A Vision for Future Software
Open Source and Design Thinking at NASA

Jay Trimble
NASA Ames Research Center
Frontiers 2017
Personal Milestones

1960’s
NASA JSC Intern
NASA Johnson Space Center

1981
Science Ops Voyager Neptune
NASA Jet Propulsion Laboratory

1989
Lead Ops Director Space Radar Lab 1
NASA Ames Research Center

1994
Lunar Rover MOM

Now
Mission Control: The Icon
Mission Control for Mars Rovers
The Light Speed Constraint

- Earth-Moon: ~6 - 25s
- Mars: ~14 - 40 Min
- Jupiter: ~80 Min
- Neptune: ~8 hours
Mission Control v. Star Trek

- NASA
  - Flight Director
  - Systems
  - Trajectory
  - Payloads/POCC
  - INCO

- Star Trek
  - Captain (Kirk, Janeway, Picard, Cisco, ...)
  - Engineering (Mr. Scott)
  - Navigation (Chekov)
  - Science Officer (Spock)
  - Communications (Uhura)
Mission Control Famous Calls
Mission Control Famous Calls
The Mission

repair a malfunctioning satellite

In orbit capture and repair has not been done

It's made possible by the Space Shuttle
The First Epiphany
Evolution
Pasadena Early 1990’s

The Mission

Earth Observations Using Synthetic Aperture Radar

Two missions on Space Shuttle Endeavor
The Second Epiphany

Write software requirements

Customer signs requirements

Expectations and mental models diverge

MOS shall track the orientation of the solar panels with respect to Sun (+/- TBR arcmin)
Expectations Meet Reality

About to get a look → Users see the software → Why this reaction?
There must be a better way

Follow the (as yet undefined for us) road to user centered agile or, take a long vacation
Early 2000’s Mars Rover Ops

The Mission

Mars Exploration Rovers (JPL)

Human Centered Computing (ARC)

**We proposed methods**, not specific solutions or tools

We called it Human Centered Computing, inspired by Don Norman, *The Invisible Computer*
Mars Exploration Rover Scenario

Users on Earth

Rover on Mars

Max round trip light time ~40 min
Acceptance

To fund MER HCC, we had to “sell” the ideas to our funders at NASA Ames, to the Mars Exploration Rover Project at JPL and to the users.

We focused on outcomes and touched on the methods using analogies.

Easier to market an artifact or a result than an idea.

Mental model example - Ethnography = User observations - what people say and what they do are often different. How often do you exercise?

Goals - Mission productivity, communications, safety.

Note no mention of design thinking, this is 2000.
Key Lessons so far

This is a small community and most people know each other

Each mission is its own community, somewhat like the cast in a performance

Speak the stakeholders language

Be careful with generalizations like “the invisible computer” or software that adapts to users rather than the other way around

Most of the stakeholders care only about what your product or method does for their mission

Most of users don’t care about design, but they may care about the results

Users who are used to a way of doing things, even an inefficient way, will resist change. Don’t give them change unless it adds significant value.

Don’t go against established conventions, no change for changes sake, use established, mental models

Do not try to take away existing tools. Give them new tools in shadow mode.

Be careful about getting too excited about your cool new technology
Next

We now believe we need new technology, not just methods and process

So we embark on a new course and instead of proposing methods we propose tools...
We are trying to “fix”

Multiple heterogeneous applications create walls, turning users into integrators
The Selling Points

Decrease Cost

Save on maintenance by retiring existing applications, make the users productive

Empower the users

Compose your own displays without programming, all your stuff in one place

Top Down v. Bottom Up

The top provides the funding

The Bottom provides the advocacy (remember this is a small community)

The problem that we could not see yet

The management funded the project based on the retirement of existing applications

Users are open to new technology but less so when they are told that they are going to lose the current capability on which they depend
Participatory Design

Designers facilitate design process, users are domain experts

We used The Bridge Method

Built a shared language

Built shared mental modes

Enabled us to design solutions with users

Created a tight bond between the design team and participatory users

Shared ownership

Created an us v. them between the participatory team and the larger user community
Agile User Centered Design

12 Week Release

3 Week 3 Week 3 Week 3 Week

Rank JIRA's
Optional mid-iteration gestation
Feature Freeze
Code Freeze

Sprint n

Coding
UE & Tech Spec Updates
Update issues as needed
Continuous build, feature notifications for testing during rollout

Feedback loop for current sprint

Customer access to continuous build
Customer Feedback
Design Updates, Fixes

Test Previous Sprint

Customer installs previous sprint
Customer Accept/Reject
Feature Complete

Testathon

24 hr Test

Deliver
Did we help the users?

