

Truly Curious: Examining the Rover Compute Element and the Mars Science Laboratory Software Development Process

Developing bug free software can be hard. It gets even more complicated when the software that you are writing will be responsible for the cruise, entry, descent, landing, and daily operations of NASA's Curiosity rover. How can such a complex problem be solved and, perhaps more importantly, what does it take to make sure a coding error does not put the rover and the mission in danger? As any good software engineer knows, the first step in coding a solution is to understand the specifications.

The Mars Science Laboratory (MSL) has three separate configurations. These configurations are all controlled by the same software and computer system (Cichy 2010). They include the cruise stage, the EDL (entry, descent and landing) stage, and the final rover stage.

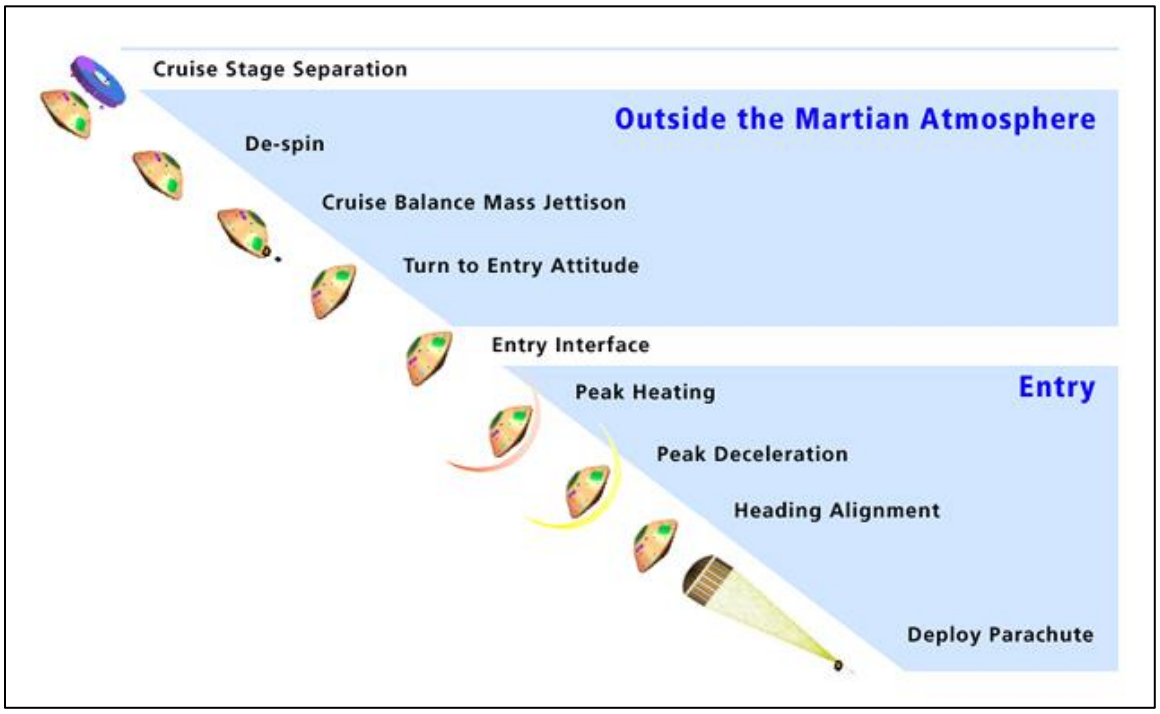


NASA/JPL 2013

From left to right are four elements: the cruise stage, the aeroshell (including the backshell and the heatshield on the far right), the descent stage, and the rover.

The cruise configuration is responsible for navigating through space for 354 million miles and making between five and six trajectory corrections before communicating to the entry vehicle that it has reached Mars' atmosphere (NASA/JPL 2013).

