Towards Modeling for Design:
Using a Real-time Collaborative Environment in CAPS

John F. Dannenhoffer, III [*]
Aerospace Computational Methods Laboratory, Syracuse University, Syracuse, NY 13244

Nitin D. Bhagat [†]
University of Dayton Research Institute, Dayton, OH, 45409

The use of geometric models is becoming central in the analysis and design of complex configurations, such as aerospace vehicles. Traditionally these models have been developed by a single engineer, working alone because of the way the modeling tools have been designed. While collaboration is possible via file sharing or screen sharing, none of the traditional design environments facilitate real-time collaboration. A similar problem in software development (programming) has been fixed with the introduction of pair-programming, wherein two programmers sit side by side, and work together. Pair programming has been shown to greatly improve the software development process, while only incurring a small cost penalty. Herein, the pair-programming concepts have been applied to the development of geometric models in the Engineering Sketch Pad. The efficacy of this new approach is demonstrated via two case studies.

I. Nomenclature

$CAPS$	=	Computational Aircraft Prototype Synthesis
ESP	=	Engineering Sketch Pad
AIM	=	Analysis Interface Module
OML	=	Outer Mold Line, suitable for fluid analysis
BEM	=	Built-up Element Model, suitable for structural analysis

II. Introduction

For the past few decades, many organizations have developed environments for the multi-disciplinary analysis and design of complex systems, such as aerospace vehicles. Amongst them is the Computational Aircraft Prototype Synthesis (CAPS) system at the Air Force Research Lab[1]. Using CAPS, an aircraft designer can build a feature-rich geometry that is ready for many different types of analysis, spanning many fidelities and many disciplines. This is done by a series of models that are linked via persistent attributes that are created at the time of model creation. In order to be successful building this feature-rich geometry demands collaboration between geometry creation and the various discipline-specific experts. A simplified version of what is embedded in the geometry using CAPS is shown in Fig. 1.

In CAPS, the steps involving geometry creation, embedding discretization requirements for mesh generation, and analysis readiness can be automated. The automation is facilitated by an appropriate combination of parameterization and attribution to generate specific geometry representations called "views", as shown in Fig. 1. These views are extracted via Analysis Interface Modules (AIMs) that are links to specific analyses. This is typically done by generating an analysis input deck prescribed for an intended analysis, along with the metadata related to analysis input that is directly embedded in a specific view. For example, boundary conditions for fluid analysis can be extracted via attributes that are embedded on specific surfaces of the geometry. Similarly, material properties can be prescribed to certain surfaces, which can be identified via specific attributes. The process of parameterizing and attributing has been overly simplified to focus on the need for collaboration. For a complete description on the process, please refer [1]. In summary, this workflow shows how the design team can interact with the geometry to achieve automation during the design process.

[*]Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow
[†]Research Engineer, Research Engineer, Air Vehicle Optimization Group (AMD), AIAA Senior Member

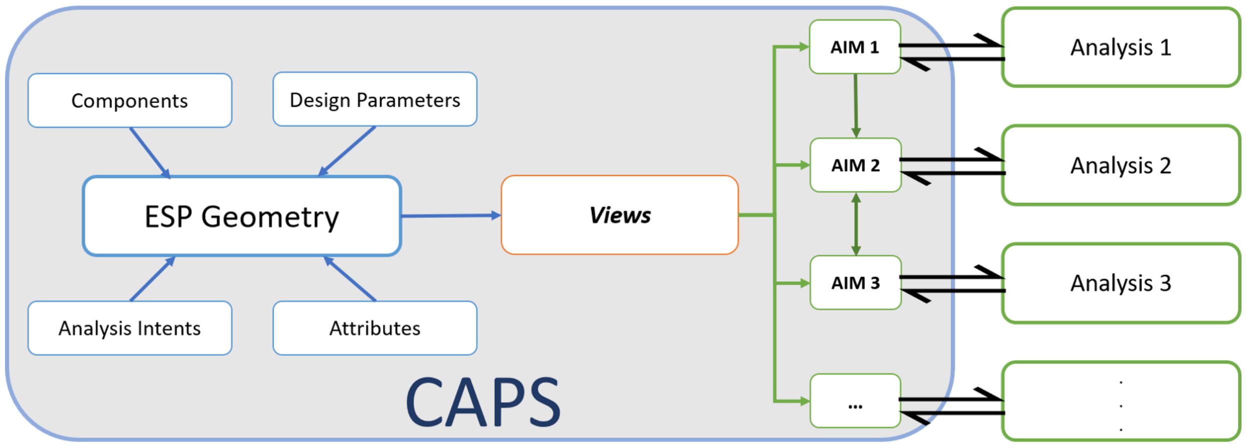


Fig. 1 Geometry in CAPS

Traditionally the geometry representation is created by a geometry modeler in isolation, with inadequate and/or vague knowledge on what the downstream analysis intents are. In addition, there is a tedious intermediate step of meshing the geometry for which the requirements are dictated by analysis software and restrictions are imposed by the available fidelity of the geometry representation. Thus, the geometry "model" is then required to be tweaked and sometimes fixed to make it suitable for analysis. This process typically breaks the relation between the discrete representation and the original geometry model. This disconnect between the design/analysis intent the actual geometry creates a burden with respect to resources, efforts, and time on both analysis experts and the geometry modeler.

