Introduction to Testing scientific software

Stefano Cozzini

Democritos and SISSA/eLAB - Trieste

Outline

- Introducing the problem:
  - What does testing software mean ?
  - Why,where,when,who, what to test ?
- Scientific software testing..
  - What is scientific computing ?
  - Why is so difficult to test simulations ?
  - Reviewing verification/validation
  - Some idea/suggestions
  - Homework

Aim of this lecture

- Share some interesting ideas i read about
- Tell about some current effort to apply some of such ideas to a scientific software development
- Convince you that testing scientific software is needed

- Disclaimer:
  - Content of these slides come from many sources all around the web; some slides are mine, some other are taken from other presentations without modification, some other are slightly modified

First definitions : errors and faults

Error

is a measure of the difference between a measured or calculated value of a quantity and what is considered to be its actual value.

Faults

Code faults are mistakes made when abstract algorithms are implemented in code.

Faults are not errors, but they frequently lead to errors.

Second definition: testing

- Software testing is a process by which one or more expected behaviors and results from a piece of software are exercised and confirmed. Well chosen tests will confirm expected code behavior for the extreme boundaries of the input domains, output ranges, parametric combinations, and other behavioral edge cases.

  (from yesterday Tommy's lecture)

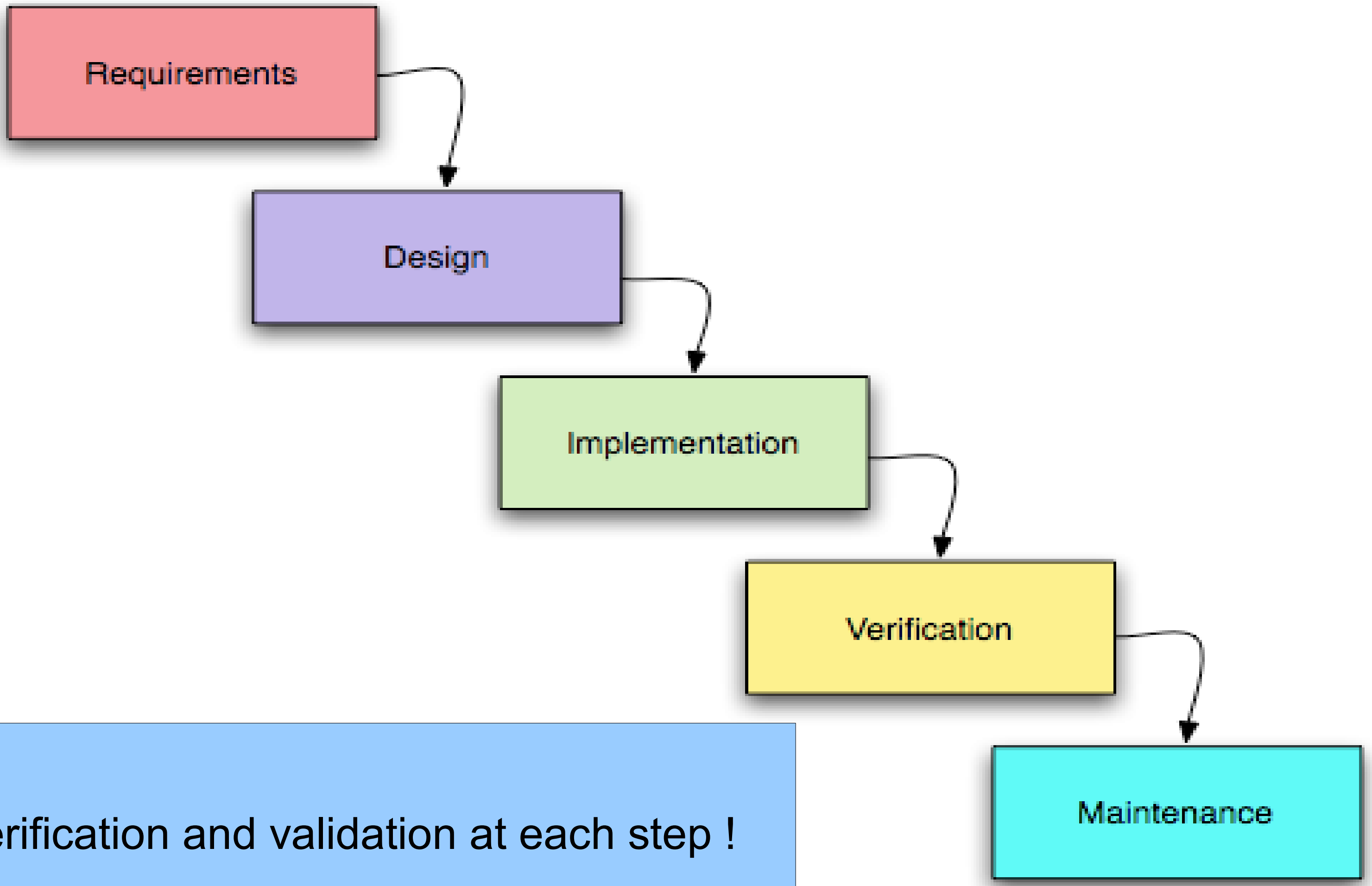
- Software testing can be stated as the process of validating and verifying that a software program/application/product:
  - meets the requirements that guided its design and development;
  - works as expected; and
  - can be implemented with the same characteristics.

(from wikipedia)

Third definition: Verification validation

- Verification: "Are we building the product right ?"
  - The software should conform to its specification
- Validation: "Are we building the right product ? "
  - The software should do what the user really requires
- V & V must be applied at each stage in the software process
- Two main objectives:
  - Discovery of defects in a system
  - Assessment of whether the system is usable in an operational situation

Software stages: waterfall model



Verification and validation at each step !

Is all the above true for scientific codes/packages/ simulation method ???

- Sure !
- But something more is required...

The 5 W of testing: Why testing ?

- To find faults
- To provide confidence
  - of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults
- Other issues include:
  - – Performance of systems (i.e. use of resources like time, space, Bandwidth,…).
  - "…ilities" can be the subject of test e.g. usability, learnability, reliability,availability, etc..

The 5 W of testing: When testing ?

- When:
  - Always !
- Different granularity:
  - When I/they change of the code
  - at every night to check possible problem
  - When a new feature/release is available
  - When we start using a new package for scientific research

The 5 W of testing: Who should test ?

- Who:
  - You as developer
  - You as part of a developing team:
    - Try to test things you did not write
    - Find some other to test you own software
  - You as user:
    - Is this software/package/routines/code what I really need ?
  - You as scientific user (never use scientific code without your own test !!!)

The 5 W of testing: Where to test ?

- Software point of view
  - On any single function of your code:
    - Unit testing
  - On the code as a whole
    - Regression tests
- Hardware point of view
  - On all the possible platforms you have at disposal
    - Ensure portability of software and of scientific output !

The 5 W of testing: What should I test ?

- Software characteristics:
  - Usability
  - Portability
  - Performance
  - Reliability
  - Scalability
- Scientific Software correctness

Yet another definition: Scientific Computing

- Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems.[Wikipidia]
- Distinguish features:
  - concerned with variables that are continuous rather than discrete
  - concerned with *approximations* and their effects
- Approximations are used not just by choice: they are inevitable in most problems

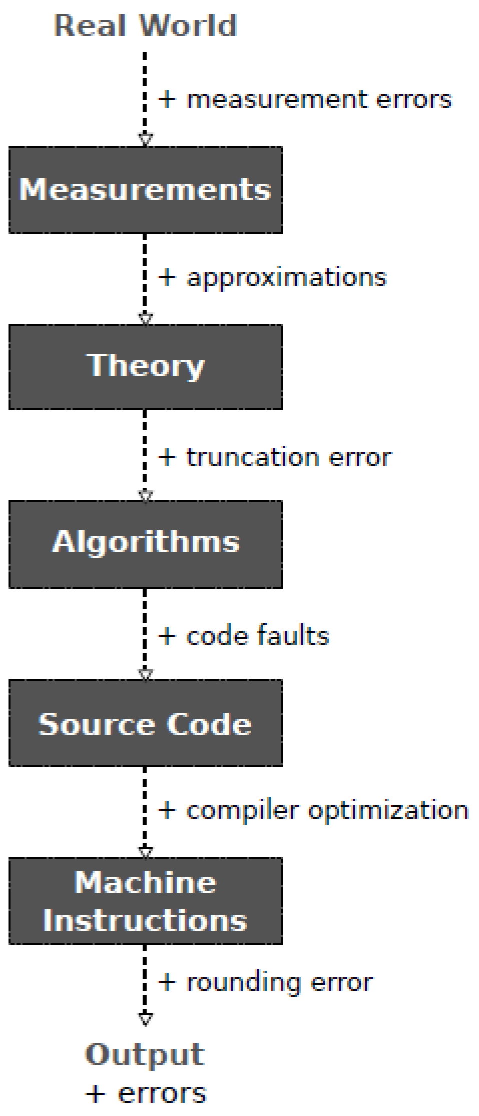
What do you need to do scientific computing ?

- Computers
  - (generally very huge/large and/or many of them)
- Software:
  - Something that allow you to use the computer (operating system/ middleware/compilers/libraries)
  - Something that allow you to simulate/compute what you needed (scientific programs / codes)
- Goodwill:
  - Learn how to use the right tools among a huge number of them
  - Learn how to write/use scientific program

Source of approximations

- Before computation begins:
  - Modeling: *neglecting certain physical features*
  - empirical measurements: *can’t always measure input data to the desired precision*
  - previous computations: *input data may be produced from error-prone numerical methods*
- During computation:
  - truncation: *numerical method approximate a continuos entity*
  - rounding: *computers offer only finite precision in representing real numbers*

Vocabulary



Scientific software development involves a number of model refinements in which errors may be introduced.

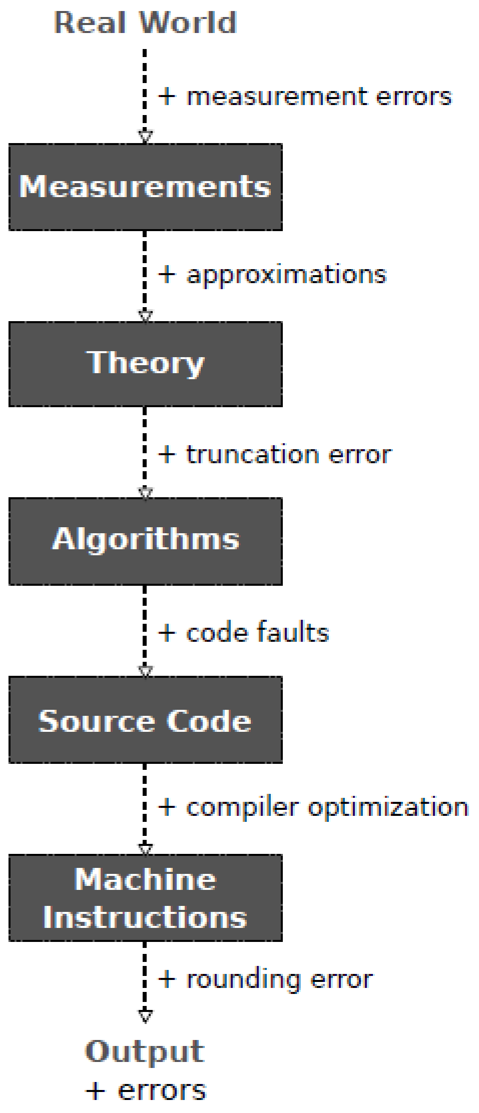
Program outputs include the accumulation of all these errors.

Stupid example

- Computing surface area of Earth using formula $A=4\pi r^2$
- This involves several approximations:
  - Modeling: Earth is considered as a sphere…
  - Measurements: value of radius is based on empirical methods
  - Truncation: value for π is truncated at a finite number..
  - Rounding: values for input data and results of arithmetic operations are rounded in computer.

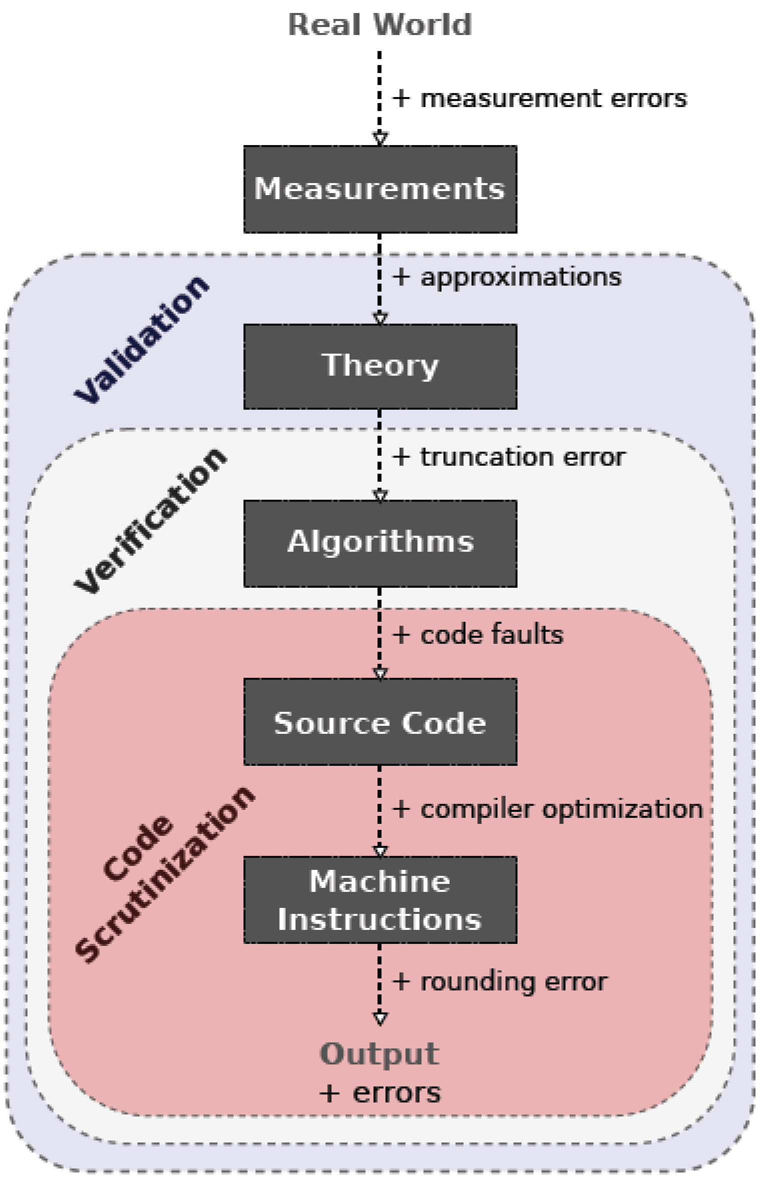
Rounding error

- Difference between result produced by a given algorithm using exact arithmetic and result produced by the same algorithm using rounded arithmetic
- Due to the inexact representation of real numbers and arithmetic operations upon them
- To understand where and how they turn out we need to know how computers deals with numbers..
- Error analysis techniques: how are your equations sensitive to roundoff errors ?
    - Forward error analysis: what errors did you make ?
    - Backward error analysis: which problem did you solve exactly ?



