

Applying IV&V Lessons Learned: Phoenix through MAVEN

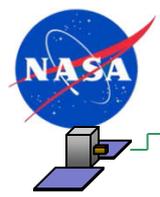
September 13, 2011

Steve Larson

Jet Propulsion Laboratory, California Institute of
Technology

Steve Raque

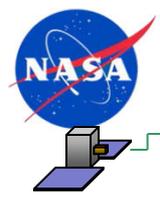
NASA IV&V



Introduction



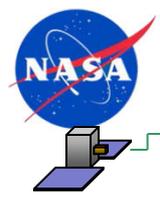
- Motivation
- GRAIL and MAVEN Project Overview
- Analysis of Phoenix IV&V and Post-Launch Anomalies
- Phoenix Lessons Applied to GRAIL
- Phoenix/GRAIL Post IV&V Comparison
- GRAIL Post-Launch Experience
 - Informal discussion, assuming GRAIL has launched
- Moving on to MAVEN



Motivation



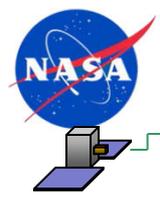
- Both GRAIL and MAVEN are cost-capped missions
 - IV&V is recognized as a contributor to mission reliability, but funds were limited
 - Improving the efficiency of the IV&V effort by learning from earlier missions would increase the value added



GRAIL Project Overview



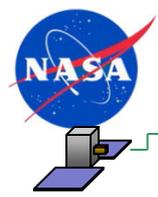
- GRAIL = Gravity Recovery And Interior Laboratory
- Launch September 2011
- Mission
 - Determine the structure of the lunar interior, from crust to core
 - Advance understanding of the thermal evolution of the Moon.
 - Extend knowledge gained from the Moon to the other terrestrial planets.
- Project Management: JPL
- PI: Dr. Maria Zuber, MIT
- Spacecraft: Lockheed Martin S&ES
- Instrument: JPL



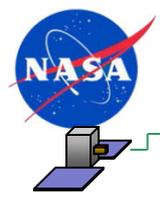
MAVEN Project Overview



- MAVEN = Mars Atmosphere and Volatile Evolution
- Launch November 2013
- Mission
 - Explore the planet's upper atmosphere, ionosphere and interactions with the sun and solar wind
 - Determine the role that loss of volatile compounds over time, giving insight into the history of Mars atmosphere and climate, liquid water, and planetary habitability
- Project Management: GSFC
- PI: Dr. Bruce Jakosky of the University of Colorado (LASP)
- Spacecraft: Lockheed Martin S&ES
- Instruments: UC Berkeley, GSFC, CU/LASP



- NASA IV&V provided a dump of all issues written for the Phoenix project
 - 893 issues categorized into 16 bins
 - Multiple categorizations allowed
 - Analyzed according to whether the project responded by changing the affected artifacts or using as-is.
 - Results compared to a similar assessment of post-launch anomalies documented by the project.

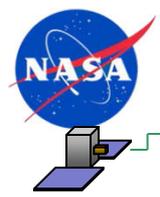


Analysis of Phoenix IV&V Results

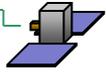


- Code-related issues were much more likely to be accepted “as-is” by the project
- Issues that could be addressed by updating documentation were more likely to be fixed
- Possible explanations:
 - Code issues developed late in the development process
 - Lower effort barrier to changing documents vs. code (e.g., don’t need to regression test documents)

Acronym	Description	Fix	UAI	Total
AB	Array Bounds violation	6	5	11
CCS	Conflicting Code Statements	3	12	15
CE	Coding Error	11	55	66
DC	Dead Code	13	31	44
DCD	Design/Code Discrepancy	13	10	23
LP	Loss of Precision	1	4	5
ML	Memory Leak	0	1	1
NP	NULL Pointer	0	7	7
TM	Type Mismatch	4	12	16
UV	Uninitialized Variable	13	19	32
	Code-related Subtotal	64	156	220
DD	Document Discrepancy	70	63	133
DRD	Design/Requirements Discrepancy	56	23	79
MR	Missing Requirement	34	13	47
NV	Requirement Not Verified	98	7	105
RQ	Requirements Quality	101	59	160
RT	Requirements Trace	36	25	61
	Documentation-related Subtotal	395	190	585
	Combined Total	459	346	805