As the interfacing of preliminary and detailed design stages is getting strongly fused with the conceptual design, suitable representations of geometry (a) is required to be more realistic, hence getting complex, (b) build process is no longer an isolated activity, and (c) needs to embed analysis and design intent a priori to fulfill the fidelity requirements. Computational Aircraft Prototype Synthesis (CAPS) provides a geometry-centric environment to allow the design team to produce such feature-rich geometry that includes analysis intent.

III. Pair-programming

Traditionally, the development of computer programs has been a solitary endeavor, wherein an individual programmer takes a set of requirements (and perhaps a set of test cases) and crafts computer code that satisfies the requirements. In order to make the code that an individual programmer wrote part of an overall system, organizations have implemented coding standards, testing standards, and code reviews so that the computer program was depersonalized and "owned" by the organization.

The first major problem with this approach is that programming involves a series of high-level (strategic) design choices and low-level (tactical) coding decisions that are hard for a single individual to effectively balance. This often leads to code in which good strategic choices are often overlooked because of the focus on the tactical decisions.

The second major problem with this approach is that there is one individual who understands all the intricacies and tactic assumptions in the design and implementation of the code. Inevitably, after that individual moves on to another project or organization, there are changes in the requirements that require that the original program be modified or extended. A new programmer who is assigned to the task has to read the original code to understand what is going on so that a suitable modification/extension can be made.

To solve this problem (and others), Kent Beck introduced eleven "Extreme Programming" practices[2] for improving the software development process. The one practice that addresses the problem listed above is "pair-programming". Simply put, pair-programming involves two individuals working side-by-side during the coding process. One individual (typically called the driver) sits at the keyboard and actually types the program (that is, makes the tactical decisions as to the naming of variables and the exact structure of the statement in the program). The other individual (typically called the navigator) sits and watches the overall process, typically thinking at a higher (strategic) level about whether or not the whole approach is going to work, about situations that might cause the program to fail, or ways of simplifying the approach. After doing this for a short period of time (perhaps a few hours or a whole morning), the individuals switch roles, which has been found to be extremely important to keep the pair functioning at a high level.

This pair-programming strategy has been reported to have many benefits[3–5] including that it boosts efficiency through collaboration, that knowledge is shared, and that there is less of a tendency to rush and cut corners because of peer pressure. Users of pair-programming also report greater job satisfaction.

Of course there are those who feel that pair-programming is wasteful, primarily because the navigator is not actually producing code, and therefore the navigator's labor hours are an unwarranted cost.

To actually measure the costs and benefits of pair-programming, several careful studies have been performed and documented in the literature[6, 7]. They were conducted in a computer science class at a major university, wherein one third of the students performed the assignments alone, and the other two thirds were paired and performed the same assignments. Their overall findings were that the programs developed by pairs were of significantly higher quality (as judged by the number of test cases that the programs passed) than those produced by individuals. Also, every pair always submitted their assignments on time, which was not true for the assignments prepared by individuals. The cost of pair-programming was that the pairs only spent 1.6 (for the first program) to 1.2 (for the last program) times as much time as the individuals. The individuals in pairs also felt that pairing was often the primary reason for their improved performance.

A. ESP Viewer as a Collaborative Tool

The creation of complex models, especially ones suited for analysis, require the expertise of more than one person. Analogous to the pair-programming, a geometry modeler can be responsible for creating a unified template for geometry views, while analysis and design experts provide their expertise to dictate the role of those views downstream in the design process. But unlike pair-programming, the generation of models might involve the interaction of more than two individuals. This would allow the design team to incorporate consistent concepts/views that may need to interact with each other, thus resulting in a unified geometry model.

The geometry system within CAPS is called the Engineering Sketch Pad (ESP), which is a system for creating, modifying, and viewing multi-fidelity and multi-disciplinary models of complex configurations. The models are parametric solid models, which are fully attributed. ESP is implemented using a client-server architecture, where the server runs on any Windows, Linux, or OSX computer and the client is any modern web-browser. The use of a web-browser makes it possible for more than one user to connect to a single session. Therefore, multiple users in multiple locations can easily collaborate in real time.

