Model-Based Systems Engineering:

Some Messages for Digital Transformation in Government

Timothy C. Lethbridge, PhD, P. Eng, ISP, FCIPS

Timothy.Lethbridge@uottawa.ca

http://www.eecs.uottawa.ca/~tcl

MPLE uOttawa CIPS

Who am I?

Professor of Software Engineering at uOttawa

- Research: code generation, usability, enterprise architecture
- Textbook author: Software Engineering

Professional societies

- CIPS; Board member, Former head of Accreditation Council
- Senior member of IEEE, ACM
- P.Eng.

Industrial Experience

- Mid 1980's Government of New Brunswick
- Later 1980's BNR (Nortel)
- 1990's: Research/consulting with Boeing, Mitel, Federal Government
- 2000's: Research with IBM, GM, KDM Analytics

What I will talk about

Key problems we need to address

- Government is a complex system of systems

Ways to help make digital government systems more successful:

—Use systems thinking (consider interconnectedness, interactions, patterns)

—Generate systems from models

—Be agile

—Attract staff with the right skills

—”Buy, don’t build” is not always the right answer

- Better balance procuring/developing

—Think in terms of user experience (UX)

System

A logical entity

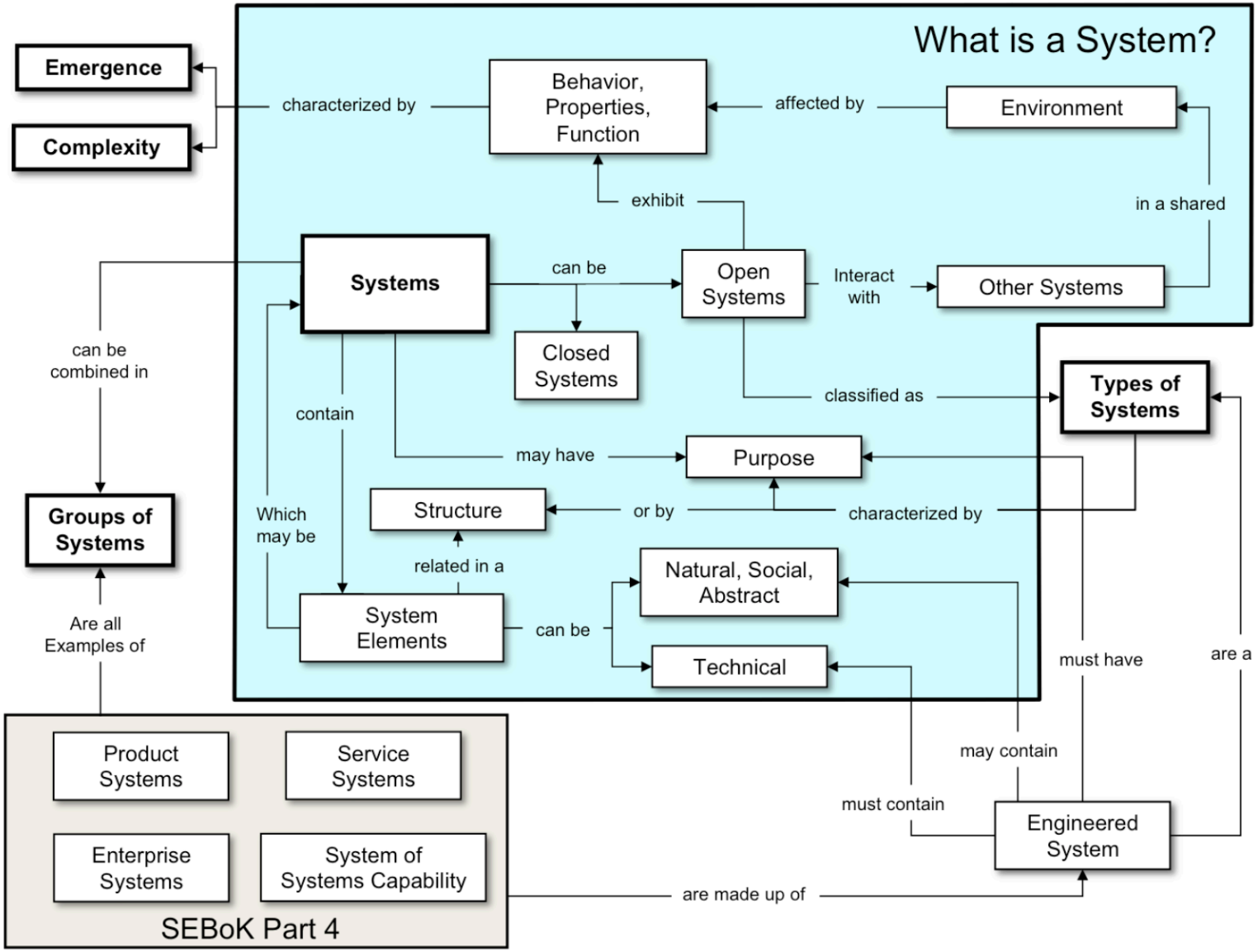
- having a set of definable responsibilities or objectives,
- and which can be divided into interacting components or subsystems.

Hardware, software, social, natural or a combination.

Natural systems have evolved to survive in a changing environment

Artificial systems must be designed to achieve their goal while surviving in a changing environment

Types and Characteristics of Systems



Model

A (simplified/abstract) representation of a system

- Machine-analyzable (ideally)
- Can be
  - —Descriptive
  - —Prescriptive: used to generate artifacts (such as code, manufactured products) or to control processes
- **Views**: diagrammatic, mathematical, tabular, textual, or physical
  - —Have: abstract and concrete syntax
  - —Have: semantics

Systems Engineering

Solving problems

by procuring, designing, adapting, configuring, integrating and evolving …

complex software, hardware, networking, and business process components …

within quality, cost, time, human-nature and other constraints

The application of engineering principles to systems

A core knowledge source for Systems Engineering: SEBoK

The Systems Engineering Body of Knowledge

- www.sebokwiki.org
- Version 2.0 available since June 1, 2019
- Published by
  - —INCOSE
  - —SERC
  - —IEEE Computer Society

- Some diagrams and concepts in this presentation come from SEBoK

Software Engineering

A special case of Systems Engineering

Solving problems by designing, adapting, configuring, integrating and evolving complex software, in a broader environment, within quality, cost, time, human-nature and other constraints

