Rekayasa Perangkat Lunak untuk Ilmu Komputasi:

Dulu, Sekarang, Masa Depan

Arne N. Johanson
Solusi Pemasaran XING GmbH

Wilhelm Hasselbring
Universitas Kiel

Editor:
Jeffrey Carver, carver@cs.ua.edu ; Damian Rouson, damian@sourceryinstitute.org

Meskipun semakin pentingnya eksperimen in silico untuk proses penemuan ilmiah, praktik rekayasa perangkat lunak canggih jarang diadopsi dalam ilmu komputasi. Untuk memahami penyebab yang mendasari situasi ini dan untuk mengidentifikasi cara memperbaikinya, kami melakukan survei literatur tentang praktik rekayasa perangkat lunak dalam ilmu komputasi. Kami mengidentifikasi 13 karakteristik kunci yang berulang dari perangkat lunak ilmiah pengembangan yang merupakan hasil dari sifat tantangan ilmiah, keterbatasan komputer, dan lingkungan budaya pengembangan perangkat lunak ilmiah. Temuan kami memungkinkan kami untuk menunjukkan kekurangan dari pendekatan yang ada untuk menjembatani kesenjangan antara rekayasa perangkat lunak dan ilmu komputasi dan untuk memberikan pandangan tentang arah penelitian yang menjanjikan yang dapat berkontribusi untuk memperbaiki situasi saat ini.

Dengan kemampuan komputer modern yang terus meningkat, eksperimen in silico menjadi lebih kompleks dan memainkan peran yang lebih penting dalam proses penemuan ilmiah.[1]Sebagai konsekuensinya, kompleksitas dan masa pakai perangkat lunak ilmiah semakin meningkat, serta kebutuhan agar keluarannya dapat direproduksi dan diverifikasi. Hal ini meningkatkan pentingnya menerapkan praktik rekayasa perangkat lunak yang baik dalam pengembangan perangkat lunak ilmiah untuk menjamin hasil ilmiah yang andal dan akurat. Namun, survei menunjukkan bahwa metode rekayasa perangkat lunak canggih jarang diadopsi dalam ilmu komputasi.[2,3]Untuk memahami penyebab yang mendasarinya dan untuk mengidentifikasi cara untuk memperbaiki situasi saat ini, dalam artikel ini, kami melakukan survei literatur tentang rekayasa perangkat lunak dalam ilmu komputasi dan mengidentifikasi karakteristik utama yang unik untuk pengembangan perangkat lunak ilmiah.

Untuk memberikan dasar untuk survei kami, kami menguraikan sejarah perkembangan hubungan antara disiplin ilmu rekayasa perangkat lunak dan ilmu komputasi. Hubungan ini, sebagian besar, dicirikan oleh isolasi antara dua disiplin ilmu yang mengakibatkan

Diterbitkan bersama oleh IEEE CS dan AIP
1521-9615/18/$33 ©2018 IEEE

produktivitas dan kredibilitas krisis dalam ilmu komputasi. Atas dasar sudut pandang ini, kami meninjau publikasi tentang studi kasus dan survei yang dilakukan di antara ilmuwan komputasi untuk mengidentifikasi 13 karakteristik utama pengembangan perangkat lunak ilmiah yang menjelaskan mengapa teknik rekayasa perangkat lunak canggih kurang diadopsi dalam ilmu komputasi. Temuan survei literatur kami memungkinkan kami untuk mengidentifikasi kekurangan dari pendekatan yang ada untuk menjembatani kesenjangan antara rekayasa perangkat lunak dan ilmu komputasi dan untuk memberikan pandangan tentang arah penelitian yang menjanjikan yang dapat berkontribusi untuk memperbaiki situasi saat ini.

PENGEMBANGAN SEJARAH TEKNIK PERANGKAT LUNAK DAN ILMU KOMPUTASI

Ketika insinyur perangkat lunak mulai memeriksa praktik pengembangan perangkat lunak dalam ilmu komputasi, mereka melihat "jurang yang lebar"[4]antara bagaimana kedua disiplin ini memandang pengembangan perangkat lunak. Stuart Faulk dan rekan-rekannya[5]menggambarkan jurang antara dua subjek ini menggunakan alegori yang menggambarkan ilmu komputasi sebagai pulau terpencil yang telah dijajah tetapi kemudian ditinggalkan selama beberapa dekade:

> *Pengunjung yang kembali (insinyur perangkat lunak) menemukan penghuninya (programmer ilmiah) tampaknya berbicara dalam bahasa yang sama, tetapi komunikasi—dan dengan demikian kolaborasi—hampir tidak mungkin; teknologi, budaya, dan semantik bahasa itu sendiri telah berkembang dan beradaptasi dengan keadaan yang tidak diketahui oleh penjajah asli.*

Fakta bahwa kedua budaya ini “dipisahkan oleh bahasa yang sama” menciptakan kesenjangan komunikasi yang menghambat transfer pengetahuan di antara mereka. Akibatnya, praktik rekayasa perangkat lunak modern jarang digunakan dalam ilmu komputasi.

Asal Usul Jurang

Asal usul keretakan antara ilmu komputasi dan rekayasa perangkat lunak dapat ditelusuri kembali sejauh awal komputasi modern di tahun 1940-an. Pada saat itu, "komputasi ilmiah" adalah sebuah pleonasme: komputer digital elektronik diciptakan semata-mata untuk memecahkan masalah matematika yang kompleks untuk kemajuan sains dan teknik.[6]Ketika disiplin ilmu komputer muncul pada akhir 1950-an dan awal 1960-an, ia berjuang untuk membedakan dirinya dari teknik elektro dan matematika terapan — disiplin yang secara tradisional terlibat dalam "studi komputer." [7]Untuk membedakan dirinya dari disiplin terapan ini, ilmu komputer menciptakan "stigma dari semua hal 'terapan'" dan ditujukan untuk umum dalam semua metode dan tekniknya.[8]Pendekatan ini seharusnya memastikan bahwa “inti ilmu komputer [...] akan tetap menjadi bidangnya sendiri, di depan, dan terpisah dari spesialis domain aplikasi.”[8]Keterasingan pertama ilmu komputer dari bidang yang kemudian menjadi ilmu komputasi ini diadopsi dan bahkan diperkuat oleh rekayasa perangkat lunak.

Syarat *rekayasa Perangkat Lunak*—dan subdisiplin yang sesuai dari *ilmu Komputer*—dilembagakan oleh sebuah konferensi dengan judul tersebut di atas yang diselenggarakan oleh NATO pada tahun 1968.[9]
Bahwa NATO adalah sponsor konferensi ini menandai jarak relatif dari rekayasa perangkat lunak dari komputasi dalam konteks akademis. Persepsinya adalah bahwa sementara kesalahan dalam aplikasi pemrosesan data ilmiah mungkin merupakan "kerepotan", semuanya dapat ditoleransi. Sebaliknya, kegagalan dalam sistem militer mission-critical mungkin menelan korban jiwa dan sejumlah besar uang.[6]

Berdasarkan sikap ini, rekayasa perangkat lunak—seperti halnya ilmu komputer secara keseluruhan—bertujuan untuk umum dalam metode, teknik, dan prosesnya dan berfokus hampir secara eksklusif pada bisnis dan perangkat lunak yang disematkan.[10]Karena idealitas umum ini, pertanyaan tentang bagaimana ilmuwan komputasi secara khusus harus mengembangkan perangkat lunak mereka dengan cara yang dirancang dengan baik mungkin akan membingungkan seorang insinyur perangkat lunak, yang jawabannya mungkin: "Yah, sama seperti perangkat lunak aplikasi lainnya."

Untuk beberapa waktu, koeksistensi ilmu komputasi dan rekayasa perangkat lunak dalam ketidaktahuan relatif satu sama lain berlanjut tanpa gangguan yang lebih besar. Satu pengecualian penting

dari ini adalah makalah dari Les Hatton dan Andy Roberts[11] di mana mereka memeriksa keakuratan lebih dari 15 perpustakaan perangkat lunak komersial besar dari algoritma numerik. Temuan mereka tidak sesuai dengan asumsi khas ilmuwan komputasi bahwa perangkat lunak mereka akurat dengan ketepatan aritmatika mesin. Hatton dan Roberts menemukan bahwa, dalam sampel mereka, perbedaan numerik antara hasil yang diharapkan dan yang dihitung meningkat sekitar 1 persen per 4.000 baris kode (LOC). Namun, pada awalnya, hasil ini tidak menemukan gaung yang lebih besar di kedua komunitas.

Krisis Produktivitas

Secara umum, ilmuwan komputasi tidak melihat alasan untuk mengkhawatirkan kualitas perangkat lunak mereka. Sikap ini mulai berubah kira-kira 10 tahun yang lalu, ketika semakin banyak kekurangan mengenai produktivitas pengembangan perangkat lunak ilmiah menjadi jelas. Kekurangan-kekurangan ini, yang menyebabkan beberapa orang berbicara tentang "kemacetan produktivitas"[5] dalam ilmu komputasi, sebagian besar terungkap oleh dua perkembangan paralel:

- pertemuan batas kecepatan clock untuk prosesor single-core dan, dengan itu, pengenalan multicore dan heterogen, sistem komputasi terdistribusi; dan
- integrasi semakin banyak efek ke dalam simulasi ilmiah yang mengatur perilaku sistem yang diteliti.

