Human Computer Interaction course

Part 3

ENSEEIHT

December 2024

Course content

- Course 1
  - HCI, HSI, distributed systems, interactive software engineering
  - First contact with Ingescape
  - Presentation of the exam
- Course 2
  - Exam groups
  - HCI & UX methodologies
  - Visual programming with Ingescape
- **Course 3**
  - **Software architecture for HCI development**
  - **Generating code and crossing models for interactive applications**
  - **Verification & Validation applied to interactive systems**
- Course 4
  - Methodologies for multidisciplinary and iterative System Engineering, notions of HSI
  - Human Factor assessments, why and how
  - Co-simulation and data record/replay with Ingescape
- Course 5
  - Practical exchanges on your exam projects using system architecture models

Software architecture for HCI development

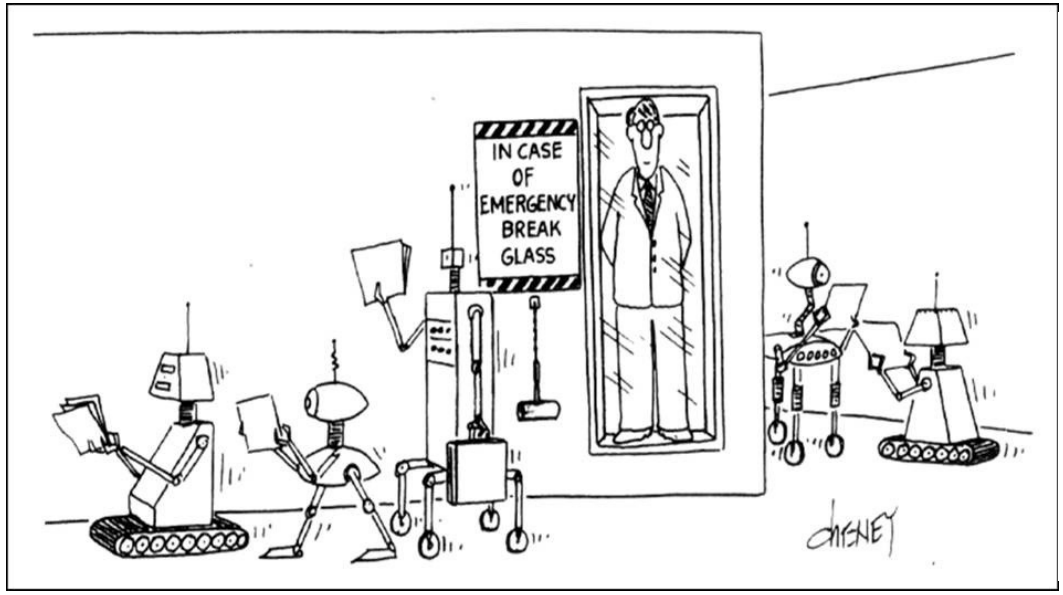
Interactive system

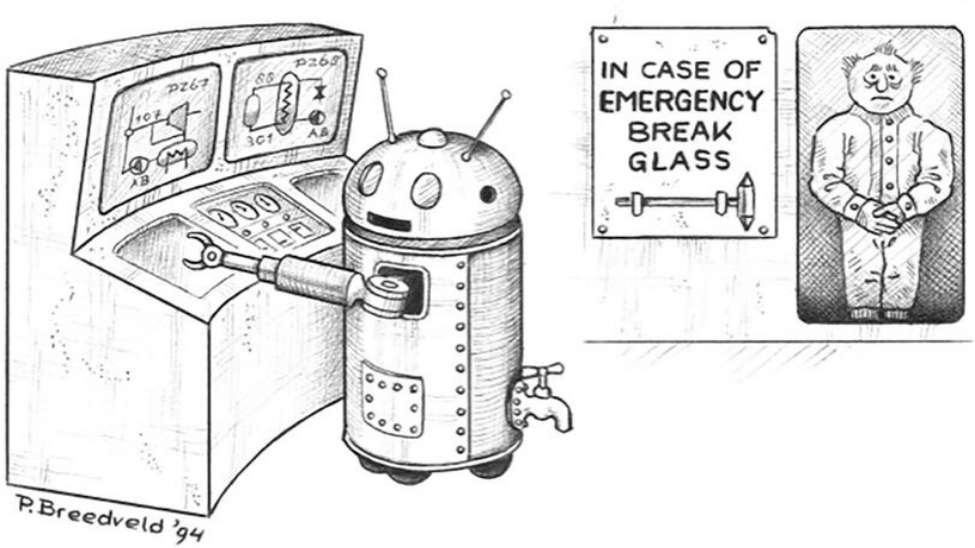
- A computer system that:
  - Holds an internal state
  - Creates perceivable representations of parts of this state
  - Reacts to user input (almost) immediately

- Properties of interactive systems:
  - **Reactive**: user provides input to system; system must process it immediately and generate output to user
  - **Open**: dependencies between system's output and user's future inputs are unknown to system
  - **Asymmetric**:
    - User does not have to react immediately to system
    - User must be able to know / anticipate the dependencies between input and output

Two conceptions of human-computer systems

- "Human in the loop"
  - System-centric view where the user must conform to the system's rules, e.g. provide input in a specific order or format
  - Addresses operational tasks where the user performs actions that the system cannot (yet) do





Two conceptions of human-computer systems

- “Computer-in-the-loop”
  - Human-centric view where the computer must be adapted to the capabilities of the user
  - Addresses creative tasks where the computer extends or augments the capabilities of the user

How it started

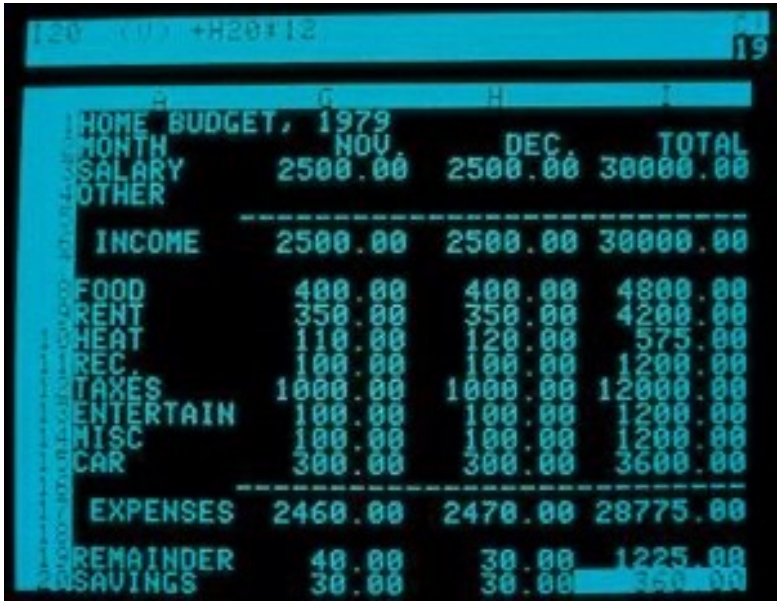
- Ways to organize code rather than tools
- Separation of UI and rest of software
- Helps think about modularization and organization
- “Models” of a UIMS (User Interface Management System)
  - Term coined after Data Base Management System (DBMS)

“The DBMS mediates between programmer and data, enforcing consistency of technique among all programmers in accessing that data. It provides portability because only the lowest-level DBMS routines are hardware-dependent. Similarly, the UIMS mediates between the applications programmer and the input events, encouraging a consistency both of graphical layout representation and of input processing mechanisms.”

W. Buxton, M. R. Lamb, D. Sherman, and K. C. Smith. Towards a comprehensive user interface management system. ACM Computer Graphics - Proc. SIGGRAPH, 17:35–42, July 1983

State of HCI at the beginning of the 80s - Industry

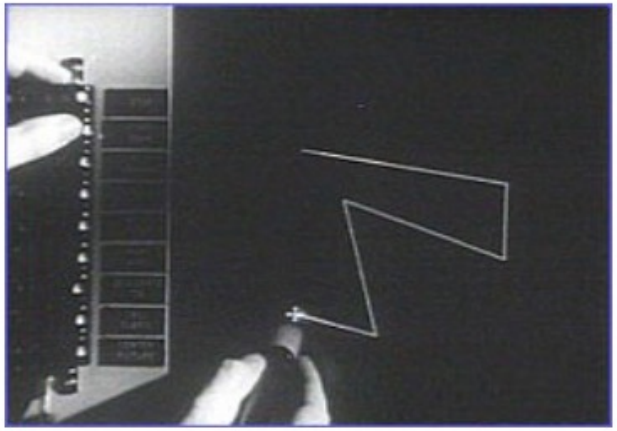
- Visicalc – Dan Bricklin (1979)
  - First spreadsheet app

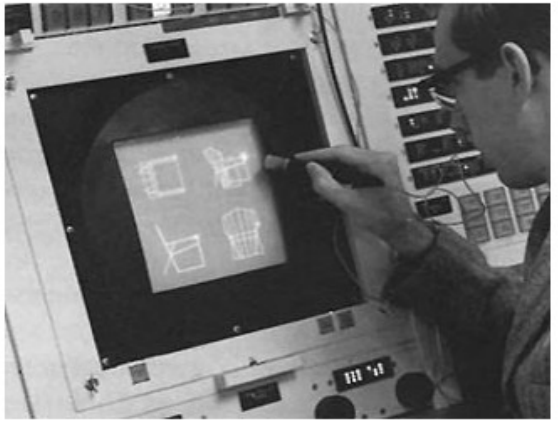


- CATIA – Dassault Systèmes (1981)

State of HCI at the beginning of the 80s - Research

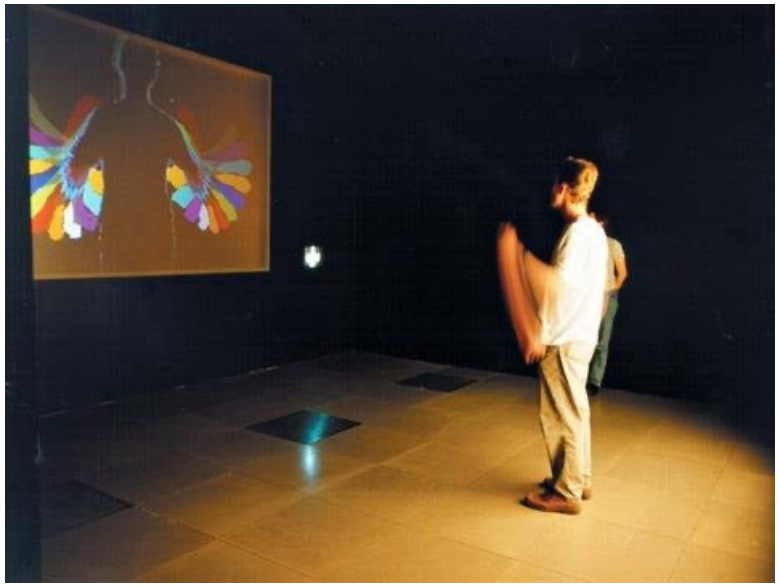
- Sketchpad – Ivan Sutherland (1963)
  - Direct manipulation of geometric shapes, zoom, click-drag, etc.