**Process steps**
What actions does it take to build and test a display?

**Process time**
How long does it take to accomplish those steps?

<table>
<thead>
<tr>
<th></th>
<th>Legacy</th>
<th>MCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>Manual data entries</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>External tools used</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

60% reduction in steps

80% reduction in manual entry

90% reduction in time

Manual data entry is the primary source of errors / risk

<table>
<thead>
<tr>
<th></th>
<th>Legacy</th>
<th>MCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minutes to complete</td>
<td>65</td>
<td>6</td>
</tr>
</tbody>
</table>
Design is not enough

End user composition alone is not enough, it must be mixed with the specific job enabling features that users want. The combination is powerful.

The term end user composition is nerdy and does not grab people, the popular lexicon on this shifts… “mashups,” “dashboards” and can confuse the message

Unknown cultural differences can have a big impact - our first user test, though we stated it as such, was thought by users to be the final software because this is the only mental model they had

New capabilities take a long time to catch up to “old” capabilities, benefits must outweigh the inconvenience

Don’t take away “old” capabilities, let new co-exist with old in shadow mode, for a period of time

Customers will map what you say into their own expectations, creating a mental model that varies across groups and that may be unknown to the design team

Show constant progress, make it visible and accessible

If it’s not easy, people won’t even try it

Customers want and expect new capabilities, they also want all of their legacy capabilities

Openness increases with time and use

A new mental model, even a better one, at first will be confusing to users
It’s all so simple

Succeed

Know who your stakeholders are, focus

Fail

Try to solve too many problems for an undefined stakeholder base

We did better creating generalizations from instances than creating instances from generalizations - start by solving real problems not generalizations
Rebuilding

The desktop version is ultimately cancelled

We rebuild, our funders are now in California
New Stakeholders

Jet Propulsion Lab
Multi-Mission Ground Systems
Multiple missions use the software over time, at many NASA centers

Jet Propulsion Lab
Many Flight Projects
Each one concerned about success of their mission

NASA Ames Research Center
Resource Prospector
Successful Mission

Open Source Community
NASA, Commercial, Other
The success of their project
Our users' mental model in the early 2000’s was that software is delivered and that's what you get (remember those inflexible displays). We conducted a user test on early software with unforeseen consequences.

A designer thinks of a prototype as a question rendered as an artifact, the expectation is that there will be many. A system engineer thinks of a prototype as a risk reduction exercise to buy down risk associated with system requirements, expectation is that there will be few because they tend to be expensive.

Popular mental models, such as dashboards and mashups affect user perception.

Say it then sim it.
Mental Model Map Example

System Engineering

Requirements (tendency fewer ideas)

Prototypes for Risk Reduction, typically few

Review
Build
Train, Fly

Design Thinking

Observations
Ideation
Synthesis (more ideas)

Prototypes - questions rendered as artifacts, typically many

Try/Use ("Say it then sim it")
Iterate
Train, Fly

Train, Fly
Open Mission Control Technologies

Goals

Provide users with an all your data in one place solution

Empower users to compose their own displays

Create new opportunities for collaboration and community involvement using open source

Take what has been a closed and hence mysterious world and open it up

https://nasa.github.io/openmct/
Initial Mission Users

Jason-3

Resource Prospector

Mars 2020 (expected testbed)

https://nasa.github.io/openmct/
All Your Data in One Place

https://nasa.github.io/openmct/
Create & Compose

Example of user object types

Layout is the users canvas

https://nasa.github.io/openmct/
User-Built Compositions
User Testing
For Fun

2001: A Space Odyssey

Open MCT
Sprint

GV Style Design

Sprint
The Community

https://nasa.github.io/openmct/

60 Visitors per week then..

User Reddit Post

20k visitors in two days

Outside contributors

Collaborations inside and outside of NASA that were not possible or practical before open source
The Role of Failure

“Failure is not an option” - Gene Kranz

Referring to human space flight operations
Design Thinking

...is now an accepted part of our organization, though it is only practiced by a small number of teams.

My team is moving design thinking from software, where we first established it, to the design and development of the mission system for a lunar prospecting rover.

“Say it then sim it”