Other contemporary model-building systems claim to support collaboration, but do it in one of two ways:

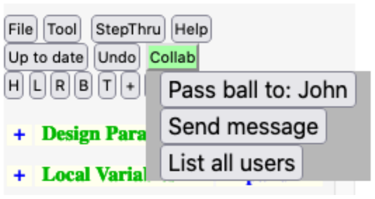
1) Cloud-based applications, wherein team member A creates a model and then publishes it in the cloud; team member B then gets the model from the cloud, add value to it, and publishes an updated model to the cloud; etc. In this way, multiple members can work on the same model, but can only so do sequentially. One recent example is the VOLTA platform[8], from Esteco, which enables an organization to share and re-use engineering knowledge via a web-based environment; and
2) Screen-sharing applications (like Zoom) allow one person to control and other to view the interactions with the modeling software. While better than nothing, these systems do not allow the collaborators to seamlessly change "who is in control"; therefore, there is really one user who can actively contribute.

In ESP, the use of a web-browser makes it possible for more than one user to connect to a single session. Therefore, multiple users in multiple locations can easily collaborate in real time. One user "has the ball", and controls changes to the model; other users can see what the user with the ball is doing and have the option of either synchronizing their display with the user with the ball, or exploring the model independently. Unlike the screen-sharing approaches, the user with the ball can pass it to any other team member, at any time, therefore allowing every team member to contribute at the appropriate time.

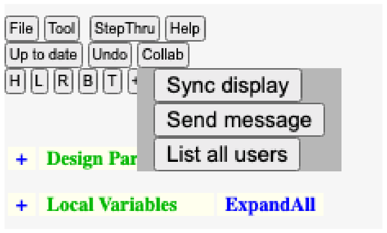
The user with the ball has a little green icon on the screen, which when pressed gives the menu shown in Fig. 2(a); users without the ball have either a white (not synchronized) or yellow (synchronized) icon, which has the attached menu shown in Fig. 2(b). Other than these icons, there is no difference in ESP when there is only one user or when there are collaborators.

IV. Case Study 1: Creating Geometry Concept from Scratch

During the summer of 2021, two Research Experience for Undergraduates (REU) students at Syracuse University were given the task of evaluating the newly-created collaboration mode in ESP. These two students, who had no previous experience with ESP, were tasked with generating two similar models, given the 3-views shown in Fig. 3. The students



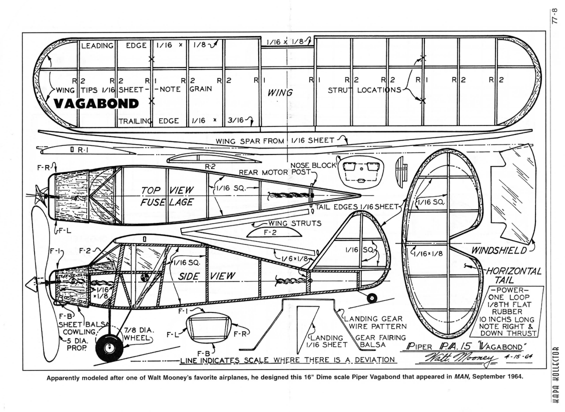
(a) menu for user with ball
(using Firefox)



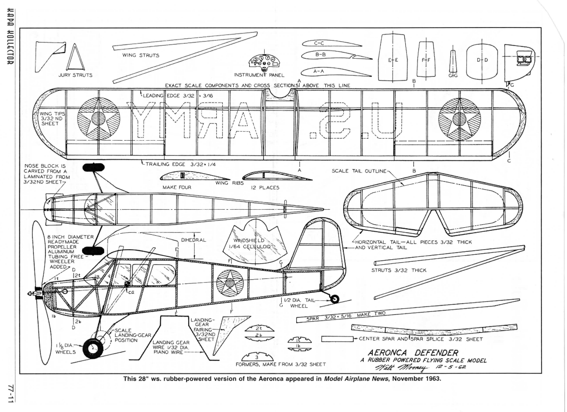
(b) menu for synchronized user
(using Google Chrome)

Fig. 2 Menus on ESP screen when collaboration is active.

were located in different cities, and therefore could only collaborate via web-based means. For the first model, they were told to use whatever tools with which they were comfortable, and to record their experiences. For the second model, they were told to use the new ESP collaboration mode. Since the two models that they produced, which are shown in Fig. 4, were their first two models, it is not fair to compare the times associated with the two models (since there was quite a learning curve associated with making such complex models).


VAGABOND
WING
TOP VIEW FUSELAGE
SIDE VIEW
WINDSHIELD
HORIZONTAL TAIL
PIPER P.A. 15 VAGABOND
Apparently modeled after one of Walt Mooney's favorite airplanes, he designed this 16" Dime scale Piper Vagabond that appeared in MAN, September 1964.

(a) Piper Vagabond[9]


U.S. ARMY
AERONCA DEFENDER
A RUBBER POWERED FLYING SCALE MODEL
This 28" ws. rubber-powered version of the Aeronca appeared in Model Airplane News, November 1963.