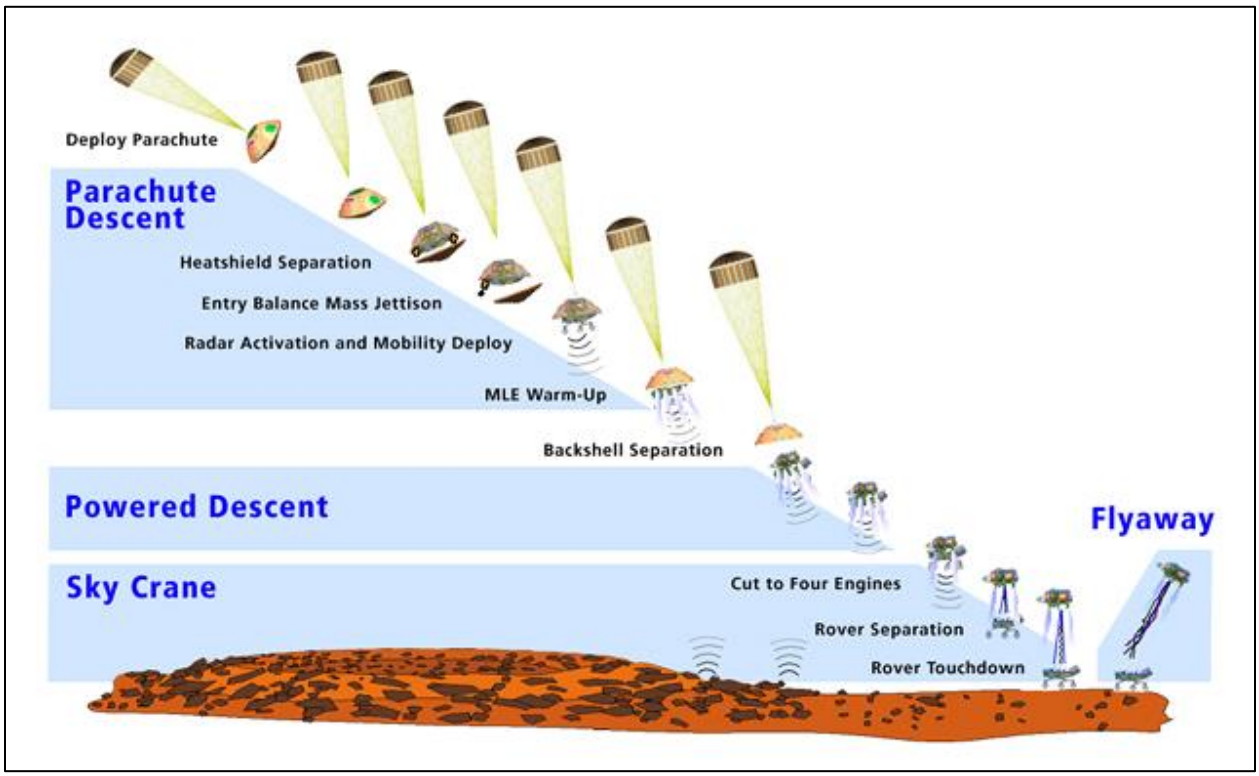
The EDL configuration performs all the necessary actions for landing the spacecraft on the surface of Mars. This means slowing the spacecraft from thirteen-thousand miles per hour down to zero. And, because MSL is far heavier than any of the previous rover missions, the airbag landing mechanisms used in the past will not work. An all new landing procedure, which uses the largest supersonic parachute ever created by NASA, must occur (NASA/JPL 2012). The first half of this procedure is shown below.



NASA/JPL 2009

The entire EDL procedure has to be controlled autonomously by the on board computer, known as the rover compute element (RCE). No interactivity can occur during this phase because the communications between Earth and the RCE take fourteen minutes and EDL must complete the landing maneuver in only seven (Cichy 2010). NASA calls this phase the "Seven

Minutes of Terror" because when the signal from the cruise configuration denoting contact with Mars' atmosphere reaches Earth, the rover has either landed successfully on the surface or been destroyed for seven minutes (NASA/JPL 2012). Below, the second half of the EDL phase.



NASA/JPL 2009

In this half of EDL, the operations that must be performed by the RCE become more evident. The exact timing of the parachute deployment and heatshield separation must be calculated by the RCE (Steltzner 2012). The RCE also must begin monitoring the Martian surface with radar to determine the velocity and altitude of the EDL phase. This determines when to cut the parachute and begin powered descent (NASA/JPL 2013). After cutting the chute, the descent stage performs an evasive maneuver with rocket boosters in order to avoid colliding with the parachute and backshell. Finally, the RCE must determine the exact landing location and begin powered descent toward the chosen area. Unfortunately, the descent stage cannot lower the rover all the way to the ground because doing so would create a large dust cloud on the surface

and potentially damage the rover's instruments (NASA/JPL 2012). So, a sky crane landing procedure will be used. In this procedure, the descent stage hovers seven and a half meters above the surface and lowers the rover down on an umbilical. The RCE sends a signal to the descent stage when the rover has separated and touch down has occurred. The descent stage then fires pyrotechnics to cut the umbilical and flies away to crash a safe distance from the Curiosity rover (NASA/JPL 2013).