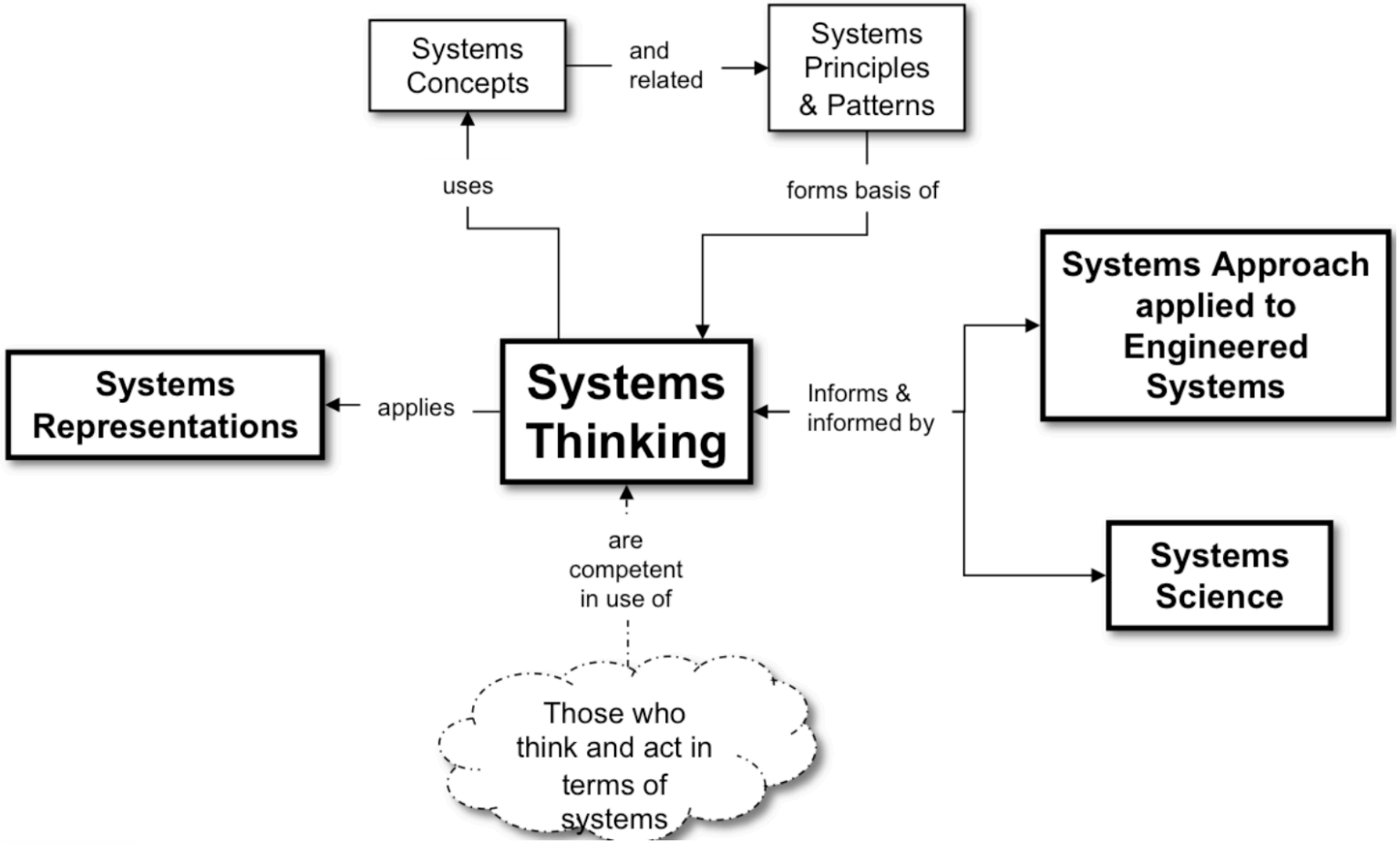
Software tends to be the most failure prone system type

- —It is hard to know what is needed (requirements)
- —Talent is lacking and/or expensive
- —Complexity and technical debt can grow rapidly
- —User experience is key

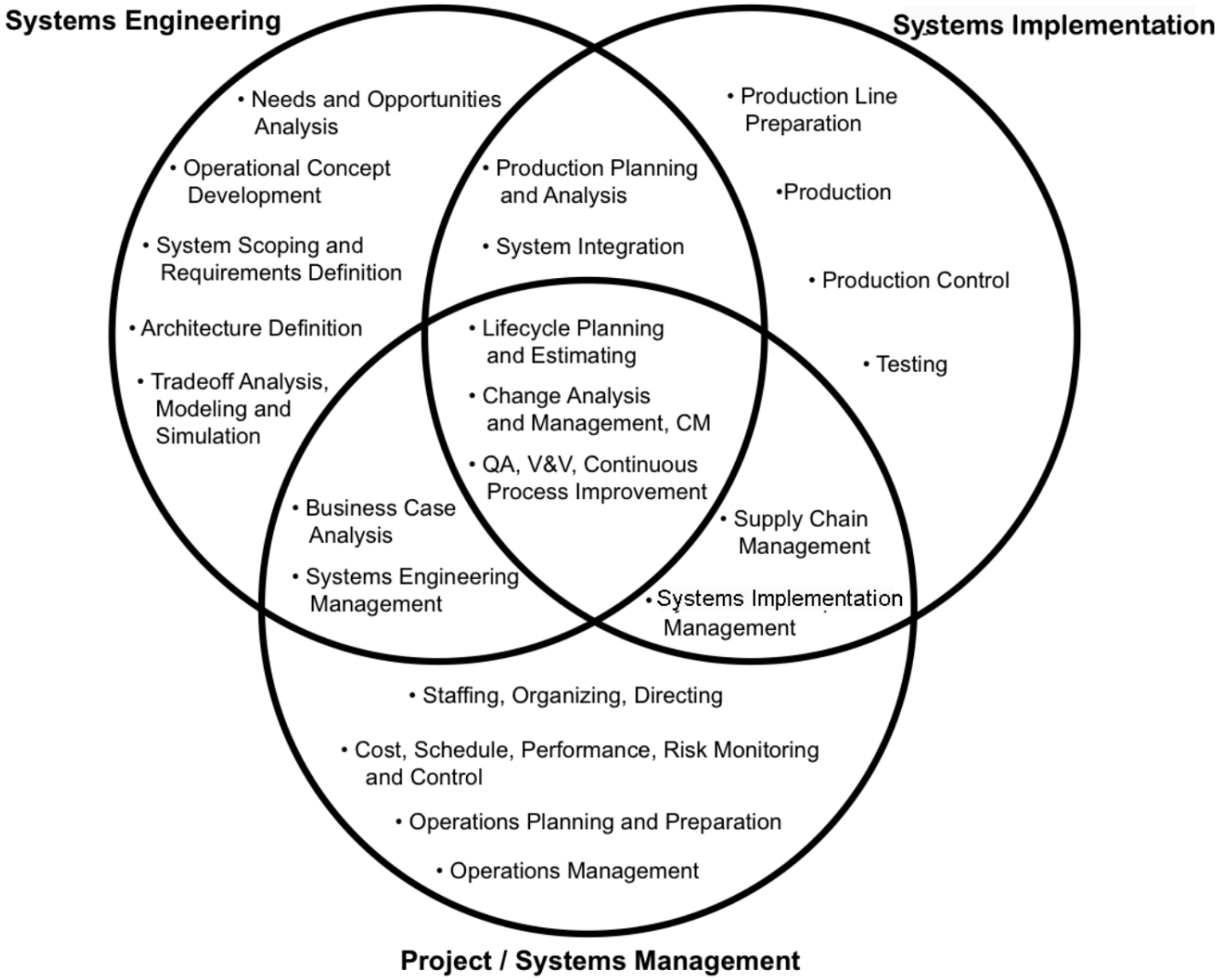
Systems Thinking

Sensibility to subtle interconnectedness, interactions, patterns

Especially in control, hierarchy, emergence and complexity



A View of of Systems Engineering and related Disciplines (from SEBoK)