(b) Aeronca Defender[10]

Fig. 3 3-views given to students for case study 1

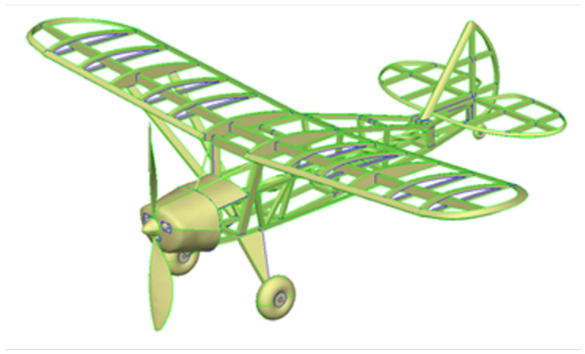
A. Traditional Collaboration Methods

Pair programming

In this method, the two students worked simultaneously on one model. Since this was the first collaboration mode, the two students were learning how to use ESP together. They used a screen-sharing application (like Zoom), where one student was the driver and the other was the navigator. The parts of the model that they developed using pair programming are shown in Fig. 5(a).

The advantages that they noted were:

- they could learn to use ESP at the same time. The driver mainly focused on the model being developed, while the navigator could look things up in the User's Manual, etc.;
- there was clear communication between the two students; and
- they caught errors earlier.


(a) Piper Vagabond

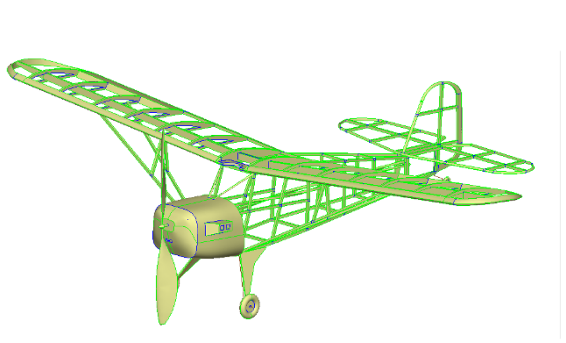
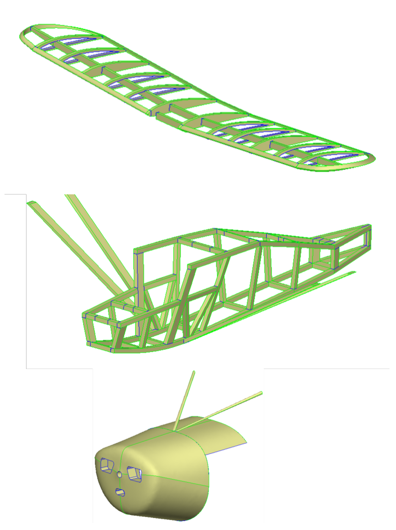

(b) Aeronca Defender

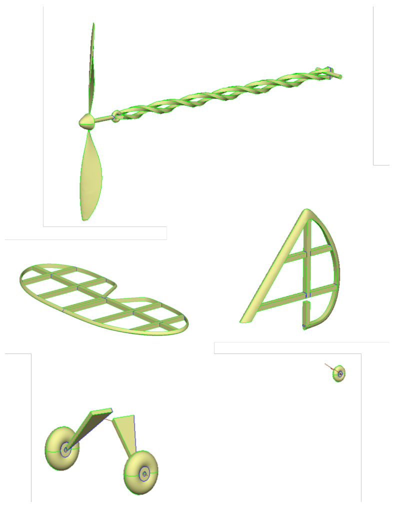
Fig. 4 `ESP` **models produced by students for case study 1**

The disadvantages that the noted were:

- the pace was perceived to be relatively slow, since there were two of them working on the same model at the same time;
- they could not easily switch driver/navigator roles, and hence there was the tendency for the navigator to become less focused; and
- working in this way required a rigid schedule, since both students needed to be involved in the task at the same time.

Nonetheless, the models they were able to produce were quite remarkable for first-time users of ESP.


(a) pair programming


(b) individual development

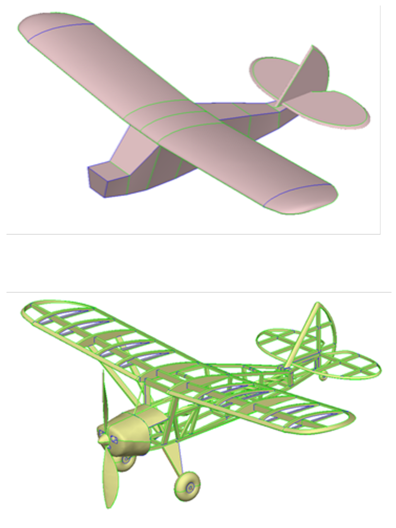

(c) parallel development

Fig. 5 Parts of model developed in each phase for case study 1

Individual Development

In this method, the two students worked on his own schedule and own computer. The `.csm` scripts that they developed were each written in the student's own style. The parts of the configuration that were made during this process were the propeller, landing gear, and tail section. The students used this development method because they were working from a common model of the wing and fuselage, and because the parts of the configuration that they were working on did not interact with each other. Hence, they anticipated that merging their changes would be an easy task. The parts of the model that they developed using individual development are shown in Fig. 5(b).