Scientific software development involves a number of model refinements in which errors may be introduced.

Program outputs include the accumulation of all these errors.



Validation and verification activities are applied in an attempt to identify or bound these errors.

In addition to validations and verifications, we also suggest that computational scientists should conduct *code scrutinizations*.

Scientific simulation Context

➢ Our simulations provide *approximate* solutions to problems for which we do not know the exact solution.

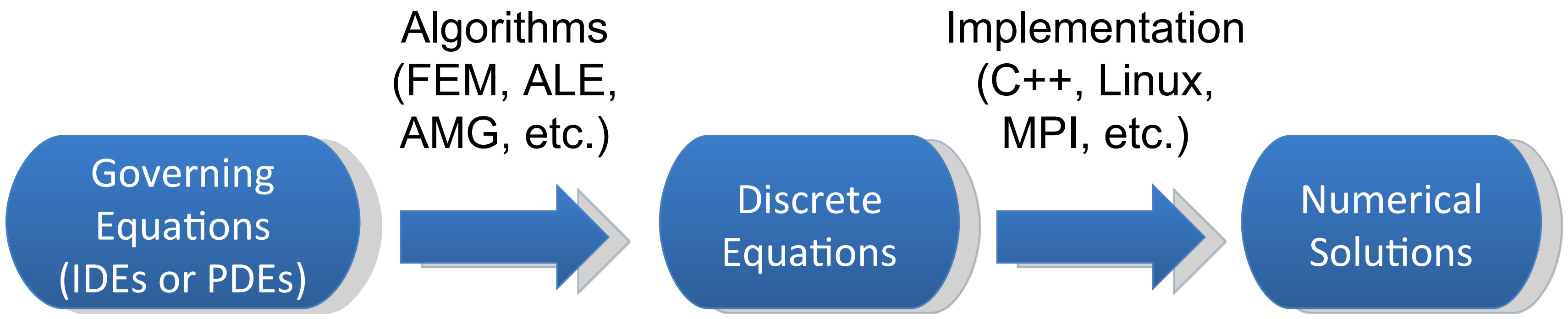
This leads to two more questions:

- How good are the approximations?
- How do you test the software?

A Numerical Simulation is the Conclusion of a Long Development Process

Model: Governing Equations, ICs, BCs, Submodels (constitutive models, closure relations, etc.)
Here, model does *not* mean code

Implementation: Compute the approximate solution


Algorithms (FEM, ALE, AMG, etc.)
Implementation (C++, Linux, MPI, etc.)
Governing Equations (IDEs or PDEs)
Discrete Equations
Numerical Solutions

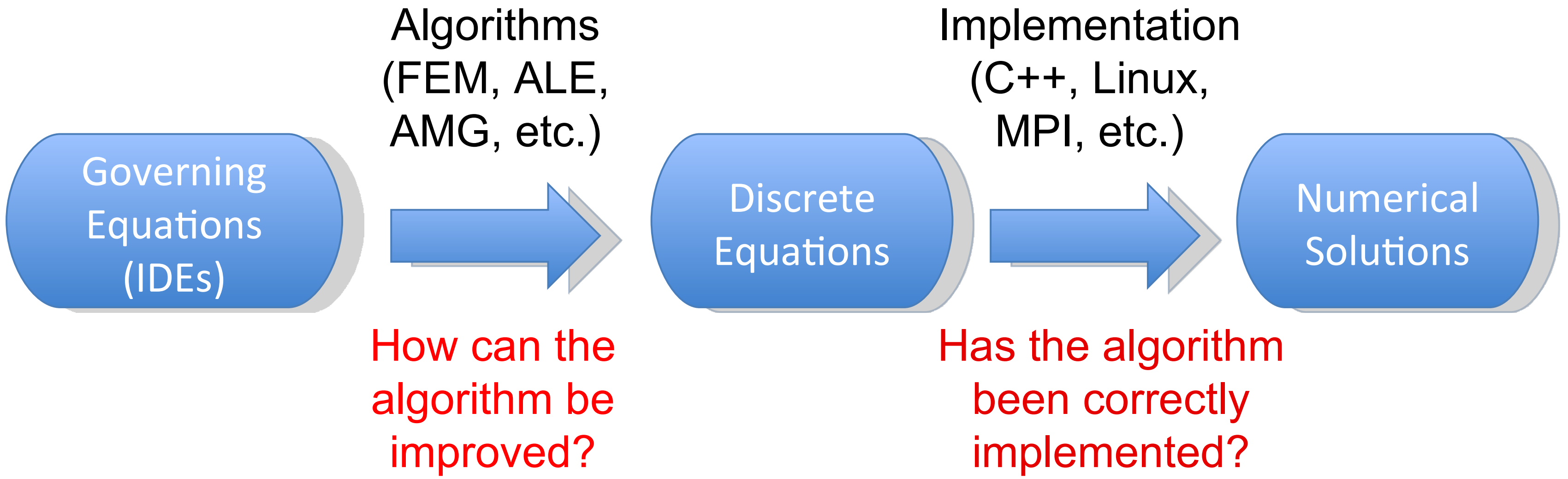
Algorithms: Generate a solvable discrete system; solution of the discrete system is an approximate solution of the model

What is code verification?

How good is your code?

Code Verification

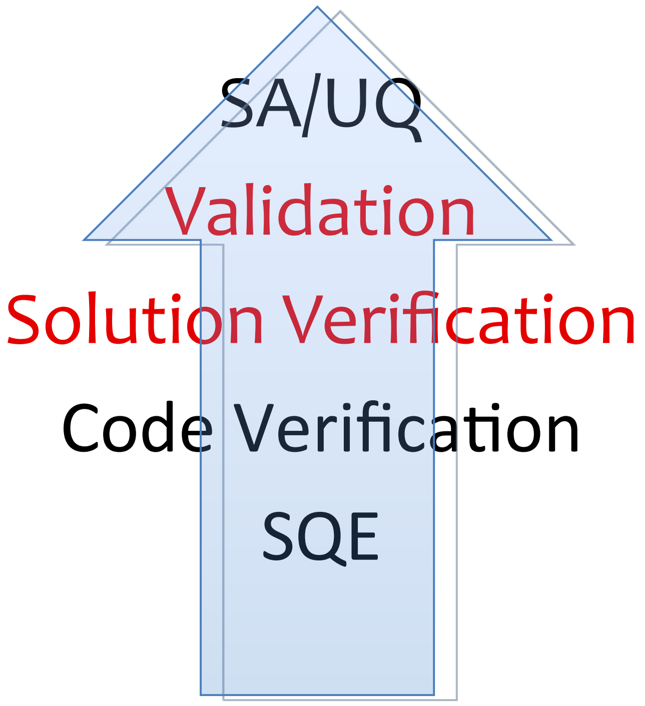
SQE


Governing Equations (IDEs)
Algorithms (FEM, ALE, AMG, etc.)
Discrete Equations
Implementation (C++, Linux, MPI, etc.)
Numerical Solutions
How can the algorithm be improved?
Has the algorithm been correctly implemented?

What else do we need?

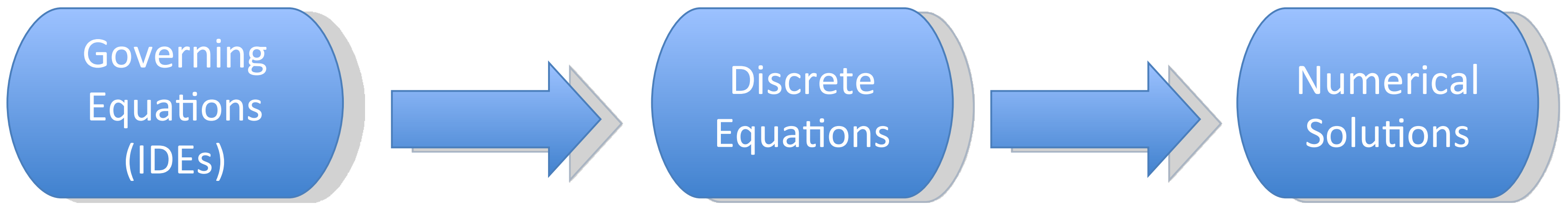
How good is your simulation?

How good is your code?



Are these equations adequate?

How large is the numerical error?



Verification and Validation (V&V) Definitions

Verification: Are the equations solved correctly?
(Math)

Validation: Are the equations correct?
(Physics)

- Verification: The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.
- Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

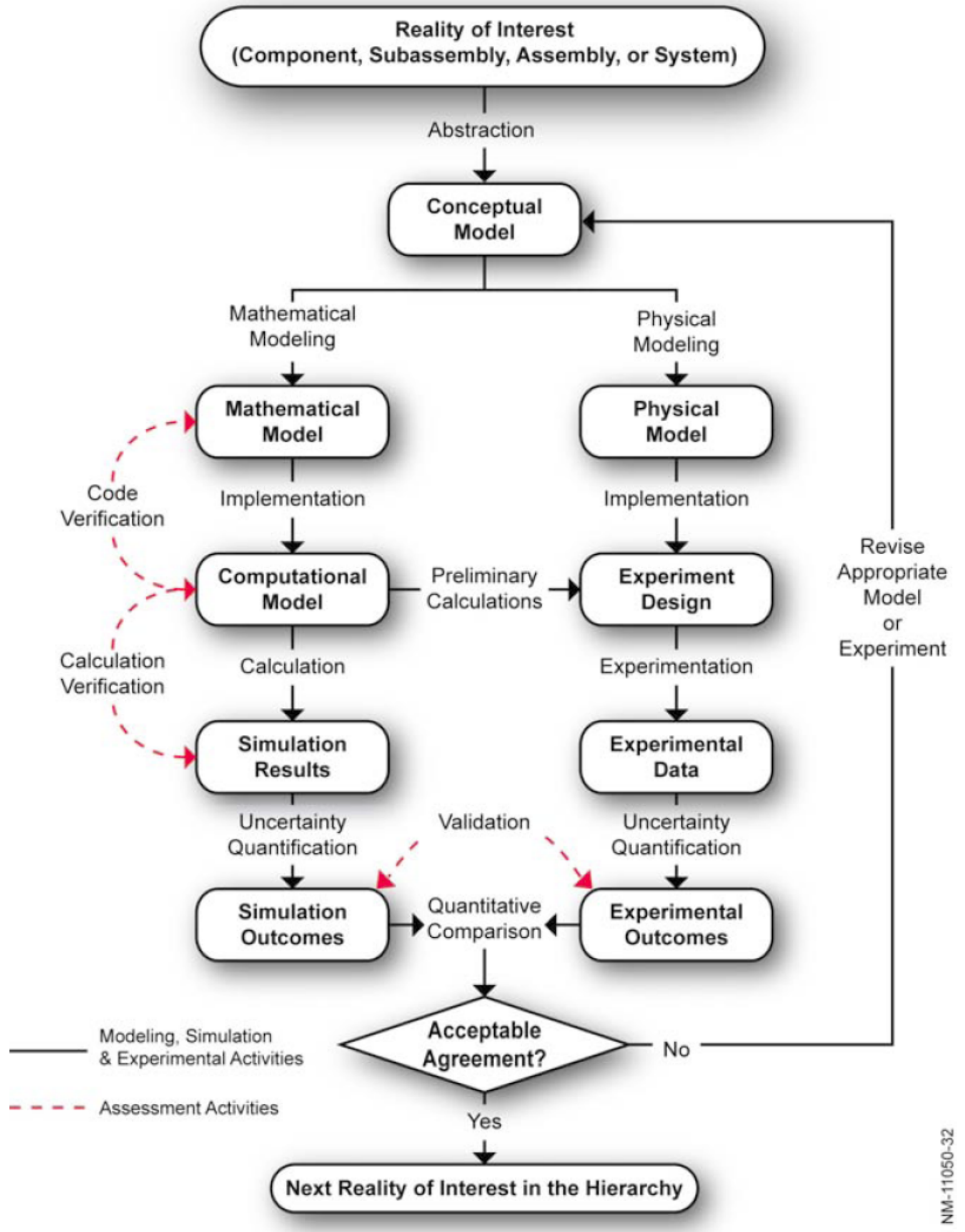
V&V targets applications of codes, not codes.

a few more informal Definitions

Software Quality Engineering (SQE)	Manage software complexity
Code Verification	Assess algorithms and their implementation vs. exact solutions
Solution Verification	Estimate discretization error
Validation	Assess physics models vs. experimental data
SA / UQ	Assess sensitivity of answer to input parameters

An ingredients list for predictive simulation, *not a menu*.

