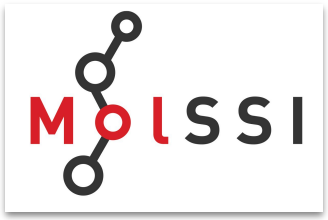
Sustainability definition

Theresa Windus
Iowa State University
Ames Laboratory
twindus@iastate.edu



What is sustainable software?

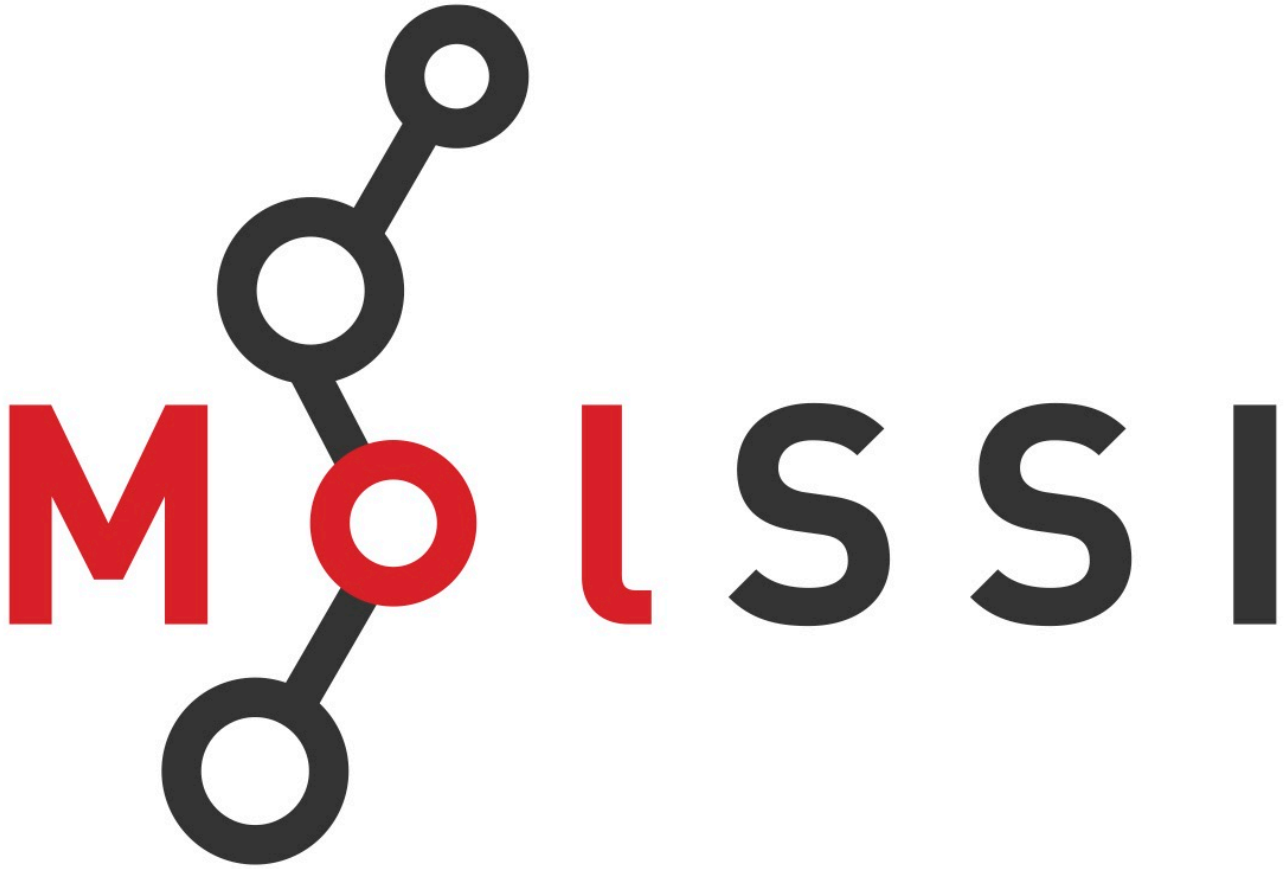
- Dan Katz blog 09/13/2016: "Here I offer a quick definition of sustainability in the context of software: **the capacity of the software to endure**. In other words, **sustainability means that the software will continue to be available in the future, on new platforms, meeting new needs**."
- Patricia Lago at WSSSPE4 with tweaks from Neil Chue Hong
  Sustainable software is software which is:
  – Easy to evolve and maintain
  – Fulfils its intent over time
  – Survives uncertainty
  – Supports relevant concerns (Political, Economic, Social, Technical, Legal, Environmental)

What is sustainable software?