They noted several advantages of this collaboration mode:
- fast pace, since each student could work on his own schedule; and
- the workload was shared in that each student could work on a different part of the model.

The disadvantages that they noted were:
- there were a few compatibility issues between the various parts of the model, even though they had sought to have agreed-upon conventions to limit them; and
- their models were more prone to errors, since it is often hard for a user to see the errors in his own work.

Parallel Development

In this method, the students worked alone, but from the same script. Parallel versions of the `.csm` script were edited by the students, and then merged. The specific tack at hand was to add skins to the existing model. They used conference software (such as Zoom) to brainstorm solutions and resolve any discrepancies that might arise. The parts of the model that they developed using parallel development are shown in Fig. 5(c).

The advantages of this collaboration mode included:
- the students could edit the script on his own schedule, thereby alleviating the need to synchronize their schedules;
- on-line collaboration time was focused on strategic issues, such as brainstorming; and
- this seemed to be faster than pair programming.

The disadvantages included:
- compatibility issues arose, even though they thought they had sufficient communications to avoid them;
- errors in the scripts were hard to find; and
- it was slower than individual collaboration.

B. `ESP` Collaboration

For the second model that they collaboratively developed (Fig. 4(b)), the new `ESP` collaboration mode was used. Since they were at remote locations, they used a shared virtual machine that was made available by Syracuse University. Access to this machine was not very easy, as they needed to log into the Syracuse network and then log into the shared virtual machine. Even so, their overall impression of the new `ESP` collaboration mode was favorable.

They cited these advantages:
- the `.csm` scripts were stored in a common workspace (on the shared virtual machine), and thus were accessible to both students at all times;
- since the `ESP` server was on the shared machine, the only software that they needed on their own computers were the browser-base code (`.html` and `.js` files). As a result, the fact that they had relatively slow personal computers did not slow them down;
- the driver (user with the ball) could make any required changes and the results of the changes would be seen by both students almost immediately;
- this kept both users actively engaged in the process;
- the navigator tended to focus on big-picture (strategic) issues while the driver focused on typing appropriate commands, or tactical issues; and
- overall they felt that having two brains working on the same problem was beneficial.

The disadvantages that they noted included:
- users cannot simultaneously edit a script;
- the process of saving a script was fragile, and required a workaround;
- accessing the shared virtual machine sometimes took considerable time; and
- there was only one version of the `.csm` script, so if a hard-to-find error was injected into the script, there was not a readily available backup.

Based upon their evaluation, several improvements were made to the `ESP` collaboration environment:

- all users can see the `.csm` script as it is being edited (even though only the user with the ball can edit it at any time); and
- several bugs associated with “passing the ball” were resolved.

Also based upon their evaluation, the following best practices were established:
- the computer on which the ESP server is running should be easily accessible by all users;
- there should be a shared location in which all scripts are stored; and
- models should be versioned so that recovery from a hard-to-find error is possible.

V. Case Study 2: Creating Multi-disciplinary Model from Scratch

For the second case study, the authors of this paper set out develop a multi-disciplinary model of the wing of a transport aircraft, similar to the Boeing 787. The case study starts from the 3-view shown in Fig. 6. The first few steps in the process mimic the way models are typically developed, that is via sequential collaboration. For the last two steps, the model was quite a bit more complicated and so the ESP collaboration environment was used. While the advantages of the new collaboration tool are anecdotal, they are consistent with the advantages reported in the literature for pair programming.

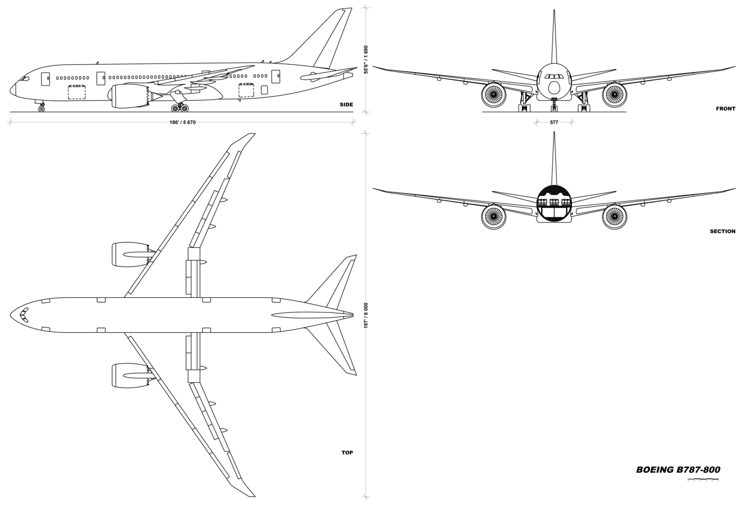


Fig. 6 Line Drawing of Aircraft [11]