Perangkat lunak ilmiah memiliki umur yang cukup panjang (dari tahun ke dekade) karena sering kali merangkum akumulasi pengetahuan dan upaya para ilmuwan atau kelompok penelitian. Dengan demikian, biasanya hidup lebih lama dari perangkat keras komputasi yang awalnya dirancang. Selama lebih dari dua dekade, ini bukan masalah karena dapat diterima begitu saja bahwa setiap generasi baru mikroprosesor akan sangat meningkatkan kinerja perangkat lunak para ilmuwan tanpa modifikasi kode sumber yang lebih besar. Dengan ditemukannya batas kecepatan clock untuk prosesor single-core sekitar tahun 2004, perkembangan ini terhenti.[12]

Untuk mencapai lebih banyak kekuatan pemrosesan, perancang chip mulai menskalakan jumlah inti komputasi prosesor. Pada awalnya, ini hanya direalisasikan untuk CPU komputer (era multicore). Langkah terbaru adalah membantu CPU multicore tersebut dengan menyediakan perangkat akselerator eksternal dengan jumlah core komputasi yang lebih besar (berdasarkan urutan besarnya!) yang terinspirasi oleh desain prosesor grafis modern (era sistem heterogen). Untuk memanfaatkan kekuatan yang disediakan oleh sistem multicore dan terutama heterogen untuk simulasi yang semakin lebih rinci, ilmuwan komputasi sekarang menghadapi tantangan untuk mengadaptasi perangkat lunak mereka untuk mengeksploitasi paralelisme pada tingkat granularity yang semakin baik. Dorongan cepat ke mesin paralel terdistribusi besar-besaran di akhir 1990-an menyebabkan masalah produktivitas yang serupa. Proses ini mengungkapkan kurangnya pengetahuan tentang rekayasa perangkat lunak di antara para ilmuwan, yang mengakibatkan pemeliharaan yang buruk dari "kode" mereka. Kurangnya pemeliharaan ini menghambat kemampuan para ilmuwan untuk berhasil menskalakan simulasi mereka melalui adaptasi ke arsitektur perangkat keras baru.[13]

Perkembangan kedua yang mencegah ilmuwan komputasi mengabaikan teknik rekayasa perangkat lunak modern jika mereka ingin tetap—atau menjadi lagi—produktif terkait dengan kompleksitas model mereka. Agar komputasi memenuhi perannya sebagai "pilar ketiga penyelidikan ilmiah,"[14] ilmuwan harus meningkatkan kemampuan prediksi model mereka dengan mengintegrasikan lebih banyak efek ilmiah ke dalamnya. Hal ini menyebabkan kebutuhan untuk menggabungkan atau bahkan mengintegrasikan kontribusi dari tim ilmuwan multidisiplin ke dalam satu aplikasi simulasi. Seperti perangkat lunak ilmiah sebelumnya dikembangkan oleh tim kecil ilmuwan terutama untuk penelitian mereka sendiri, modularitas, pemeliharaan, dan koordinasi tim sering dapat diabaikan tanpa dampak yang besar. Pergeseran menuju tim interdisipliner yang lebih besar menjadikan aspek pengembangan perangkat lunak yang sering diabaikan ini penting bagi para ilmuwan dan, sekali lagi, memperlihatkan kesenjangan pengetahuan di antara mereka yang hanya dapat diatasi dengan terlibat dalam dialog dengan rekayasa perangkat lunak.[1]

Krisis Kredibilitas

Tantangan mengenai kualitas perangkat lunak ilmiah tidak hanya menyebabkan penurunan kinerja pengembangan tetapi juga mengganggu kredibilitas hasilnya. Aspek ini menjadi sangat penting karena dampak sosial dari simulasi komputer telah berkembang belakangan ini, yang

dapat dicontohkan oleh apa yang disebut skandal "Climategate". Skandal itu meletus setelah peretas membocorkan korespondensi email para ilmuwan dari Unit Penelitian Iklim di Universitas East Anglia tidak lama sebelum Konferensi Perubahan Iklim Perserikatan Bangsa-Bangsa 2009. Sementara tuduhan bahwa data dipalsukan untuk konferensi ini ternyata tidak berdasar, email tersebut mengungkap kurangnya keterampilan pemrograman di antara para peneliti dan memaparkan kepada khalayak publik yang besar praktik yang diterapkan secara luas dalam ilmu iklim untuk tidak merilis kode dan data simulasi bersama dengan publikasi yang sesuai.[15]Ini sendiri, tentu saja, cukup untuk melemahkan pekerjaan para ilmuwan, karena kemampuan prediksi simulasi hanya sebaik kualitas kode mereka.[11]dan kode mereka bahkan tidak tersedia untuk peer review—belum lagi review publik.

Dalam komunitas ilmiah, Climategate memprakarsai debat tentang reproduktifitas hasil komputasi yang juga menarik perhatian komunitas rekayasa perangkat lunak (kami membahas debat ini nanti).

Menjembatani kesenjangan

Baik krisis produktivitas maupun kredibilitas memperjelas bahwa koeksistensi yang terisolasi dari ilmu komputasi dan rekayasa perangkat lunak tidak dapat berlanjut. Masalah dengan pemrograman dan desain perangkat lunak ilmiah tidak boleh diabaikan sebagai kerumitan lagi agar ilmu komputasi maju dan menepati janjinya sebagai paradigma ketiga untuk penemuan ilmiah. Meskipun peristiwa dekade terakhir telah memulai dialog antara rekayasa perangkat lunak dan ilmu komputasi, kemajuan masih lambat. Misalnya, Jed Brown dan rekan-rekannya[16]berikan deskripsi yang cukup sarkastis tentang status terkini perangkat lunak ilmiah dan praktik pengembangan terkait: bayangkan perangkat lunak ilmiah sebagai browser web tanpa kotak entri URL—Anda memasukkan alamat web ke dalam file konfigurasi. Peramban dapat menggunakan http atau https tetapi tidak keduanya secara bersamaan, yang dikendalikan oleh sakelar #ifdef yang mengharuskan Anda untuk mengkompilasi ulang peramban dengan versi kedua dari terakhir dari kompiler Fortran77 oleh vendor tertentu. Pada prinsipnya, Anda dapat mengubah semua itu karena perangkat lunaknya adalah open source, tetapi, sayangnya, pengembangan bersifat pribadi dan Anda harus mengajukan permohonan untuk diberikan akses ke kode sumber, yang hanya akan Anda peroleh jika niat Anda sesuai dengan itu. dari pengembang utama.

Sementara pilihan desain seperti itu tampak tidak masuk akal bagi kami untuk aplikasi modern, Brown dan rekan menyimpulkan bahwa mereka "mewakili status quo dalam banyak paket perangkat lunak ilmiah" dan sering "dibela dengan keras."[16]Namun, ketidakpercayaan ilmu komputasi terhadap teknik rekayasa perangkat lunak modern tidak sepenuhnya tidak berdasar: karena rekayasa perangkat lunak ditujukan untuk umum dalam semua metode dan prosesnya, ia mengabaikan tuntutan unik dari ilmu komputasi.[10]Oleh karena itu, para ilmuwan terlalu sering mengalami penawaran metodologis rekayasa perangkat lunak sebagai penuh dengan "kerumitan yang tidak disengaja" alih-alih membantu.[17]Oleh karena itu, ketidakpercayaan dan prasangka masih sering ditemukan di kedua sisi “jurang perangkat lunak” yang selama ini belum tertutup kembali.[18]

Untuk memahami pendekatan mana yang mungkin cocok untuk menjembatani kesenjangan antara dua disiplin ilmu, kita harus memeriksa dengan cermat karakteristik pengembangan perangkat lunak ilmiah dan memperhatikan persyaratan khusus dari ilmu komputasi secara serius. Hanya jika kita—dari sudut pandang rekayasa perangkat lunak—meninggalkan "stigma dari semua hal 'terapan'"[8]dan mengakhiri perjuangan tanpa syarat untuk umum yang tidak melakukan ilmu komputasi, keadilan apa pun yang dapat kita harapkan untuk memperbaiki situasi saat ini.

KARAKTERISTIK PENGEMBANGAN PERANGKAT LUNAK ILMIAH

Literatur tentang karakteristik pengembangan perangkat lunak ilmiah muncul hanya setelah artikel berpengaruh oleh Douglass Post dan Lawrence Votta,[19]yang menemukan bahwa disiplin ilmu komputasi yang relatif baru masih "sangat tidak matang." Topik diselidiki terutama dengan melakukan studi kasus dan beberapa studi survei. Meninjau dan mengintegrasikan pengamatan dari studi yang berbeda ini memungkinkan kita untuk mengurangi risiko utama yang umumnya terkait dengan studi kasus: kurangnya generalisasi. Menggabungkan dan mengkontraskan temuan-temuan dari

beberapa studi di lingkungan yang berbeda memungkinkan untuk mengidentifikasi serangkaian karakteristik yang mungkin melekat pada pengembangan perangkat lunak ilmiah secara umum. Meskipun sebagian besar literatur tentang praktik pengembangan perangkat lunak dalam ilmu komputasi berasal dari tahun 2006 hingga 2009, pengamatan yang dilakukan tampaknya benar hingga saat ini.[20]

Makalah yang dimasukkan ke dalam survei literatur kami diidentifikasi dengan menanyakan database seperti ACM Digital Library (dl.acm.org), IEEE Computer Society Digital Library (www.computer.org/csdl), dan Google Scholar (scholar.google. com). Selain itu, kami mencari jurnal yang kami harapkan memiliki minat tertentu, seperti*Komputasi dalam Sains & Teknik*, dan prosiding konferensi dan lokakarya seperti Konferensi Internasional tentang Rekayasa Perangkat Lunak dan Lokakarya Internasional tentang Rekayasa Perangkat Lunak untuk Ilmu dan Rekayasa Komputasi mulai tahun 2005 dan seterusnya. Beberapa makalah yang disarankan oleh rekan-rekan atau diidentifikasi oleh referensi dari artikel lain. Batasan dari strategi ini adalah bahwa kami harus bergantung pada sejumlah kata kunci dalam kueri basis data kami. Kami mencoba mengurangi risiko ini dengan memvariasikan frasa pencarian dan, misalnya, menggunakan sinonim yang berbeda (seperti komputasi ilmiah untuk ilmu komputasi dan sebagainya).