- http://catalogue.pearsoned.co.uk/samplechapter/0321286081.pdf "Sustainable software development is a mindset (principles) and an accompanying set of practices that enable a team to achieve and maintain an optimal development pace indefinitely. "
- Robert Heine in https://www.energypedia-consult.com/en/blog/robert-heine/what-sustainable-software "The concept of sustainability is based on three pillars: the ecological, the economical and the social. This means that for a software to be sustainable, we must take all of its effects – direct and indirect – on the environment, the economy and the society into account. In addition, the entire life cycle of a software has to be considered: from planning and conception to programming, distribution, installation, usage and disposal."

What is sustainable software?

- Joost Visser in https://www.infoq.com/news/2018/04/sustainable-software-agile/ *"So for software, "sustainable" actually means "evolvable"."*
- Tom Dufour in https://www.infoq.com/news/2018/04/sustainable-software-agile/ *"To me, it is software that can be understood quickly and can be edited easily."*



The Molecular Sciences Software Institute

... a nexus for science, education, and cooperation for the global computational molecular sciences community.

Codes Are Developed and Used Globally

MolSSI

ECP
EXASCALE COMPUTING PROJECT

Codes Are Developed and Used Globally

These codes represent decades of development by thousands of programmers, and are used by hundreds of thousands of scientists worldwide.

MolSSI

ECP
EXASCALE COMPUTING PROJECT

Code Complexity and Historical Legacy

- CMS programs contain **millions of lines of hand-written code** and require **hundreds of programmers** to develop and maintain.
- Incredible **language diversity**: F77, F90, F95, HPF, C, C++, C++11, C++14,C++17, Python, perl, Javascript, etc.
- Incredible **algorithmic diversity**: structured and unstructured grids, dense and sparse linear algebra, graph traversal, fast Fourier transforms, MapReduce, and more.
- The packages have evolved in an ad hoc manner **over decades** because of the intricacy of the scientific problems they are designed to solve.

Rapidly Evolving Computing Hardware

- **Multi- and many-core architectures** are the norm, but many CMS codes are developed with limited view to parallel task management.
- Reduced-power solutions will also require improved **error recovery and checkpointing** at the software level – capabilities absent in nearly all CMS codes.
- Anticipated architectural innovations will yield **even greater hardware complexity** – more advanced accelerators, specialized computing cores, reconfigurable logic…
- Many CMS codes (especially for quantum chemistry) are limited to shared-memory paradigms and cannot yet take advantage of GPUs or **large-scale distributed-memory systems**.

Some examples of unsustainability

- **No plan for design** – no thought to modularity, separation of concerns, reusability, extensibility, or ease of use. Just get the job done.
- **Picking the wrong horse in the language/library race** – perl and Fortran examples; which on-node parallel library to use (OpenMP, OpenAcc, CUDA, Kokkos)
- **Programming too close to the hardware** – assembly as the extreme
- **Programming too abstractly** – can't get performance on any hardware
- **Inefficient usage of resources** – hardware and human
- **Lack of well defined interfaces** – multiple implementations that don't get used; FFT example

Some examples of unsustainability

- **Lack of code development oversite** – may have rules, but if not enforced…
- **Depending on other libraries** – especially if they are not sustainable; version control nightmare
- **No or very little documentation** – both user and developer
- **Inadequate testing** –Just get the job done; potential incorrect results for other users
- **Lack of user communication during development** –Might have a great idea, but if doesn't meet the end user need, then it hasn't served its purpose
- **Single point of failure** – Hit by a bus syndrome

Not all software should be sustained

- **Society has evolved past the use of the software** – software no longer meets a specific need
- **Training** – software will not be good the first time and is likely highly redundant
- **Prototype** – often is used as the final solution, but that is not its original purpose
- **Software should die gracefully** – Zach Mathew https://medium.com/building-freshbooks/great-software-isn-t-built-to-last-it-s-built-to-die-gracefully-594df9c3a470

Acknowledgments

- T. Daniel Crawford; the Board of Directors, and the wonderful MolSSI staff scientists;
- The NWChem and NWChemEx developers;
- The dozens of members of the CMS community who helped to develop the vision;
- NSF ACI-1547580;
- Exascale Computing Project