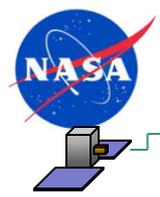
Analysis of Phoenix IV&V Results



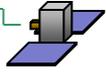
- Fix vs. Use As-Is decisions were correlated with issue severity
 - All Severity 2 issues were addressed by the project
 - IV&V agreed with the 2 UAI responses, and closed the issues
 - Severity 5 issues are the exception
 - Less numerous, may have been fixed as part of normal process or addressing higher severity issues
 - For the most part, IV&V agreed with UAI decisions
- Fix/Use as-is decisions were correlated with the analysis approach
 - Roughly 2/3 of issues found with automated tools were accepted as-is
 - The proportion was reversed in the case of manual analysis
 - Lockheed code relatively mature, often able to show that code would behave properly
 - Later in life cycle, more difficult to fix
 - Manual analysis more likely to be applied to tests, documentation
 - Earlier in life cycle and/or easier to fix

Severity	Disposition	Count	Percent
2	F	24	92.3%
2	UAI	2	7.7%
3	F	329	66.1%
3	UAI	169	33.9%
4	F	80	33.2%
4	UAI	161	66.8%
5	F	26	65.0%
5	UAI	14	35.0%

ToolUsed	Fix	UAI
Flexe-Lint, V8.00Q	26	54
Klocwork inSpect	19	49
Manual Analysis	415	239
Understand for C/C++	0	4

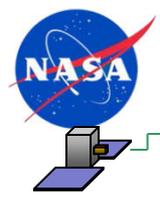


Analysis of Phoenix IV&V Results



- Payloads accounted for roughly $\frac{3}{4}$ of all issues, and had a somewhat higher proportion of higher severity issues
- Assuming that issue frequency is a predictor of in-flight performance, one might predict that payloads would account for the majority of post-launch FSW anomalies

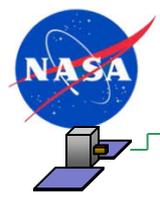
CSCI	Severity				Totals	
	2	3	4	5	Count	%
GN&C	1	25	5	2	33	15.4%
I/O	0	28	5	0	33	15.4%
OS	0	16	9	1	26	12.1%
Spacecraft	5	38	50	3	96	44.9%
Telecom	0	14	12	0	26	12.1%
Spacecraft Subtotal	8	124	85	11	214	
Spacecraft % Distribution	4%	58%	40%	5%	27%	
Mardi	0	4	4	0	8	1.4%
MECA	4	130	44	4	182	30.8%
MET	0	39	9	3	51	8.6%
RA	5	48	32	7	92	15.6%
SSI/RAC	6	75	27	15	123	20.8%
TEGA	5	81	44	5	135	22.8%
Payloads Subtotal	20	377	160	34	591	
Payloads % Distribution	3%	64%	27%	6%	73%	



Analysis of Phoenix IV&V Results



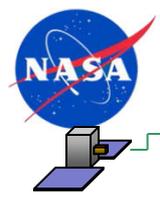
- Review of immature and/or non-controlling documentation generated large numbers of issues in some cases
 - 35 issues related to the TEGA S/W design
 - Closed by clarifying that the design document was the controlling document not the CDR charts
 - 60 issues related to MECA S/W testing
 - Analysis began with initial release of test plan
 - Requirements continued to change, and updates were to be expected
 - Issues were generally closed as part of normal process



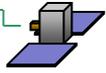
Phoenix Anomaly Analysis



- 369 unique post-launch ISA (Incident/Surprise/Anomaly) reports were analyzed
 - Binned into 8 categories of contributing factors
 - Factors identify where in the development & test process a defect was likely to have been introduced, or could have been corrected but was not
 - Multiple factors allowed
 - Binned according to whether issue was discovered on flight vehicle or on the ground
 - 31 ISAs determined to be in flight software (next slide)
 - 7 in spacecraft, 24 in payloads
 - » Mirrored IV&V ration of spacecraft/payload issues
 - Most ISAs concerned ground software and hardware, and were not included in the study

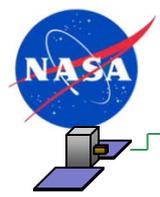


Contributing Factor Distribution



	Contributing Factors								Total # Incidents	Occurred in Flight?
	Complexity	Heritage Process	Missing Requirement	Design	Inadequate Testing	Implementation	System Engineering	Insufficient Information		
Spacecraft-related	0 0%	2 29%	3 43%	4 57%	3 43%	2 29%	4 57%	1 14%	7 23%	6 86%
Payload-related	3 13%	1 4%	9 38%	4 17%	11 46%	14 58%	7 29%	3 13%	24 77%	12 50%
Combined	3 10%	3 10%	12 39%	8 26%	14 45%	16 52%	11 35%	4 13%	31 100%	18 58%