Model-Based Systems Engineering (MBSE)

Models are the primary artifacts

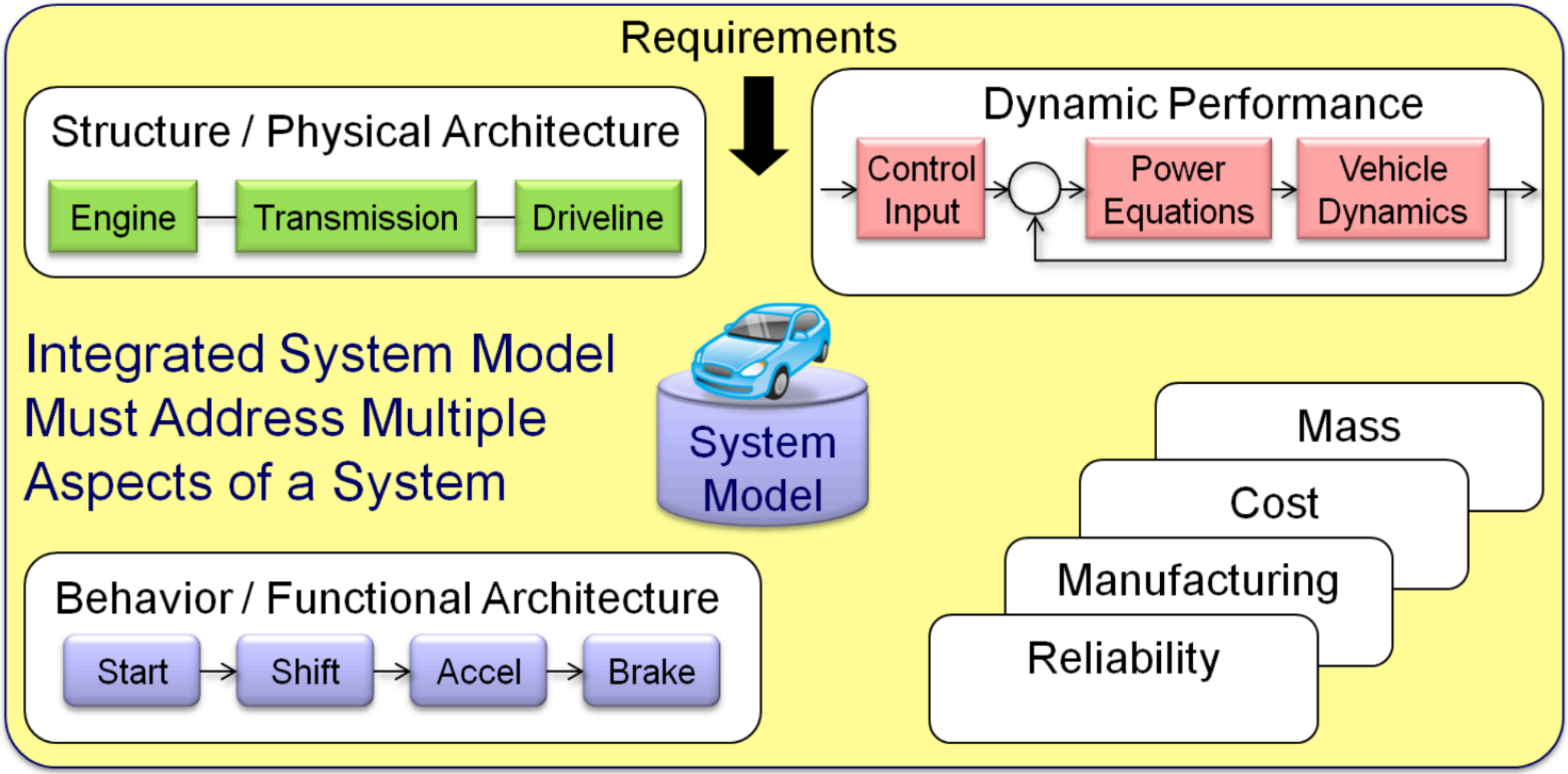
- As opposed to documents (which dominated early decades of systems engineering)

Models of systems allow us to

- Think abstractly
- Simulate before implementing
- View the system from different perspectives
- See the subtle interactions
- Analyze, test, validate to find defects
- Generate the final system, and each subsequent version

Some Models in an Automotive System (Paradis 2011)

Think of the subtle interactions!



Types of Models in a Payroll System

Think of the subtle interactions!

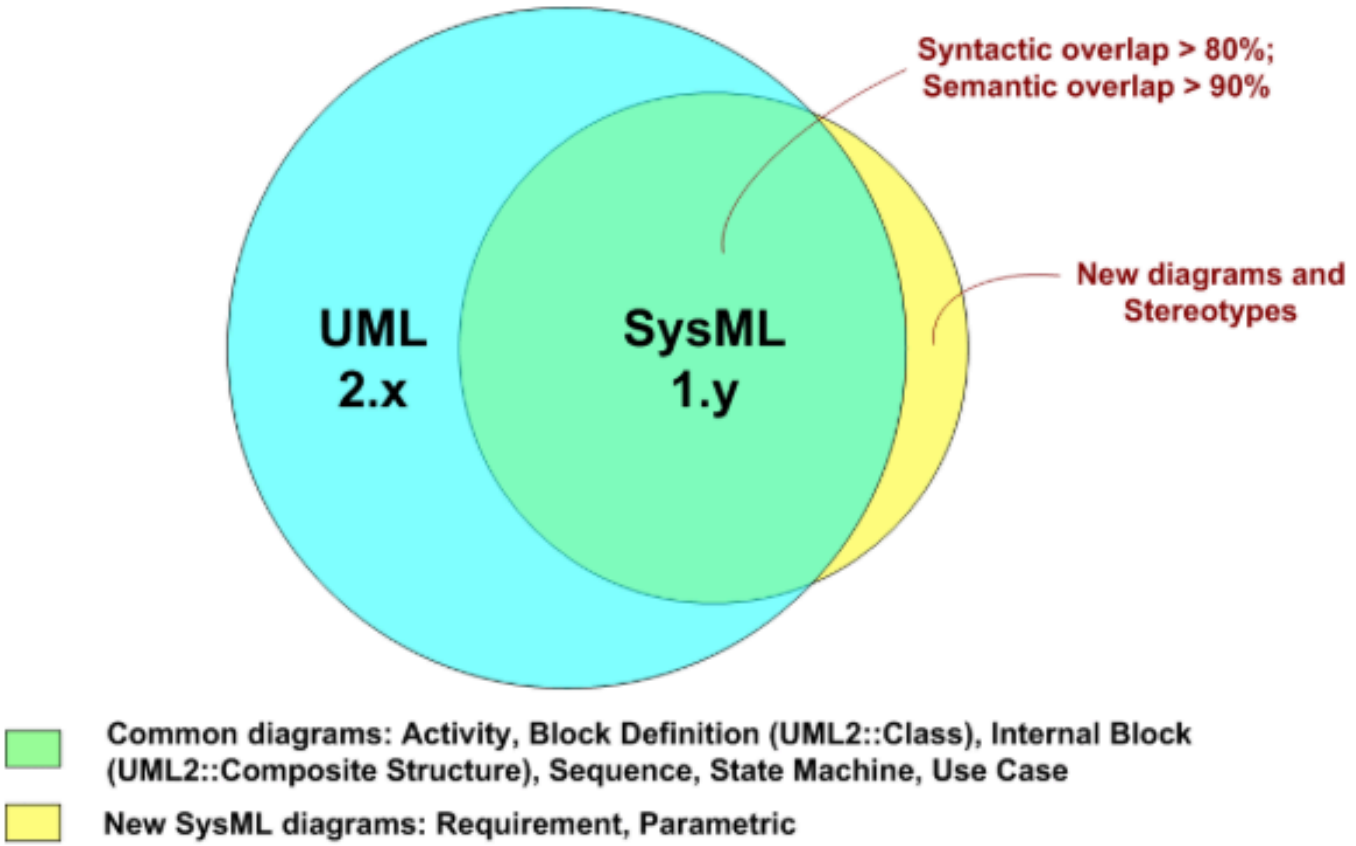
Data models (Class/ERD)