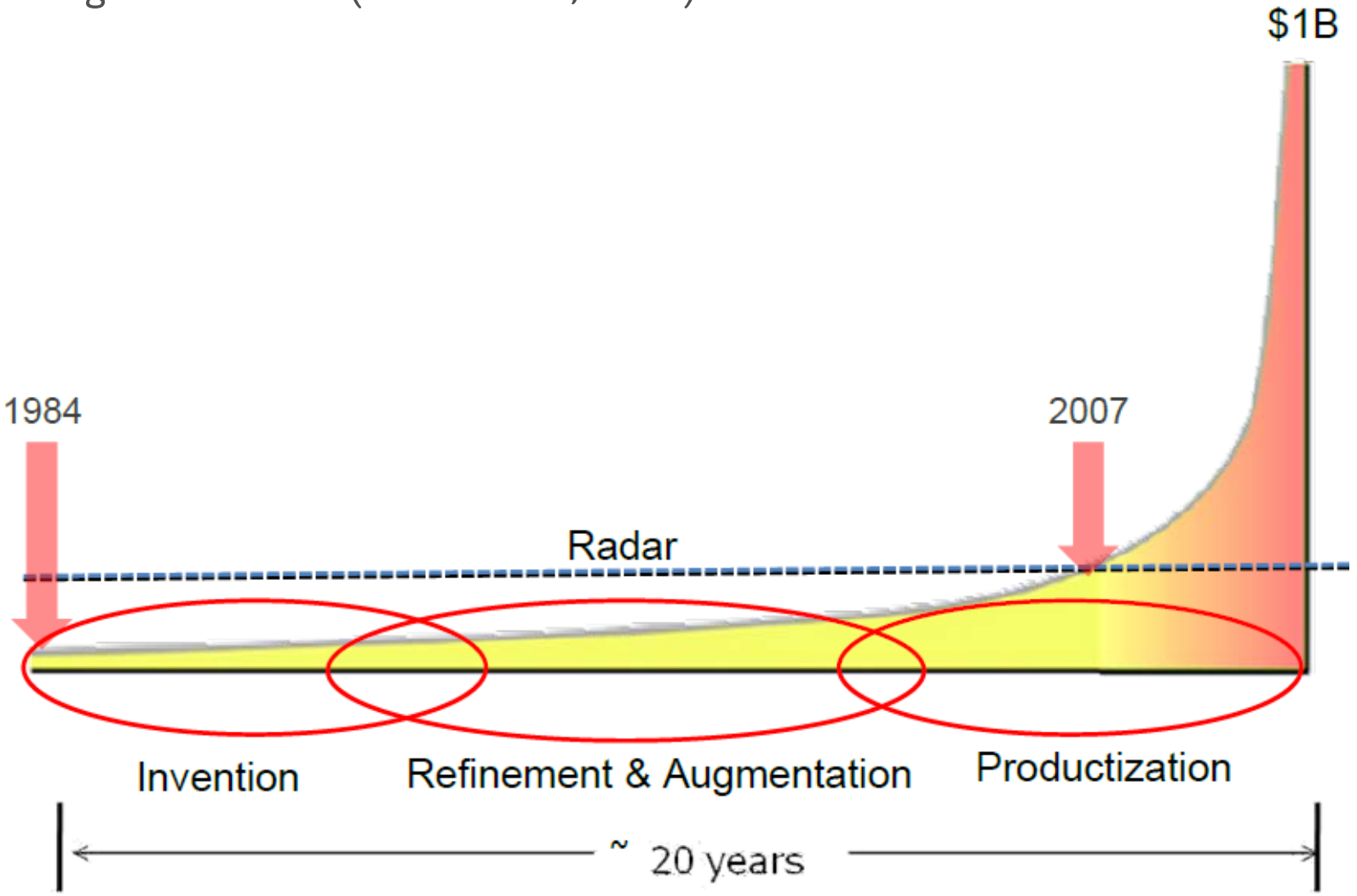
- Put-that-there: Bolt (1980)
  - Multimodal interactions

VideoPlace – Krueger (1983)
Multiple fingers, hands and people interactions



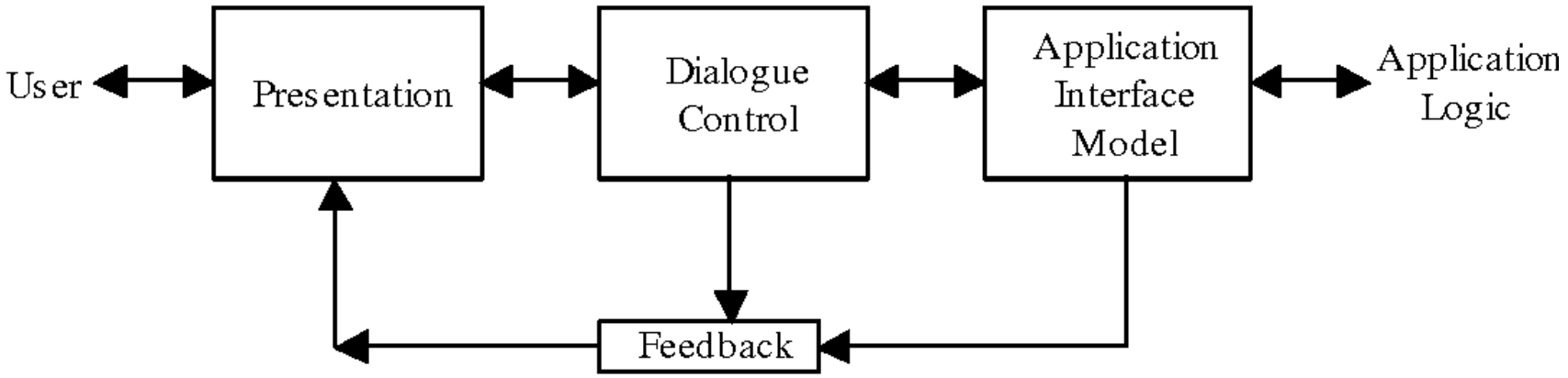
The long nose of innovation

- Visions are important
  - E.g. multitouch (Bill Buxton, 1984)



Seeheim model

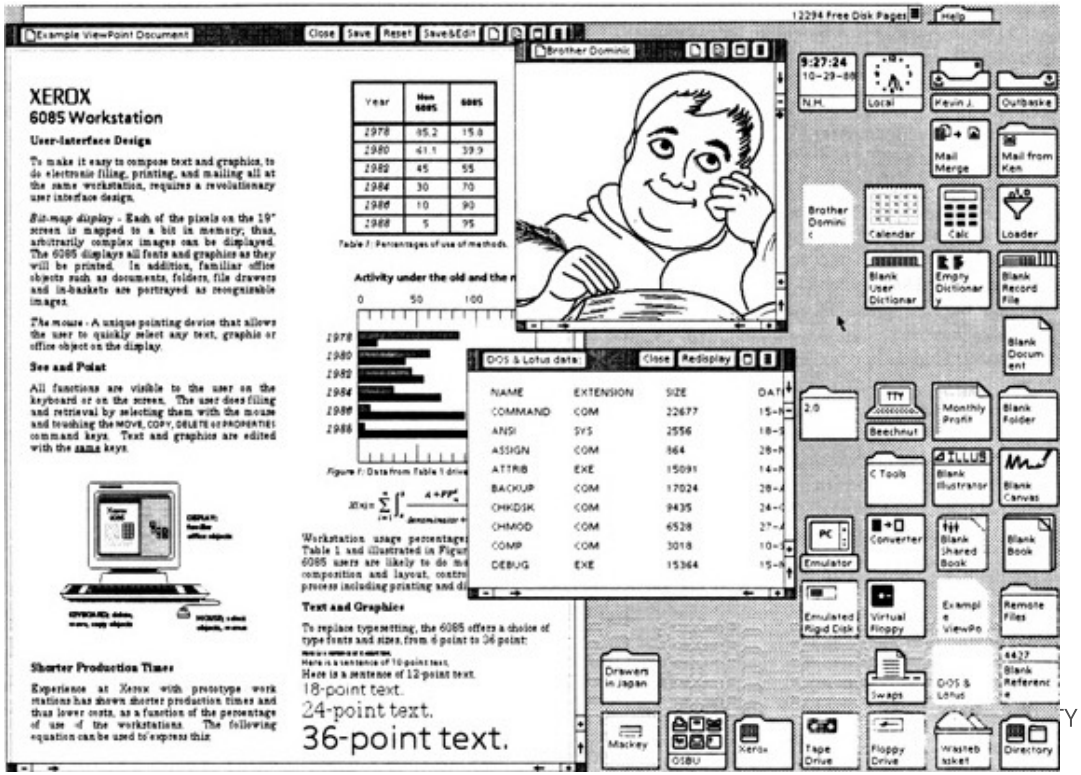
- Resulted from the 1st UI software tools workshop which took place in Seeheim, Germany (Nov 1-3, 1983)



- **Presentation**:
  - Manage input and display at a low level
- **Dialogue control**:
  - Validates input and transforms it into commands
  - Transforms responses from the Application Interface Model into graphical capabilities
- **Application Interface Model**:
  - Adapts the functional core (app logic) to the needs of the UI