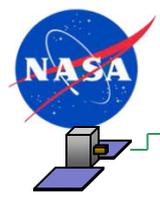
- Most ISAs had more than 1 contributing factor
 - For example, the heritage process introduced a defect in spacecraft battery control that testing should have caught, had fidelity to flight conditions been adequate
- Majority of ISAs were payload-related, mirroring the proportion of IV&V findings
- Systems Engineering and S/W Design were leading causes of spacecraft issues
 - Inverse of what might be predicted from IV&V distribution (58% for Implementation vs. 37% for Requirements & Design)
- Code & Test were leading causes of payload issues
 - Also inverted from IV&V issue distribution (68% for Requirements & Design vs. 16% for Implementation)
- Effects of complexity is an industry-wide concern, but did not appear to be a dominant factor
 - Complexity-related issues in the payloads did not fit the classic “Normal Accidents” model



Predictive Skill



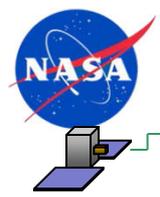
- IV&V broadly predicted (based on the data) that payloads would be the primary source of in-flight anomalies
 - This was borne out in flight
- Inverse relationship between distribution of IV&V issues and contributing factors for flight anomalies suggests additional analysis needs to be done to understand this relationship.



Phoenix Lessons Applied to GRAIL



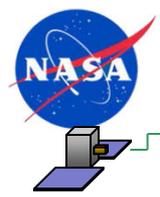
- “Newness” is a risk
 - Payload issues dominated, and were all either first-of-a-kind or modifications to previous products
 - PBRA process used on GRAIL emphasized “newness” as a risk
- Product Line FSW should be approached differently than first-of-a-kind/low heritage software
 - Discussion of the effects of heritage allowed us to close issues more easily
- Unnecessary issues can be avoided by waiting for products to mature
 - Structured discussion of potential issues generated by review of early versions of FSW and requirements allowed us to resolve a large number of issues without excess formality
- Problems that escape both the developer and IV&V tend to be “difficult”
 - Hardware interfaces of greatest concern, received thorough IV&V review
 - However, proprietary nature of source data limited the analysis



IV&V Results—GRAIL/Phoenix



- Upon disposition of the last GRAIL issue, Phoenix (spacecraft only) and GRAIL issue distribution was analyzed
 - Overall roughly 10% drop in number of issues
 - Change could simply be due to different IV&V team or bundling strategies
 - Overall increase in Fix rate
 - Previous IV&V work on code base likely reduced number of false positives in code
 - Better communication eliminated more false positives in all categories
 - Big drops in Dead Code, Design/Code Discrepancy, Document Discrepancy categories
 - Better communication helped eliminate false positives and issues due to examining immature products
 - Dead code reduction may be due to prior IV&V work, but not analyzed for cause
 - A new category (Code/Requirements Discrepancy) was introduced for the GRAIL analysis
 - Better alignment with the way IV&V does their work
 - On Phoenix, these would have shown up in either Design/Code Discrepancy or Coding Error
 - Increase in Requirements Quality, Requirements Trace, Missing Requirements categories
 - IV&V changed approach, started with modeling and top-down requirements assessment (many more documents examined)
 - Different IV&V personnel may have also contributed

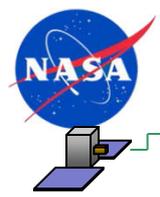


GRAIL Post-launch Experience



<Placeholder for informal discussion>

BACKUP



Analysis of Phoenix IV&V Results



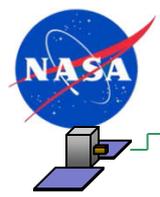
- Most (> 75%) of IV&V findings were closed*, and about half of the remainder were left in a terminal state that indicated no further IV&V action
- Approximately 12% ended in the “Project Accepts Risk” state
- Majority of issues were Sev 3, with Sev 4 being the next most numerous

Table X. TIM Resolutions

State	Count	Percent
Closed*	683	76.5%
Closed Before Submitted	3	0.3%
Not an Issue	40	4.5%
Not To Be Verified	17	1.9%
Project Accepts Risk	106	11.9%
Withdrawn	44	4.9%
	893	100.0%

Severity	Count	Percent
2	26	3.20%
3	498	61.90%
4	241	29.90%
5	40	5.00%
	805	100.00%

* “Closed” did not always mean “fixed”. The number of issues resolved to “Use As-Is” exceeded the number of “Closed” issues.