The ASME proposed a V&V workflow.



- This V&V process flowchart is taken from the ASME Solid Mechanics V&V guide.

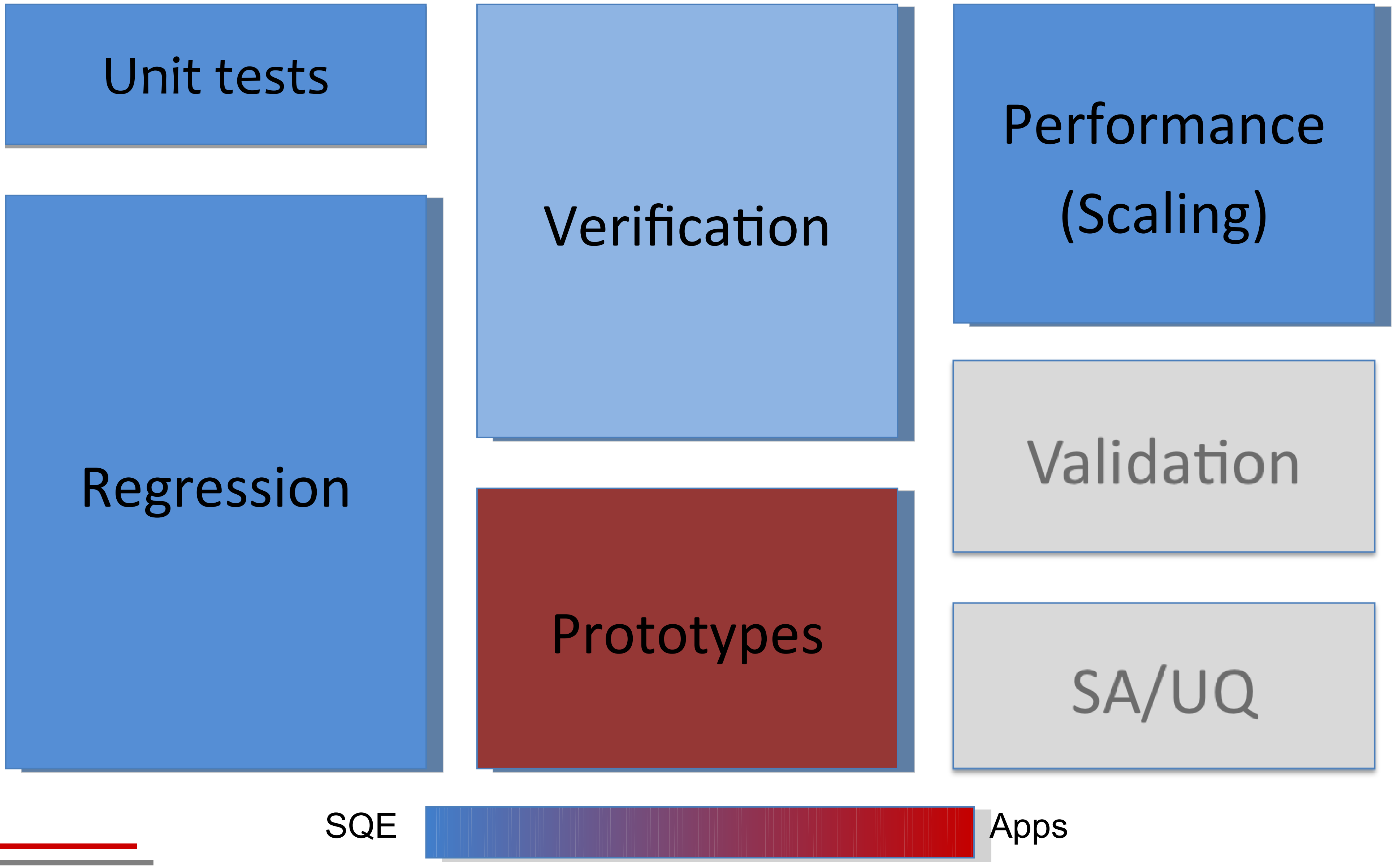
- Note the positions of code verification, calculation verification, validation, and UQ in this workflow diagram.

ASME, *V&V 10-2006 Guide for Verification and Validation in Computational Solid Mechanics*, American Society of Mechanical Engineers (2006).

Question: is ASME approach ok for everything?

- Homework #1:
  - Try to answer this question on your blog : can apply ASME workflow to my computational project ?

Tests for simulations...



Code Verification As A Continuous Process

- To set up a verification problem once takes significant effort – steep learning curve, infrastructure is not in place

```
while 1:
   …
   run verification_suite
```

- Running a verification analysis you have maintained takes minimal work
- Without regular, automated verification testing, verification results go stale quickly - they do not reflect the current state of the code

Code Verification Is Not Free

Principal Costs:

- Infrastructure development
- Test development

Recurring Costs – A tax on development:

- Maintenance of existing tests
- Code development becomes a very deliberate process

Sustainable verification: Benefits outweigh costs

Code Verification Identifies Algorithmic Weaknesses

One purpose of code verification is to find bugs.

- Code verification often finds bugs that are subtle and otherwise difficult to identify.
- The eyeball norm finds most obvious bugs quickly.

Perhaps a better use of code verification is to guide code development.

- Some bugs are algorithmic and conceptual.
- Code verification identifies algorithmic weaknesses.
- Large errors are a weakness.

The Most Efficient Code Verification is Done by Code Developers

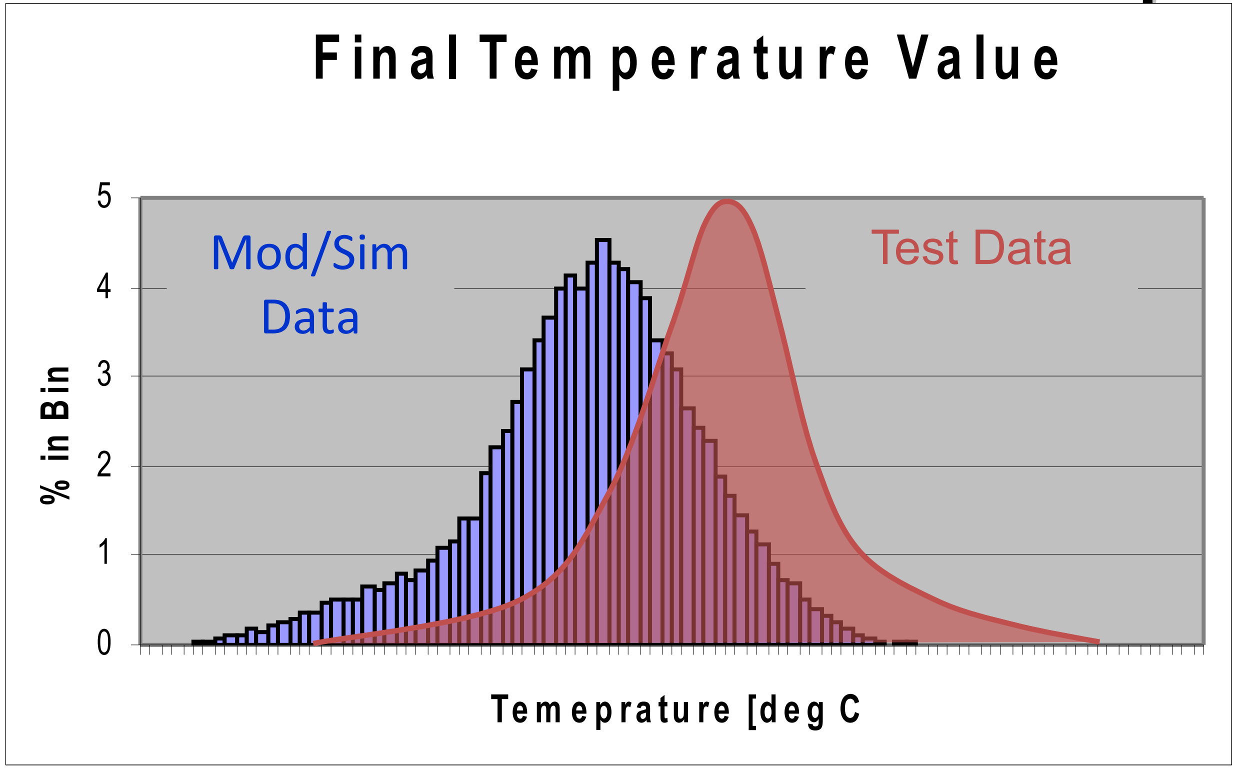
- Code developers best understand the numerical methods they are using

- Code developers are best able to use the results of code verification (and other forms of assessment) to improve the algorithms they use

Verification Testing Must Be a Team Ethic

- Discipline is required to keep a “clean” test suite – to keep all tests passing; “stop-the-line” mentality
- If only part of the team values verification, that part is always carrying the other developers
- Maintaining an automated verification test suite is probably necessary but *definitely not sufficient*
- Developers should be using verification tests interactively

A notional example of validation analysis illustrates the incorporation of uncertainty.



Validation:

- Compare simulation data histogram to a test data histogram.
- Quantify amount of "overlap" between histograms.
- Assess sufficiency of overlap.

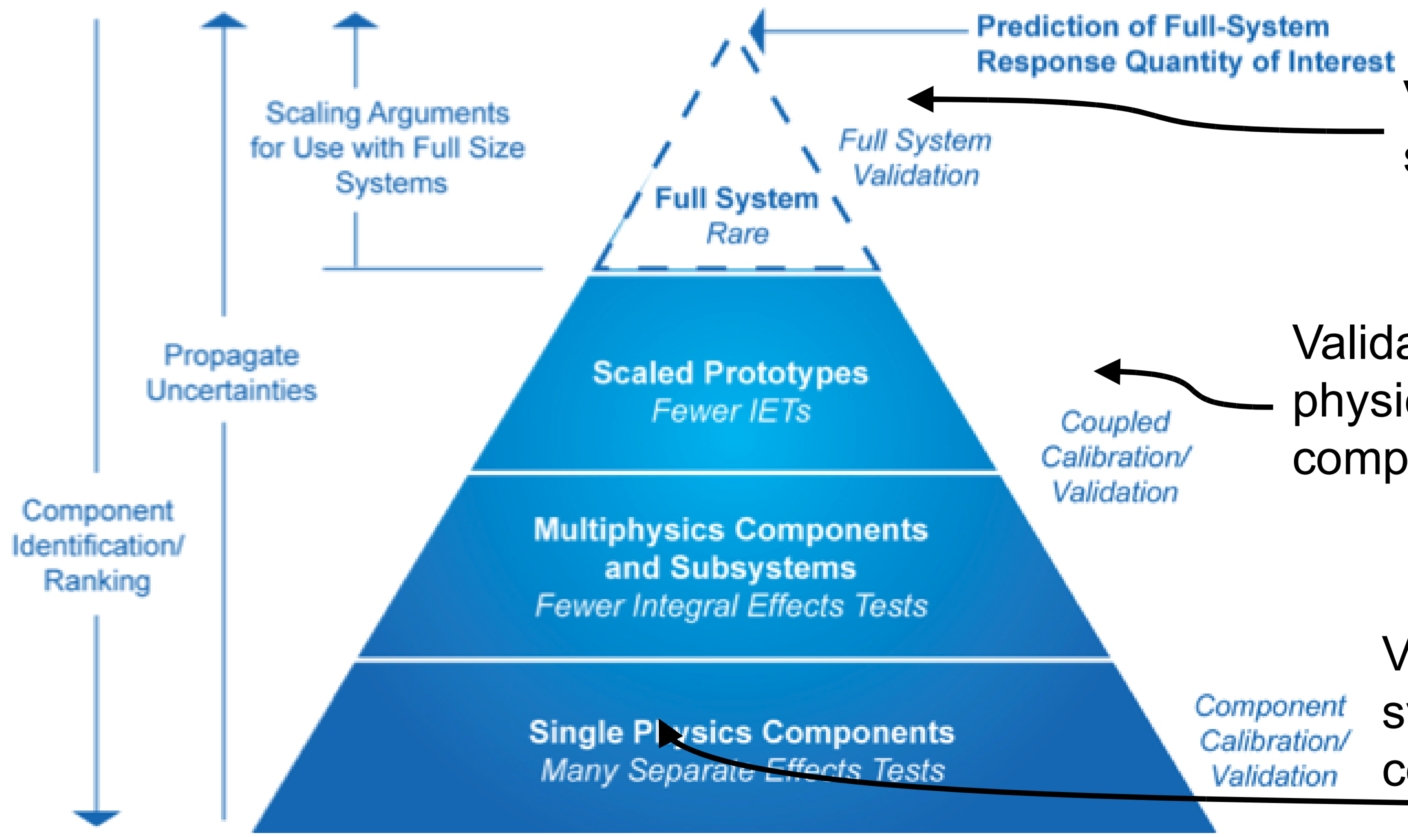
Uncertainty Quantification:

- UQ methods generate an ensemble of mod/sim data.

- UQ methods are used to generate statistical information on the code output.
  - Probability distribution on Temperature, *given various $x_1,\ldots,x_N$ inputs.*
  - Correlations (i.e., trends) of Temperature vs. $x_1,\ldots,x_N$.
  - Mean(T), StdDev(T), Probability($T > T_{critical}$)

Credit: Tim Trucano

Complex problems require a hierarchical approach to validation.