Karena berbagai perangkat lunak ilmiah dan aplikasinya besar,[21]ilmuwan komputasi tidak membentuk kelompok yang homogen. Para ilmuwan mengembangkan perangkat lunak mulai dari skrip untuk analisis data skala kecil hingga simulasi multifisika gabungan kompleks yang dijalankan pada perangkat keras kelas atas. Dalam survei literatur kami, kami sebagian besar fokus — meskipun tidak secara eksklusif — pada kelompok terakhir, yang membentuk komunitas komputasi kinerja tinggi (HPC). Alasan untuk mengarahkan perhatian kita pada kelompok ini adalah karena kelompok ini paling terpengaruh oleh krisis produktivitas dan kredibilitas yang digambarkan di bagian sebelumnya.

Dari survei literatur kami, kami mengidentifikasi 13 karakteristik kunci berulang dari pengembangan perangkat lunak ilmiah yang dapat dibagi menjadi tiga kelompok:

- karakteristik karena sifat tantangan ilmiah;
- karakteristik karena keterbatasan komputer; dan
- karakteristik karena lingkungan budaya pengembangan perangkat lunak ilmiah.

Dalam subbagian berikut, kami merinci temuan kami berkaitan dengan ketiga kelompok ini dan menjelaskan bagaimana pendekatan rekayasa perangkat lunak untuk ilmu komputasi dapat mempertimbangkan karakteristik ini.

Karakteristik Karena Sifat Tantangan Ilmiah

Semua karakteristik pengembangan perangkat lunak dalam ilmu komputasi yang kami sebutkan di sini dihasilkan dari fakta bahwa perangkat lunak ilmiah merupakan bagian integral dari proses penemuan. Ketika Anda mengembangkan perangkat lunak untuk mengeksplorasi fenomena yang sebelumnya tidak diketahui, sulit untuk menentukan secara pasti apa yang harus dilakukan perangkat lunak tersebut, bagaimana seharusnya keluarannya terlihat, dan bagaimana melanjutkan selama pengembangannya.

Persyaratan tidak diketahui sebelumnya

Dalam sains, perangkat lunak digunakan untuk membuat penemuan baru dan untuk memajukan pemahaman kita tentang dunia. Karena perangkat lunak ilmiah tertanam dalam ke dalam proses eksplorasi, Anda tidak pernah tahu ke mana perkembangannya akan membawa Anda. Dengan demikian, sulit untuk menentukan persyaratan untuk perangkat lunak semacam ini di muka seperti yang diminta oleh proses perangkat lunak tradisional. Dengan demikian, sebagian besar persyaratan—kecuali yang paling jelas tingkat tinggi—ditemukan hanya selama proses pengembangan dalam proses yang sangat berulang.[21]Alasan untuk ini adalah bahwa sementara teori ilmiah yang mendasari sudah mapan di sebagian besar proyek perangkat lunak ilmiah, tidak jelas sebelumnya bagaimana teori ini dapat diterapkan pada masalah spesifik yang dihadapi.[22]Ketika satu-satunya tujuan proyek adalah untuk lebih memahami domain, hasil pasti dari proyek tersebut—menurut definisi—tidak diketahui.

Tujuan utama pengembangan perangkat lunak dalam ilmu komputasi bukanlah untuk menghasilkan perangkat lunak tetapi untuk memperoleh hasil ilmiah. Untuk alasan ini, tidak mengherankan bahwa programmer ilmiah

mengatakan bahwa mereka "memprogram secara eksperimental."[23]Model ilmiah serta implementasinya diperlakukan sebagai teori yang berkembang untuk menguji hipotesis tertentu.[24]Jadi, wawasan yang diperoleh dari satu versi perangkat lunaklah yang menentukan apa yang dibutuhkan untuk versi berikutnya dalam iterasi yang relatif singkat.[25]Sifat berulang dari proses pengembangan perangkat lunak ilmiah tidak, oleh karena itu, menunjukkan kurangnya keterampilan pemrograman di antara para ilmuwan melainkan mencerminkan pemahaman yang berkembang tentang persyaratan sebagai perangkat lunak berkembang.[26]

Bahwa para ilmuwan jarang melihat desain dan analisis persyaratan sebagai langkah yang berbeda dalam pengembangan perangkat lunak[27]sebagian karena fakta bahwa banyak aplikasi ilmiah dimulai sebagai proyek yang sangat kecil dan mulai tumbuh hanya atas dasar keberhasilan ilmiah mereka.[28]Jadi, persyaratan untuk versi pertama perangkat lunak sering kali berasal dari pengalaman seorang ilmuwan dan biasanya tidak dijelaskan oleh orang tersebut. Jika perangkat lunak terbukti bermanfaat bagi komunitas yang lebih luas, anggotanya cenderung menyarankan fitur untuk dimasukkan ke dalam perangkat lunak dan, dengan demikian, menambahkan persyaratan. Persyaratan ini, bagaimanapun, tidak dijelaskan dengan cara yang cukup rinci untuk membentuk dasar dokumen kontrak seperti yang disyaratkan dalam proses rekayasa perangkat lunak yang mapan.[29]Jika organisasi sponsor menuntut dokumentasi desain dan proses analisis persyaratan, para ilmuwan biasanya menulis dokumen-dokumen ini setelah perangkat lunak hampir selesai.[27]

Verifikasi dan validasi itu sulit dan sangat ilmiah

Dalam konteks perangkat lunak ilmiah, verifikasi berarti menunjukkan bahwa penerapan algoritma dan persamaan yang terkandung di dalamnya benar. Jadi, verifikasi murni berkaitan dengan konstruksi teoretis. Sebaliknya, validasi berarti menunjukkan bahwa perangkat lunak dan model matematika yang diwakilinya berhasil menangkap semua efek ilmiah yang relevan dengan benar. Oleh karena itu, validasi harus memastikan bahwa keluaran perangkat lunak cukup sesuai dengan pengamatan dari dunia nyata.[22] Verifikasi dan validasi menimbulkan tantangan serius di semua bidang pengembangan perangkat lunak tetapi sangat sulit dalam ilmu komputasi karena kurangnya oracle uji, karena lingkungan perangkat keras terdistribusi yang kompleks dengan dukungan alat yang tidak memadai, dan karena penilaian para ilmuwan terhadap perangkat lunak secara umum.[30]

Validasi sangat menantang karena para ilmuwan sering kekurangan data pengamatan untuk membandingkan hasil model mereka—bagaimanapun juga, mereka menggunakan simulasi justru karena subjek yang ada “terlalu kompleks, terlalu besar, terlalu kecil, terlalu berbahaya, atau terlalu mahal untuk dijelajahi di dunia nyata."[21] Tetapi bahkan jika pengamatan tersedia, mereka masih bisa tidak lengkap atau salah dan mereka tidak pernah meluas ke masa depan, yang menjadi perhatian banyak simulasi.[27]Terakhir, jika penyimpangan dari pengamatan terjadi, sulit untuk melacak penyebabnya, yang dapat terletak pada dimensi berikut, atau kombinasinya:[22]

- model matematis realitas bisa jadi tidak mencukupi, artinya aspek ilmiah salah;
- algoritme yang digunakan untuk mendiskritkan masalah matematika mungkin tidak memadai (misalnya, memiliki masalah stabilitas);
- implementasi algoritma bisa salah karena kesalahan pemrograman; atau
- ketika model untuk proses fisik yang berbeda digabungkan, kesalahan mungkin menyebar melalui sistem sehingga menjadi sulit untuk melacak penyebab kesalahan.

Oleh karena itu, pemeriksaan ekstensif terhadap kode dan model ilmiah harus dilakukan selama pengembangan, yang menyoroti pentingnya verifikasi yang tepat.[31]

Untuk verifikasi, ilmuwan komputasi dapat mengandalkan metode pengujian yang telah ditetapkan (misalnya, pengujian unit dan pernyataan). Selain pendekatan tradisional ini, mereka menggunakan pemeriksaan untuk menguji apakah hasil yang dijamin secara teoritis benar (proposisi mengenai stabilitas dan kualitas aproksimasi, konservasi kuantitas fisik tertentu, dan sebagainya). Namun, pengujian sistem khususnya diperumit oleh kenyataan bahwa perangkat lunak simulasi sering berjalan pada perangkat keras terdistribusi yang tidak didukung dengan baik oleh alat untuk debugging dan pembuatan profil.[28]

Karena kesulitan yang terkait dengan pengujian dan karena pengabaian umum untuk kualitas kode (lihat di atas), prosedur verifikasi formal tidak umum dalam ilmu komputasi.[26]
Prakash Prabhu dan rekan-rekannya[3]melaporkan bahwa, menurut survei mereka, para ilmuwan menghabiskan lebih dari

setengah dari waktu pemrograman mereka untuk menemukan dan memperbaiki kesalahan tetapi hanya menggunakan metode debugging dan pengujian "primitif". Pengujian yang dilakukan hanya bersifat sepintas dan terdiri dari pemeriksaan manual untuk jawaban atas pertanyaan seperti "Apakah perangkat lunak melakukan apa yang saya harapkan untuk dilakukan dengan input dari jenis yang saya harapkan untuk digunakan?"[29]Dalam konteks ini, visualisasi data keluaran adalah alat yang paling umum untuk tujuan verifikasi dan validasi. Namun, visualisasi dapat memberikan tidak lebih dari pemeriksaan kewarasan yang menunjukkan bahwa kode tersebut berperilaku "wajar".[32]