Analysis of Phoenix IV&V Results



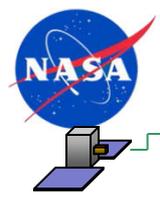
- Comparison of when defects were introduced vs. where they were found generally showed that IV&V caught issues in the same phase they were introduced

- Suggests IV&V did a good job of keeping up with the project
- Suggests that the development teams did not have a significant problem with defects escaping from one phase into the next

PhaseFound	PhaseIntroduced	Count	Percent
Design	Design	346	98.9%
Design	Implementation	1	0.3%
Design	Requirements	3	0.9%
	Design Subtotal	350	
Implementation	Implementation	130	100%
	Implementation Subtotal	130	
Requirements	Requirements	1	100%
	Requirements Subtotal	1	
SW Detailed Design	System Requirements/Design	3	100%
	SW Detailed Design Subtotal	3	
SW Implementation	Requirements	5	2%
SW Implementation	Subsystem Requirements/Design	16	6%
SW Implementation	SW Detailed Design	1	0%
SW Implementation	SW Implementation	162	62%
SW Implementation	SW Requirements Design	76	29%
SW Implementation	System Requirements/Design	1	0%
	SW Implementation Subtotal	261	
SW Preliminary Design	SW Preliminary Design	1	2%
SW Preliminary Design	SW Requirements Design	46	98%
	SW Requirements Design Subtotal	47	

- Obvious exception is code defects associated with requirements

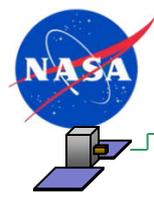
- The cause for this was not determined for Phoenix
- GRAIL experience was that this was due to on-the-fly changes to requirements made in code implementation that were not reflected back into requirements documents (see Code/Requirements Discrepancy category, slide 21)



Serious Spacecraft Incidents



- There were only two spacecraft ISAs that appeared to pose a potentially serious threat to the spacecraft, and both were addressed with in-flight FSW modifications
 - Battery charging
 - Early in the mission anomalous telemetry relating to the battery were observed
 - Investigation revealed that incorrect battery parameters had eluded the heritage review and development/test process
 - No way for IV&V to know the parameters were wrong
 - Discovery via test would require using flight hardware in a scenario exceeding typical ATLO resources
 - Telemetry wrap-around
 - Boot times were found to be slowly increasing after landing
 - Problem traced to bug in telemetry generation under certain special conditions
 - Logic error very unlikely to be found by automated tools
 - Circumstances were difficult to foresee
 - » Easy to express generic concerns about departure from heritage
 - » Much harder for IV&V or developer to develop specific scenarios

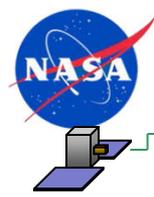


IV&V Results—GRAIL/Phoenix (Data)



- Upon disposition of the last GRAIL issue, Phoenix (spacecraft only) and GRAIL issue distribution was analyzed

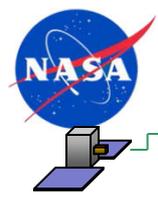
Acronym	Description	GRAIL				Phoenix--Spacecraft Only			
		Fix	UAI	Total	Percent Total	Fix	UAI	Total	Percent Total
AB	Array Bounds violation	3	1	4	2.1%	2	4	6	2.8%
CCS	Conflicting Code Statements	0	0	0	0.0%	1	8	9	4.2%
CE	Coding Error	8	6	14	7.4%	5	32	37	17.3%
CRD	Code/Requirements Discrepancy	15	17	32	17.0%				0.0%
DC	Dead Code	1	6	7	3.7%	6	19	25	11.7%
DCD	Design/Code Discrepancy	0	0	0	0.0%	10	5	15	7.0%
DD	Document Discrepancy	2	4	6	3.2%	19	15	34	15.9%
DRD	Design/Requirements Discrepancy	0	0	0	0.0%	0	0	0	0.0%
LP	Loss of Precision	0	1	1	0.5%	0	0	0	0.0%
ML	Memory Leak	0	0	0	0.0%	0	1	1	0.5%
MR	Missing Requirement	9	10	19	10.1%	9	5	14	6.5%
NP	NULL Pointer	1	2	3	1.6%	0	5	5	2.3%
NV	Requirement Not Verified	7	9	16	8.5%	9	0	9	4.2%
RQ	Requirements Quality	26	34	60	31.9%	4	27	31	14.5%
RT	Requirements Trace	9	11	20	10.6%	1	0	1	0.5%
TM	Type Mismatch	0	0	0	0.0%	2	11	13	6.1%
UV	Uninitialized Variable	1	5	6	3.2%	9	5	14	6.5%
		82	106	188	100%	77	137	214	100%
		44%	56%			36%	64%		