Revolution: WIMP paradigm / desktop metaphor

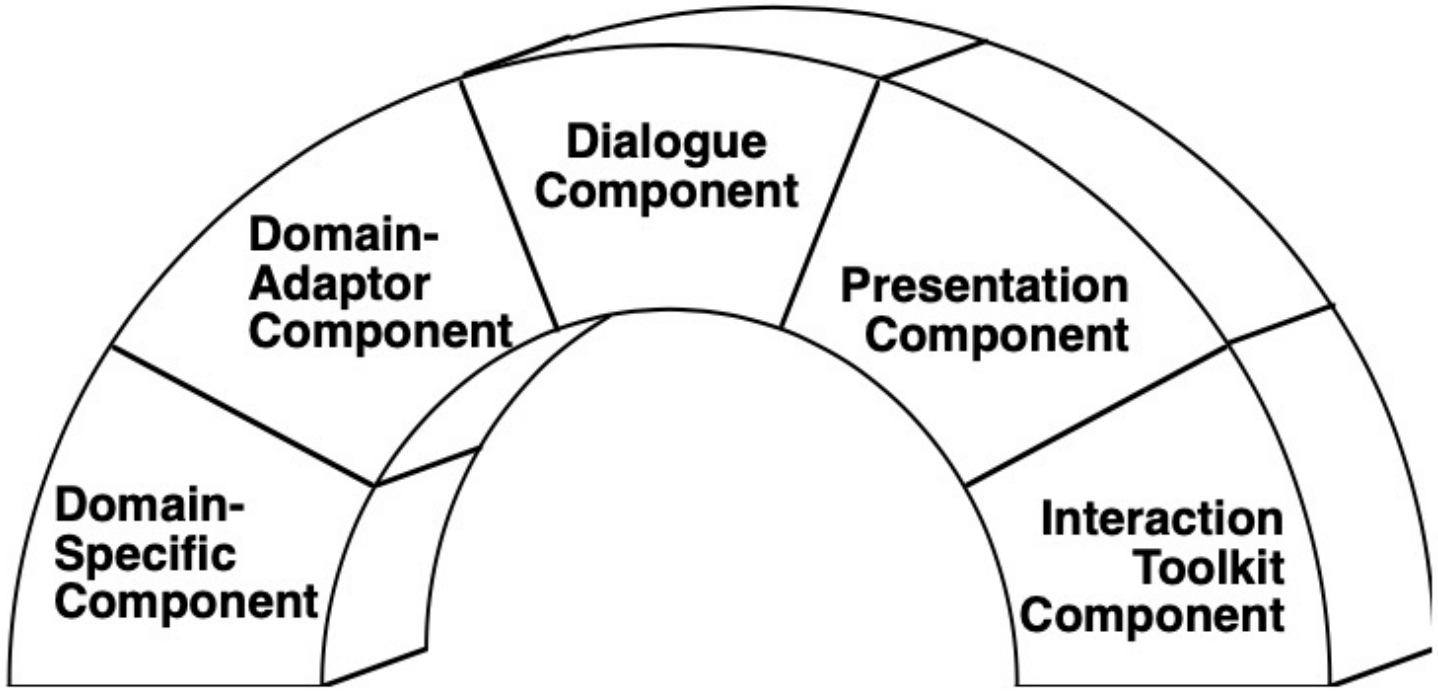
- Windows, Icons, Menus, Pointer
- Direct manipulation of graphical objects
- **WYSIWYG**: What You See Is What You Get
- Invented by the Xerox PARC
  - Xerox Alto (1973): first computer with an OS based on a graphical user interface
  - Xerox Star (1981): first commercial graphical workstation
- Popularized by Apple with the Macintosh (1984)



Arch/Slinky model

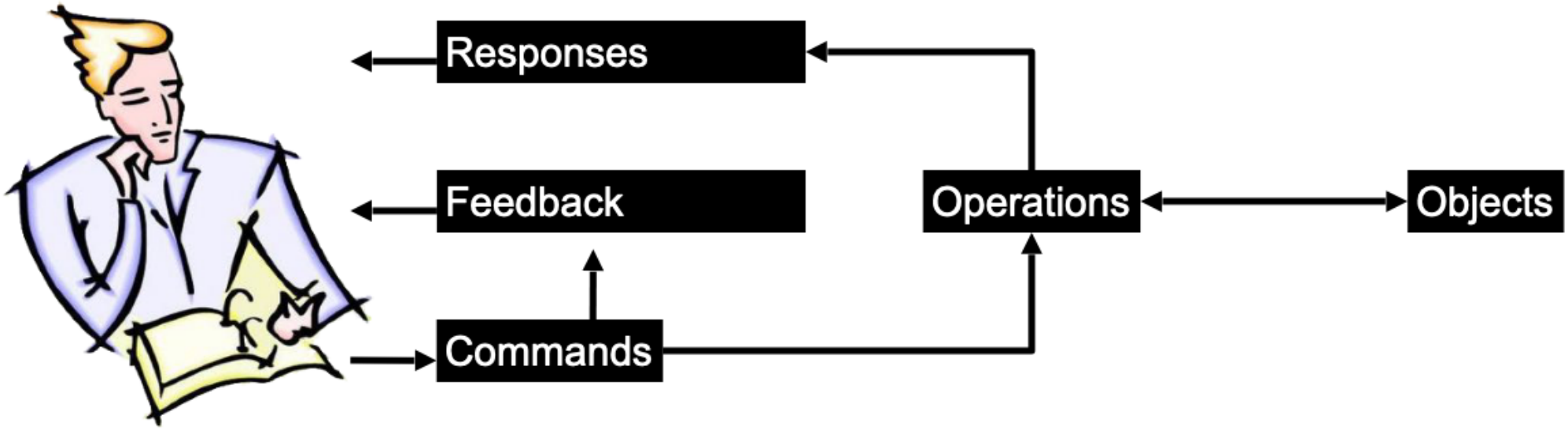
Bass, R. Faneuf, R. Little, N. Mayer, B. Pellegrino, S. Reed, R. Seacord, S. Sheppard, and M. Szczur, 1992. "A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop", ACM SIGCHI Bulletin. 24 (1), 32–37. Jan, 1992

- Revision of the Seeheim model
  - Acknowledges the existence of UI toolkits
  - Adaptor components to ensure a better independence between parts
  - Components can be of different sizes, or even non-existent



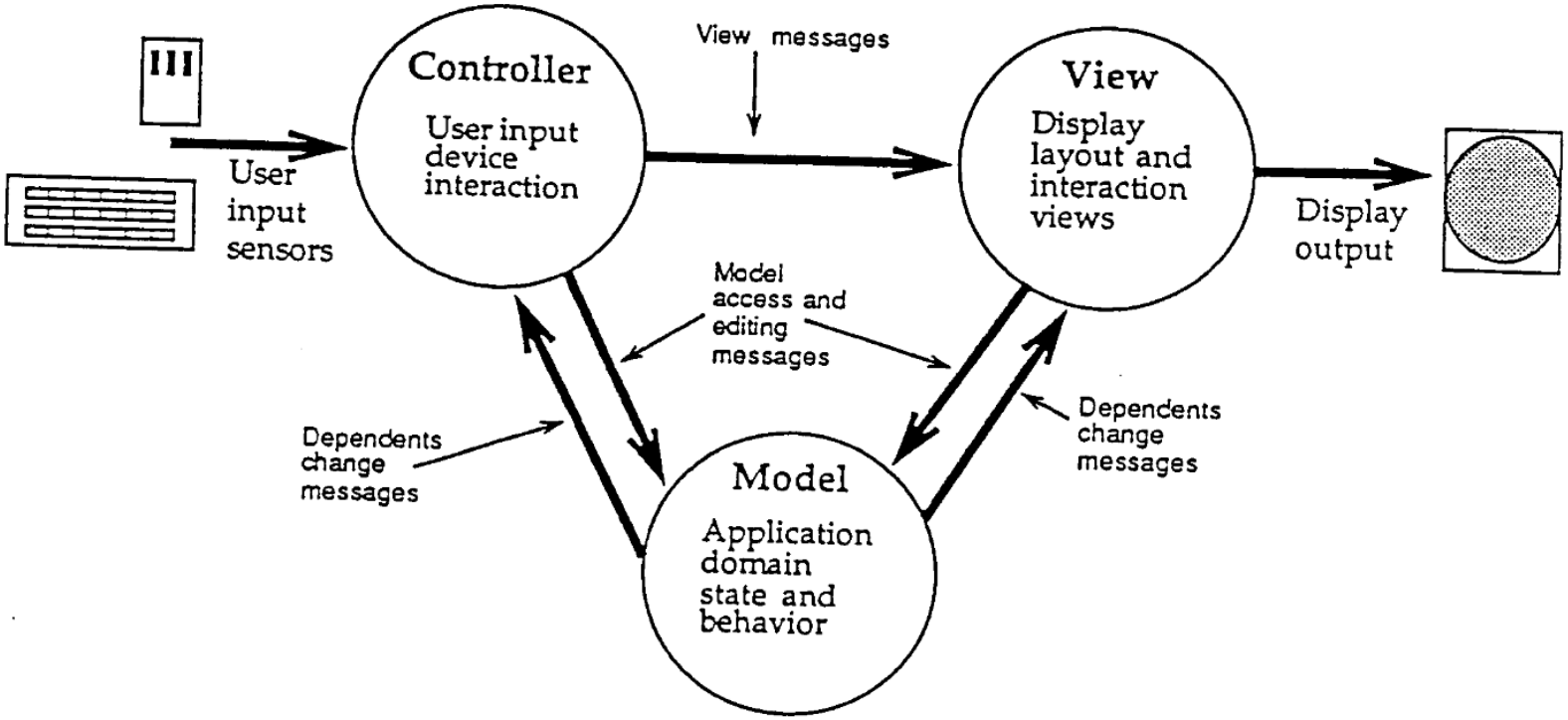
Conceptual model

- Model of how the system operates
  - Ideally, matches the user's mental model



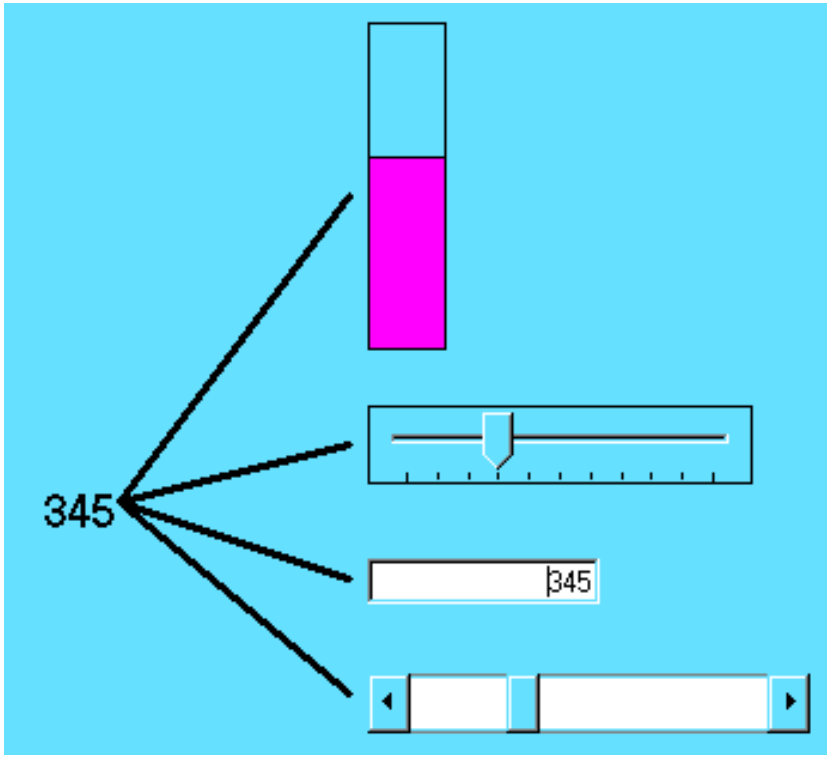
MVC: Model-View-Controller (1/2)