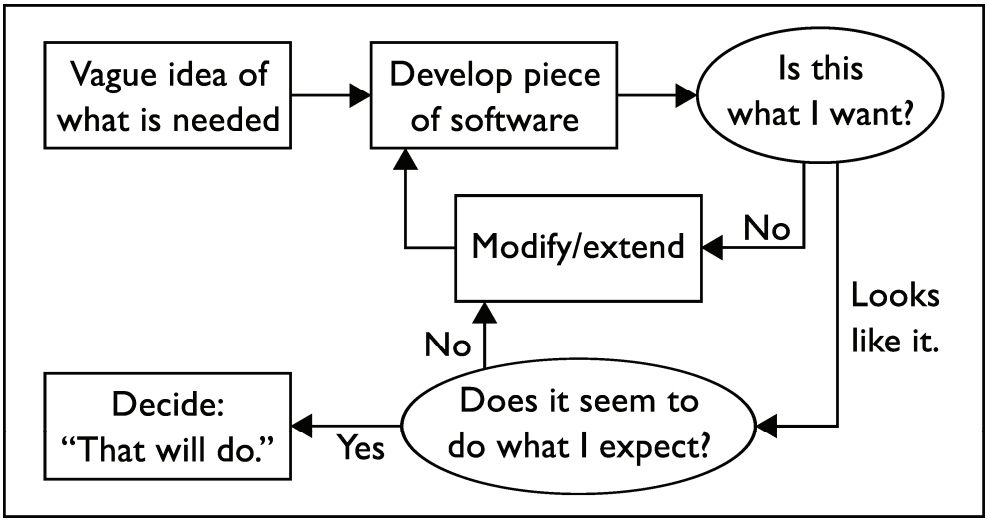
Salah satu alasan kurangnya pengujian disiplin ini dapat dilihat pada para ilmuwan tentang perangkat lunak mereka sebagai teori yang berkembang tidak sempurna yang memungkinkan mereka untuk menguji hipotesis. Dari sudut pandang ini, mereka menilai cacat model dan algoritme jauh lebih penting daripada cacat pengkodean.[24,28] Ini juga menjelaskan mengapa hampir semua strategi pengujian yang digunakan oleh para ilmuwan sangat ilmiah.[5] Karena mereka tidak menganggap kode sumber sebagai entitas dalam dirinya sendiri, melihatnya sebagai representasi langsung dari teori ilmiah yang mendasarinya (lihat sebelumnya), mereka hanya melihat output dari perangkat lunak dan memeriksa apakah itu sesuai dengan teori mereka saat ini. Para ilmuwan memperlakukan perangkat lunak seperti peralatan eksperimen (fisik) lainnya yang biasanya diharapkan berfungsi dengan baik. Asumsi ini hanya dipertanyakan jika data bertentangan dengan apa yang kira-kira diharapkan oleh para ilmuwan.[29]Untuk alasan ini, pendekatan rekayasa perangkat lunak untuk ilmu komputasi harus menarik perhatian programmer untuk pentingnya peran kebenaran kode sumber.[33]Ini dapat dicapai, misalnya, dengan menyediakan metode yang mudah digunakan untuk menguji pernyataan yang bermakna bagi para ilmuwan pada tingkat ilmiah.

Proses perangkat lunak yang terlalu formal membatasi penelitian

Proses pengembangan perangkat lunak tradisional yang menggunakan pendekatan desain besar di awal—seperti model air terjun—tidak cocok untuk ilmu komputasi.[24]Pengembangan perangkat lunak dalam sains sangat tertanam dalam metode ilmiah, yang membuat spesifikasi persyaratan di muka menjadi tidak mungkin dan menimbulkan tantangan dengan verifikasi dan validasi implementasi. Karena perangkat lunak ilmiah terus berkembang, tidak ada analisis kebutuhan yang jelas, desain, atau fase pemeliharaan yang dapat dilihat[26]dan pengembang membutuhkan fleksibilitas untuk bereksperimen dengan cepat dengan pendekatan solusi yang berbeda.[22]

Alih-alih proses rekayasa perangkat lunak yang mapan, para ilmuwan menerapkan proses informal yang tidak standar (lihat Gambar 1). Metode mereka sangat berulang, mulai dari gagasan yang kabur tentang masalah ilmiah mana yang harus dipecahkan oleh perangkat lunak dan oleh karena itu, aplikasi apa yang harus dilakukan. Berdasarkan ide ini, prototipe dikembangkan dan terus ditingkatkan, dipandu oleh pertanyaan "Apakah itu melakukan apa yang saya inginkan?" dan “Apakah itu membantu memecahkan masalah ilmiah yang dihadapi?”[29] Ketika perangkat lunak mencapai keadaan matang yang memungkinkan untuk menjawab pertanyaan penelitian, itu dikenakan pengujian sepintas seperti yang dijelaskan sebelumnya. Jika keluaran perangkat lunak tidak memenuhi harapan pengembang, modifikasi menjadi perlu sampai keluaran yang "masuk akal" tercapai. Perhatikan bahwa modifikasi ini hampir selalu melibatkan kode dan teori ilmiah yang mendasarinya.[27]Oleh karena itu, dan karena kode sering dianggap hanya sebagai representasi teori dan bukan sebagai entitas yang berdiri sendiri, metode pengembangan ilmuwan dapat, dalam arti tertentu, dianggap terutama sebagai metode pengembangan teori daripada metode pengembangan perangkat lunak. . Para ilmuwan menganggap proses perangkat lunak informal mereka harus mengikuti dari penerapan metode ilmiah ke penalaran ilmiah dengan bantuan komputasi.[34]

Beberapa peneliti menunjukkan bahwa pendekatan pengembangan yang lazim dalam ilmu komputasi memiliki beberapa kesamaan dengan metode rekayasa perangkat lunak tangkas (agilemanifesto.org), seperti pemrograman ekstrem. Banyak ilmuwan komputasi telah beroperasi dengan filosofi tangkas jauh sebelum istilah tersebut diperkenalkan dalam rekayasa perangkat lunak.[22]Namun, semua proses pengembangan yang sudah mapan—bahkan yang gesit—umumnya ditolak oleh masyarakat karena dianggap terlalu formal karena para ilmuwan merasa bahwa proses ini membatasi mereka dalam bereksperimen dengan perangkat lunak mereka.[29]Oleh karena itu, pendekatan pengembangan apa pun yang akan diadopsi oleh komunitas ilmu komputasi harus sangat ringan dan terintegrasi dengan baik dengan metode perangkat lunak/teori yang digambarkan pada Gambar 1.



Gambar 1. Model pengembangan perangkat lunak ilmiah. Diadaptasi dari Judith Segal dan Chris Morris.[21]

Karakteristik Karena Keterbatasan Perangkat Keras Komputer

Pada bagian ini, kita membahas karakteristik pengembangan perangkat lunak dalam ilmu komputasi yang disebabkan oleh keterbatasan sumber daya komputasi yang tersedia dan pemrograman yang efisien.

Pengembangan didorong dan dibatasi oleh perangkat keras

Perangkat lunak simulasi kompleks tidak pernah dianggap selesai oleh para ilmuwan komputasi. Karena itu hanya bisa menjadi representasi yang tidak sempurna dari realitas yang sangat kompleks, orang dapat terus berharap untuk meningkatkan perangkat lunak dan outputnya dengan memodelkan lebih banyak proses ilmiah yang relevan atau meningkatkan resolusi diskritisasi. Oleh karena itu, perangkat lunak ilmiah biasanya tidak dibatasi oleh teori tetapi oleh sumber daya komputasi yang tersedia dan pemanfaatannya yang efisien.[24]

Pengembangan perangkat lunak ilmiah tidak hanya terbatas tetapi juga didorong oleh perangkat keras komputasi yang tersedia dalam dua cara. Pertama, setiap kali perangkat keras baru yang meningkatkan daya komputasi dengan urutan besarnya tersedia, jenis simulasi multifisika gabungan yang benar-benar baru tiba-tiba menjadi mungkin. Ini memerlukan implementasi perangkat lunak simulasi baru atau, setidaknya, penggabungan simulasi dengan cara yang lebih kompleks. Kedua, platform perangkat keras baru secara teratur memperkenalkan perubahan dalam arsitektur perangkat keras yang mendasarinya. Memanfaatkan kekuatan arsitektur baru ini biasanya memerlukan adaptasi perangkat lunak simulasi yang ada untuk optimasi kinerja.[5]

Penggunaan bahasa dan teknologi pemrograman "lama"

Aplikasi HPC lawas cenderung ditulis dengan standar lama untuk bahasa pemrograman seperti Fortran dan bahasa tingkat rendah seperti C, dan menggunakan teknologi lama seperti antarmuka penyampaian pesan (MPI; www.mpi-forum.org). Ini karena beberapa alasan, salah satunya adalah umur panjang perangkat lunak HPC. Dalam konteks ini, Fortran dan C tampaknya menjadi pilihan yang aman karena setiap platform perangkat keras kemungkinan akan mendukung bahasa-bahasa ini selama bertahun-tahun yang akan datang.[5]Pemrogram ilmiah skeptis tentang teknologi baru karena sejarah HPC penuh dengan alat dan bahasa pemrograman yang menjanjikan peningkatan produktivitas tetapi segera dihentikan. Selain itu, tingkat abstraksi C yang rendah dan versi Fortran yang lebih lama menyiratkan bahwa pengembang beroperasi lebih dekat ke platform perangkat keras yang mendasarinya. Oleh karena itu, bahasa ini memberikan kinerja yang dapat diprediksi dan memungkinkan lebih banyak pengoptimalan kinerja buatan tangan.[28]Dapat juga diamati bahwa beberapa komunitas ilmiah besar bergerak menuju kerangka kerja C++ seperti Trilinos,[35]atau bahkan kerangka kerja Python seperti Jupyter.[36]