The last stage of the MSL is the rover stage. This stage occurs when the rover has made it safely to the surface of Mars and will begin proceeding with mission objectives. An interesting point to note regarding the RCE and this last configuration of the MSL is that the RCE did not have enough storage space to fly and land with the full version of the software that will be used on the surface. So, once the Curiosity landed, the RCE deleted the EDL and cruise software and replaced it with version two of the surface operations software (Steltzner 2012).

When in the rover configuration, the RCE must be able to navigate the Martian surface and avoid obstacles without human interactivity (Cichy 2010). Also, because the on board power source does not generate enough energy for the rover to be on at all times, complex behaviors must be implemented into the software to keep the RCE functioning in low power situations. The RCE should not attempt a task that will use more power than it has available. Lastly, the software on the RCE must be able to operate each of the ten science instruments in Curiosity's payload (Cichy 2010).

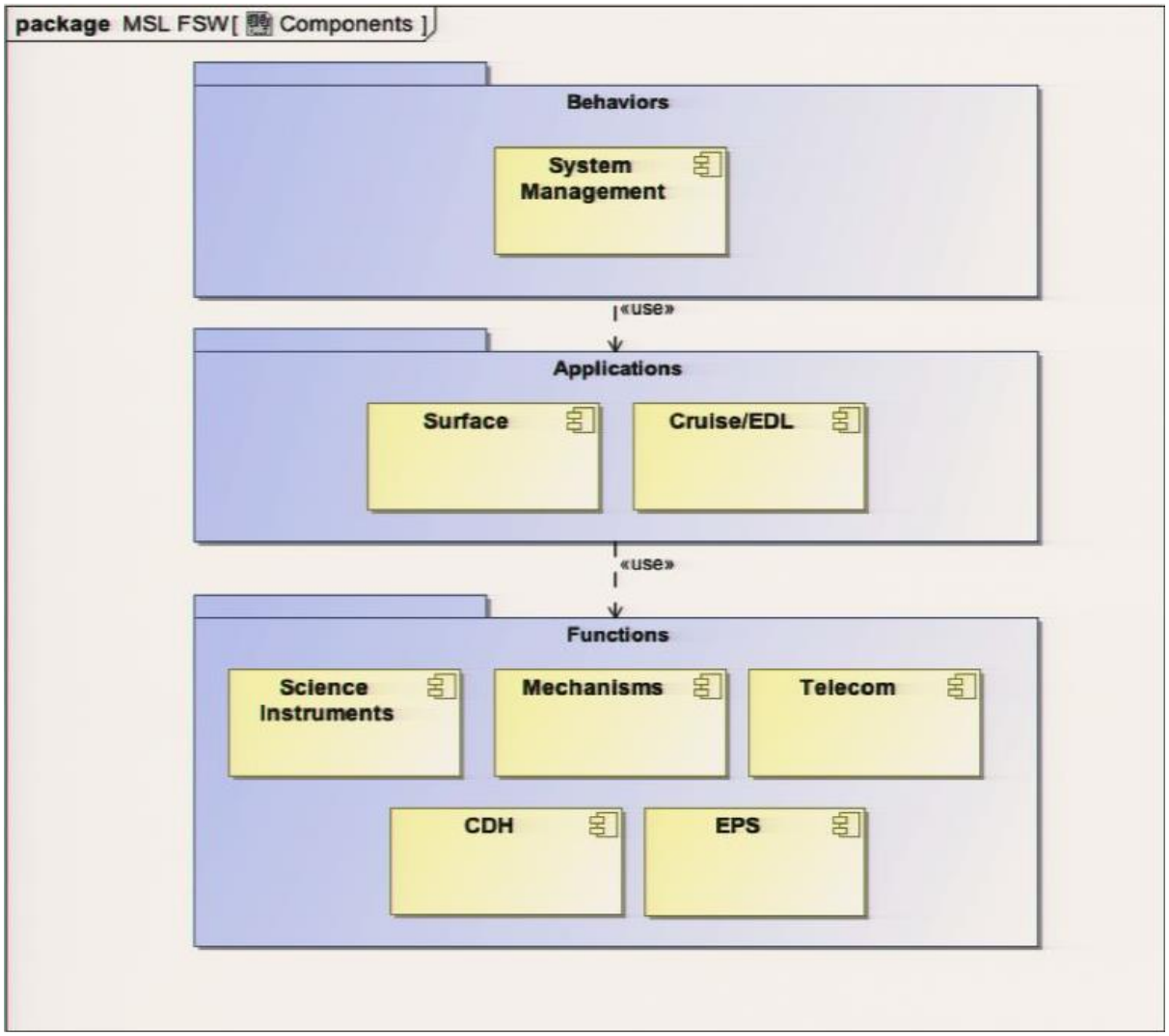
Benjamin Cichy (2010), Chief Flight Software Engineer for the MSL, describes the creation of MSL's flight software as a process of building upon the software and hardware that was used on the Mars Pathfinder Rover (MPF) and Mars Exploration Rover (MER).

