ExxonMobil

Collegeville Workshop 2020: Developer Productivity

The Art of Serving HPC Products to Business while they are still hot

a story of HPC DevOps

Vadim Dyadechko

Disclaimer (just in case)

Certain parts of the presentation may look to you:

- too obvious
- too general
- too specific

Well, the audience is too diverse, hopefully everyone will find something useful.

Some statements may sound over-simplified or arguable;
it was done on purpose,
to make the message unambiguous
and spark a discussion.

I understand that the right answer to most real-life questions is “It depends”;
unfortunately, this answer does not help to stay focused
and deliver quality products on time & on budget.

The opinions expressed here are solely my own
and do not express the views or opinions of my EMployer.

The weak link

HPC bottlenecks (in ascending order of severity):

- Instruction pipeline
- Memory bandwidth
- Network bandwidth
- Storage throughput
- Power source, heat sink
- People who develop, deploy, and run s/w

We, humans, are
the least scalable and the least reliable
component* of HPC (well, of any) technology.

*** as Fred Brooks pointed out, humans are not a resource…**

GettyImages-172194820

Coding: neither scales up nor accelerates

Frederick P. Brooks Jr., the head of IBM OS/360 project and the author of

The Mythical Man-Month (1975)
a.k.a. "The Bible of Software Engineering"

"Everybody quotes it,
some people read it,
and a few people go by it."
(Daniel Roth)

FAQ:
Q1. Do you quote Fred Brooks in every talk?
A1. Pretty much…
Q2. Did you read other books on s/w project management?
A2. What other books?

The Brooks' Law:

"Adding manpower to a late s/w project makes it later."
(complex s/w projects cannot be perfectly partitioned into independent tasks)

No Silver Bullet (1986):

"There is no … [s/w development] technology or … technique,
which … promises … order of magnitude improvement …
in productivity, in reliability, in simplicity."

Productivity: beyond coding

This year Collegeville Workshop focuses on developer productivity;
the theme suggests that the major s/w cost is due to development / coding.

While improving developers' productivity is important,
coding is just one part of the larger picture;
there are many other aspects of s/w technology
that constantly toll the budget and beg to be improved.

I strongly believe that
the bigger prize is in optimizing
the entire s/w product pipeline / ecosystem.

At the end of the day,
search through a larger space
always yields a more optimal result.

Productivity: beyond coding

One does not need a looking glass to spot a candidate for improvement.

There is a widespread misconception that s/w is ready for use
once the coding is completed,
and putting s/w to work is a trivial routine step
that somehow happens on its own.

Reality check:
the road from developer's work copy
to the end user
is long and rocky…

Murphy's Law: "whatever can go wrong, will go wrong"

"Things will go wrong in any given situation, if you give them a chance"

- build is broken due to the system / library update
- production cases fail after the new release
- recent trunk updates break downstream integration tests
- tests passing on developer's machine but failing on the production server
- today's results are different from the ones obtained yesterday
- seemingly identical inputs A and B yield different results
- new team member onboarding takes days
- the full build takes hours

These mishaps / mysteries / inefficiencies
are quite common;
and, **if left unchecked**, they start steadily
hog your time, drain resources, and destroy your reputation.

"A horse! A horse! My kingdom for a horse!" (Richard III)

"For want of a nail the shoe was lost.
For want of a shoe the horse was lost.
For want of a horse the rider was lost.
For want of a rider the message was lost.
For want of a message the battle was lost.
For want of a battle ***the kingdom was lost.***
And all for the want of a horseshoe nail."
(13th century A.D.)

GettyImages-1132749725

The old nursery rhyme below a perfect example of how **simple, invisible, everyday problems can derail a critical mission**:

- 1991: a Patriot missile missed the target due to a round-off error
- 1996: the f64 → i12 cast triggered self-destruction of Ariane 5 rocket
- 1999: NASA lost Mars Climate Orbiter due to the units disagreement

"Alice? Who ^@&%$@$ is Alice?" (Smokie/Gompie)

Lots of productivity / contingency / quality control issues
can be mitigated / eliminated
with proper DevOps processes, policies, and tools.

Wikipedia:

"DevOps is an agile and highly automated process
that combines software development (Dev) and IT operations (Ops)
to shorten the systems development life cycle
while delivering features, fixes, and updates frequently
in close alignment with business objectives."