A. Step 1: Initial Non-Parameterized Outer Mold Line (OML)

As is typical in the development of a new model (especially one crafted by a student), the original model, shown in Fig 7(a) was created by measuring a printed-out copy of the 3-view (shown in Fig. 6) and creating a non-parameterized model. The script for this model is:

```
# collab1
# Engineer A: basic wing OML with hardwired dimensions

# dimensions in mm based upon printed 3-view
#DESPMTR   wing:area
#DESPMTR   wing:aspect
#DESPMTR   wing:ybreak
#DESPMTR   wing:taper_ob
#DESPMTR   wing:sweep
#DESPMTR   wing:dihedral
```

```
#DESPMTR   winglet:area
#DESPMTR   winglet:aspect
#DESPMTR   winglet:taper
#DESPMTR   winglet:sweep
#DESPMTR   winglet:dihedral

UDPRIM    naca      thickness 0.12   camber 0.06
SCALE     31.5
ROTATEX   90

UDPRIM    naca      thickness 0.12   camber 0.04
SCALE     31.5-24.0/62.5*43.0
ROTATEX   90
TRANSLATE 24.0/62.5*43.0   24.0   24.0/62.5*8.0

UDPRIM    naca      thickness 0.04   camber 0.04
SCALE     48.0-43.0
ROTATEX   90
TRANSLATE 43.0   62.5   8.0

UDPRIM    naca      thickness 0.04   camber 0.00
SCALE     52.0-50.0
ROTATEX   90
TRANSLATE 50.0   65.0   9.0

RULE

# scale to ft
SCALE     197/2/65

END
```

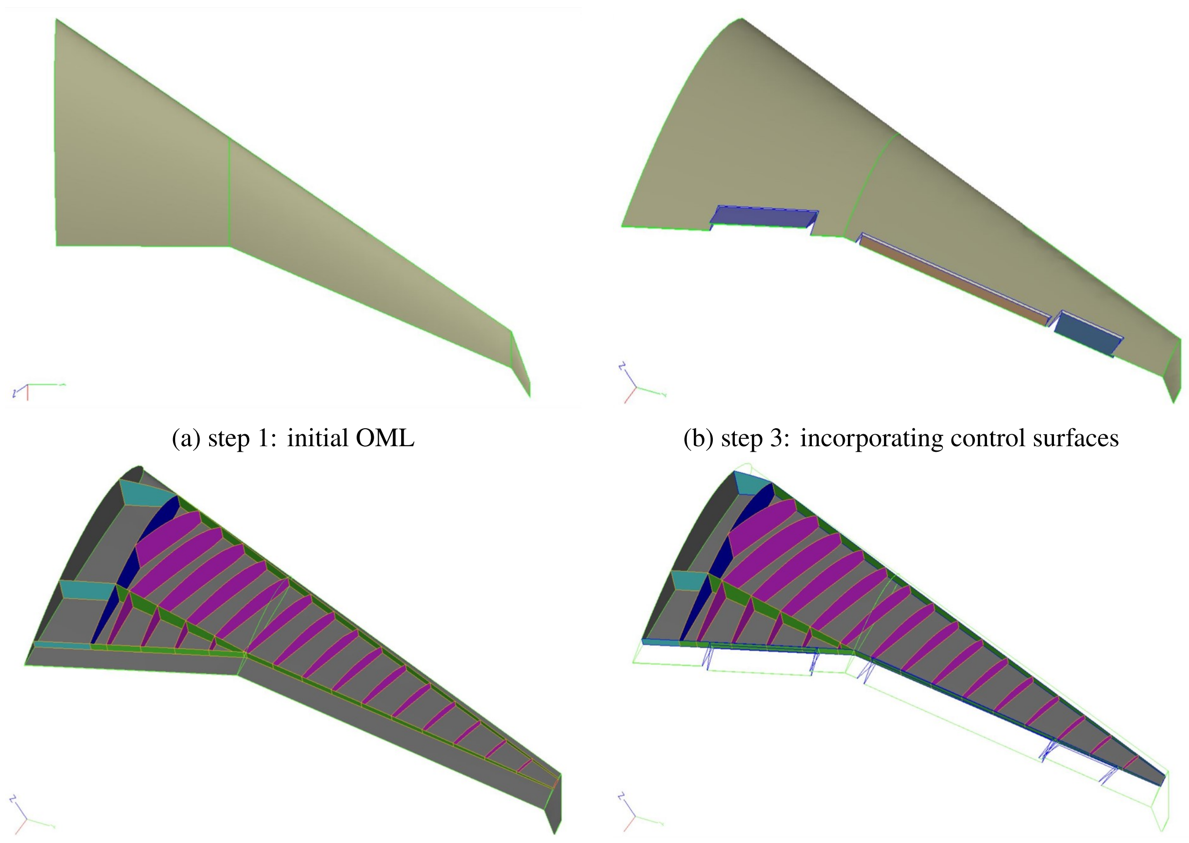
Note that there are comments for the expected design parameters (DESPMTRs) near the top of the script. This model was created by engineer A. The model was kept as version 1.