MER and MPF both used a RAD6K radiation-hardened processor which ran a real-time operation system (RTOS) called VxWorks. MSL's RCE was the first dual-string rover computer. This means the RCE has two computer systems. One is the prime system and the other is used as a backup if the first one fails. The RCE uses VxWorks on two RAD750 radiation-hardened processors (Holzmann 2013). Each RCE has 4 gigabytes of flash memory, 128 megabytes of RAM, and a clock speed of 133 megahertz (Holzmann 2013).

MPF and MER's flight software was created according to a list of architectural principles. The principles dictated that the software would be broken down into modules. Modules are defined as a collection of software for a singular purpose that is owned by a single developer. The modules can communicate with each other through a conflict-free message passing interface only. Also, messages may only be sent by void return functions. Tasks executed by a module will be scheduled based on their deadlines, and a task may not have a deadline that occurs before its worst-case execution time. Also, whenever possible, memory will be allocated statically, including stack space (Cichy 2010).

The MPF flight software was composed of thirty-two modules. The MER flight software more than doubled this number to seventy-eight. MSL's flight software is composed of over one-hundred and fifty modules (Cichy 2010). Over 75% of this code was automatically generated. One of the methods used created extensible markup language files (XML) from unified modeling language (UML) state charts (Holzmann 2013). To generate this code, a UML state chart was created in a program called MagicDraw. Once the state chart was complete, the program could output an XML file. This XML file was sent to the "MSL statechart autocoder" which generated source code and a header files in the C language for the state chart's XML file (Benowitz 2012).

Cichy (2010) determined that the number of modules in MSL's software made it difficult to manage. He developed new rules which were appended to the original set of architectural principles. For MSL's software, modules are grouped together into components and have their own interfaces. Modules that have a similar functionality are elected to form a single element called a component. Also, because MSL has a dual-string computer system, the redundancy must be masked behind the interface of the components. This means the component interface will operate the same regardless of which computer system is in use. Another principle is that the components will be organized into layers.



Cichy 2010

The image above illustrates the layering of the final components. Each component is categorized as a function, a collection of functions known as an application, or a collection of activities known as a behavior. Functions perform a specific action only. They are not very useful on their own. Applications are "the primary operations interface to the vehicle" (Cichy 2010). Behaviors perform a higher level task by combining applications. After working to manage the complexity of MSL's software hierarchy by combining modules into components, Cichy (2010) decided "somewhat arbitrarily" to allow no more than ten top-level software components. After development was complete, the MSL software abstracted over one-hundred and fifty modules into just eight top-level components.

The final architectural principle for MSL states that "each component must specify requirements on resource usage in time and space with rate groups for each required real-time processing deadline" (Cichy 2010). This means that the component must inform the RTOS how much time will be required for it to execute and how much space is necessary. Additionally, the software component must state its priority. This helps to ensure that the programs complete their critical tasks in time and that the software remains reliable.

The Laboratory for Reliable Software (LaRS) at the Jet Propulsion Laboratory (JPL) in California was created to guarantee the dependability of flight software. Gerard J. Holzmann, LaRS member, described MSL's software development management process as a unique challenge when compared to rovers of the past. According to Holzmann, "Managing the development of a few million lines of critical code carries very different challenges from the development of a few thousand or even a few hundred thousand lines" (Wright 2013). To manage this software, LaRS made use of many several software reliability techniques.

Because the flight software has to function in conditions that cannot be easily recreated on Earth, NASA contracted the creation of the Mars Science Laboratory Workstation Test Set (WSTS). The WSTS is a computer program that simulates flight avionics hardware. It is completely software based and runs on a workstation Linux PC (NASA 2009). According to Holzmann (2012), the WSTS was used in the majority of all the flight software testing. The other test beds used for flight systems stress testing include a specific test for the EDL called GSTS, a test with part of the hardware known as MSTB, and ATLO. ATLO is the final test of the flight system. It tests the software with all of the actual hardware.