Prediction of Full-System Response Quantity of Interest
Scaling Arguments for Use with Full Size Systems
Full System Validation
Full System
Rare
Propagate Uncertainties
Scaled Prototypes
Fewer IETs
Coupled Calibration/ Validation
Component Identification/ Ranking
Multiphysics Components and Subsystems
Fewer Integral Effects Tests
Single Physics Components
Many Separate Effects Tests
Component Calibration/ Validation

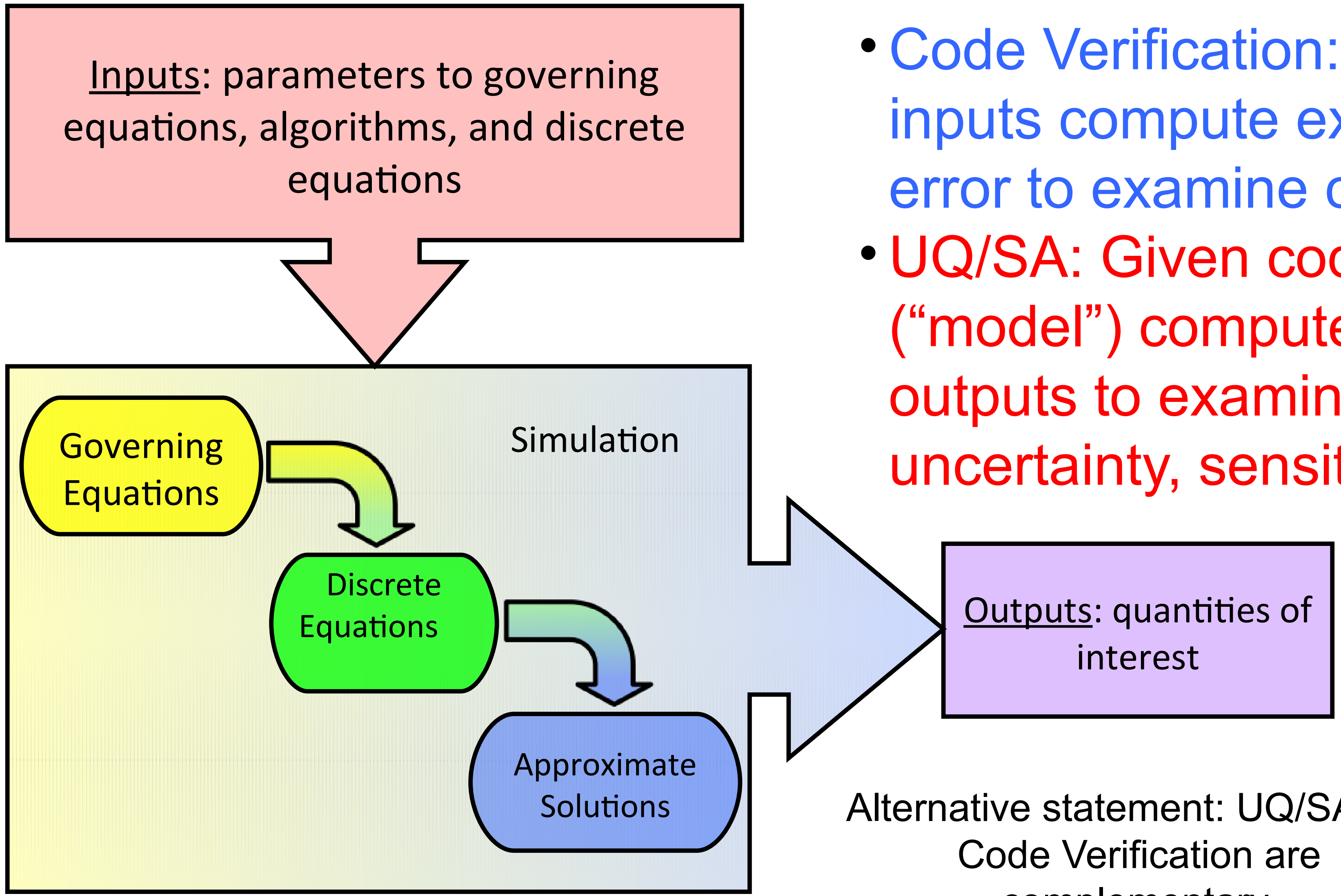
Validation of “full-physics” systems is usually rare.

Validation of “coupled-physics” systems is often complex and challenging.

Validation of “single-physics” systems is the most common analysis.

Credit: D. Kothe et al., Consortium for Advanced Simulation of Light Water Reactors (CASL), 2010.

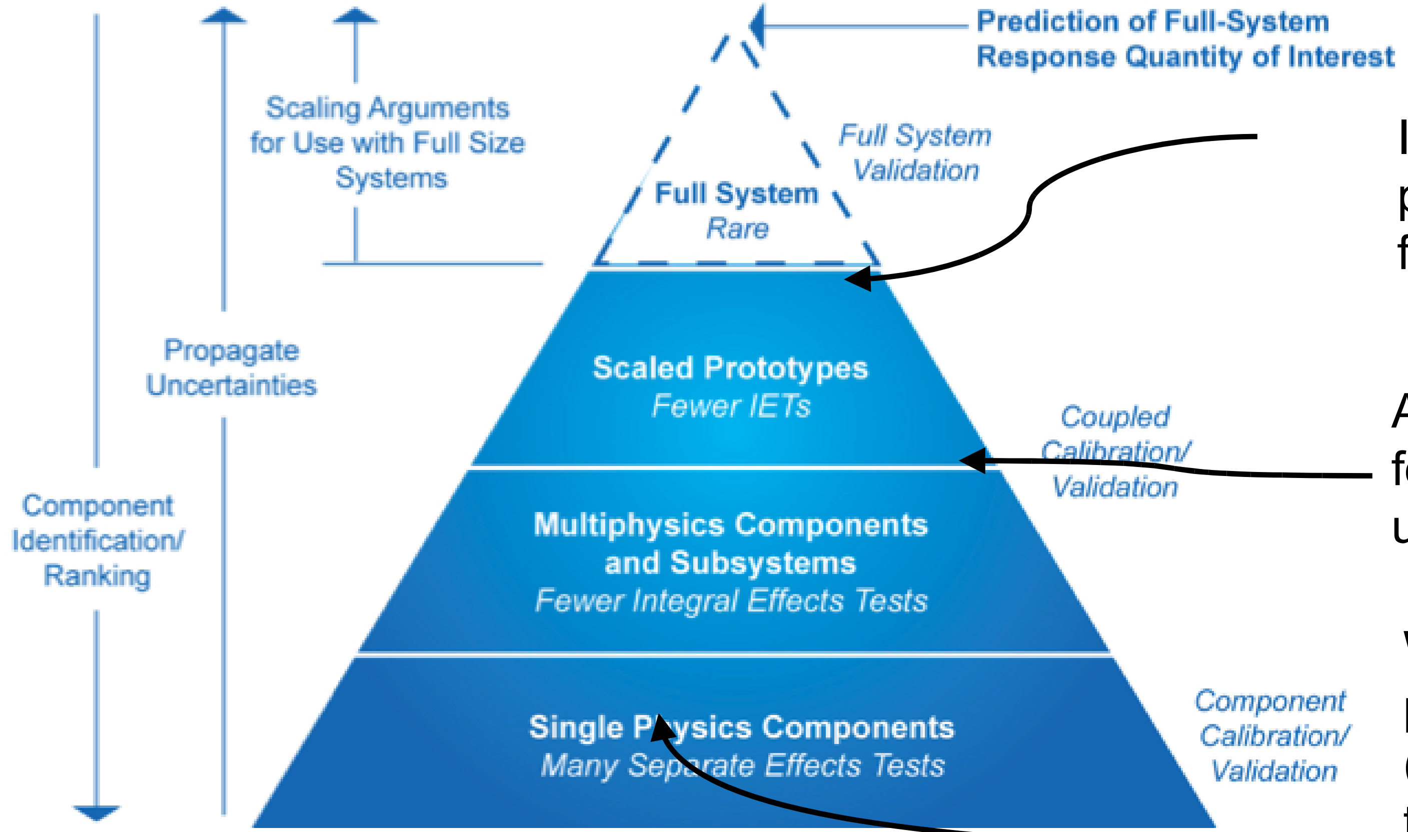
Verification is orthogonal to UQ and SA


Inputs: parameters to governing equations, algorithms, and discrete equations
Simulation
Governing Equations
Discrete Equations
Approximate Solutions
Outputs: quantities of interest

- Code Verification: Given inputs compute exact error to examine code.
- UQ/SA: Given code ("model") compute outputs to examine uncertainty, sensitivity.

Alternative statement: UQ/SA and Code Verification are complementary.

The fundamental tension: find out as much as you can, but recognize you can't eliminate all risk


Prediction of Full-System Response Quantity of Interest
Scaling Arguments for Use with Full Size Systems
Full System Validation
Full System
Rare
Propagate Uncertainties
Scaled Prototypes
Fewer IETs
Coupled Calibration/ Validation
Component Identification/ Ranking
Multiphysics Components and Subsystems
Fewer Integral Effects Tests
Single Physics Components
Many Separate Effects Tests
Component Calibration/ Validation

If we could validate here, we wouldn't need simulations.

Is there any evidence for the possibility of a catastrophic failure?

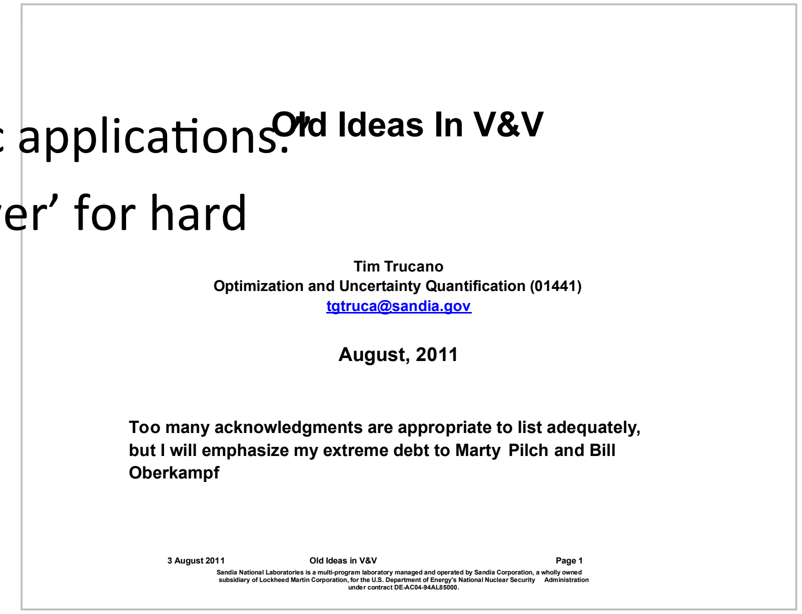
Assessments tell where to focus resources to reduce uncertainties.

We can often make predictions with confidence (but it takes a lot more work than you might think.)

Image: Kothe et al., Consortium for Advanced Simulation of Light Water Reactors (CASL) proposal, 2010.

Tim Trucano's observations on V&V…

- Key V&V themes have not changed "for decades":
  - "Codes are not solutions, people are solutions."
  - "Credibility of computational simulations for defined applications is evolutionary…"
  - "… at worst, credibility is non-existent in specific applications."
  - "Single calculations will never be 'the right answer' for hard problems."
  - "Real V&V and real UQ are a lot of work."



Old Ideas In V&V

Tim Trucano
Optimization and Uncertainty Quantification (01441)
tgtruca@sandia.gov

August, 2011

Too many acknowledgments are appropriate to list adequately, but I will emphasize my extreme debt to Marty Pilch and Bill Oberkampf

3 August 2011 Old Ideas in V&V Page 1

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

- Trucano's four insights on V&V:
  1. "V&V — pay me now or pay me later."
  2. "Journal editorial policies and practices must change."
  3. "Ask 'What's good enough?'"
  4. "Saying you don't need verification is like saying you don't need oxygen."

Final ideas: what should we provide....

A software infrastructure to support testing...

- Ability to run a large number of simulations repeatedly
- Ability to compare results with a baseline
- Ability to report pass/diff/fail
- Run on platforms of interest at regular intervals

Homework

- Describe on the blog:
  - which kind of testing procedure you adopted so far in your code
  - Which kind verification are you plan to use ?
  - Which kind of validation are you plan to use ?

eLab
Joint DEMOCRITOS/SISSA
Laboratory for e-Science

Introduction to Testing scientific software

Stefano Cozzini

Democritos and SISSA/eLAB - Trieste

Outline

- Introducing the problem:
  - What does testing software mean ?
  - Why,where,when,who, what to test ?
- Scientific software testing..
  - What is scientific computing ?
  - Why is so difficult to test simulations ?
  - Reviewing verification/validation
  - Some idea/suggestions
  - Homework

Aim of this lecture

- Share some interesting ideas i read about
- Tell about some current effort to apply some of such ideas to a scientific software development
- Convince you that testing scientific software is needed

- Disclaimer:
  - Content of these slides come from many sources all around the web; some slides are mine, some other are taken from other presentations without modification, some other are slightly modified

First definitions : errors and faults

Error

is a measure of the difference between a measured or calculated value of a quantity and what is considered to be its actual value.

Faults

Code faults are mistakes made when abstract algorithms are implemented in code.

Faults are not errors, but they frequently lead to errors.

Second definition: testing

- Software testing is a process by which one or more expected behaviors and results from a piece of software are exercised and confirmed. Well chosen tests will confirm expected code behavior for the extreme boundaries of the input domains, output ranges, parametric combinations, and other behavioral edge cases.