- Employees / Positions / Supervisors / Substantives / Acting Roles / Temporary Assignments / Departments / Unions / Jurisdictions / Collective Agreements / Job descriptions / Work units / Pay Scales / Progress Through Ranks / Overtime / Pay computations / Pay periods / Benefits / Deductions / Taxes / Pay Actions / Error Corrections / Reports

Process Models: What has to happen, in what order, under what constraints, by what time

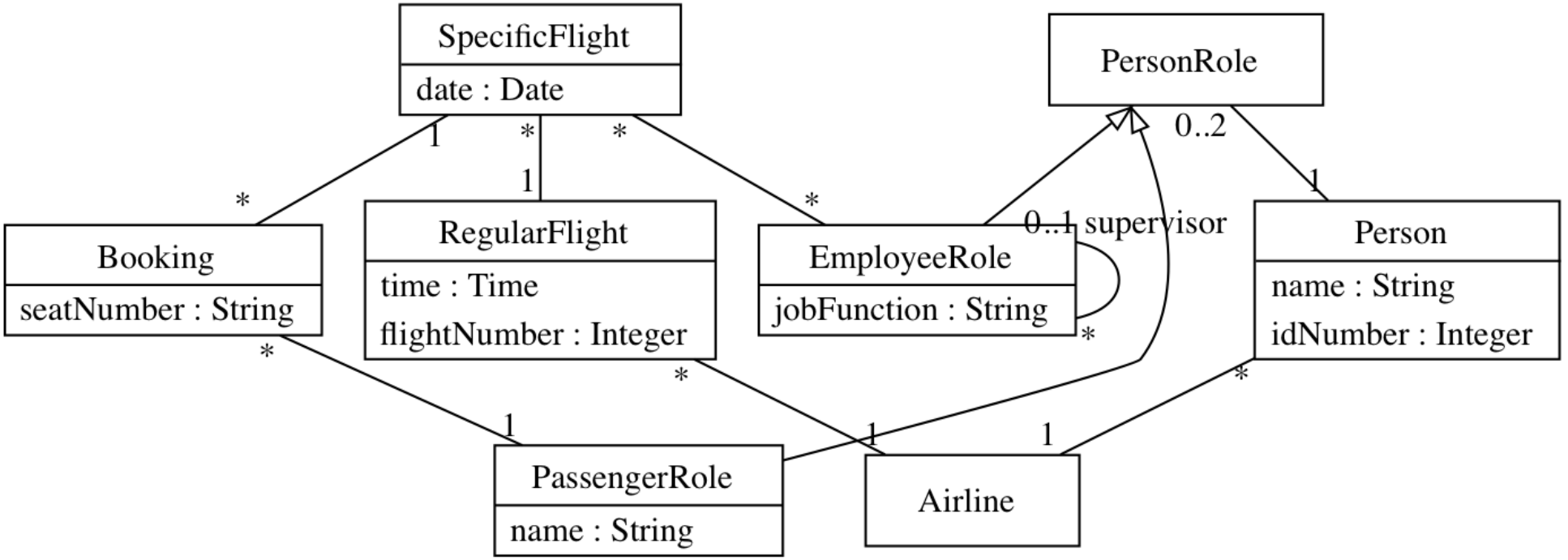
State models: What state can each data item can be in and when does it change state

SysML and UML: Modeling languages



Some UML models: Class diagrams

- Show classes, attributes, generalizations and interrelationships among data (associations) in ways hard to see by just looking at code

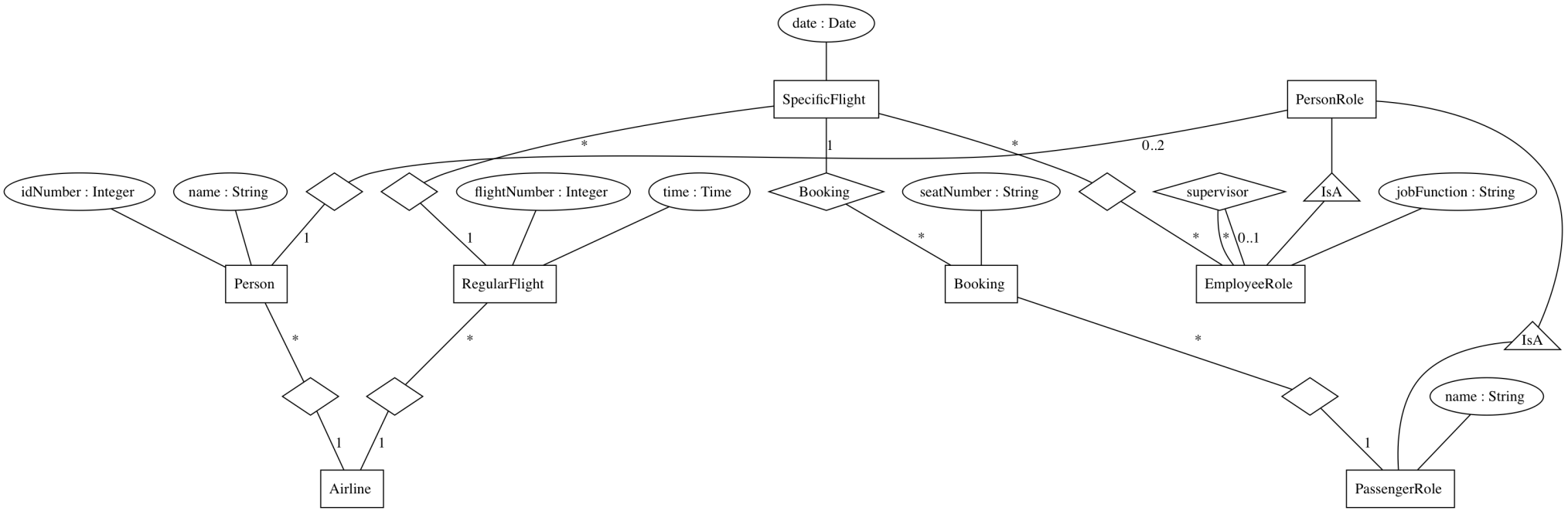


- Model at: http://try.umple.org/?example=Airline&diagramtype=GvClass

Entity-Relationship Diagrams (ERD)