Para ilmuwan tidak melihat alasan untuk mengadopsi bahasa pemrograman yang lebih baru karena bahasa pemrograman yang sudah mapan mudah dipelajari (yang penting untuk pembelajaran mandiri), dan sejumlah besar kode warisan ditulis dalam bahasa tersebut.[22]Keputusan mereka juga sangat dipengaruhi oleh tradisi dan kepercayaan budaya: narasumber Rebecca Sanders dan Diane Kelly[27]melaporkan bahwa orientasi objek tidak "membeli [mereka] apa pun" dan bahwa "beberapa baris C akan membutuhkan banyak kode C++." Agar dapat diterima oleh komunitas ilmu komputasi, bahasa pemrograman baru harus mudah dipelajari, menawarkan kinerja yang cukup tinggi, menunjukkan stabilitas, dan mengubah konstruksi bahasa menjadi instruksi mesin dengan cara yang dapat diprediksi.[22]

Komunitas HPC menggunakan bahasa tingkat yang lebih tinggi seperti Matlab hampir secara eksklusif untuk algoritme prototipe, yang kemudian diimplementasikan kembali untuk kinerja yang lebih tinggi menggunakan bahasa tingkat yang lebih rendah.[34]Dalam disiplin ilmu yang kurang terkait teknologi — seperti biologi atau psikologi — bahasa yang lebih baru seperti Matlab dan Python lebih banyak diadopsi untuk proyek skala kecil.[3]Untuk proyek yang lebih besar, teknologi baru lebih mungkin diterima jika mereka dapat hidup berdampingan dengan yang lebih lama dan tidak segera memerlukan persetujuan penuh. Ini menjelaskan mengapa kerangka kerja yang mendikte pengguna bagaimana menyusun program mereka jarang digunakan. Para ilmuwan lebih suka menerapkan kembali banyak fungsi yang ada daripada menyerahkan kendali atas kode yang ingin mereka uji.[28]

Ketika mengadaptasi metode rekayasa perangkat lunak untuk ilmu komputasi, kita harus mempertimbangkan keengganan para ilmuwan, khususnya pengembang HPC, untuk menggunakan teknologi apa pun yang tidak teruji oleh waktu dan yang berisiko berhenti didukung. Oleh karena itu, penting untuk membuat semua perangkat lunak yang ditujukan untuk pemrogram ilmiah tersedia di bawah lisensi sumber terbuka dan tidak memaksa mereka untuk menggunakan bahasa pemrograman yang lebih baru. Hal ini memungkinkan para ilmuwan untuk, setidaknya pada prinsipnya, terus memelihara perangkat lunak yang dihentikan sendiri. Juga, persetujuan bertahap terhadap teknologi yang diusulkan harus dimungkinkan.

Pembauran logika domain dan detail implementasi

Penggunaan bahasa pemrograman prosedural yang lebih tua dalam ilmu komputasi dan fokus pada kinerja sering menghambat pemisahan logika domain dan detail implementasi dalam artefak solusi. Hal ini membuat sulit untuk mengembangkan teori ilmiah dan aspek implementasi tertentu (seperti pengoptimalan untuk platform perangkat keras tertentu) secara mandiri dan pada akhirnya mengarah ke perangkat lunak yang sulit untuk dipelihara. Ini juga menghasilkan masalah keahlian: jika semua aspek implementasi bercampur, pengembang harus—tetapi jarang—sama-sama mahir dalam semua aspek tersebut, mulai dari pengetahuan domain hingga metode numerik hingga spesifikasi desain prosesor tertentu.[5]Pendekatan rekayasa perangkat lunak, dengan demikian, harus fokus pada pemisahan masalah ini tanpa mempengaruhi tingkat kinerja secara negatif.

Persyaratan kualitas perangkat lunak yang saling bertentangan

Standar ISO/IEC 25010 mencantumkan delapan kategori karakteristik kualitas produk yang dapat dievaluasi oleh perangkat lunak: kesesuaian fungsional, keandalan, efisiensi kinerja, kegunaan, keamanan, kompatibilitas, pemeliharaan, dan portabilitas. Dalam studi lapangan mereka, Jeffrey Carver dan rekan-rekannya[22]menemukan bahwa pengembang perangkat lunak ilmiah memberi peringkat karakteristik berikut sebagai yang paling penting, dalam urutan menurun:

- kebenaran fungsional,
- pertunjukan,
- portabilitas, dan
- pemeliharaan.

Jelas, para ilmuwan menganggap kebenaran hasil perangkat lunak mereka sebagai prioritas utama. Bagaimanapun, hasilnya seharusnya secara akurat mewakili proses di dunia nyata dan digunakan sebagai titik awal untuk penalaran ilmiah.

Khususnya dalam konteks HPC, juga tidak mengherankan jika para ilmuwan menghargai kinerja karena simulasi besar dapat memakan waktu berhari-hari atau bahkan berbulan-bulan untuk dijalankan. Namun, betapapun berharganya kinerja bagi para ilmuwan, itu bukanlah tujuan itu sendiri—tujuan sebenarnya adalah melakukan sains. Oleh karena itu, yang paling ade-

metrik kinerja yang memadai untuk perangkat lunak ilmiah tidak diberikan dalam operasi floating point per detik (FLOPS) melainkan dalam "hasil yang berguna secara ilmiah per waktu kalender."[28,32]Selain itu, kinerja bertentangan dengan portabilitas dan pemeliharaan karena biasanya dicapai dengan memperkenalkan pengoptimalan khusus perangkat keras yang mengurangi keterbacaan kode. Atribut kualitas tambahan dari portabilitas dan pemeliharaan juga sangat penting bagi para ilmuwan karena perangkat lunak ilmiah berumur panjang. Selama masa pakainya yang lama, platform perangkat keras sering berubah, yang membatasi kemungkinan penyetelan kinerja khusus perangkat keras.[34]

Konflik antara kinerja dan portabilitas dialami sebagai masalah oleh para ilmuwan. Namun, rekayasa perangkat lunak dapat, sejauh ini, menawarkan sedikit panduan dalam aspek ini karena kinerja dan portabilitas adalah salah satu karakteristik kualitas yang paling tidak signifikan untuk sebagian besar pendekatan rekayasa perangkat lunak.[5]Oleh karena itu, adaptasi teknik rekayasa perangkat lunak untuk ilmu komputasi harus memberikan perhatian khusus untuk mengurangi masalah kinerja/portabilitas.

Karakteristik Karena Lingkungan Budaya Pengembangan Perangkat Lunak Ilmiah

Karakteristik yang dibahas dalam bagian ini dihasilkan dari lingkungan budaya di mana pengembangan perangkat lunak ilmiah terjadi. Lingkungan ini dibentuk, misalnya, oleh pelatihan ilmuwan komputasi dan skema pendanaan proyek penelitian ilmiah.

Hanya sedikit ilmuwan yang terlatih dalam rekayasa perangkat lunak

Judith Segal[26]menggambarkan ilmuwan komputasi sebagai "pengembang pengguna akhir profesional" yang bekerja di domain yang sangat teknis dan "kaya pengetahuan" dan biasanya mengembangkan perangkat lunak semata-mata untuk memajukan tujuan profesional mereka sendiri. Kesamaan mereka dengan pengembang pengguna akhir konvensional adalah bahwa kebanyakan dari mereka tidak memiliki pelatihan ilmu komputer formal dan tidak menganggap diri mereka sebagai insinyur perangkat lunak melainkan sebagai ahli domain meskipun mereka menghabiskan banyak waktu penelitian mereka untuk mengembangkan perangkat lunak. Berbeda dengan kebanyakan pengembang pengguna akhir konvensional, bagaimanapun, ilmuwan komputasi jarang mengalami kesulitan belajar bahasa pemrograman tujuan umum.