- Invented in Smalltalk (1979-1983)
  - Still the most commonly-used architecture (revised MVC or variants)
- **Idea**: separate out presentation (View), user input handling (Controller) and logics (Model) which does the work
- **Goal**: modular design
  - Program a new model, and then re-use existing views and controllers
  - Use multiple, different kinds of views on same model



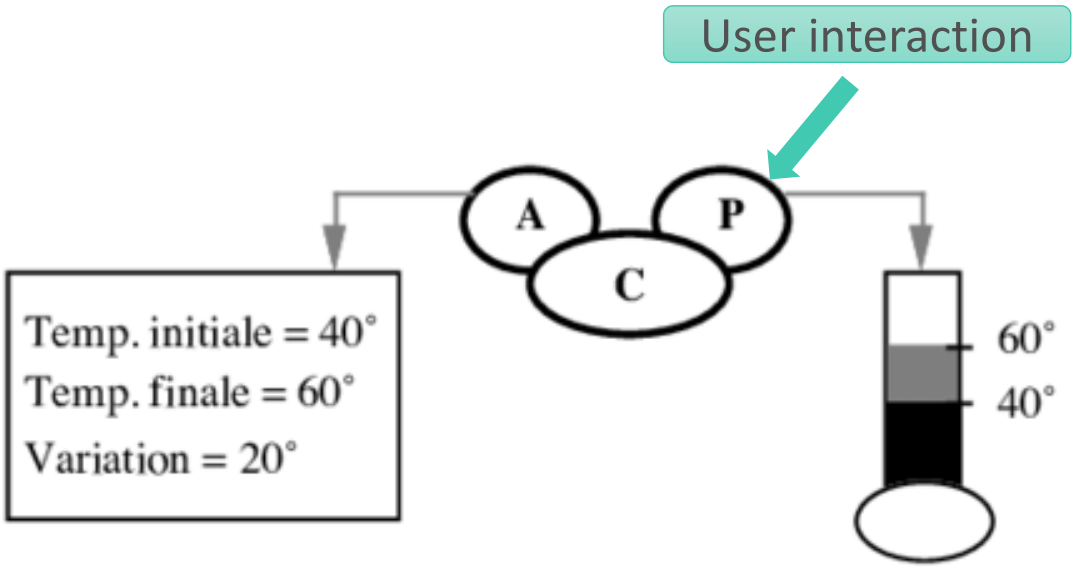
MVC: Model-View-Controller (2/2)

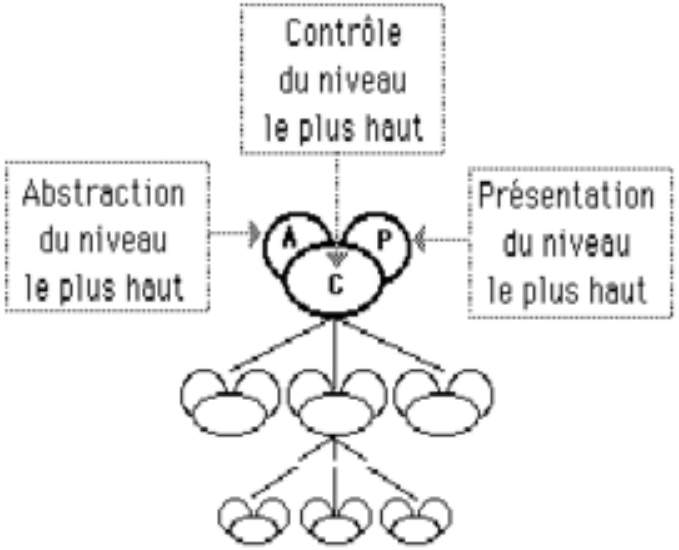
- Fairly straightforward in principle, hard to carry through
  - Views are closely associated with controllers
  - VCs know about their model explicitly
    - Each VC has one M; one M can have many VCs
  - Complexities with views with sub-parts, controllers with sub-controllers, models with sub-models



PAC: Presentation – Abstraction - Control

- Invented by Joëlle Coutaz, 1987
  - Numerous evolutions: PAC-Amodeus, PAC*, CoPAC, PAC-C3D, etc.
- Tree of agents with 3 facets each:
  - **Presentation** (≃ VC)
  - **Abstraction** (≃ M)
  - **Control**: handles flows between Presentation and Abstraction, and communicates with other agents through their control part





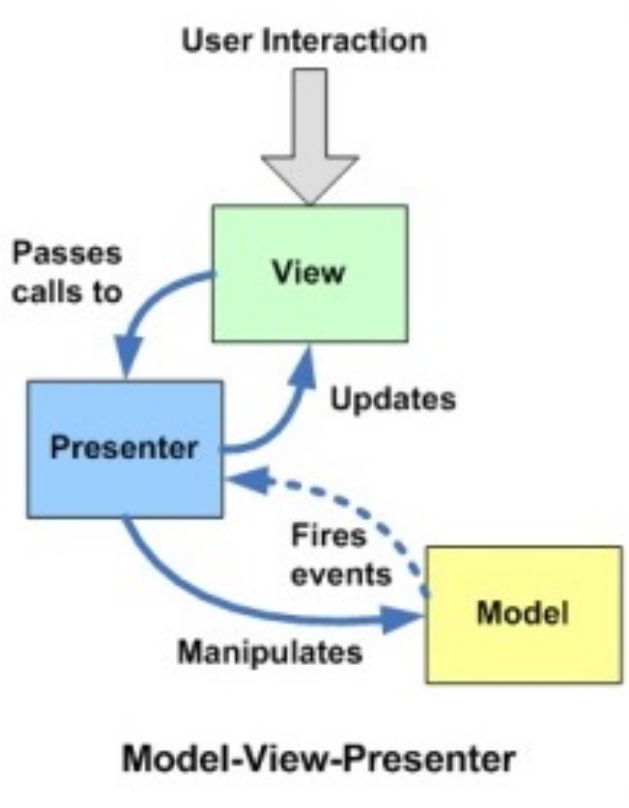
Other famous variants

MVP

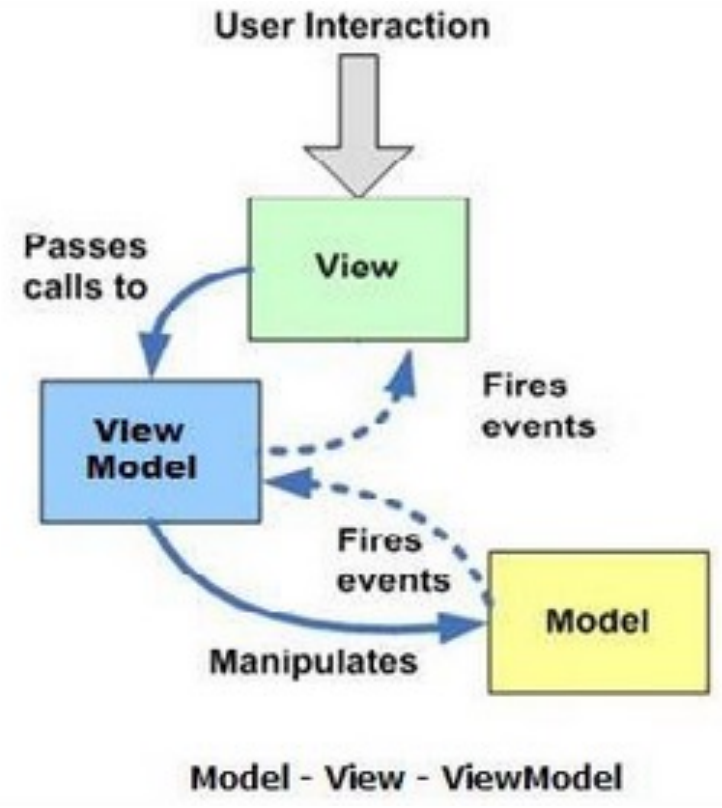
- Taligent, 1990
- Facilitate automated unit testing
- Java Swing, Windows Forms, etc.

MVVM (a.k.a. Model-View-Binder)

- Microsoft, 2005
- Remove all code from the **View** to use a markup language to describe it
  - *Separation of roles and tools*
- WPF, etc.



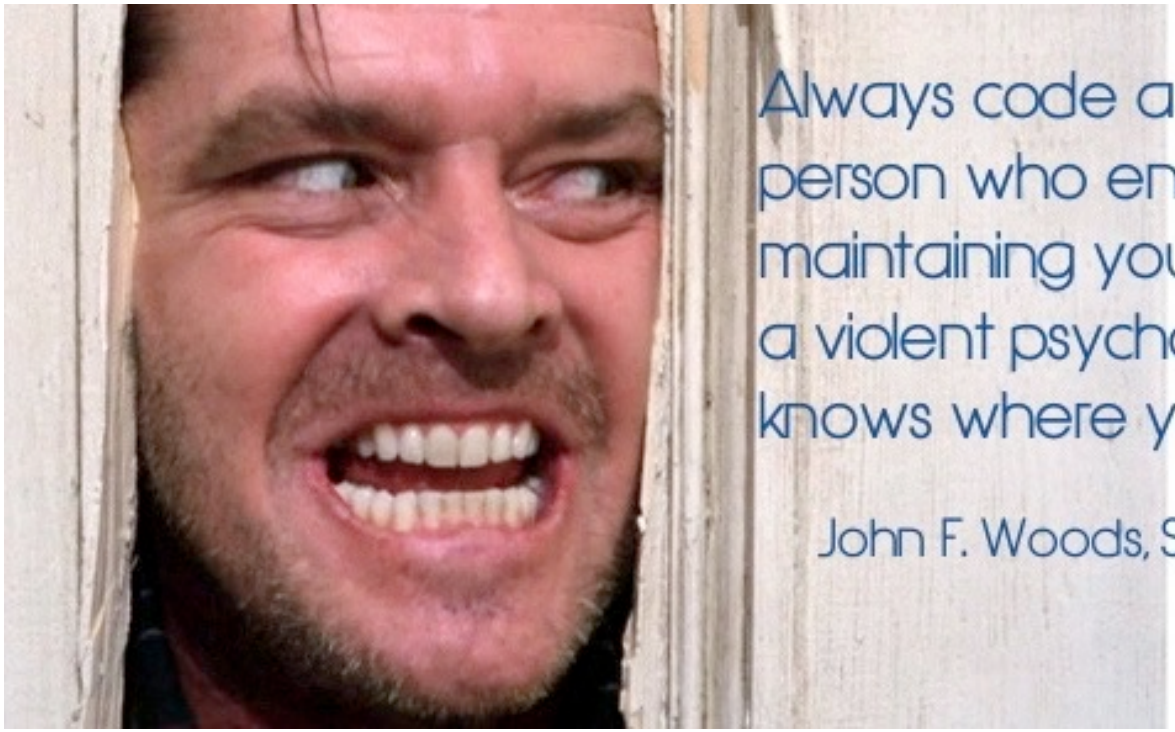
Model-View-Presenter



Model - View - ViewModel

How to choose ?