  (from yesterday Tommy's lecture)

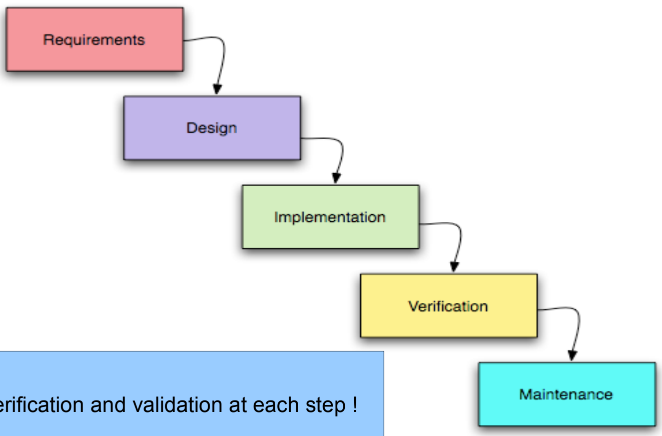
- Software testing can be stated as the process of validating and verifying that a software program/application/product:
  - meets the requirements that guided its design and development;
  - works as expected; and
  - can be implemented with the same characteristics.

(from wikipedia)

Third definition: Verification validation

- Verification: "Are we building the product right ?"
  - The software should conform to its specification
- Validation: "Are we building the right product ? "
  - The software should do what the user really requires
- V & V must be applied at each stage in the software process
- Two main objectives:
  - Discovery of defects in a system
  - Assessment of whether the system is usable in an operational situation

Software stages: waterfall model

- 

Verification and validation at each step !

Is all the above true for scientific codes/packages/ simulation method ???

- Sure !
- But something more is required…

The 5 W of testing: Why testing ?

- To find faults
- To provide confidence
  - of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults
- Other issues include:
  - – Performance of systems (i.e. use of resources like time, space, Bandwidth,...).
  - "...ilities" can be the subject of test e.g. usability, learnability, reliability,availability, etc..

The 5 W of testing: When testing ?

- When:
  - Always !
- Different granularity:
  - When I/they change of the code
  - at every night to check possible problem
  - When a new feature/release is available
  - When we start using a new package for scientific research

The 5 W of testing: Who should test ?

- Who:
  - You as developer
  - You as part of a developing team:
    - Try to test things you did not write
    - Find some other to test you own software
  - You as user:
    - Is this software/package/routines/code what I really need ?
  - You as scientific user (never use scientific code without your own test !!!)

The 5 W of testing: Where to test ?

- Software point of view
  - On any single function of your code:
    - Unit testing
  - On the code as a whole
    - Regression tests
- Hardware point of view
  - On all the possible platforms you have at disposal
    - Ensure portability of software and of scientific output !

The 5 W of testing: What should I test ?

- Software characteristics:
  - Usability
  - Portability
  - Performance
  - Reliability
  - Scalability
- Scientific Software correctness

Yet another definition: Scientific Computing

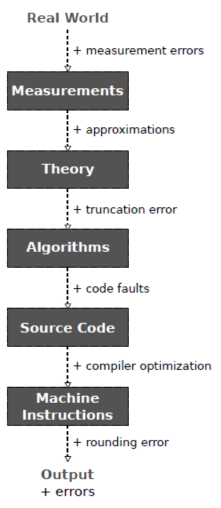
- Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems.[Wikipidia]
- Distinguish features:
  - concerned with variables that are continuous rather than discrete
  - concerned with *approximations* and their effects
- Approximations are used not just by choice: they are inevitable in most problems

What do you need to do scientific computing ?

- Computers
  - (generally very huge/large and/or many of them)
- Software:
  - Something that allow you to use the computer (operating system/ middleware/compilers/libraries)
  - Something that allow you to simulate/compute what you needed (scientific programs / codes)
- Goodwill:
  - Learn how to use the right tools among a huge number of them
  - Learn how to write/use scientific program

Source of approximations

- Before computation begins:
  - Modeling: *neglecting certain physical features*
  - empirical measurements: *can't always measure input data to the desired precision*
  - previous computations: *input data may be produced from error-prone numerical methods*
- During computation:
  - truncation: *numerical method approximate a continuos entity*
  - rounding: *computers offer only finite precision in representing real numbers*



Scientific software development involves a number of model refinements in which errors may be introduced.

Program outputs include the accumulation of all these errors.

Stupid example

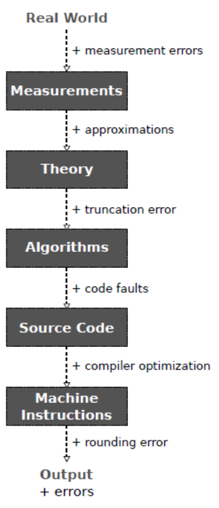
- Computing surface area of Earth using formula $A=4\pi r^2$

- This involves several approximations:
  - Modeling: Earth is considered as a sphere…
  - Measurements: value of radius is based on empirical methods
  - Truncation: value for π is truncated at a finite number..
  - Rounding: values for input data and results of arithmetic operations are rounded in computer.

Rounding error

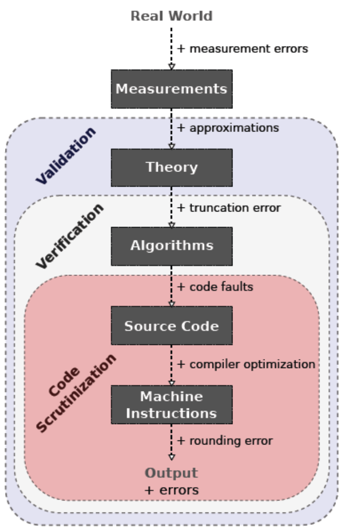
- Difference between result produced by a given algorithm using exact arithmetic and result produced by the same algorithm using rounded arithmetic
- Due to the inexact representation of real numbers and arithmetic operations upon them
- To understand where and how they turn out we need to know how computers deals with numbers..
- Error analysis techniques: how are your equations sensitive to roundoff errors ?
  - Forward error analysis: what errors did you make ?
  - Backward error analysis: which problem did you solve exactly ?

Vocabulary



Scientific software development involves a number of model refinements in which errors may be introduced.

Program outputs include the accumulation of all these errors.



Validation and verification activities are applied in an attempt to identify or bound these errors.

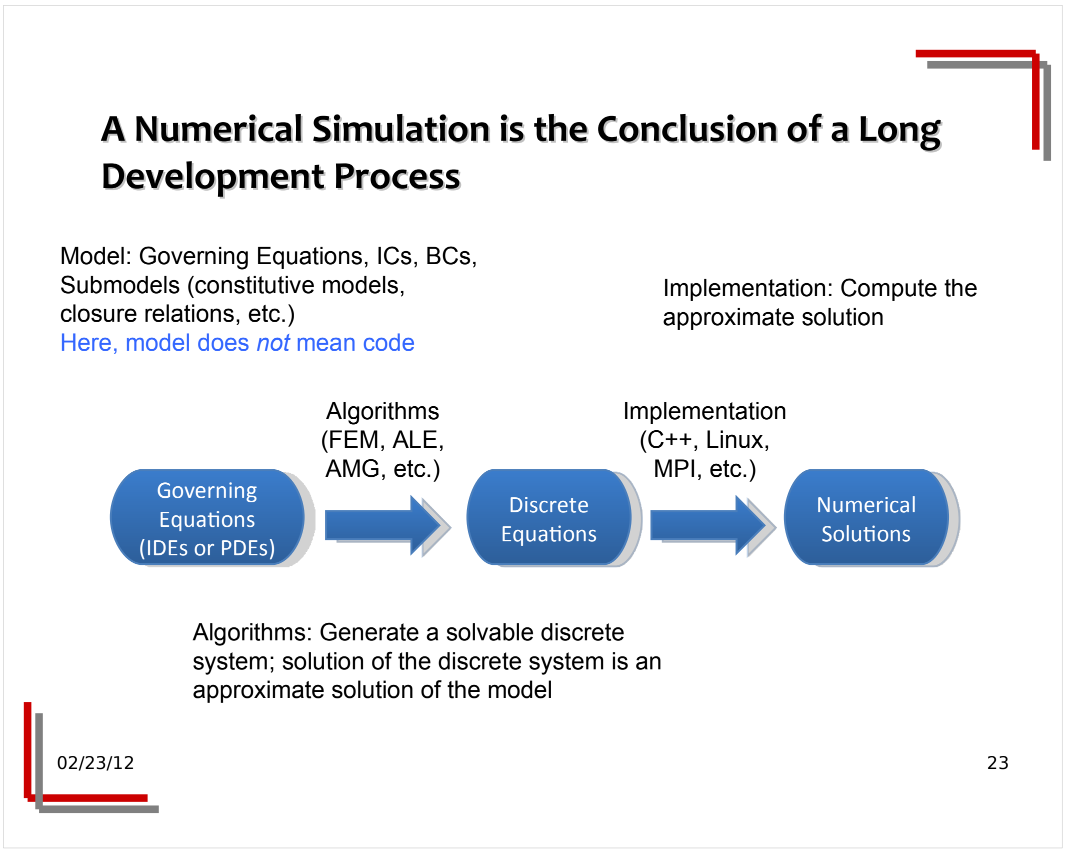
In addition to validations and verifications, we also suggest that computational scientists should conduct *code scrutinizations*.

Scientific simulation Context

➢ Our simulations provide *approximate* solutions to problems for which we do not know the exact solution.

This leads to two more questions:

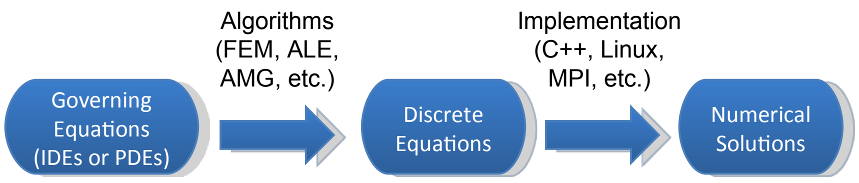
- How good are the approximations?
- How do you test the software?



A Numerical Simulation is the Conclusion of a Long Development Process

Model: Governing Equations, ICs, BCs, Submodels (constitutive models, closure relations, etc.)
Here, model does *not* mean code

Implementation: Compute the approximate solution

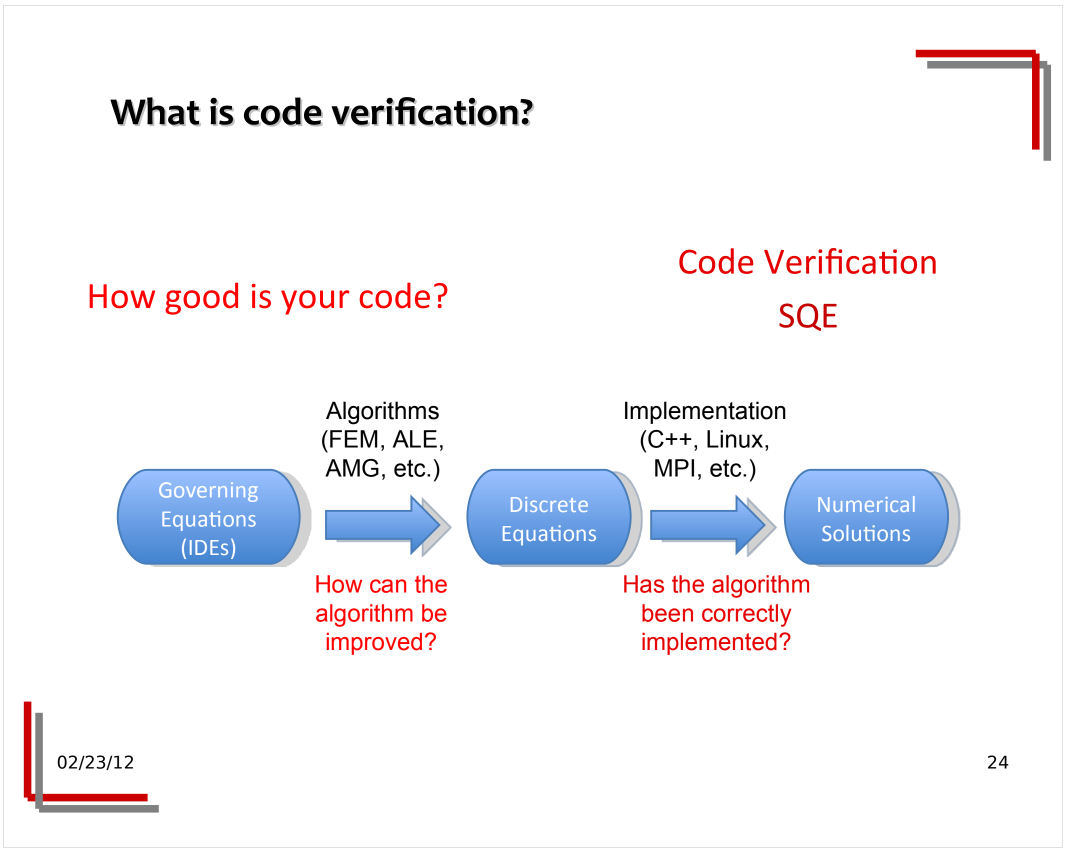

Algorithms (FEM, ALE, AMG, etc.)
Implementation (C++, Linux, MPI, etc.)
Governing Equations (IDEs or PDEs)
Discrete Equations
Numerical Solutions

Algorithms: Generate a solvable discrete system; solution of the discrete system is an approximate solution of the model

02/23/12 23

Another piece, not illustrated in this diagram, is that the model(s), algorithms, and implementation require input parameters which are uncertain.

Parameter Uncertainty

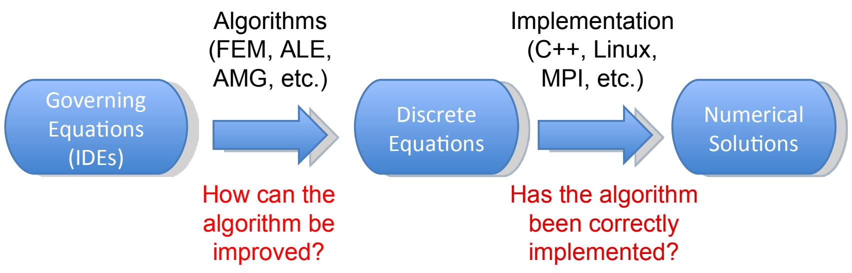


What is code verification?