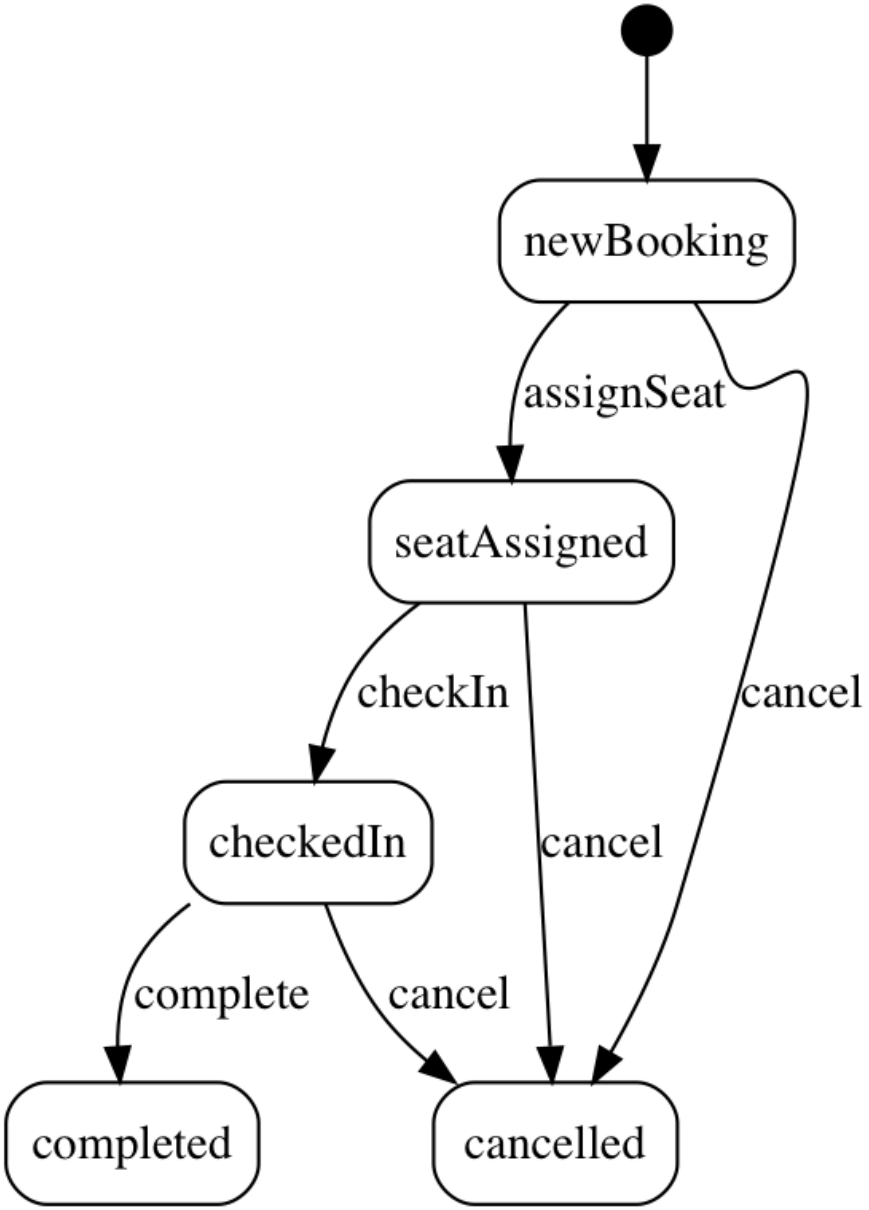
For many purposes, interchangeable with class diagrams

- Used heavily in database community (Not UML)

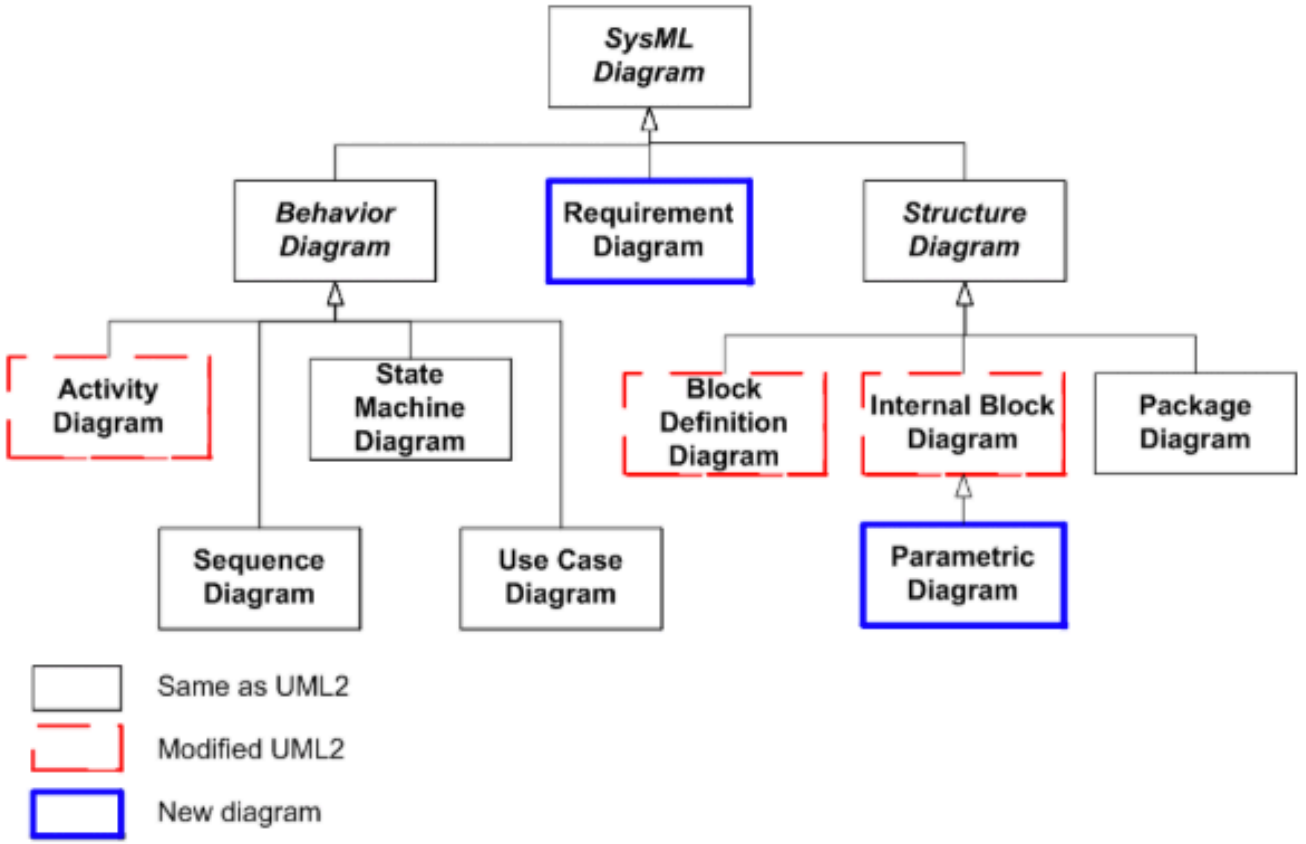


State Machines

- Show behaviour
- Behaviour changes in response to events



SysML Diagrams



SysML Diagram Taxonomy

A Selection of systems engineering tools

- Papyrus SysML
- Modelio
- Rational Rhapsody
- MagicDraw
- Sparx EA
- Enterprise Architect