- KISS (Keep It Simple, Stupid): Be pragmatic, don't go overkill
  - According to use-cases, quality requirements and chosen technologies
  - Trade-off between efficiency and modularity
  - A real world app is rarely a perfect implementation of a given software architecture model
  - The whole system can contain heterogeneous apps and thus implies different software design patterns



Generating code and crossing models for interactive applications

Why use models ?

- Increasing complexity and **weight** of interactive systems

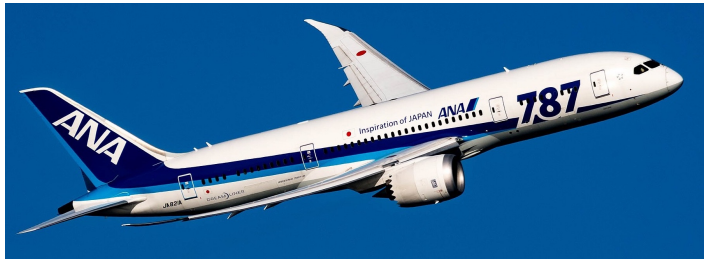
Margaret Hamilton
Apollo 11 in-flight system (1969)
145 000 loc

Boeing 787 Dreamliner (2007-present)
Avionics and support systems: **6.5 Mloc**
All (IFES, etc.): **14 Mloc**

Why use models ?

- Increasing complexity and **weight** of interactive systems

Margaret Hamilton
Apollo 11 in-flight system (1969)
145 000 loc



Boeing 787 Dreamliner (2007-present)
Avionics and support systems: **6.5 Mloc**
All (IFES, etc.): **14 Mloc**

6.7 Mloc

36 Mloc

Microsoft Office 2013
44 Mloc

Why use models ?

Large Hadron Collider, CERN

VS

Ford F-150

(high end connected car)

Why use models ?

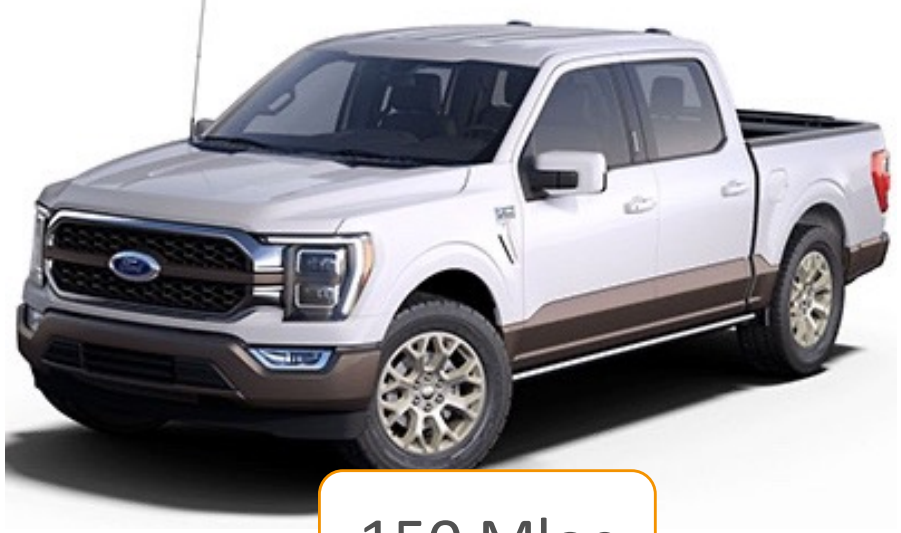
Large Hadron Collider (2008), CERN

50 Mloc

VS

Ford F-150

(high end connected car)



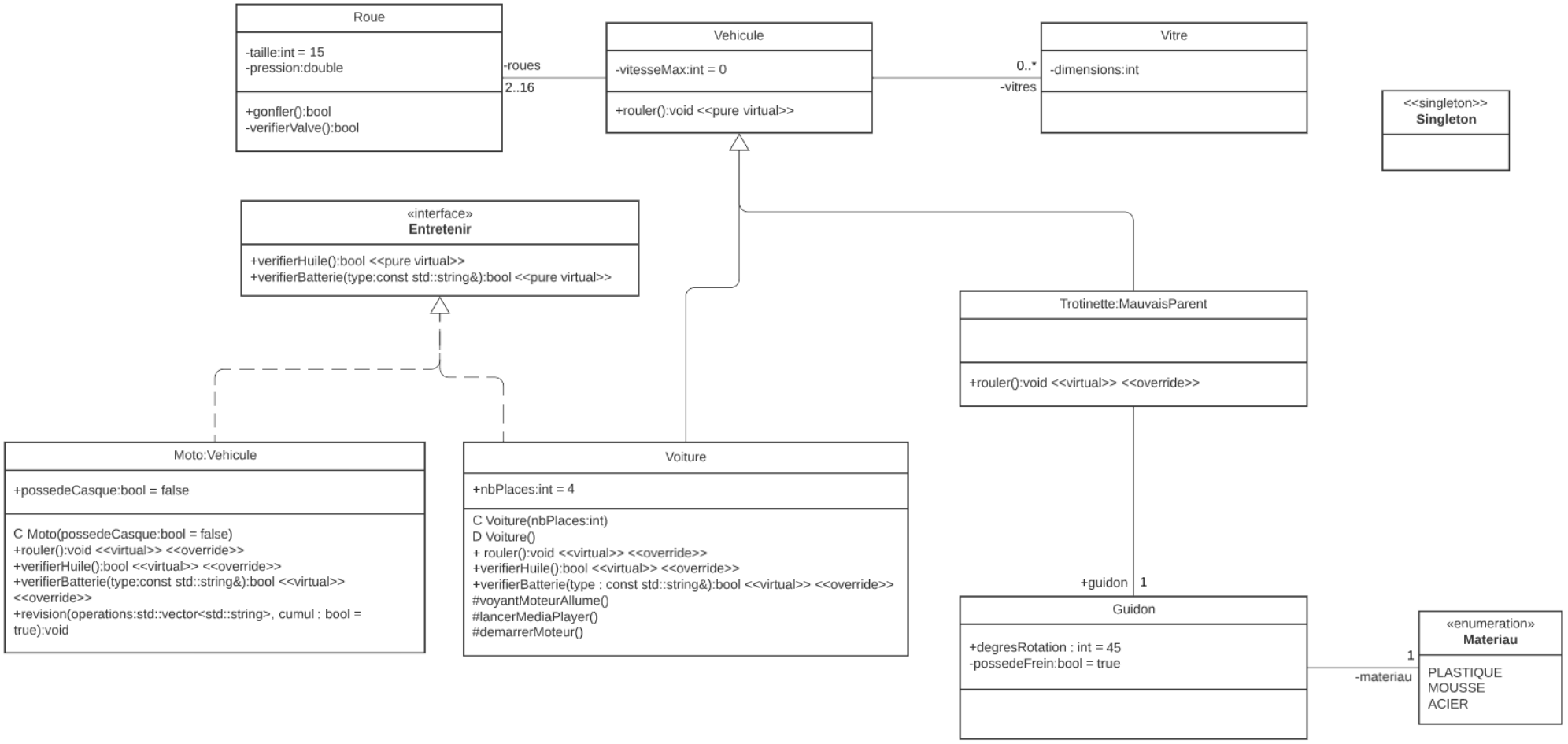
150 Mloc

Why use models ?

- Overcome complexity with collaboration
  - Accelerate system knowledge build-up
  - Much easier to reason about models (vs code)
  - Discuss with all stakeholders
  - Split work between team members
- Ease reusability of parts of the system (platform independent models)
- Generate a complete or partial system implementation from models
  - Improve maintainability and portability
  - Reduce costs and decrease the time to certification (e.g. ARINC 661)