How good is your code?

Code Verification

SQE

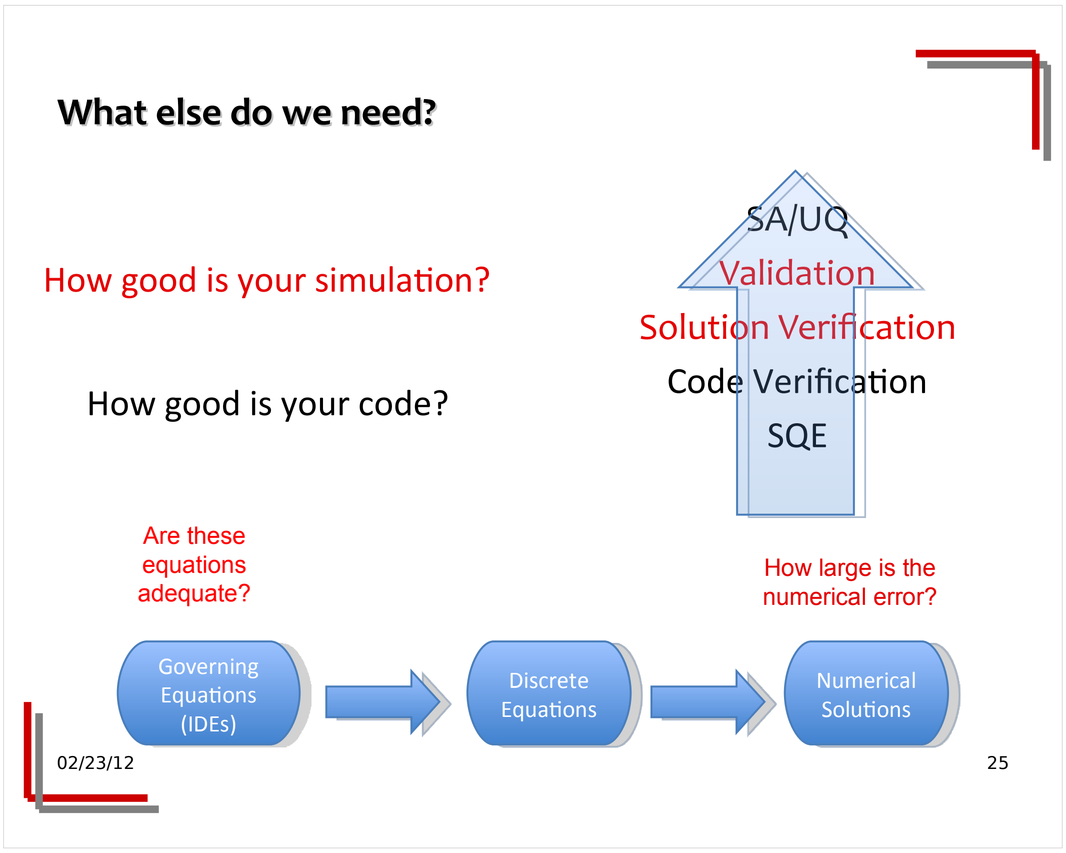

Algorithms (FEM, ALE, AMG, etc.)
Implementation (C++, Linux, MPI, etc.)
Governing Equations (IDEs)
Discrete Equations
Numerical Solutions
How can the algorithm be improved?
Has the algorithm been correctly implemented?

02/23/12

24

Code verification is a requirement for solution verification, validation, and UQ

Strict and loose code verification, definitions and practice



What else do we need?

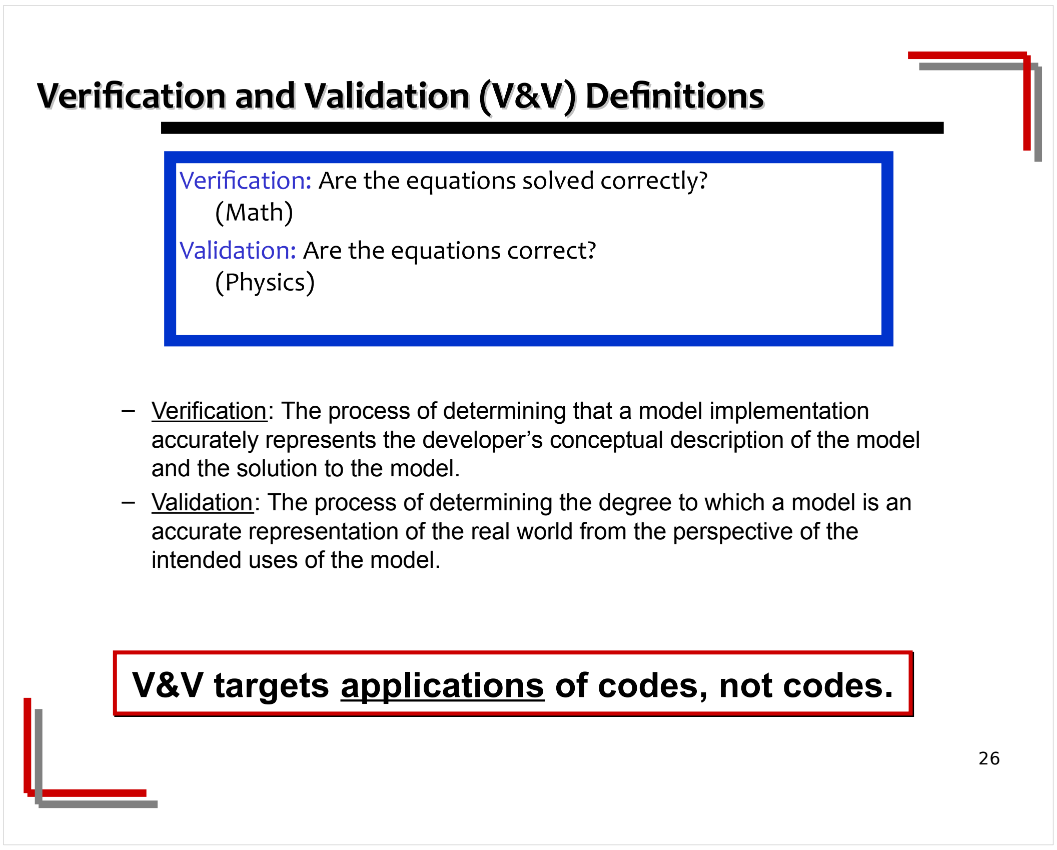
How good is your simulation?

How good is your code?

SA/UQ
Validation
Solution Verification
Code Verification
SQE
Are these equations adequate?
How large is the numerical error?
Governing Equations (IDEs)
Discrete Equations
Numerical Solutions

Code verification is a requirement for solution verification, validation, and UQ

Strict and loose code verification, definitions and practice



Verification and Validation (V&V) Definitions

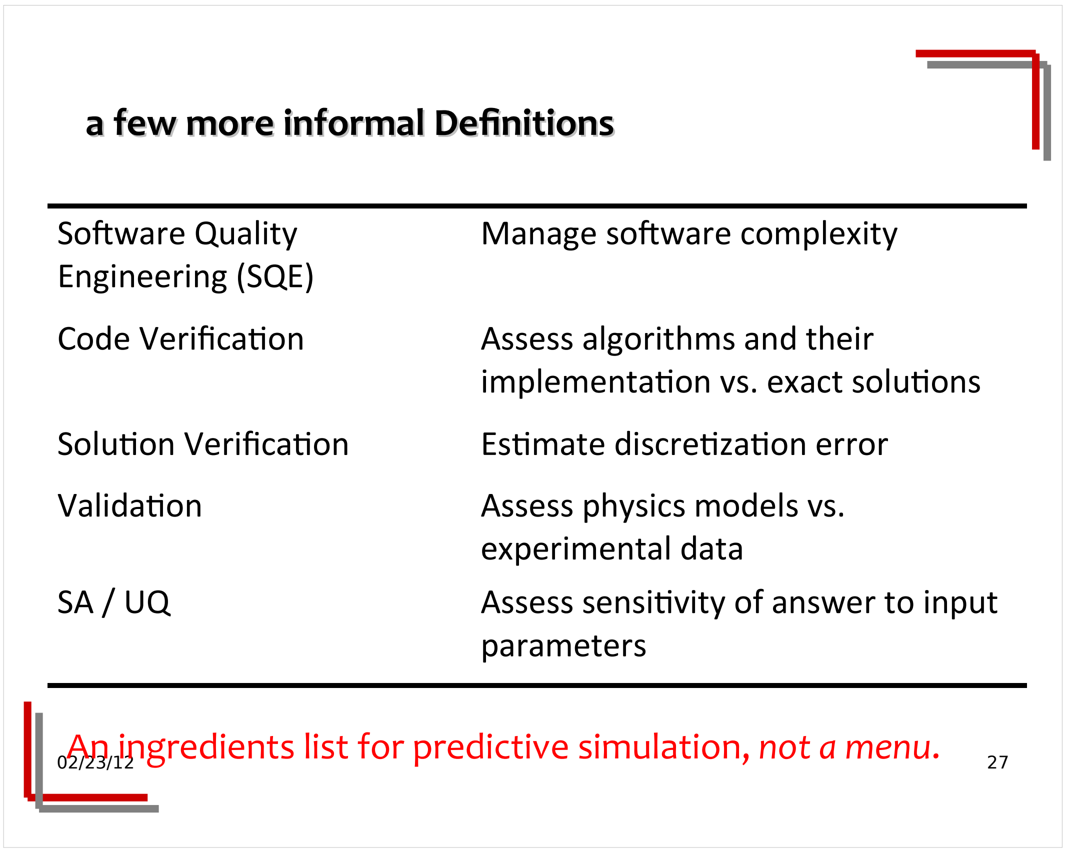
Verification: Are the equations solved correctly?
(Math)

Validation: Are the equations correct?
(Physics)

- Verification: The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.
- Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

V&V targets applications of codes, not codes.

26



a few more informal Definitions

Software Quality Engineering (SQE)	Manage software complexity
Code Verification	Assess algorithms and their implementation vs. exact solutions
Solution Verification	Estimate discretization error
Validation	Assess physics models vs. experimental data
SA / UQ	Assess sensitivity of answer to input parameters

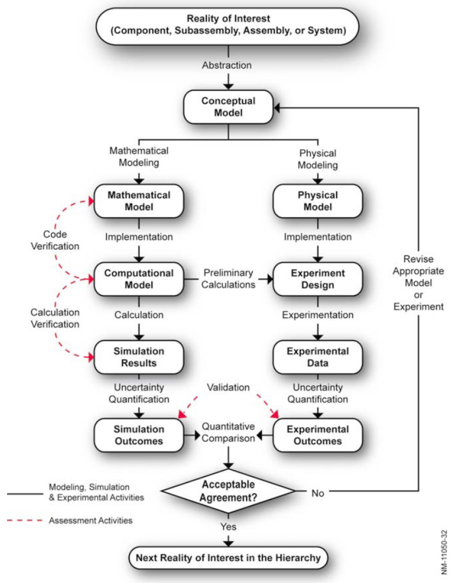
An ingredients list for predictive simulation, *not a menu.*

02/23/12

27

Each method of assessment is independent, each answers a different question.

The ASME proposed a V&V workflow.



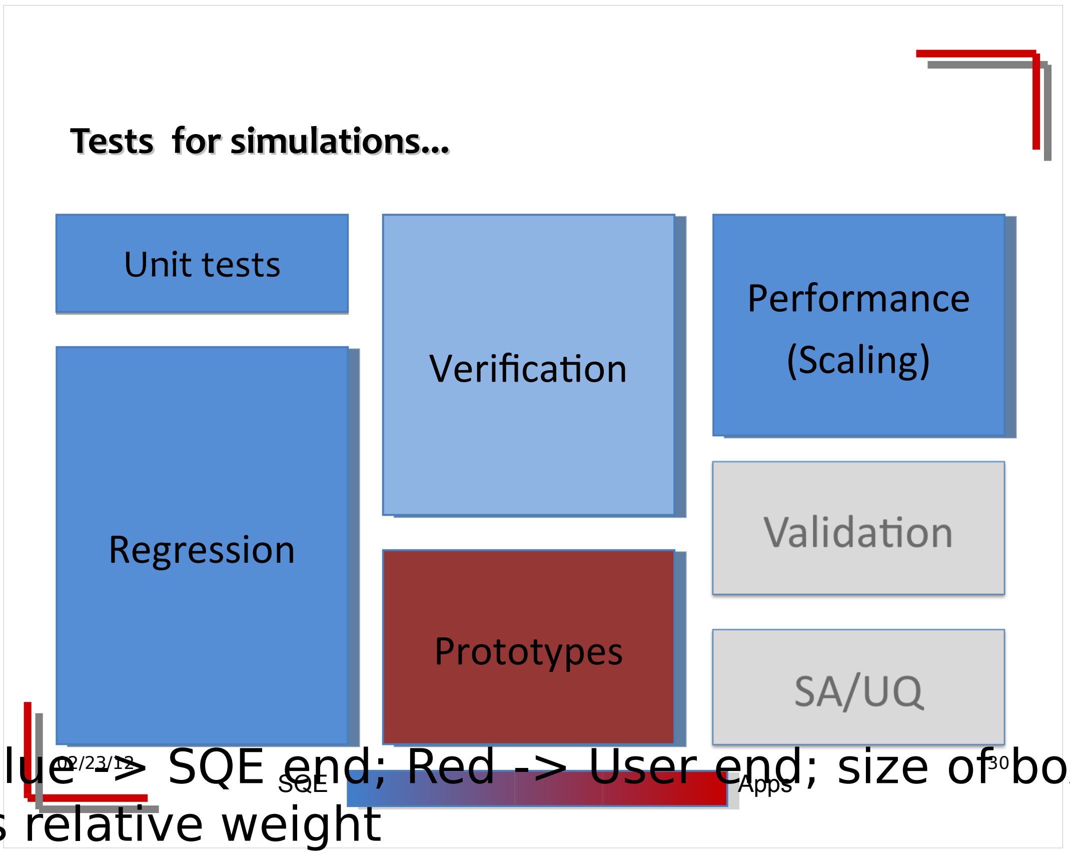
- This V&V process flowchart is taken from the ASME Solid Mechanics V&V guide.