B. Step 2: Parameterizing the OML

Engineer B then had the task of parameterizing the model. The first step that engineer B performed was to examine the `.csm` script in order to understand the organization that engineer A used. This was done via code inspection as well as stepping-through the build process and observing on the evolution of the model. Ultimately the standard definitions for wing area, aspect ratio, taper ratio, leading edge sweep, and dihedral were used and the appropriate equations added to the script. In addition, values were computed for the DESPMTRs that were consistent with the original non-parameterized dimensions. The model created by the resulting script:

```
# collab2
# Engineer A: basic wing OML with hardwired dimensions
# Engineer B: update OML to use DESPMTRS  (wingOML)

# dimensions in mm based upon printed 3-view
DESPMTR   wing:area        1886.
DESPMTR   wing:aspect      8.285
DESPMTR   wing:ybreak      24.0
DESPMTR   wing:taper_ob    0.333
DESPMTR   wing:sweep       34.5
DESPMTR   wing:dihedral    7.30
```


(a) step 1: initial OML

(b) step 3: incorporating control surfaces

(c) step 4: initial structural model

(d) step 5: final model

Fig. 7 Model development throughout case study 2

```
DESPMTR   winglet:area     17.5   # both
DESPMTR   winglet:aspect   1.43
DESPMTR   winglet:taper    0.40
DESPMTR   winglet:sweep    70.3
DESPMTR   winglet:dihedral 21.8

SET        wing:span           sqrt(wing:area*wing:aspect)
SET        delta               wing:ybreak*tand(wing:sweep)
SET        num                 "(wing:area+delta*wing:ybreak\
                               +delta*(1+wing:taper_ob)*(wing:span/2-wing:ybreak))"
SET        den                 "(2*wing:ybreak\
                               +(1+wing:taper_ob)*(wing:span/2-wing:ybreak))"
SET        c1                  num/den
SET        c2                  c1-delta
SET        c3                  c2*wing:taper_ob

SET        winglet:span        sqrt(winglet:area*winglet:aspect)
SET        c4                  c3*winglet:taper

...
```

was compared visually with the first model to ensure that no mistakes were made. This new model, which was visually identical to the model shown in Fig. 7(a), was kept as version 2.

C. Step 3: Incorporating Control Surfaces

After conferring with engineer A, engineer B then added control surfaces (an aileron and inboard and outboard flaps) to the model, as shown in Fig. 7(b). This "sequential" mode was done because engineer B had extensive experience adding control surfaces to a model, whereas engineer A hadn't done so for several years. The original control-surface parameterization was in terms of span and chord for each control surface. After team discussions, it was determined that a slightly different parameterization:

```
DESPMTR   fuselage:width  6.5      # inboard span begins from here
DESPMTR   flapgap         0.2      # mm

DESPMTR   ibflap:spanbeg  0.2      # fraction of inboard span
DESPMTR   ibflap:spanend  0.8      # fraction of inboard span
DESPMTR   ibflap:chord    0.20     # fraction of tip chord
DESPMTR   ibflap:angle    20.0

DESPMTR   obflap:spanbeg  0.05     # fraction of outboard span
DESPMTR   obflap:spanend  0.65     # fraction of outboard span
DESPMTR   obflap:chord    0.50     # fraction of tip chord
DESPMTR   obflap:angle    25.0

DESPMTR   aileron:spanbeg 0.67     # fraction of outboard span
DESPMTR   aileron:spanend 0.85     # fraction of outboard span
DESPMTR   aileron:chord   0.25     # fraction of tip chord
DESPMTR   aileron:angle  -10.0
```

was preferred. Note (for future reference) that the chordwise extent of each control surface could be specified independently (that is, the outboard flaps had a much bigger chordwise extent than the aileron did); this turned out to be a problem in future steps. (Also, there was a hidden error in the model that was not detected until step 5.) This new model was saved as version 3.

D. Step 4: Adding Structural Model

Adding the structural model was quite a bit more challenging than either the OML or the control surfaces. As such, two-heads were better than one, making step 4 ideal for the new ESP collaboration mode. This process started off with a sketch, which showed the basic layout and the associated labeled "points" on the layout. Zoom was used to share the sketch and for person-to-person communication. Engineer B was the driver and engineer A was the navigator.