During EDL, which was arguably the most dangerous part of the mission, LaRS decided to make use of both of the RCE processors. They did this by creating a version of the EDL software component that only had the most essential functions coded into it. In the event that the prime RCE experienced a critical error and had to shut down, control was given to the backup RCE and it ran this software, known as "Second-Chance" (Holzmann 2012). This software did not have to be used when Curiosity landed, but it is an interesting method of fault protection.

As previously mentioned, much of the code was automatically generated from UML state charts. These charts describe the flow of action in a software module. Generating code from this created model ensures that the software documentation and code to always remain in sync. This increased the reliability of the code because it encouraged communication between many departments within NASA. Actual code did not have to be understood by systems engineers or software testers. They only needed to understand the state machine diagrams (Benowitz 2012).

Another method used to ensure MSL would have capable software was to enroll developers in a flight software developer certification course. All developers were required to pass this course before they could modify any of the flight software. It covered the coding

standard and its rationale as well as general computer science principles and the basic structure of the spacecraft control software (Holzmann 2013).

Despite all of these techniques, defects can still make their way into the software. In order to combat this, Holzmann (2013) employed the use state of the art detection tools including static source code analyzers, background tool-based code reviewing software, and Holzmann's own Spin logic model checker.

The background tools ran continuously to check for common errors, risky code patterns, or noncompliance with the coding standards implemented. The static code analyzers determine the meaning and function of each statement within the code. Holzmann describes them as "employing an additional, very conscientious and tireless developer on your team…that never tires of pointing out new subtle flaws" (Wright 2013). For MSL, the LaRS used four different static code analyzers in order to detect as many different types of flaws as possible. Surprisingly, each static code analyzer had very little overlap in errors detected. Lastly, the Spin logic model checker formally verified the multi-threaded software.

After these tools ran on the software source code, the error reports generated were made available to the software team in an interface known as Scrub. This tool took all of the errors and peer reviews for each module and funneled them to the developer responsible for that module. Each developer had to respond to every criticism, error, or automatically generated report with an "agree", "disagree", or "discuss". About 80% of all reports were responded to with an "agree" from the module owner, illustrating the effectiveness of the error detection process (Holzmann 2013).

Writing reliable software may be hard, but understanding the specifications, developing sound architectural principles, reducing the complication brought about by numerous modules,

and using state of the art code analyzing tools makes it achievable. Of course, credit to the software team itself is owed. Numerous clever solutions were no doubt devised behind the scenes of each module. The result of all of this difficult work culminates in satisfying perhaps the greatest human drive: curiosity.

References

Benowitz, E. (2012). UML State Chart Autocoding for the Mars Science Laboratory (MSL) Mission. *2012 Workshop on Spacecraft Flight Software.* Lecture conducted from California Institute of Technology. Pasadena, CA

Cichy, B. (2010). The Evolution of the Mars Science Laboratory Flight Software. *2010 Workshop on Spacecraft Flight Software.* Lecture conducted from California Institute of Technology. Pasadena, CA

Havelund, K., Groce, A., & Barringer, H. (2009). Monitoring the Execution of Space Craft Flight Software. Pasadena, CA: Jet Propulsion Laboratory. Retrieved November 2, 2013, from http://compass.informatik.rwth-aachen.de/ws-slides/havelund.pdf

Holzmann, G. (2013). Landing a Spacecraft on Mars. *IEEE Software.* 17-20.

Holzmann, G. (2012) Mars Code. *HotDep 2013.* Lecture conducted from Hollywood, CA

NASA (2009) Mars Science Laboratory Workstation Test Set. *NASA Tech Briefs 2009.* 50.

NASA/JPL. (2013) Mars Science Laboratory Curiosity Rover. Retrieved November 4, 2013, from http://mars.jpl.nasa.gov/msl/mission/

NASA/JPL. (2012). *Seven Minutes of Terror.* [Video] Retrieved November 3, 2013, from http://www.jpl.nasa.gov/video/index.php?id=1090

NASA/JPL. (2009). *MSLEntry1-2.* [Diagram] Retrieved April 28, 2009, from http://marsprogram.jpl.nasa.gov/msl/mission/spacecraft/edlconfig/

Steltzner, A. & Entry, Descent, and Landing Team Panel. (2012). Curiosity News Briefing. Lecture conducted from the Jet Propulsion Laboratory. Pasadena, CA

Wright, A. (2013). Revving the Rover. *Communications of the ACM.* 14-16.