- EA tools such as Archimate

Plus other tools used for design of software, hardware, business processes

Sample UML modeling tools in wide use

- Papyrus, and other Eclipse-based tools
  - —Open source leaders
- IBM Rhapsody, current most widely used IBM tool
- ArgoUML
  - —Developed 1999-2011, but not maintained
- Astah
- Magic Draw
- StarUML
- PlantUML
- Visual Paradigm

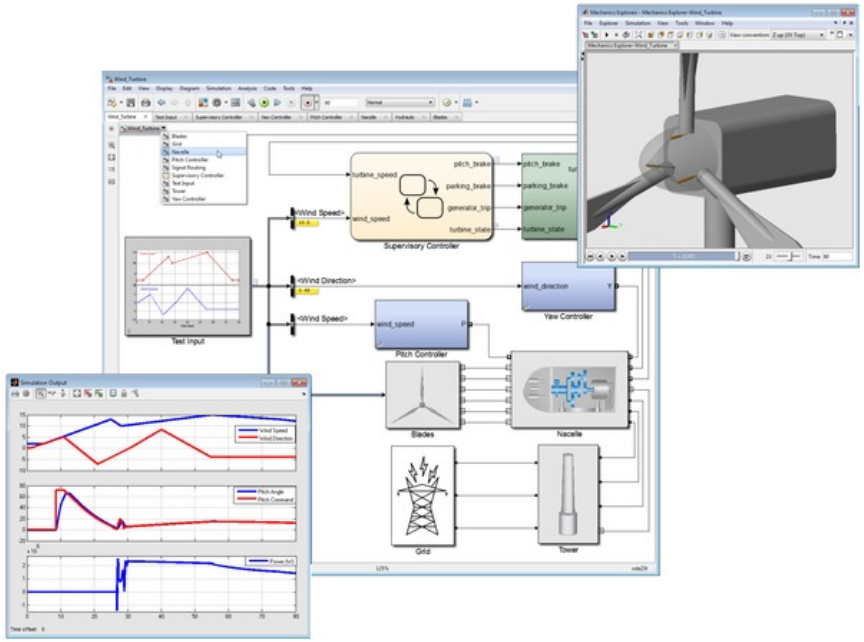
For more details and links see

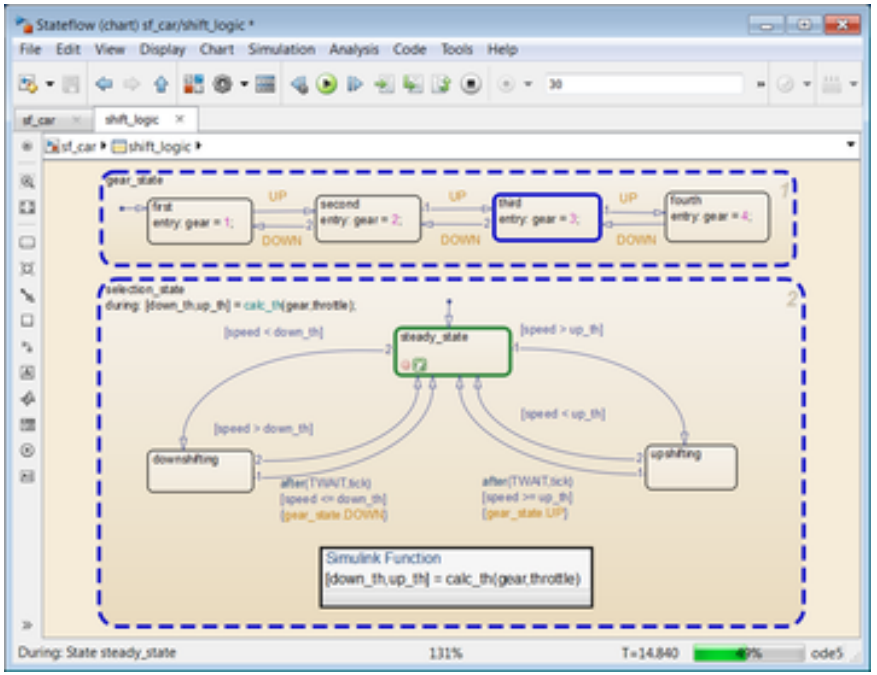
https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

Other MDD technologies (language+tool): *Simulink and Stateflow*

Part of MathWorks' MATLAB suite

- Widely used in automotive and other safety-critical domains
- But expensive and not taught much to general software engineers





User Experience

Usability + Aesthetics + The Right Functionality

Central to project success

- Requires awareness and investment

Usability = Learnability + Efficiency of Use + Error avoidance

UX talent is in short supply

- Training is needed

Model-Driven Systems Engineering likely beats “Buy, don’t Build”

Corporate products, especially COTS, are legacy too!

- Not designed for purpose
- Corporate employees are often not familiar with the systems,

  —just as lacking in skills as public servants.

Data and processes in government tend to be unique and complex

Integration of systems requires enterprise-wide models:

- Enterprise Architecture

Agility

Developing very small changes to systems, and releasing frequently,

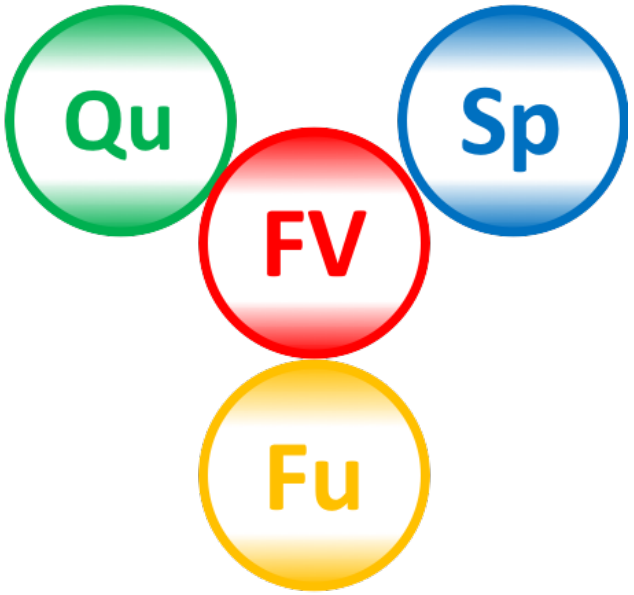
while responding to stakeholder needs,

maintaining high quality

and keeping overhead and technical debt low

It's about balancing

1. Speed/Velocity (dates for delivery)
2. Functionality
3. Quality
4. Financial viability



Adjust 1 and 2, avoiding risks from sacrificing 3 and 4

So why be agile?