The first step was to create a strategy for positioning the forward and rearward spars. After much discussion, it was decided to create a construction line along the leading edge, offset it by a prescribed distance, and then to define the forward spar to fall along the displaced construction line and terminate at the aircraft centerline and the wing tip. A similar strategy was used for the trailing edge spar.

Then the roles of the navigator and driver were reversed. The next step was to create the spar along the trailing edge of the inboard wing section, as well as `nrib` ribs. In the process of doing this, a typographical error was introduced into the script, which the driver could not figure out. By working together, the navigator was able to see the error and told the driver how to fix it.

Again the roles of the navigator and driver were switched and the rearward ribs as well as the structure that is inside the fuselage were created.

After switching roles again, the surfaces of the wing were scored by the ribs and spars into panels and the ribs and spars were intersected by the OML to provide their proper shapes.

After the final switch of roles, the various structural elements were colored to visually identify the ribs, spars, skin panels, fuselage header, and carry-through structure. The resulting model is shown in Fig. 7(c) and was saved as version 4.

E. Step 5: Finalizing the Model

To finalize the model, engineer A performed various clean-ups. Recall that the original dimensions were in millimeters, as measured on the printed copy of Fig. 6; the final step in the build process was to scale the model up to its actual size. This was going to be problematic long term, since the dimensions used during the build were not based upon reality. Therefore the DESPMTRs were converted to feet and the scaling at the end was eliminated. Also, the part of the structure forward of the forward spar, rearward of the rear spar, and outboard of the tip were removed from the structural model, giving the model shown in Fig. 7(d). This was saved as version 5.

Once the model was developed, various design parameters were changed and the model re-built to ensure (at least visually) that the model was properly developed. At this point, it was discovered that there was a typographical error introduced during a cut-and-paste operation, wherein the aileron deflection was actually being controlled by the `obflap:angle` parameter. This was subsequently fixed in the final model.

VI. Summary

A new collaboration environment, patterned after the pair-programming paradigm, has been implemented in the Engineering Sketch Pad (`ESP`). In this new capability, which is enabled by `ESP`'s use of a web browser for its user interface, multiple users can simultaneously connect to the same session. Control of `ESP` can easily be switched to another user by "passing the ball". This new capability was tested using two case studies, with the advantages and disadvantages noted. Overall, the new collaboration environment was found to have several benefits, including a shared-ownership of the model, a tendency to take fewer short-cuts, and a lower error rate. Shared ownership is particularly important for models that evolve over time due to changing requirements. While the total labor time increase was generally small, there was a situation (in case 2, step 4) in which the labor time actually decreased because of the complexity of the task.

Acknowledgments

This work was supported by the EnCAPS Project (AFRL Contract FA8650-20-2-2002): "EnCAPS: Enhanced Computational Prototype Syntheses", with Ryan Durscher as the Technical Monitor. The second author would like to thank the Aerospace Systems Directorate, at the Air Force Research Laboratory, for support. The authors would also like to thank Cody VanNostrand and Marco Svolinsky (who executed) and Paul Mokotoff (who supervised) the first case study.

References

[1] Bryson, D.E., Haimes, R., and Dannenhoffer, J.F., III, "Toward the Realization of a Highly Integrated, Multidisciplinary, Multifidelity Design Environment", AIAA SciTech 2019 Forum, American Institute of Aeronautics and Astronautics, San Diego, CA, 2019.

[2] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[3] "Extreme programming", *https://en.wikipedia.org/wiki/Extreme_programming*

[4] Sharpe, J., "Introduction to Pair Programming", *https://blog.jonrshar.pe/2017/Oct/13/ada-college-pairing.html*

[5] Chong, J., Plummer R., Leifer, L., Klemmer, S., Eris, O., and Toye, G., "Pair programming: When and why it works", *The 17th Workshop of the Psychology of Programming Interest Group*, 2005

[6] Cockburn, A., Williams, L., "The costs and benefits of pair programming", *In eXtreme Programming and Flexible Processes in Software Engineering XP2000*, 2000

[7] Williams, L., Kessler, R., Cunningham, W., Jeffries, R., "Strengthening the Case for Pair Programming", *IEEE Software*, Volume 17, Issue 4, 2000

[8] "Volta Data Manager", *https://engineering.esteco.com/volta/volta-capabilities*, 2021.

[9] Buffardi, L.N., (editor) "The Kapa Kollector", *www.flyingacesclub.com/KAPA/ISSUE77.pdf*, March 2012.

[10] *ModelAirplane News*, November, 1963.

[11] Scavini, J., "Schematics of Boeing 787-800. Side, top, front, section", *https://commons.wikimedia.org/wiki/File:B787-800v1.0.png*, 2011