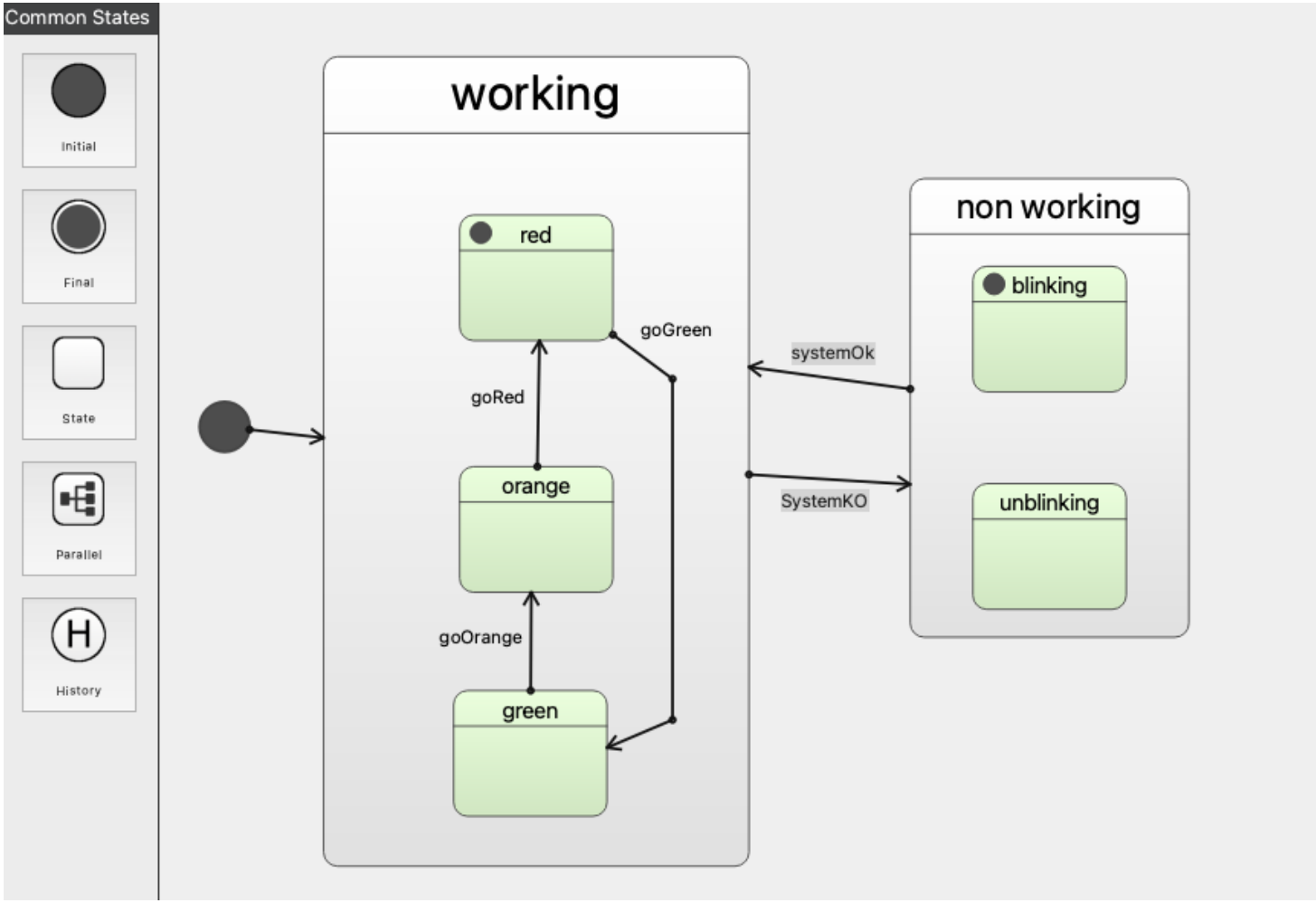
Examples of models

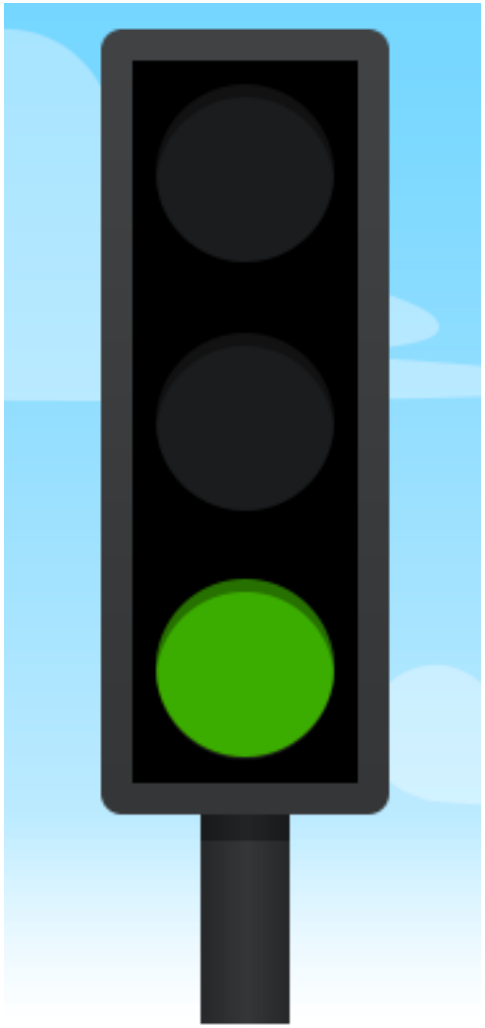
- UML (Unified Modeling Language)
  - E.g. Class diagrams which show classes and the associations between them



Examples of models

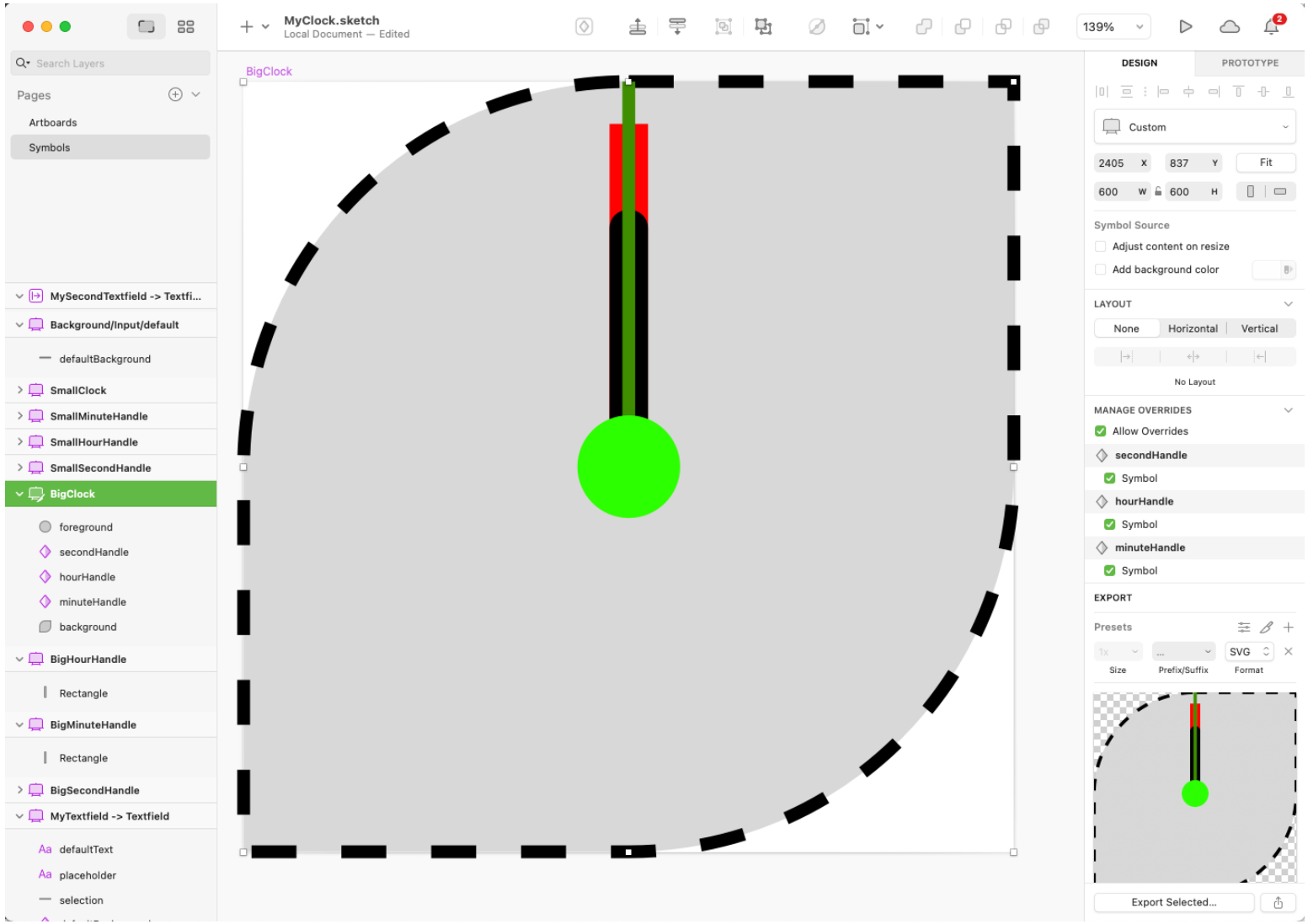
- SCXML (StateChart XML)





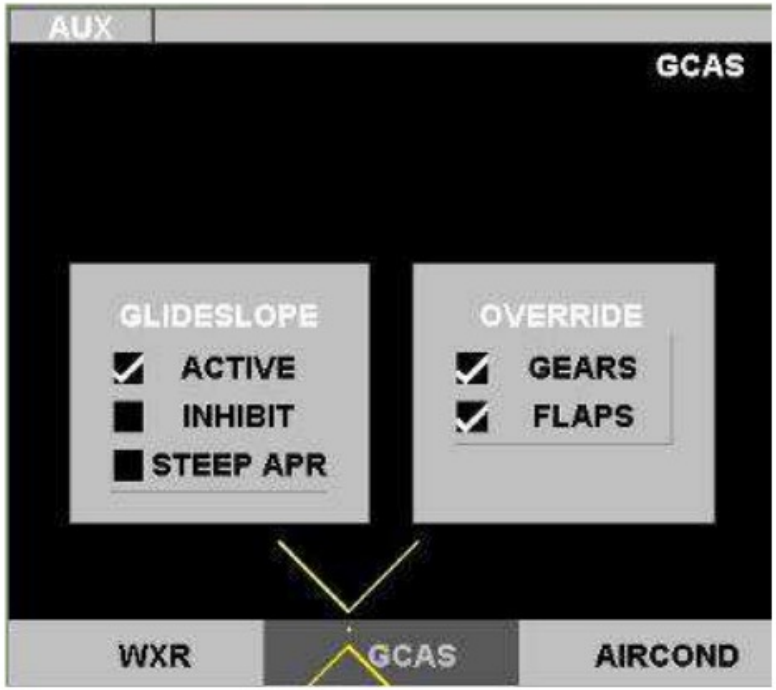
Examples of models

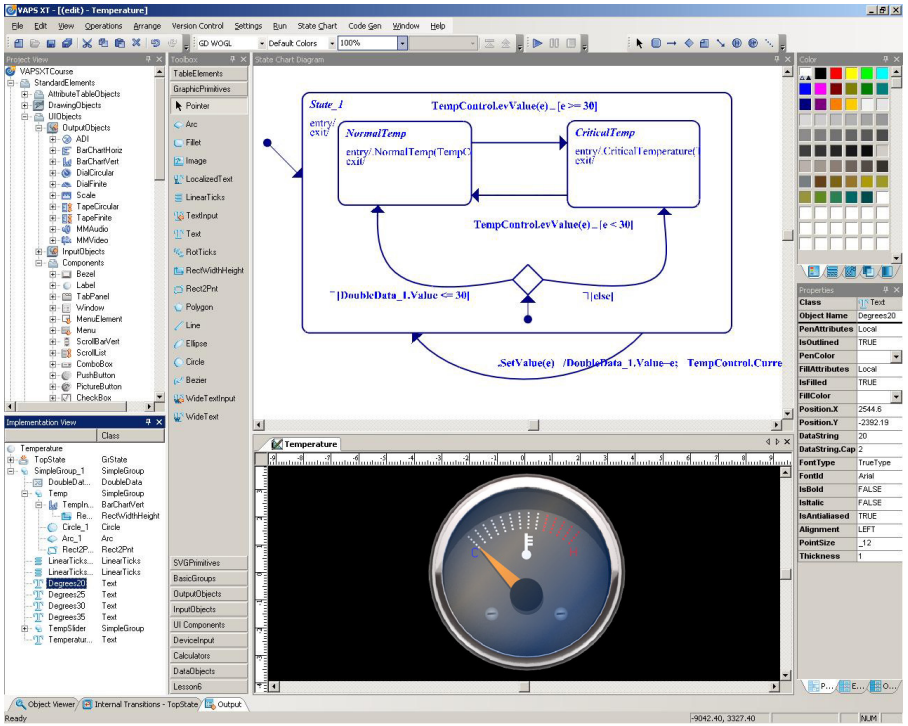
- SVG (Scalable Vector Graphics)
  - XML-based vector image format
  - Used by most graphic designer tools (Illustrator, Sketch, Figma, etc.)