- Delivers core stakeholder value earlier
- Enables better quality of life by reducing stress and overtime

- Better manages stakeholder needs, which change and are usually not fully knowable

- Allows teams to build quality into systems without being held to ransom by unreasonable delivery requirements

- Keeps costs under control
- Lets any project fail fast if going in the wrong direction

"Waterfall" doesn't work for most projects

How to be agile (1)

Plan small updates (based on issue tracking and 'sprints').

- Or even better, Continuous Integration (CI)

Deliver automated tests with all increments

- Run every time anyone builds
- Even better: Test-Driven Development

Design, document and manage with low overhead

- Collaborative design with wikis and issue comments
- Generate documentation from models and embedded comments

How to be agile (2)

Automate, automate, automate

- —Version control
- —Continuous integration
- —Design (tools)
- —Software code generation
- —Testing
- —Analysis (metrics, defect finding)
- —Documentation generation
- —Review processes
- —Release/distribution
- —Management of dependencies

How to be agile (3)

Involvement of and responsiveness to end-users

- Understand and respond to their needs
- So as to deliver them a good user experience

- Have them 'in house' to
  - —Discuss plans, designs
  - —Try out prototypes

- Get a working early system version into their hands fast

Any challenges to being agile (1)?

Tricky if you are given a binder of pre-written requirements!

- Some solutions:
  - —Negotiate to be agile
  - —Decline to accept a contract for unchangeable requirements

Any challenges to being agile (2)?

Tricky if we are asked to adopt a giant pre-existing system

- That is not designed for agility

Some solutions

—Develop from scratch with agility from the start

—Roll out slowly to small groups of stakeholders

Any challenges to being agile (3)?

For some embedded systems:

- Model and simulate safety and security: non-negotiable
- Certification of in-field changes is required

For systems with many or complex interfaces to other systems

- Modeling of interfaces avoids chaos

Test driven development (TDD)

Tests are the specification of what system should do with automatically verifiable results.

Process:

- Write the tests
- Verify they fail

- Design and implement the system increment
- Verify the tests pass

- Perform other refactoring or changes as needed
- Always verify the tests keep passing

Continuous integration (CI)

Integrate agile increments into customer-facing systems as soon as they are ready (maybe multiple times a day!)

- 'Ready' means checks have passed such as:
  - —100% of pre-written tests
  - —Reviews, inspections
  - —Automated analysis for vulnerabilities

Options:

- Slow releases as well as continuous releases
- A/B testing – some users given experimental version

Curation of issues / backlog

Bugs, features, enhancements are all *issues*

Can come from end-users, customers, marketing, developers

- Developer issues might include refactoring and dealing with technical debt

There is a need to work on the highest-value issues first

- Both bugs and enhancements
- Not do ‘too much’
- Regular sorting ‘planning game’ needed

Planned reduction of technical debt

Technical Debt (TD)

= required future development needed to avoid excess costs (interest) from software evolution and support

Examples

- Inflexible/non-scalable design choices
- “quick and dirty” implementation
- The need to upgrade obsolescent dependencies