- Note the positions of code verification, calculation verification, validation, and UQ in this workflow diagram.

ASME, *V&V 10-2006 Guide for Verification and Validation in Computational Solid Mechanics*, American Society of Mechanical Engineers (2006).

Question: is ASME approach ok for everything?

- Homework #1:
  - Try to answer this question on your blog : can apply ASME workflow to my computational project ?



Blue -> SQE end; Red -> User end; size of box is relative weight

Clarify the metric for regression tests: yesterday's results

Code librarian oversees building and testing across different platforms.

Nightly regression + verification is thousands of simulations, mostly small and fast. Run on Linux with multiple compilers, Sun, IBM

Long tests run weekly on one or two platforms (currently in transition)

Performance is mostly scaling. We also check runtimes piggy-backed on Regression.

Code Verification As A Continuous Process

- To set up a verification problem once takes significant effort – steep learning curve, infrastructure is not in place

```
while 1:
   …
   run verification_suite
```

- Running a verification analysis you have maintained takes minimal work
- Without regular, automated verification testing, verification results go stale quickly - they do not reflect the current state of the code

Code Verification Is Not Free

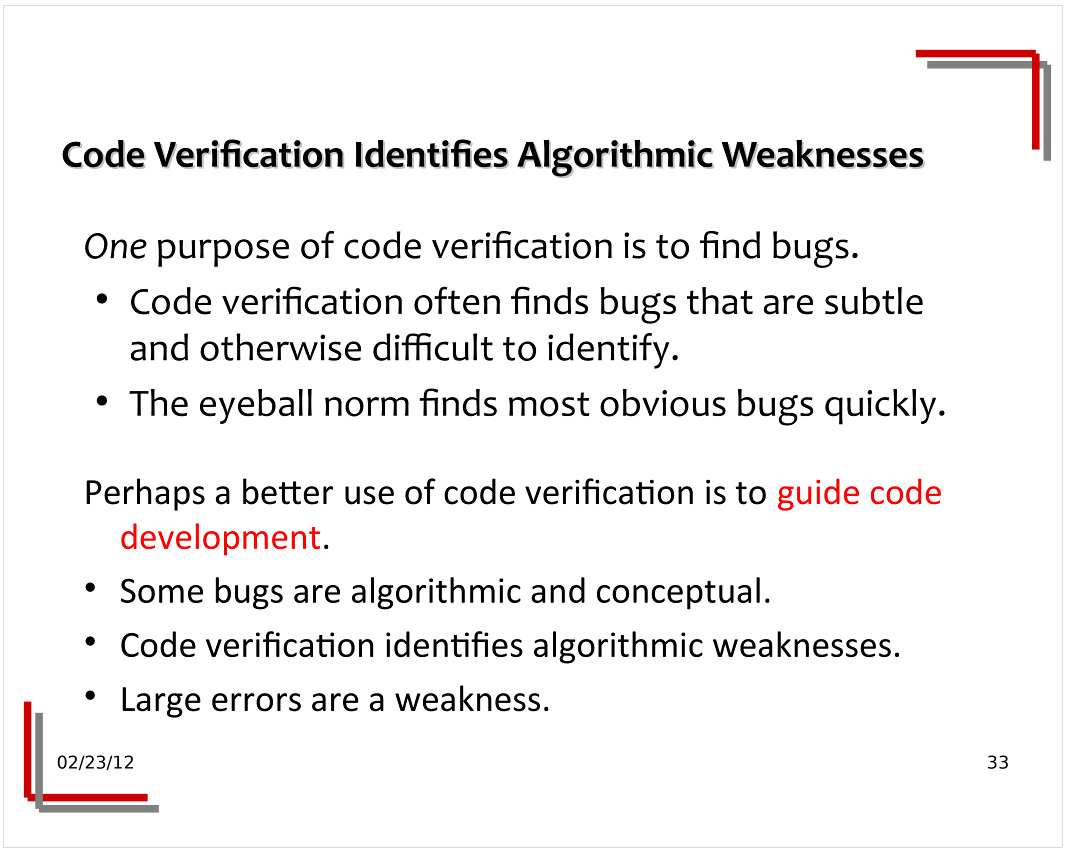
Principal Costs:

- Infrastructure development
- Test development

Recurring Costs – A tax on development:

- Maintenance of existing tests
- Code development becomes a very deliberate process

Sustainable verification: Benefits outweigh costs



Code Verification Identifies Algorithmic Weaknesses

One purpose of code verification is to find bugs.

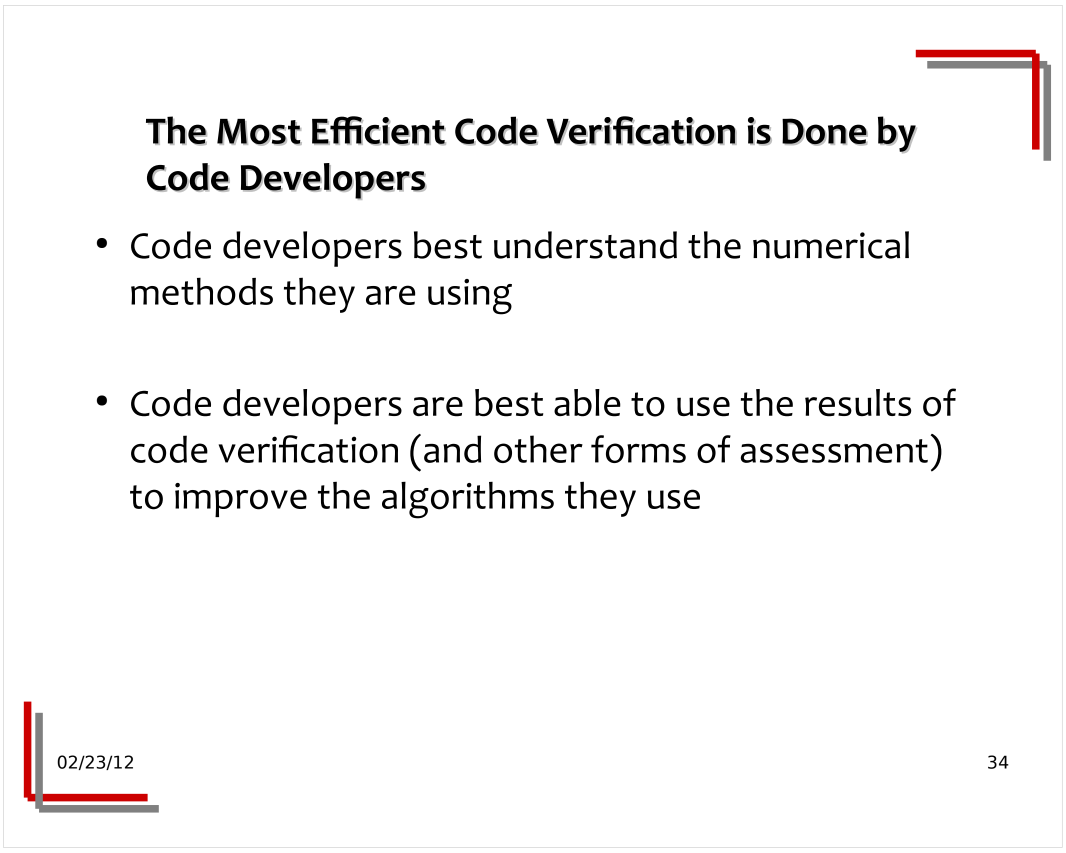
- Code verification often finds bugs that are subtle and otherwise difficult to identify.
- The eyeball norm finds most obvious bugs quickly.

Perhaps a better use of code verification is to guide code development.

- Some bugs are algorithmic and conceptual.
- Code verification identifies algorithmic weaknesses.
- Large errors are a weakness.

02/23/12 33

Enhancing code development is an additional benefit of code verification.



The Most Efficient Code Verification is Done by Code Developers

- Code developers best understand the numerical methods they are using
- Code developers are best able to use the results of code verification (and other forms of assessment) to improve the algorithms they use

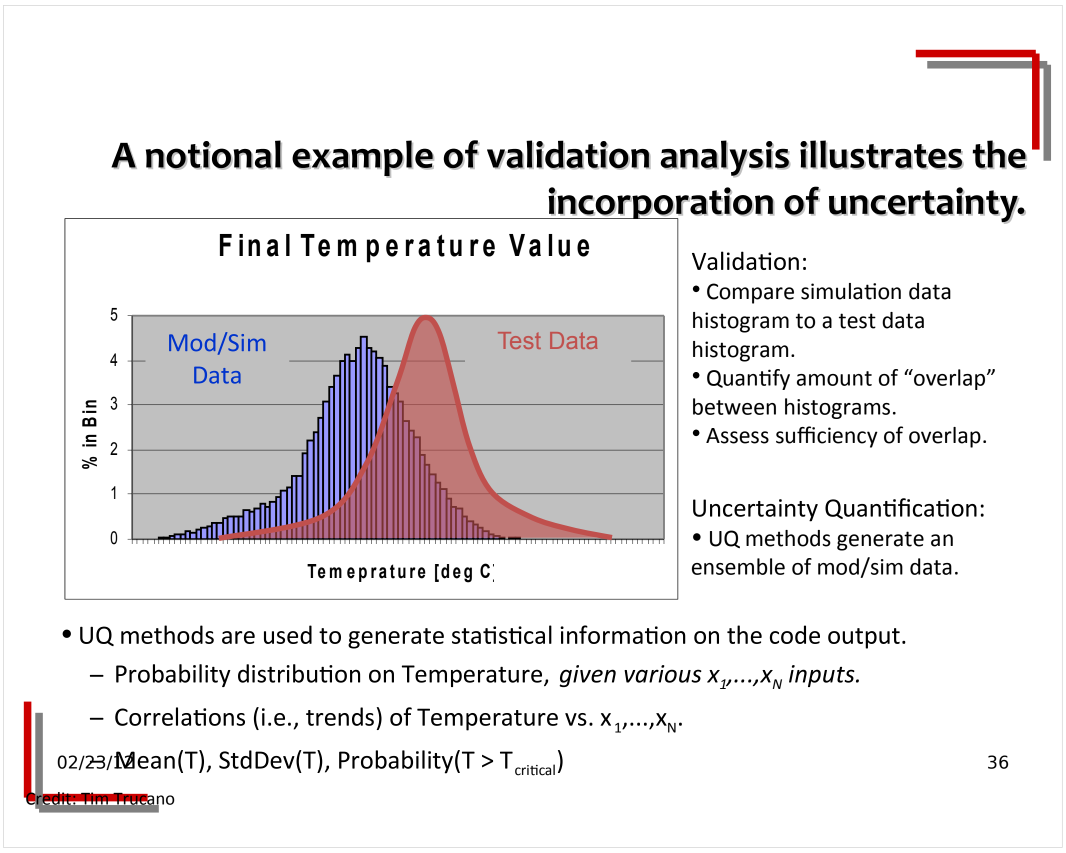
02/23/12

34

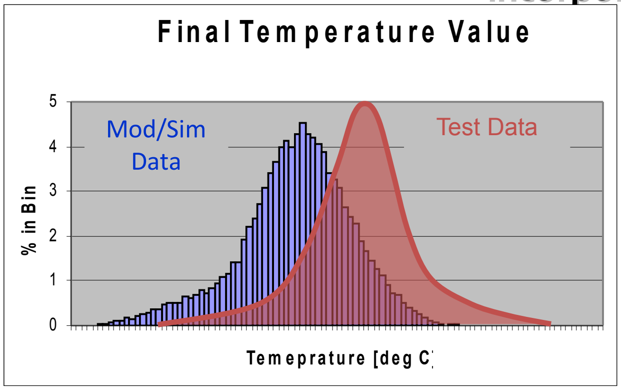
All known examples of lab verification exercises carried out by non-code-team members were very expensive and often failures.

Verification Testing Must Be a Team Ethic

- Discipline is required to keep a “clean” test suite – to keep all tests passing; “stop-the-line” mentality
- If only part of the team values verification, that part is always carrying the other developers
- Maintaining an automated verification test suite is probably necessary but *definitely not sufficient*
- Developers should be using verification tests interactively



A notional example of validation analysis illustrates the incorporation of uncertainty.



Validation:

- Compare simulation data histogram to a test data histogram.
- Quantify amount of "overlap" between histograms.
- Assess sufficiency of overlap.

Uncertainty Quantification:

- UQ methods generate an ensemble of mod/sim data.

- UQ methods are used to generate statistical information on the code output.
  - Probability distribution on Temperature, *given various $x_1,\ldots,x_N$ inputs.*
  - Correlations (i.e., trends) of Temperature vs. $x_1,\ldots,x_N$.
  - Mean(T), StdDev(T), Probability($T > T_{critical}$)

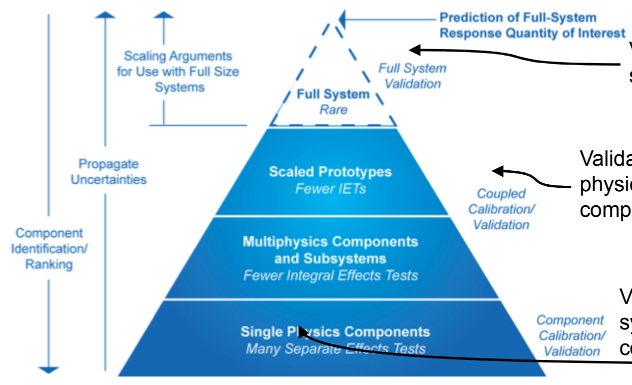
02/23/12 36

Credit: Tim Trucano

The eyeball norm is dead. Long live UQ!