Examples of models

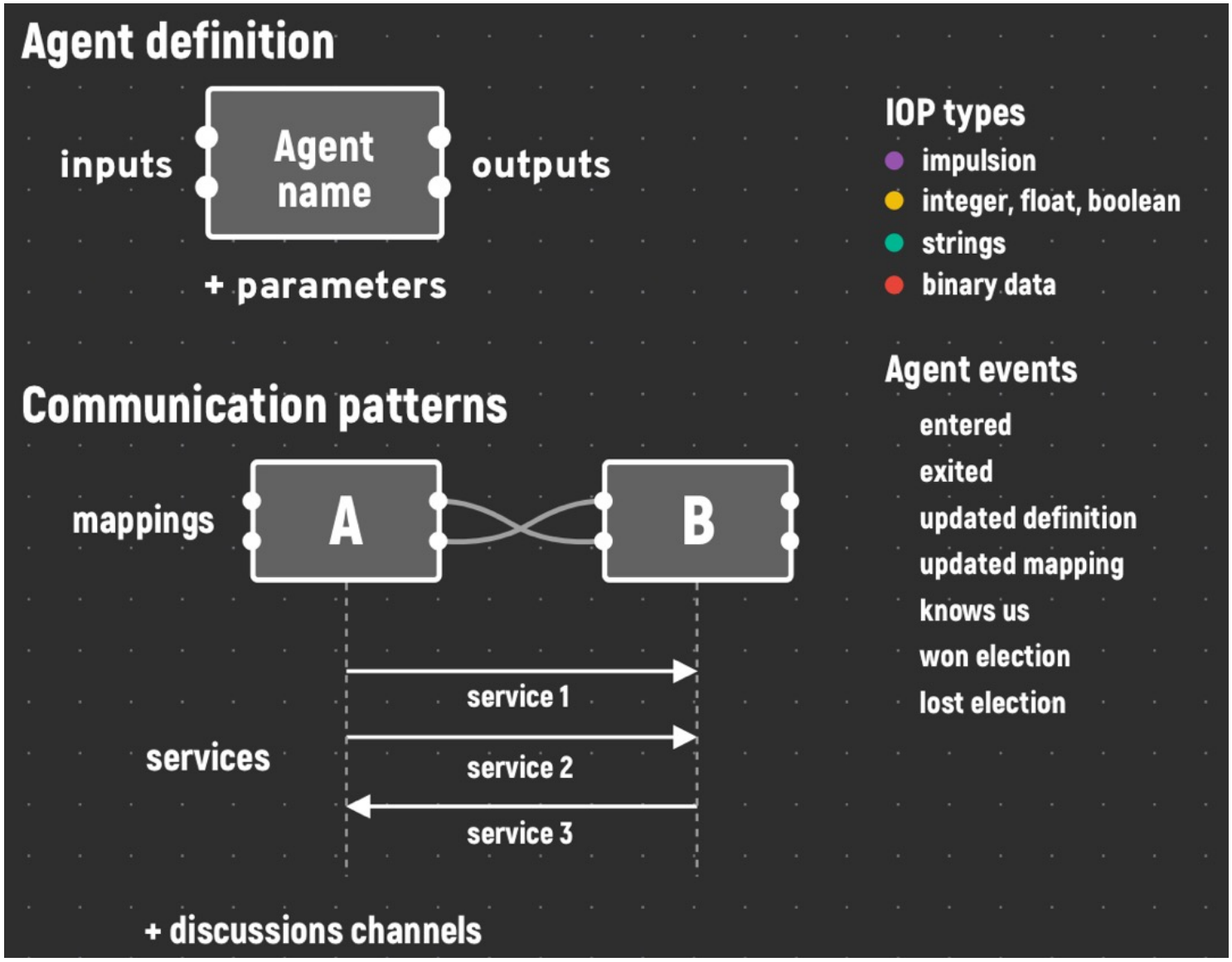
- ARINC 661 standard
  - Normalize the definition of a Cockpit Display System (CDS) and the communication between the CDS and User Applications
  - Two parts:
    - Part 1: Widgets-based UI (2001-present)
      - Airbus A380 and A400M, Boeing 787, etc.
    - Part 2: User Interface Markup Language (2020-present, work in progress)
      - Mix of SCXML (behaviors), SVG and interactive primitives (touch, pointer, etc.)



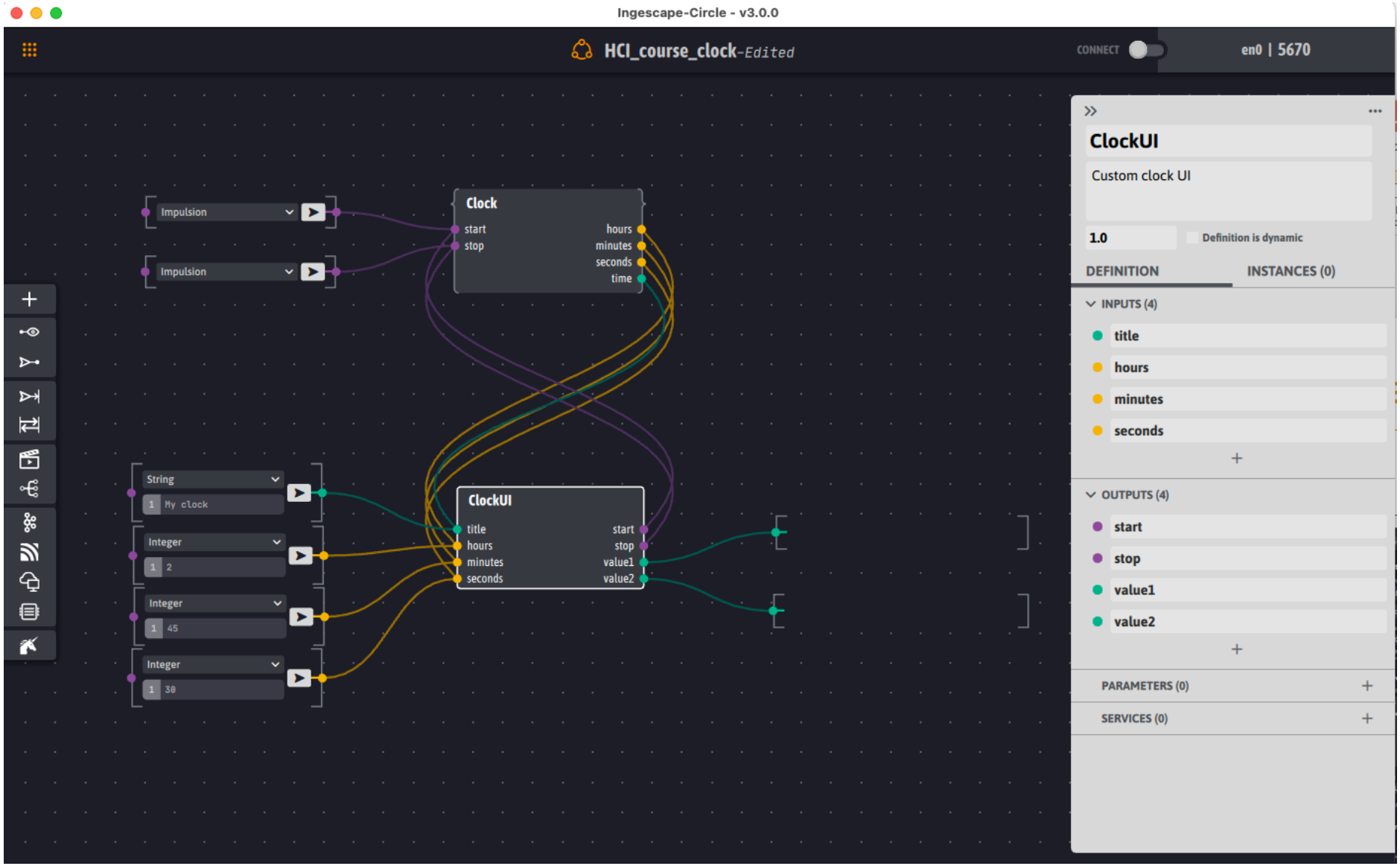


Ingescape

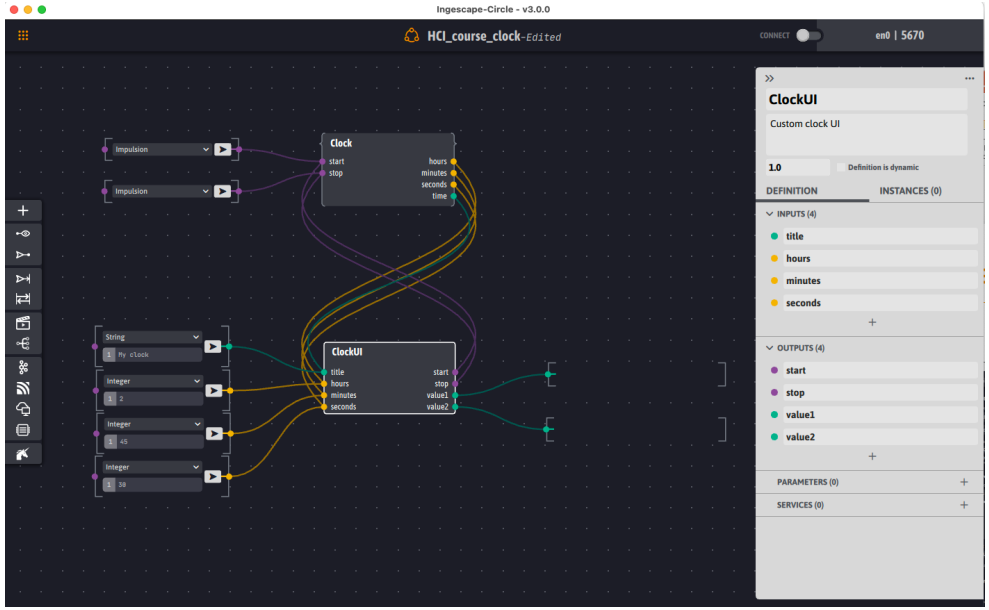
- Define agents and interactions between agents