Persepsi diri pengembang perangkat lunak ilmiah sebagai ilmuwan daripada pengembang didasarkan pada nilai-nilai budaya masyarakat: karena tujuan utamanya adalah untuk memajukan pengetahuan ilmiah, keahlian domain dipandang sebagai modal intelektual, sedangkan keterampilan pengembangan perangkat lunak hanyalah teknik—a berarti berakhir. Ini juga menyiratkan bahwa memiliki keterampilan rekayasa perangkat lunak tidak dihargai dalam hal keputusan rekrutmen dan promosi. Pekerjaan diberikan kepada kandidat yang memenuhi syarat terbaik untuk apa yang biasanya dipandang sebagai prioritas tertinggi dalam ilmu komputasi: teori ilmiah.[27]

Selain tidak dihargai, mempelajari keterampilan rekayasa perangkat lunak dianggap sebagai tuntutan yang berlebihan oleh para ilmuwan komputasi karena mereka sudah cukup melakukan perannya sebagai ilmuwan (menulis makalah dan hibah, memberikan presentasi, dan sebagainya) dan mengikuti perkembangan zaman. bidang studi mereka yang berkembang pesat.[37]Masalah ini diperkuat oleh fakta bahwa ilmu komputasi sudah menjadi lebih interdisipliner, dan dengan demikian lebih rumit, murni dari sisi ilmiah. Karena semakin banyak efek yang harus dipertimbangkan oleh simulasi yang semakin kompleks, ilmuwan komputasi harus dapat berkolaborasi dengan peneliti dari disiplin lain dan "berbicara dalam bahasa mereka."[22]Semua ini menyisakan sedikit ruang untuk pendidikan rekayasa perangkat lunak. Pengetahuan tentang bahasa pemrograman yang dimiliki para ilmuwan—yang jelas tidak identik dengan pengetahuan rekayasa perangkat lunak—biasanya diperoleh dengan belajar sendiri atau dari rekan kerja.[28,38]

Meskipun pengembangan perangkat lunak sebagian besar dianggap sebagai beban, ilmuwan komputasi tidak suka mendelegasikannya kepada orang lain. Mereka merasa memiliki keterampilan teknis yang diperlukan dan merasa lebih mudah melakukannya sendiri daripada menjelaskan kebutuhan mereka kepada orang lain.[24] Selanjutnya, proses pengembangan sangat tergantung pada pengetahuan domain.[21,39]Dianggap lebih mudah untuk mengajari para ilmuwan cara memprogram daripada membuat insinyur perangkat lunak memahami ilmu domain karena banyak aplikasi "memerlukan gelar PhD dalam fisika atau cabang teknik hanya untuk memahami masalahnya."[22]Pandangan ini didukung oleh studi Segal,[23]di mana insinyur perangkat lunak

mengimplementasikan perpustakaan perangkat lunak ilmiah berdasarkan persyaratan dan dokumen spesifikasi yang ditulis oleh para ilmuwan. Meskipun pertemuan menit resmi diadakan selama proses pengembangan untuk membangun pemahaman bersama antara para ilmuwan dan insinyur perangkat lunak, produk akhir tidak memenuhi persyaratan para ilmuwan.

Meskipun tampaknya tidak diinginkan atau tidak layak untuk mendelegasikan pekerjaan ilmuwan komputasi kepada insinyur perangkat lunak eksternal, dianggap bermanfaat untuk memiliki beberapa insinyur perangkat lunak yang bekerja di lembaga penelitian ilmiah untuk memberikan dukungan pengembangan.[37]Namun, posisi seperti itu biasanya tidak didukung oleh lembaga pendanaan di masa lalu.[22]

Terminologi yang berbeda

Karena perkembangan ilmu komputasi dan rekayasa perangkat lunak yang terisolasi, kedua bidang tersebut telah menetapkan terminologi yang berbeda bahkan untuk konsep bersama.[5]Istilah dan metafora ilmuwan komputasi biasanya diambil dari metode ilmiah itu sendiri atau dari konsep komputasi tingkat rendah. Misalnya, pemrogram ilmiah tidak menyebut aplikasi mereka "perangkat lunak" melainkan berbicara tentang "kode." "Kode serial" adalah bagian dari perangkat lunak yang tidak menggunakan paralelisme, dan "menskalakan" kode semacam itu berarti mengadaptasinya untuk eksekusi paralel dan seterusnya.

Karena terminologi mereka yang berbeda, pemrogram ilmiah terkadang (harus) menemukan kembali teknik rekayasa perangkat lunak yang ada: mereka tidak dapat menemukan metode yang ada yang sesuai dengan kebutuhan mereka karena mereka mencarinya menggunakan kosakata yang "salah". Bagi mereka, teknik ini hanyalah aspek alami dari metode penelitian daripada alat umum untuk pengembangan perangkat lunak. Oleh karena itu, para ilmuwan sering tidak menyadari bahwa mereka telah menggunakan metode rekayasa perangkat lunak jika mereka dihadapkan dengan mereka dalam kosakata rekayasa perangkat lunak.[24]

Insinyur perangkat lunak mungkin harus menyesuaikan kosakata mereka untuk dipahami dan dianggap serius dalam domain ilmu komputasi. Terminologi rekayasa perangkat lunak sering dianggap oleh para ilmuwan komputasi sebagai istilah pemasaran mewah yang menawarkan apa-apa selain janji-janji kosong.[37]

Perangkat lunak ilmiah itu sendiri tidak memiliki nilai tetapi tetap berumur panjang

Untuk ilmuwan komputasi, perangkat lunak yang mereka hasilkan tidak memiliki nilai dalam dirinya sendiri; nilainya semata-mata didasarkan pada kemampuannya untuk secara efisien memecahkan masalah yang dihadapi dan membuat penemuan-penemuan ilmiah baru.[5]Fokus pada kebaruan dan penemuan ini mengarah pada persepsi bahwa keterampilan perangkat lunak hanyalah keahlian yang diperlukan dan bahwa memperolehnya bukanlah pekerjaan nyata.[37]Sementara pengetahuan domain dianggap sebagai modal intelektual, pengetahuan pengembangan perangkat lunak hanyalah sebuah teknik, yang akibatnya membuat semua keputusan teknis menjadi tidak penting.[26]

Selain itu, banyak ilmuwan komputasi tidak menganggap perangkat lunak sebagai entitas yang berdiri sendiri. Dalam pikiran mereka, kode sumber kurang lebih merupakan representasi langsung dari teori ilmiah yang mendasarinya.[27] Dengan demikian, satu-satunya nilai bahkan sebuah kode yang telah dipertahankan selama beberapa dekade tidak berasal dari upaya rekayasa yang dimasukkan ke dalamnya tetapi dari pengetahuan ilmiah yang terakumulasi di dalamnya.

Perspektif perangkat lunak seperti itu mengarah pada situasi di mana kualitas kode juga tidak dianggap penting—meskipun sangat terkait dengan kualitas hasil ilmiah.[11]Alih-alih tingkat kerusakan, satu-satunya metrik kode yang diterapkan pada perangkat lunak ilmiah adalah hasil baru yang dapat diterbitkan per LOC.[24]Bahkan ada kasus di mana perangkat lunak nontrivial diimplementasikan hanya untuk tujuan menerbitkan satu artikel saja. Karena waktu-untuk-solusi harus rendah dalam kasus seperti itu, tidak banyak pemikiran yang dihabiskan untuk atribut kualitas seperti rawatan, diperpanjang, atau dapat digunakan kembali. Jika kode rekayasa yang agak buruk terus diperpanjang—yaitu berapa banyak kode besar yang muncul—sulit untuk memperbaiki kekurangan ini.[37]

Meskipun para ilmuwan tidak melihat nilai dalam perangkat lunak ilmiah itu sendiri, banyak kode memiliki masa hidup dalam urutan beberapa dekade. Perangkat lunak ini mungkin tidak begitu berharga, tetapi akumulasi pengetahuan para peneliti yang terkandung di dalamnya menjadikannya investasi jangka panjang.[5]

Selama siklus hidup yang begitu panjang, perangkat lunak terus perlu dikembangkan lebih lanjut untuk mencerminkan kemajuan dalam teori ilmiah dan perangkat keras komputasi.[22,24]Karena masa pakai yang berpotensi sangat lama, banyak pengembang perangkat lunak ilmiah mencoba menghindari ketergantungan pada teknologi yang

bisa menjadi tidak tersedia. Untuk alasan ini, jumlah dependensi, seperti pustaka perangkat lunak, dijaga agar tetap minimum, dan hanya alat dan bahasa pemrograman yang telah bertahan dari kerusakan waktu yang digunakan. Hal ini terutama berlaku di komunitas HPC karena kode mereka kemungkinan besar akan berumur panjang.

Meskipun umur panjang perangkat lunak ilmiah, upaya yang ditujukan untuk pemeliharaannya rendah karena fokus pada implementasi fitur-fitur baru. Melakukan tugas pemeliharaan tidak disarankan, pertama, karena tidak diberi imbalan karena tidak mengarah pada hasil baru yang dapat dipublikasikan dan, kedua, dengan membebani pengembang untuk menunjukkan bahwa perubahan mereka tidak memengaruhi keakuratan hasil simulasi.[24]Selain itu, skema pendanaan berbasis hibah di banyak cabang ilmu pengetahuan membuat sulit untuk mengasumsikan perspektif jangka panjang dalam merawat perangkat lunak ilmiah, itulah sebabnya solusi cepat dan kotor lebih disukai secara selektif.[37]Akibatnya, setiap pendekatan rekayasa perangkat lunak untuk ilmu komputasi harus memastikan bahwa sifat kualitas seperti rawatan dibangun ke dalam perangkat lunak dari awal kuasi-otomatis.

Menciptakan pemahaman bersama tentang "kode" itu sulit

Sementara semua ilmuwan bersemangat mendokumentasikan hasil ilmiah mereka dalam makalah dan laporan teknis, mereka biasanya tidak menghasilkan dokumentasi untuk perangkat lunak yang mereka implementasikan. Panduan pengguna dibuat hanya dalam kasus yang lebih jarang bahwa perangkat lunak dimaksudkan untuk digunakan oleh basis pengguna yang lebih besar di luar kelompok riset pengembang asli.[27]Alih-alih mengandalkan dokumentasi, para ilmuwan lebih memilih cara transfer pengetahuan informal dan kolegial untuk menciptakan pemahaman bersama tentang perangkat lunak. Karena pengguna dan pengembang kode ilmiah biasanya tumpang tindih, mereka dapat mengandalkan latar belakang pengetahuan bersama. Oleh karena itu, para ilmuwan merasa lebih sulit untuk membaca dan memahami artefak dokumentasi daripada menghubungi pembuat bagian tertentu dari perangkat lunak dan mendiskusikan pertanyaan mereka dengannya.[26]

Tingkat pergantian personel yang tinggi dalam pengembangan perangkat lunak ilmiah, bagaimanapun, membuat transfer pengetahuan informal seperti itu bermasalah. Sebagian besar pengembang di bidang ini adalah pemula (mahasiswa PhD dan peneliti pascadoktoral awal) karena para ilmuwan biasanya tidak mengembangkan perangkat lunak untuk seluruh karir mereka. Saat mereka menaiki tangga karier—dan sering pindah ke institusi lain—pengetahuan mereka tentang perangkat lunak menjadi lebih sulit untuk diakses.[31]Ini berarti bahwa para pemula, tanpa bantuan materi dokumentasi apapun, harus berulang kali membiasakan diri dengan kode-kode yang tidak ditulis dengan mempertimbangkan pemahaman program.[26,32]Oleh karena itu, metode rekayasa perangkat lunak untuk ilmu komputasi harus meningkatkan tingkat abstraksi artefak implementasi yang dihasilkan oleh pengembang ilmiah untuk membuat artefak ini, setidaknya sampai batas tertentu, mendokumentasikan diri.