Ultimate goal: fast delivery of high quality software to business (end user).

Frequently cited features:

- mindset on productivity and business needs
- highly automated processes
- frequent releases

DevOps: unite and conquer

(yet another) **DevOps manifest:**

- **non-coding tasks (building, testing, integration, deployment, etc.)**
  are critical steps of s/w product pipeline
- **non-coding tasks**
  are core responsibilities of the development team.

Everything else:
automation, short delivery times, mindset on productivity --
are just natural implications of
the **positive reinforcement mechanism** created by
consolidating development and operations in one hands.

Consolidation prevents diffusion of task ownership
(nothing falls between the cracks)
and encourages wide range of optimizations,
now internal to one team: …

DevOps: freedom to operate

… and encourages wide range of optimizations, now internal to one team:

- **custom automation**
  prevents otherwise inevitable human errors,
- **systematic testing** and investment in quality code
  discovers bugs before they surprise end users,
- **power to customize** development / testing / production **environment**s
  greatly simplifies the configuration and integration tasks;

gradually making your product more maintainable, testable, deployable.

Formal transfer of responsibilities does not change anything unless it grants

- **freedom to operate**,
- freedom to select the right tools,
- freedom to customize testing and production environments.

Red pill? ← make your choice → **Blue pill?**

GettyImages-1140916922

DevOps: do it yourself

Any custom automation requires
intimate knowledge of product and environment.

Each HPC system is unique and is far from being static.

While many of DevOps tasks and activities fall into the category of IT operations,
it is critical not to delegate them outside the development team.

Involvement of developers in daily operations
has enormous positive impact on the quality and usability
of DevOps components;
the best pieces of s/w around us were created for personal use:
Unix, C, Perl, TeX, Git.

DevOps: non-technical challenges

IT operations (system, build, integration, automation):

- **frequently under-appreciated**,
  mistakenly viewed to be secondary to coding
- **have hard-to-measure deliverables**
- **suffer from negative visibility** ("sysadmin curse"):
  - smooth operations are usually taken for granted
  - occasional screw-ups generate negative feedback
- **require conservative attitude**
- **require highly scattered unstructured subject knowledge**
  "They don't teach this stuff at school." (Alan Wild, HPC Systems)

Constant pressure to provide immediate solution
rather than a sustainable / manageable / scalable one

DevOps: technical challenges

Moving target:

- production environment is constantly evolving,
  DevOps layer has to catch up promptly

Robustness:

- mishaps are part of real life,
  DevOps layer should embed full diagnostics
  and have a good recovery potential

Application teams do not have root privileges:

- limits the choice of tools:

  `crontab, ssh, rsync, git, make, qsub, sh, perl, python`

- requires fair amount of tinkering for finding robust workarounds

"Failure is Not an Option" (Apollo 13)

Take full control of routine tasks,
leave developer / tester / integrator no chance for a mistake.

Automation is not a rocket science,
but it takes significant effort to fit the moving parts,
make them robust and user-friendly,
and keep them running smoothly ever after

Automation requires discipline
and imposes constraints on the workflows and the product itself

- automation does not tolerate
  - low quality components
- automation is incompatible with
  - complex workflows
  - rich configuration space
  - bureaucracy

"It's the little differences" (Pulp Fiction)

Keep it simple: settle for fit-for-purpose solutions

- focus on essentials
- minimize nice-to-haves
- routine tasks should require zero effort, robust defaults are vital

Be defensive, shield your project from surprise factors:

- version all the components
- stage all 3rd-party dependencies

Quality enablers (think of UX):

- short build times (5min max for a full build)
- short test times (5min max for the entire test suite)
- smart regression comparators (minimize false-positives)
- keep your workflows simple and transparent
- collect as much diagnostics as feasible

Closing remarks

- **Recognize the importance of non-coding tasks.**
- **Take control of little things before they take control of you.**
- **Take a red pill: do it yourself;**
  you do not need fancy tools or 3rd party services,
  everything can be done using standard Linux utilities.
- **There is no free lunch: allocate adequate resources for DevOps.**

feedback, comments are welcome: <vdyadechko@gmail.com>