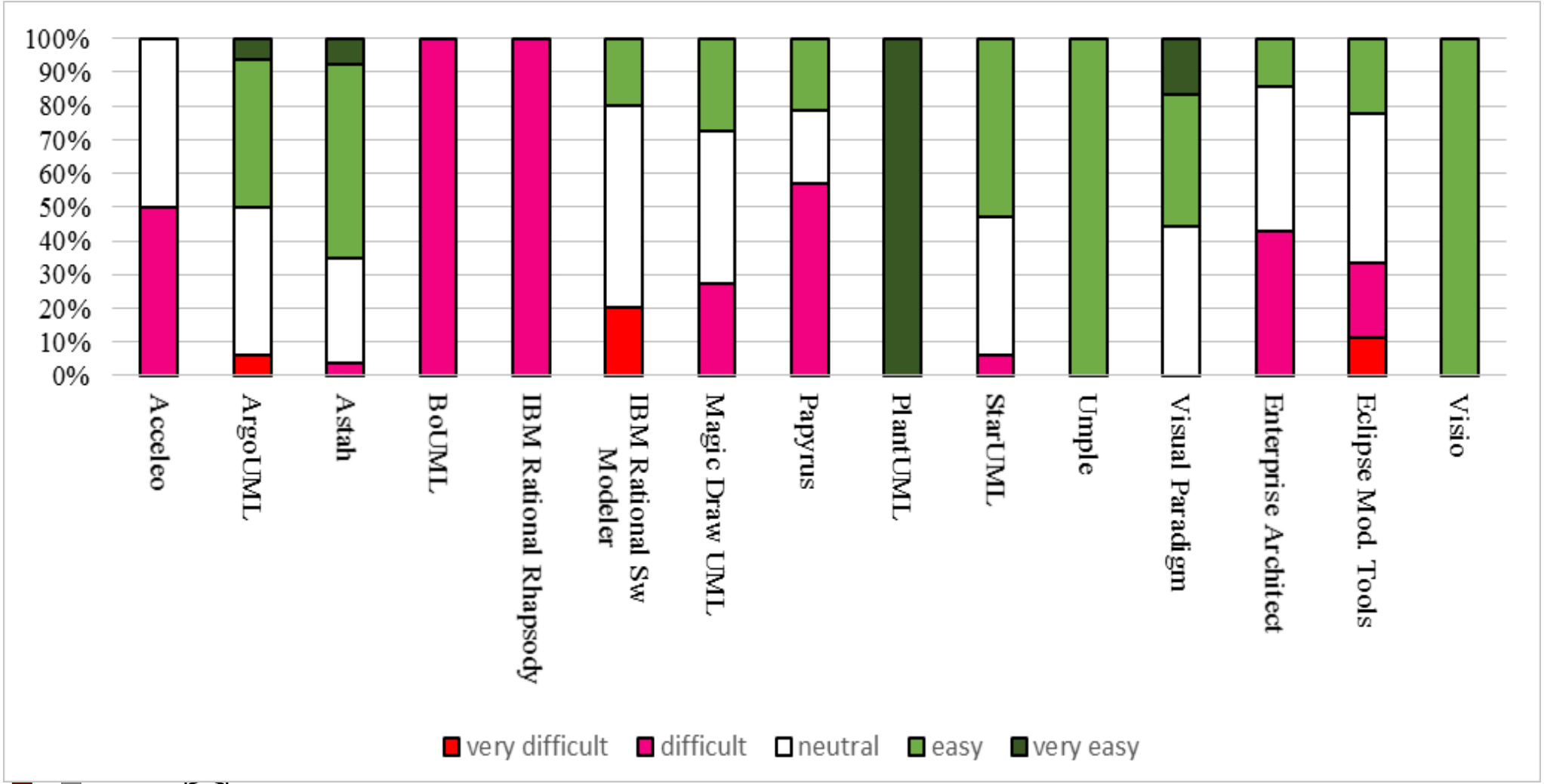
Agility helps manage TD

- By allowing scheduling of regular refactoring activities

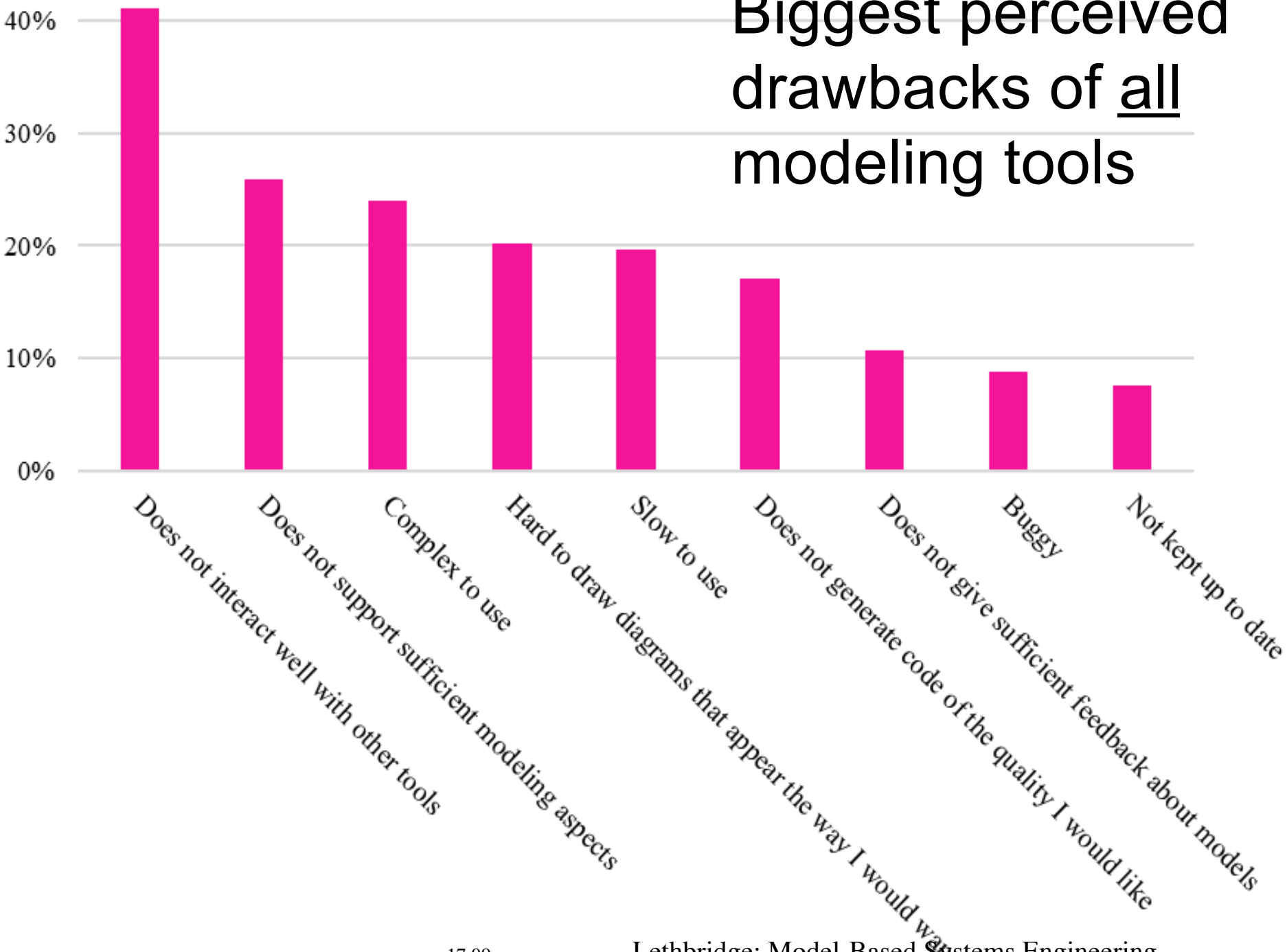
What can break agility?

- Managing documents rather than models
- Contracting out without attention to agility
- Procuring a large system and then trying to make it conform!
  - Developing CAN be better if done in an agile manner using models
- Lack of automation and other weaknesses in tools

Difficulty level of top 15 software modeling tools in a survey my team conducted (2017)



Biggest perceived drawbacks of all modeling tools



Perceived biggest drawbacks of top 7 tools

	Argo UML	Astah	Eclipse Mod. Tools	Magic Draw UML	Papyrus	Star UML	Visual Paradigm
Complex to use	25.0%	11.5%	22.2%	46.2%	42.9%	11.8%	10.5%
Slow to use	12.5%	11.5%	11.1%	15.4%	14.3%	17.6%	5.3%
Does not generate code of the quality I would like	25.0%	38.5%	11.1%	23.1%	21.4%	23.5%	10.5%
Does not support some modeling aspects	25.0%	19.2%	0.0%	30.8%	7.1%	17.6%	15.8%
Does not give sufficient feedback about models	37.5%	53.8%	22.2%	23.1%	42.9%	35.3%	42.1%
Hard to draw diagrams	25.0%	0.0%	55.6%	30.8%	28.6%	17.6%	21.1%
Does not interact well with other tools	6.3%	11.5%	22.2%	30.8%	0.0%	11.8%	10.5%
Buggy	18.8%	0.0%	11.1%	7.7%	28.6%	5.9%	5.3%
Not kept up to date	25.0%	3.8%	0.0%	0.0%	0.0%	17.6%	10.5%

	up to 10%
	10% - 20%
	20% - 30%
	30% - 40%
	more than 40%

Textual modeling, and motivation for Umple …

Textual modeling tools

Almost all modeling tools support a way to exchange models textually: XMI (XML Metadata Interchange)

- But not intended or easy for human editing

Some tools are specifically designed to allow modeling textually in the same way one would program

- Many examples:
  - —List at https://modeling-languages.com/text-uml-tools-complete-list/
  - —PlantUML: https://www.planttext.com
    - Nice, but lacks features such as complete code generation
  - —Others: USE, TextUML, yUML

Umple: Simple, Ample, UML Programming Language

1. Open source textual modelling tool set for 3 platforms
   - Command line compiler
   - Web-based tool (UmpleOnline) for demos and education
   - Eclipse plugin

2. Code generator for UML ++
   - Infinitely nested state machines, with concurrency
   - Proper referential integrity and multiplicity constraints on associations
   - Traits, mixins, aspects for modularity
   - Text generation templates, patterns, traits

3. Pre-processor to add UML, patterns and other features on top of Java, PhP, C++ and other languages

Quick demo

Entry-point: http://umple.org

UmpleOnline: http://manual.umple.org

Github: https://github.com/umple/umple

Motivation for developing Umple (1)

We want the ***best combination of features***:

- Textual editing and blending with other languages
- Ability to use in an agile process
  - —Write tests, continuous integration, versioning
  - —Combine the best of agility and modeling
- Excellent code generation
  - —Complete generation of real systems (including itself)
- Multi-platform (command line, Eclipse, Web)
- Practical and easy to use for developers
  - —Including great documentation
- Open source

Conclusions

Ways to help make digital government systems more successful

—Use systems thinking (consider interconnectedness, interactions, patterns)

—Generate systems from models

—Be agile

—Attract staff with the right skills

—”Buy, don’t build” is not always the right answer

- Better balance procuring/developing

—Think in terms of user experience