Namun, situasinya berbeda untuk laboratorium pemerintah di mana perangkat lunak berumur panjang terutama dikembangkan oleh para ilmuwan yang menghabiskan sebagian besar karir mereka di satu institusi itu. Institusi semacam itu seringkali memiliki sumber daya untuk tim ahli yang berfokus pada perangkat lunak itu sendiri, dengan tanggung jawab untuk porting, pengoptimalan, dan pemeliharaan. Untuk perangkat lunak yang tahan lama, penting untuk selalu memperbarui dokumentasi dan pengetahuan tentang perangkat lunak.[40]

Penggunaan kembali kode kecil

Pengembang perangkat lunak ilmiah jarang menggunakan kembali kode yang dikembangkan oleh orang lain. Misalnya, kerangka kerja untuk mengabstraksi dari detail penggunaan MPI yang seringkali membosankan tidak diadopsi karena mereka membuat asumsi tertentu tentang bagaimana pengguna mereka harus menyusun kode mereka. Para ilmuwan khawatir bahwa nanti dalam sebuah proyek, asumsi struktural ini bisa menjadi terlalu membatasi tetapi tidak dapat dielakkan. Jadi sebagai gantinya, para peneliti cenderung mengembangkan kembali kerangka kerja tersebut untuk setiap aplikasi agar sesuai dengan kebutuhan mereka.[28,32]Hal yang sama berlaku untuk penggunaan perpustakaan perangkat lunak. Misalnya, banyak ilmuwan mengimplementasikan pustaka aljabar linier mereka sendiri meskipun ada banyak implementasi paralel yang telah teruji dengan baik, dioptimalkan untuk cache, dan tersedia di bawah lisensi sumber terbuka. Dengan demikian, para ilmuwan ini menyia-nyiakan banyak upaya untuk menemukan kembali teknologi yang ada dan, sangat mungkin, menciptakannya kembali dengan kualitas yang lebih rendah.[3]

Penggunaan kembali kode yang ada secara terbatas diamati tidak hanya untuk perangkat lunak yang dikembangkan oleh orang lain, tetapi juga lazim bahkan jika menyangkut milik para ilmuwan. Karena sebagian besar kode ilmiah tidak diprogram dengan mempertimbangkan pemahaman, para ilmuwan lebih suka menulis ulang kode untuk proyek-proyek baru.

daripada menghabiskan banyak waktu untuk memahami kode lama—bahkan jika mereka adalah pembuat kode lama.[26]Meningkatkan tingkat abstraksi dalam artifak implementasi dapat mendorong penggunaan kembali kode di antara para ilmuwan karena menyederhanakan proses pemahaman.

Abaikan sebagian besar metode rekayasa perangkat lunak modern

Survei di antara pengembang perangkat lunak ilmiah menunjukkan bahwa mereka percaya bahwa mereka memiliki pengetahuan rekayasa perangkat lunak yang memadai untuk mencapai tujuan pengembangan mereka. Namun, ketika ditanya tentang pengetahuan dan adopsi praktik dan teknik terbaik rekayasa perangkat lunak modern tertentu (seperti pengujian, pembuatan profil, dan pemfaktoran ulang), baik pengetahuan maupun adopsi relatif rendah. Oleh karena itu, tampaknya para ilmuwan "tidak tahu apa yang tidak mereka ketahui".[4,38]Dan bahkan jika mereka terbiasa dengan alat seperti profiler, mereka jarang benar-benar menggunakannya—baik karena prasangka terhadap alat ("tidak akan membantu") atau karena mereka berpikir bahwa mereka tidak membutuhkannya ("Saya tahu di mana waktu dihabiskan dalam kode saya").[3]

Tapi bukan hanya ketidaktahuan yang mengarah pada tidak diadopsinya metode rekayasa perangkat lunak. Banyak metode dan alat tidak cocok untuk ilmuwan karena fungsinya didasarkan pada asumsi (seringkali tersirat) yang dilanggar dalam konteks ilmu komputasi.[2]Atau mereka tidak cocok karena mengabaikan persyaratan khusus yang dimiliki para ilmuwan (terutama jika menyangkut alat yang dapat mendukungnya). Contoh ketidaksesuaian karena asumsi yang salah adalah penggunaan proses rekayasa perangkat lunak yang tidak cukup mempertimbangkan siklus hidup panjang perangkat lunak ilmiah atau kurangnya persyaratan di muka.[22]Lingkungan pengembangan terpadu (IDE) untuk komunitas HPC adalah contoh ketidakcocokan karena mengabaikan persyaratan khusus untuk alat. Penggunaan IDE terbatas dalam komunitas ini karena lingkungan pengembangan biasanya tidak menampilkan dukungan yang nyaman untuk membangun, membuat profil, dan menerapkan aplikasi HPC pada sistem terdistribusi skala besar. Oleh karena itu, para ilmuwan merasa dibatasi oleh IDE dan, karenanya, tidak mengadopsinya.[3,32]

Kegagalan rekayasa perangkat lunak untuk memenuhi kebutuhan ilmu komputasi secara memadai mengarah pada situasi di mana para ilmuwan curiga terhadap klaim insinyur perangkat lunak dan sangat menyukai solusi buatan tangan.[5]Namun, jika para ilmuwan dihadapkan pada teknik rekayasa perangkat lunak tertentu yang mereka temukan cocok untuk lingkungan kerja spesifik mereka, itu akan segera diadopsi. Contohnya adalah sistem kontrol versi, kerangka pengujian regresi yang dapat disesuaikan dengan kebutuhan para ilmuwan untuk pengujian, dan penggunaan kembali di perpustakaan kecil melalui untuk pemecah persamaan, penanganan mesh, dan sebagainya.[28]Agar dapat diterima oleh para ilmuwan, alat-alat ini harus memperkenalkan minimal teknis karena para ilmuwan cukup sibuk mengikuti perkembangan pesat di bidangnya sendiri.[37]

Secara keseluruhan, kita dapat menyimpulkan bahwa pendekatan rekayasa perangkat lunak hanya akan diadopsi oleh para ilmuwan jika pendekatan ini menghormati karakteristik dan kendala yang berbeda dari pengembangan perangkat lunak ilmiah, yang kami jelaskan di atas.

TEKNIK PERANGKAT LUNAK APA YANG DITAWARKAN UNTUK ILMU KOMPUTASI

Analisis rinci kami tentang karakteristik spesifik dari pengembangan perangkat lunak ilmiah memungkinkan kami untuk mengidentifikasi beberapa kekurangan dari proposal terperinci yang ada yang berkaitan dengan menjembatani "jurang" antara rekayasa perangkat lunak dan ilmu komputasi. Selain itu, kami memberikan gambaran tentang upaya yang lebih baru untuk menutup kesenjangan antara kedua disiplin ilmu dan memberikan pandangan tentang kemungkinan arah penelitian yang dapat berkontribusi untuk memperbaiki situasi saat ini.

Menjembatani Jurang Perangkat Lunak

Upaya sebelumnya untuk mengatasi krisis kredibilitas dan produktivitas ilmu komputasi dapat dikategorikan ke dalam tiga kelompok:

- mempublikasikan dan meninjau kode sumber bersama dengan artikel ilmiah untuk memastikan reproduktifitas, atau setidaknya pengulangan, dari eksperimen in silico,

- biarkan insinyur perangkat lunak membangun atau merekayasa ulang (bagian dari) perangkat lunak ilmiah, dan
- melatih para ilmuwan untuk memungkinkan mereka menggunakan metode rekayasa perangkat lunak yang canggih.

Dalam konteks krisis kredibilitas ilmu komputasi, muncul diskusi tentang reproduktifitas hasil ilmiah yang mengandalkan komputasi.[41]dan komunitas rekayasa perangkat lunak.[42]Mampu—setidaknya pada prinsipnya—memvalidasi temuan ilmuwan lain dengan mereproduksi eksperimen mereka adalah inti dari metode ilmiah. Namun, itu adalah praktik umum di banyak bidang ilmu komputasi untuk tidak merilis kode sumber yang menjadi dasar temuan publikasi. Praktik ini menghambat reproduksi hasil yang dipublikasikan atau bahkan membuatnya menjadi tidak mungkin. Oleh karena itu, beberapa penulis telah menyarankan untuk membuat pengungkapan publik kode sumber wajib untuk publikasi peer-review, dan beberapa bahkan mengusulkan untuk memasukkan kode dalam proses peer-review.[43]Dalam komunitas rekayasa perangkat lunak, misalnya, beberapa konferensi besar baru-baru ini mulai menggunakan proses evaluasi artefak yang ditinjau oleh rekan sejawat.[44]

Saran dan upaya ini tentu saja merupakan langkah penting ke arah yang benar dan dapat membantu meningkatkan apresiasi perangkat lunak dan kualitasnya di komunitas ilmu komputasi. Namun, penerbitan kode sumber saja tidak cukup mengatasi masalah mendasar bahwa para ilmuwan tidak memiliki keterampilan rekayasa perangkat lunak untuk mengatasi masalah yang mendasari krisis kredibilitas dan produktivitas.