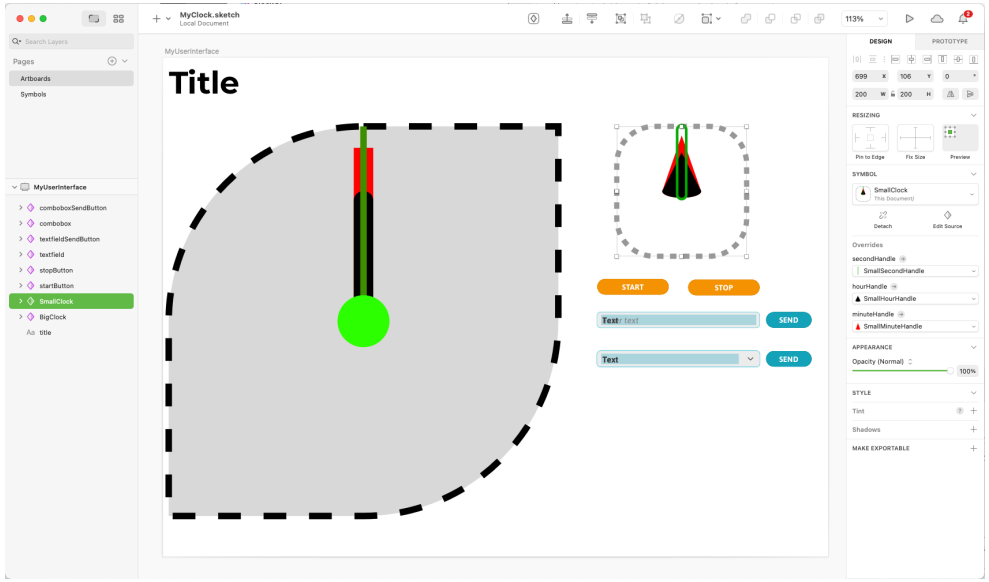


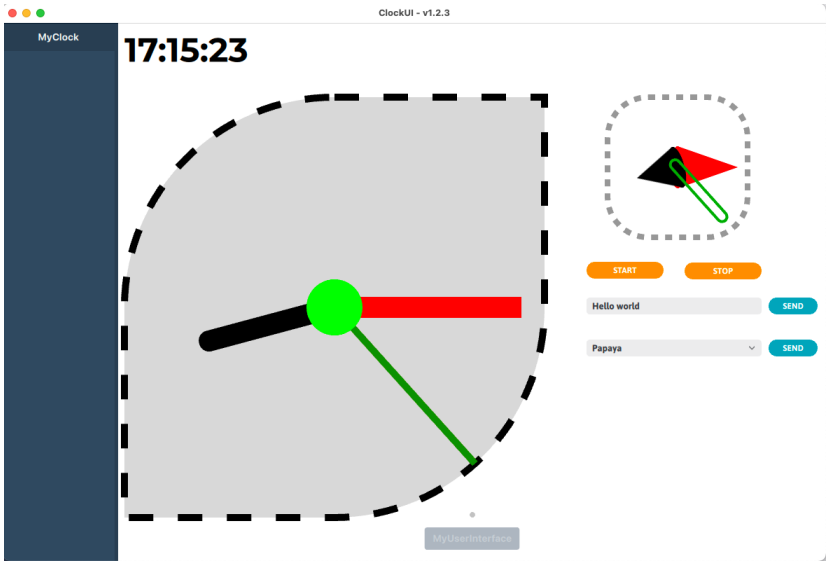
Ingescape: example



Code generation: example







Verification & Validation

Verification & Validation

- **Verification**
  - “Are we building the system right ?”
  - The system implementation meets its specification and exhibits desired properties
  - Code inspections, walkthroughs, static analysis, etc.

- **Validation**
  - “Are we building the right system ?”
  - The system meets the needs and expectations of stakeholders
  - Functional testing, performance testing, usability testing, etc.

Verification & Validation

- V&V must be applied at each stage of the development process
  - Detect defects earlier
  - Emphasize quality throughout development
  - Assess whether or not the system is usable in an operational situation

- V&V should confirm that the system is fit for purpose
  - This does NOT mean completely free of defects BUT it must be good enough for its intended use

Verification & Validation

- **Unit testing**
  - Meticulously examine the individual components of our system
  - Verify the correctness and proper functioning of each component
- **Integration testing**
  - Verify the interactions between different components (data flows, compatibility of function calls, etc.)
- **System testing**
  - Evaluate the compliance of the whole system with functional and non-functional requirements
  - Rigorous testing scenarios, stress testing, security testing, scalability testing, etc.
- **User testing**
  - Gather feedbacks and insights from end-users
  - Ensure the system aligns with user requirements

V&V using Ingescape

- Domain Specific Language (DSL) to write V&V tests:
  - Instructions to inject information and events in a given platform
  - Verifications to check that the behaviors of agents is correct

- Tool dedicated to running V&V tests on Ingescape platforms and generating reports

V&V using Ingescape: Instructions

Instruction	Code
Inject information on the input of an agent	myAgent.myImpulsion = 0 // 0 by convention but any value is valid myAgent.myInt = 345 myAgent.myBool = 1 // 0 means false and 1 means true myAgent.myDouble = 23.56 myAgent.myString = "my string" myAgent.myData = "0xaa22eeff"
Call a service	myAgent.myService(1, 23, 45.213, ”mystring", "aa32eed4")
Sleep	sleep 1000 // Wait for 1000 ms
Interrogate a user through the command line interface	ask ”Q1 without text" ask ”Q2 with a specific character from the following list" y n ? ask ”Q3 with free text" text

V&V using Ingescape: verifications

Assertion	Code
Output value	assert myAgent.myDoubleOutput >= 10.234 assert myAgent.myIntOutput == 42 assert myAgent.myStringOutput == "expected message with value = 45" assert myAgent.myStringOutput ~ "expected [^]+ with value = (\d+)"
Silence	assert silence myAgent 5000 //all the outputs of myAgent assert silence myAgent.myOutput 5000 //silence on myOutput
Service	assert getAnswer from myAgent // we received the call assert getAnswer from myAgent with arg1 = 1 // checking first argument assert getAnswer from myAgent with arg2 = 45 and arg4 = "Hello" // checking arguments 2 and 4 assert getAnswer from myAgent with arg4 = "~ str(.*)"
Interrogate a user (y or n)	assert user "are you ready ?"
Logs	expect log 15000 myAgent DEBUG == "model_write_iop;set input myDouble to double 13.890000" expect log 5000 myAgent DEBUG ~ "model_write_iop;set output value1 to (.*)"

V&V using Ingescape: blocks

	Code

Blocks	{ // Untitled block
	INSTRUCTIONS_AND_VERIFICATIONS
	}
	
	"my title" { // With a title
	INSTRUCTIONS_AND_VERIFICATIONS
	}
	
	"my title" "my description" { // With a title and a description
	INSTRUCTIONS_AND_VERIFICATIONS
	}
	
	"my title" "my description" {
	// Run multiple verifications at once based on a given event
	block.use_pool = true
	block.multi_check = true
	INSTRUCTIONS_AND_VERIFICATIONS
	}

V&V with Ingescape: example

- Whiteboard app

- V&V script (*Whiteboard.igsscript*) available in the open repository for the Whiteboard agent
  - https://gitlab.ingescape.com/learn/whiteboard

Reminders

Calendar

Friday December 6th	• Groups formation (3 students per group) • Each group registers by sending an an email to n7@ingenuity.io with the student names and subject chosen
Monday December 16th	• 1st practical work session, assisted by the Ingenuity team • Technical choices, compilation, debug environment
Wednesday December 18th	• Practical exchanges on your exam projects • Short briefs
Wednesday January 8th	• 2nd practical work session, assisted by the Ingenuity team • Continuous testing, V&V scripting, live integration
Friday January 10th	• Last practical work session, assisted by the Ingenuity team • Integration and testing with the white board and other agents
Monday February 3nd	• Project delivery to n7@ingenuity.io (less than 9MB) or Github • Documentation, ingescape platform for integration and tests, V&V scripts, source code, compiled code

Course 5 : December 18th

- Practical exchanges on your exam projects
  - Short brief
    - =~ 5 min per group
    - 1 slide per group
  - Content of your slide
    - Title: name of your agent (**without spaces**)
    - Group members
    - Clearly communicate your vision
      - Go straight to the point: few words to describe what you intend to do
      - If possible, add an illustration of your agent
        - Snapshot of your agent in Ingescape Circle, paper mockup, etc.

- Your slide **MUST** be ready by **12:00 noon on December 17th**
  - Google slides: **bit.ly/HCIBriefsN7**

Evaluation criteria

- Quality of the proposed User eXperience **/5**
  - Utility, efficiency, comfort, robustness
- Completeness of the integration with the white board **/5**
  - Use of the white board's inputs, outputs and services in your own agent
  - Bonus points if you interact with other agents for an extended user experience.
- System engineering **/5**
  - Agent requirements
  - Minimal specifications for your agent (less is more)
  - Complete V&V scripts with traceability to your requirements
- Coding **/5**
  - Documentation
  - Ability for the teachers to compile and run the code
  - Clarity, concision and robustness

Where to get Ingescape and other resources ?

- The open source Ingescape **library repository**
  - https://github.com/zeromq/ingescape
- The Ingescape **Circle installer**
  - https://repository.ingescape.com/circle-v4/
- The **license and resources for this course**
  - https://ingescape.com/n7
- The open repository for the **Whiteboard agent**
  - https://gitlab.ingescape.com/learn/whiteboard
- Cheat sheets for Ingescape
  - Python: https://ingescape.com/ingescape-python/
  - NodeJS: https://ingescape.com/ingescape-nodejs/
  - Java: https://ingescape.com/n7/java/
  - C# and HTML/CSS/JS: Ask us.