Well, that is the day we are hoping for.

Complex problems require a hierarchical approach to validation.



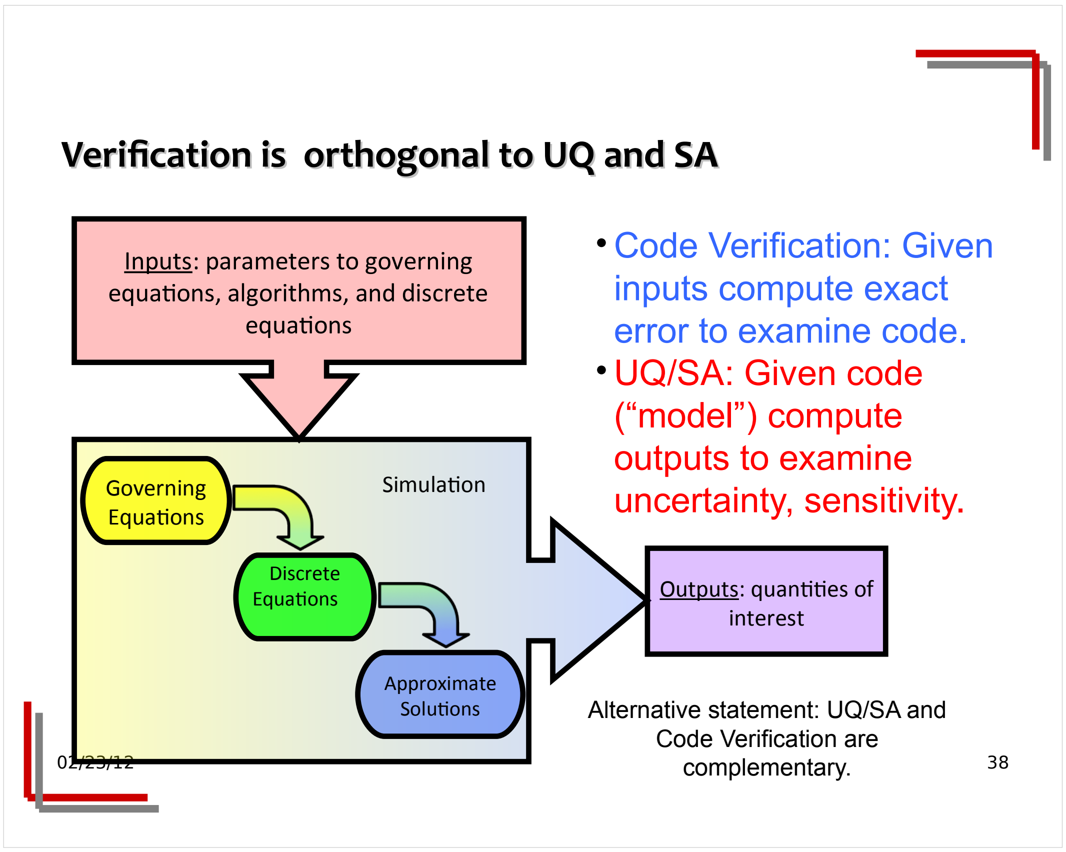
Prediction of Full-System Response Quantity of Interest
Scaling Arguments for Use with Full Size Systems
Full System Validation
Full System
Rare
Propagate Uncertainties
Scaled Prototypes
Fewer IETs
Coupled Calibration/ Validation
Component Identification/ Ranking
Multiphysics Components and Subsystems
Fewer Integral Effects Tests
Single Physics Components
Many Separate Effects Tests
Component Calibration/ Validation

Validation of “full-physics” systems is usually rare.

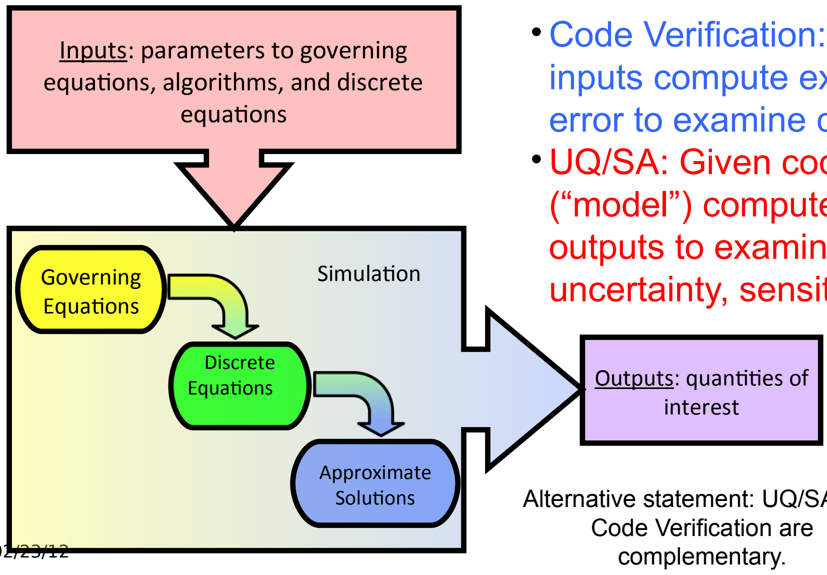
Validation of “coupled-physics” systems is often complex and challenging.

Validation of “single-physics” systems is the most common analysis.

Credit: D. Kothe et al., Consortium for Advanced Simulation of Light Water Reactors (CASL), 2010.



Verification is orthogonal to UQ and SA


Inputs: parameters to governing equations, algorithms, and discrete equations
Simulation
Governing Equations
Discrete Equations
Approximate Solutions
Outputs: quantities of interest

- Code Verification: Given inputs compute exact error to examine code.
- UQ/SA: Given code ("model") compute outputs to examine uncertainty, sensitivity.

Alternative statement: UQ/SA and Code Verification are complementary.

02/23/12 38

Governing equations includes BCs, ICs, and auxiliary models (material models)

UQ/SA does not say anything about how good your black box is – the black box is an input to UQ and SA.

Similarly, code verification does not say anything about different values of the inputs.

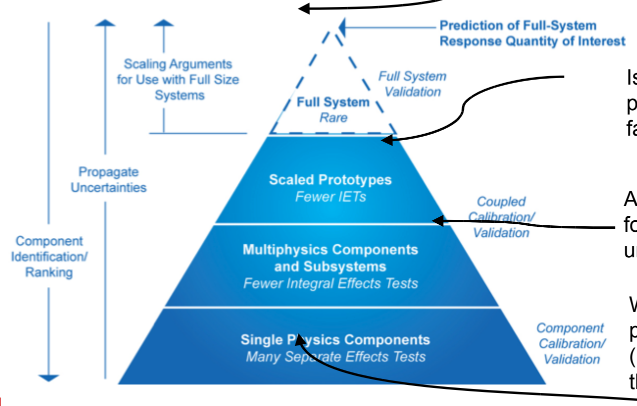
UQ, SA originated in fields where models were cheap, no deep knowledge embedded in them, very often empirical.

Applying UQ, SA to complex software systems is a new application area, and perhaps more thought is needed to identify the implications

In V&V, I say the model is the governing equations.

In UQ, SA, the model is the approximate solutions. More on the next slide.

The fundamental tension: find out as much as you can, but recognize you can't eliminate all risk


Prediction of Full-System Response Quantity of Interest
Full System Validation
Full System
Rare
Scaling Arguments for Use with Full Size Systems
Scaled Prototypes
Fewer IETs
Propagate Uncertainties
Coupled Calibration/ Validation
Component Identification/ Ranking
Multiphysics Components and Subsystems
Fewer Integral Effects Tests
Single Physics Components
Many Separate Effects Tests
Component Calibration/ Validation

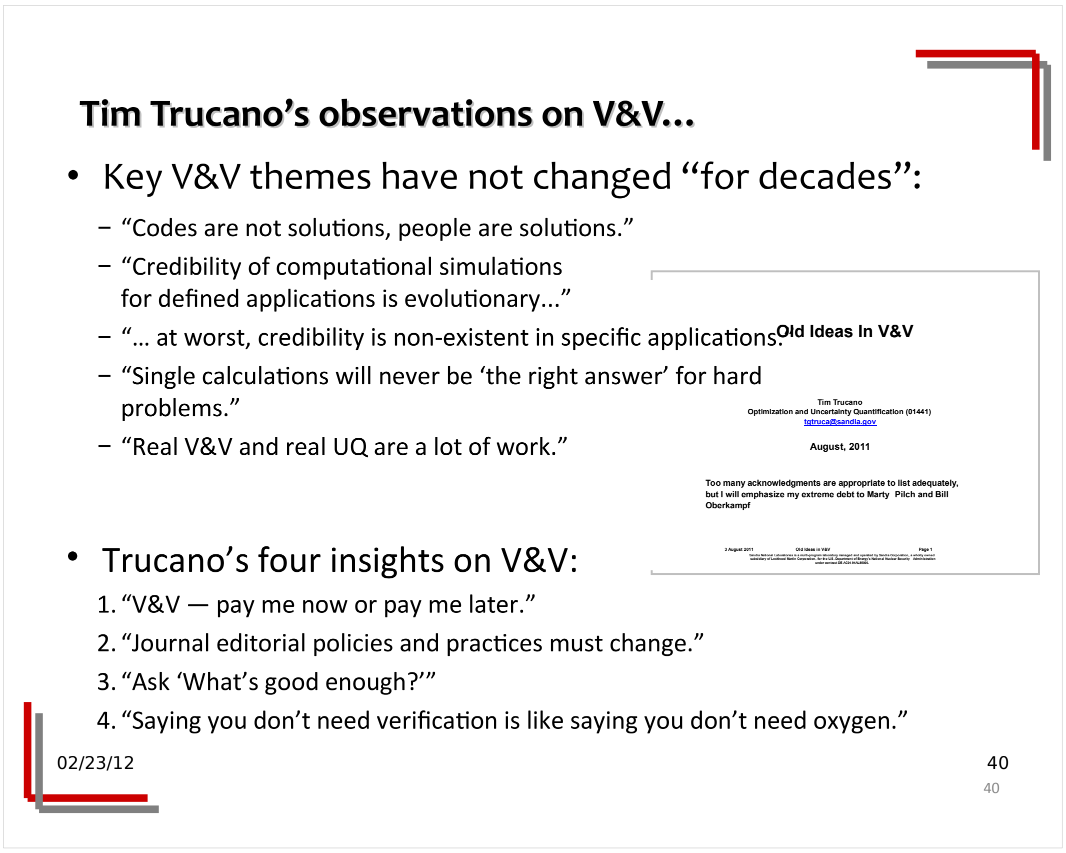
If we could validate here, we wouldn't need simulations.

Is there any evidence for the possibility of a catastrophic failure?

Assessments tell where to focus resources to reduce uncertainties.

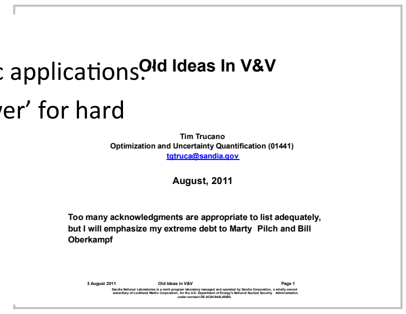
We can often make predictions with confidence (but it takes a lot more work than you might think.)

Image: Kothe et al., Consortium for Advanced Simulation of Light Water Reactors (CASL) proposal, 2010.



Tim Trucano's observations on V&V…

- Key V&V themes have not changed "for decades":
  - "Codes are not solutions, people are solutions."
  - "Credibility of computational simulations for defined applications is evolutionary..."
  - "… at worst, credibility is non-existent in specific applications."
  - "Single calculations will never be 'the right answer' for hard problems."
  - "Real V&V and real UQ are a lot of work."



Old Ideas In V&V

Tim Trucano
Optimization and Uncertainty Quantification (01441)
tgtruca@sandia.gov

August, 2011

Too many acknowledgments are appropriate to list adequately, but I will emphasize my extreme debt to Marty Pilch and Bill Oberkampf

3 August 2011 Old Ideas in V&V Page 1
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

- Trucano's four insights on V&V:
  1. "V&V — pay me now or pay me later."
  2. "Journal editorial policies and practices must change."
  3. "Ask 'What's good enough?'"
  4. "Saying you don't need verification is like saying you don't need oxygen."

02/23/12 40

40

1. **"Pay me now or pay me later"**
   - **High consequence computational science is ABOUT ANSWERS, NOT ABOUT INSIGHT (with due respect to George Box).**
1. **Journal editorial policies and practices must change**
   - **The iron law that journals don't publish V&V is an empirical fact.**
   - **Starting the "Journal of V&V" is not the solution.**
1. **Ask "What's good enough?"**
   - **Is the answer: "When I run out of energy and/or money?"**
1. "Saying you don't need verification is like saying you don't need oxygen."
   - **Standard argument:**
     1. **Show plot in which "calculation" and "experiment" appear to "agree."**
     2. **Logic: the calc/expt agreement could not possibly be that "good" if there were algorithm/code/numerical errors.**
     3. **Deduction: "I don't need verification."**
   - **This is an example of proof by intimidation/"vigorous hand-waving"/"eminent authority"/"reduction to the wrong problem"/etc.**

Final ideas: what should we provide....

A software infrastructure to support testing...

- Ability to run a large number of simulations repeatedly
- Ability to compare results with a baseline
- Ability to report pass/diff/fail
- Run on platforms of interest at regular intervals

Homework

- Describe on the blog:
  - which kind of testing procedure you adopted so far in your code
  - Which kind verification are you plan to use ?
  - Which kind of validation are you plan to use ?