Categories used to classify IV&V Findings



Acronym	Description	Notes
CE	Coding Error	Many possibilities, from not adhering to coding standards, use of obsolete functions, use of GOTO, lack of a switch default case, or just plain wrong.
NP	NULL Pointer	Pointer may be assigned a NULL value, which isn't checked
TM	Type Mismatch	Assignments and comparisons of differing types
ML	Memory Leak	Failure to release memory, or unbounded memory allocation
AB	Array Bounds violation	Possible reading or writing from/to an array or string beyond declared length
DC	Dead Code	Code is never used or can't be reached
DD	Document Discrepancy	Documents disagree with each other, or contain internal inconsistencies
DRD	Design/Requirements Discrepancy	Design and corresponding requirements spec not in agreement; possible missing or incorrect implementation
UV	Uninitialized Variable	Variable possibly not initialized before use
LP	Loss of Precision	Variant of type mismatch, where significant bits can be lost
CCS	Conflicting Code Statements	One part of the code contradicts, or repeats what is found elsewhere in the code
NV	Requirement Not Verified	Test design does not test an assigned requirement
DCD	Design/Code Discrepancy	Code and design do not agree
MR	Missing Requirement	Requirements specs do not appear to contain everything that would be expected based on contents of other reference documents
RQ	Requirements quality	Broad category encompassing clarity, completeness, use of language, etc.
RT	Requirements trace	Broader than simple problems with traceability, includes general flow-down issues

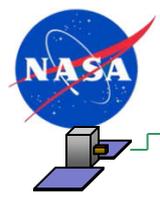


Issue types by CSCI

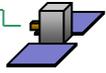


CSCI	Disposition	AB	CCS	CE	DC	DCD	DD	DRD	LP	ML	MR	NP	NV	RQ	RT	TM	UV
GN&C	F			2			9[1]				6			1		0	
	UAI			3			3				2			6		1	
I/O	F					10	4							2			2
	UAI		1	2	1	2	1			1	1	1		3		1	1
OS	F		1				1										
	UAI		2	9		1	5				1	4		1			1
Spacecraft	F	2		3	4		2				2		9	1	1	2	6
	UAI	3	4	18	17	2	2							7		8	3
Telecom	F				2		3				1						1
	UAI	1	1		1		4				1			10		1	
RA	F			3			2	5			1		16	4	2	1	1
	UAI	1	3	8	3		6	8			1	2	2	11	9		3
SSI/RAC	F	1	1	2	3		22		1		13		4	19	11		3
	UAI		2	9	5				3		5		2	7	3	1	6
TEGA	F	3	1	1	2	1	6	1			4		7	29	12		
	UAI				1		38	7	1		2		2	5	11		1
Mardi	F															1	
	UAI			3	2												2
MECA	F				2	2	17	50			7		59	20			
	UAI					5	4	8					1	5			2
MET	F						4						3	25	10		
	UAI			2	1									4	2		

[1] Boldface type is used in this table to highlight issue areas that will be discussed in the text.



Definitions of ISA Contributing Factors



Category	Description
Complexity	The complexity of the problem or solution played a role in the incident
Heritage Process	A failure in the process of inheriting the software resulted in inappropriate features being retained
Missing Requirement	If a requirement specifying the desired behavior had been written, the incident could have been averted
Design	Some feature of the implementation provided the conditions for undesirable or incorrect behavior. Similar to the Implementation category, but focuses on higher-level decisions of code design as opposed to simple programming mistakes.
Inadequate Testing	The scenario where the software flaw appeared was not tested.
Implementation	A programming error (using wrong number for constant, typos, etc.) caused to problem.
System Engineering	System engineering within the project (from project SE down to FSW SE, and including science and mission system SE) did not provide support in the problem area, leading to conditions where correct software behavior was either not recognized or not specified.
Insufficient Information	The software developers did not have access to key information that would have guided them towards the correct implementation.