Upaya kedua untuk mencari solusi adalah meminta insinyur perangkat lunak mengimplementasikan perangkat lunak untuk para ilmuwan. Pengalaman Segal,[23]yang menguji pendekatan ini, dan pertimbangan yang kami bahas sebelumnya menunjukkan bahwa ini bukan cara yang praktis.

Sejauh ini, upaya yang paling menjanjikan untuk memecahkan krisis perangkat lunak ilmiah ganda tampaknya adalah pendidikan melalui program pelatihan berbasis lokakarya yang berfokus pada mahasiswa PhD, seperti program yang diselenggarakan oleh Gregory Wilson.[45]Sementara pendekatan pendidikan mengatasi kesenjangan keterampilan yang menjadi pusat jurang perangkat lunak, hal itu dilakukan dengan cara yang tidak memadai. Analisis kami sebelumnya dengan jelas menunjukkan bahwa hanya mengekspos ilmuwan untuk metode rekayasa perangkat lunak tidak akan cukup karena metode ini sering gagal untuk mempertimbangkan karakteristik khusus dan kendala pengembangan perangkat lunak ilmiah. Oleh karena itu kami menyimpulkan bahwa kami harus memilih teknik rekayasa perangkat lunak yang sesuai dan menyesuaikannya secara khusus dengan kebutuhan ilmuwan komputasi.

Mengadaptasi Pendekatan Rekayasa Khusus Domain

Studi literatur kami dengan jelas menunjukkan bahwa ilmuwan komputasi hanya secara tidak sengaja terlibat dalam pengembangan perangkat lunak: pada akhirnya, tujuan mereka bukan untuk membuat perangkat lunak tetapi untuk mendapatkan hasil ilmiah yang baru. Namun, pada saat yang sama, mereka sangat prihatin tentang memiliki kontrol penuh atas aplikasi mereka dan bagaimana ini benar-benar menghitung hasil mereka, itulah sebabnya banyak yang lebih memilih bahasa pemrograman yang lebih tua dengan tingkat abstraksi yang relatif rendah dari perangkat keras yang mendasarinya.

Di antara teknik dan alat yang ditawarkan oleh rekayasa perangkat lunak, apa yang disebut bahasa spesifik domain (domain-specific language/DSL) adalah titik awal yang menjanjikan untuk memenuhi kebutuhan ilmuwan komputasi. Seperti bahasa tujuan umum (GPL) seperti C atau Java, DSL adalah bahasa pemrograman. Namun, tidak seperti GPL, yang dirancang untuk dapat mengimplementasikan program apa pun yang dapat dihitung dengan mesin Turing, DSL membatasi ekspresinya pada domain aplikasi tertentu. Dengan menampilkan konsep domain tingkat tinggi yang memungkinkan untuk memodelkan fenomena pada tingkat abstraksi domain dan dengan memberikan notasi yang dekat dengan domain target, DSL bisa menjadi sangat ringkas. Sintaks DSL dapat berupa tekstual atau grafis, dan program DSL dapat dieksekusi baik melalui interpretasi atau melalui pembuatan kode sumber dalam GPL yang ada.

Karena DSL dirancang untuk mengekspresikan solusi pada tingkat abstraksi domain, mereka memungkinkan para ilmuwan untuk peduli tentang apa yang paling penting bagi mereka: melakukan sains tanpa harus berurusan dengan detail teknis, implementasi spesifik. Meskipun mereka menggunakan abstraksi domain tingkat tinggi, mereka

tetap memegang kendali penuh atas proses pengembangan mereka karena merekalah yang secara langsung mengimplementasikan solusi mereka dalam bahasa pemrograman formal dan yang dapat dieksekusi (misalnya, melalui generasi). Selain itu, generasi dari bahasa formal ke GPL tingkat rendah memungkinkan pemeriksaan kode yang dihasilkan untuk melacak apa yang sebenarnya dihitung.

DSL juga dapat membantu mengatasi konflik antara persyaratan kualitas kinerja di satu sisi dan portabilitas dan pemeliharaan di sisi lain, yang bertanggung jawab atas banyak kesulitan yang dialami dalam pengembangan perangkat lunak ilmiah. Kode sumber DSL dapat dipelihara karena sering diprastrukturkan dan lebih mudah dibaca daripada kode GPL, yang membuatnya hampir didokumentasikan sendiri. Sifat kode sumber DSL yang hampir mendokumentasikan diri ini dan fakta bahwa ia dapat, idealnya, mengandalkan generator yang teruji dengan baik untuk terjemahan program memastikan keandalan hasil ilmiah berdasarkan keluaran perangkat lunak. Portabilitas kode DSL dicapai dengan mengganti generator untuk bahasa tersebut dengan yang menargetkan platform perangkat keras lain. Dengan DSL,

Dengan cara yang dijelaskan di atas, DSL yang diintegrasikan ke dalam pendekatan rekayasa perangkat lunak yang tepat dapat membantu mengatasi krisis produktivitas dan kredibilitas ilmu komputasi. Indikator pertama yang mendukung hipotesis ini dapat ditemukan dalam laporan survei Prabhu dan rekan,[3]yang menemukan bahwa para ilmuwan yang memprogram dengan DSL "melaporkan produktivitas dan kepuasan yang lebih tinggi dibandingkan dengan ilmuwan yang terutama menggunakan bahasa tujuan umum, numerik, atau skrip."

Penelitian yang ada mengenai penerapan DSL dalam ilmu komputasi mencakup desain beberapa DSL individu. Contohnya adalah Liszt,[46]yang merupakan DSL untuk pemecah persamaan diferensial parsial berbasis mesh dengan fokus pada paralelisasi otomatis, dan SESSL,[47]yang memungkinkan eksperimen simulasi model untuk memastikan reproduktifitasnya. Erik Schnetter dan rekan-rekannya[48]menjelaskan Chemora, kerangka kerja untuk memecahkan persamaan diferensial parsial pada arsitektur HPC modern. Mereka menggunakan DSL untuk memisahkan masalah desain model komputasi, diskritisasi model, dan pemetaan ke sumber daya perangkat keras (termasuk pengoptimalan kinerja).

Dalam contoh yang disebutkan sejauh ini, DSL dipandang sebagai alat yang kurang lebih terisolasi yang dapat digunakan para ilmuwan untuk membuat pengembangan perangkat lunak lebih mudah bagi mereka. Peneliti lain mengintegrasikan penggunaan DSL ke dalam pendekatan yang lebih holistik yang secara langsung menangani krisis produktivitas dan/atau krisis kredibilitas. Misalnya, Marc Palyart dan rekan-rekannya[49]memperkenalkan pendekatan rekayasa perangkat lunak yang disebut MDE4HPC yang menggunakan DSL HPCML[50]untuk membantu para ilmuwan dalam mengimplementasikan aplikasi HPC secara efisien yang tidak bergantung pada arsitektur perangkat keras HPC tertentu. Mohamed Almorsy dan rekan mengusulkan penggunaan suite DSL grafis untuk menggunakan pemodelan grafis dalam semua aspek proses pengembangan perangkat lunak ilmiah. Mereka menyediakan alat berbasis web yang membantu para ilmuwan mendefinisikan DSL sendiri. Pendekatan rekayasa perangkat lunak lain untuk ilmu komputasi adalah Sprat,[52,53]yang mengintegrasikan beberapa DSL secara hierarkis untuk memfasilitasi kolaborasi ilmuwan dari berbagai disiplin ilmu dalam pengembangan perangkat lunak simulasi yang kompleks. Dalam pendekatan ini, setiap peran pengembang dalam proyek perangkat lunak diberi DSL terpisah, yang dimaksudkan untuk mengarah pada pemisahan yang jelas dari masalah, kode yang terpelihara dengan baik, dan produktivitas tinggi karena para ilmuwan hanya perlu bekerja dengan abstraksi yang sudah ada. akrab dengan dari domain masing-masing.

Mirip dengan desain GPL, desain DSL menghadapi masalah tidak mengetahui persyaratan sebelumnya. Jadi, penting untuk mengembangkan DSL dengan metode tangkas[54]dan untuk melibatkan pengguna akhir dalam proses desain dan evaluasi.[55]

Rekayasa Kinerja Perangkat Lunak

Arah penelitian lain yang mungkin adalah memasukkan teknik yang dikembangkan oleh rekayasa kinerja perangkat lunak (SPE).[56]ke dalam pendekatan rekayasa perangkat lunak untuk ilmu komputasi. Seringkali, optimasi kinerja membutuhkan perubahan besar dalam desain perangkat lunak. Oleh karena itu, kinerja harus sudah dipertimbangkan dalam fase desain perangkat lunak pada tingkat arsitektur. Namun, sekali lagi, para ilmuwan biasanya mengembangkan perangkat lunak dengan cara yang sangat berulang dengan fokus pada masalah ilmiah yang dihadapi, yang menyiratkan bahwa biasanya tidak ada fase desain perangkat lunak yang berbeda. Masalah ini dapat dielakkan dengan menggunakan DSL untuk membangun model dari

perangkat lunak ilmiah yang akan diimplementasikan, seperti yang dibahas pada bagian sebelumnya. Jika perangkat lunak diimplementasikan menggunakan abstraksi tingkat domain, prediksi kinerja berbasis model dan teknik pengoptimalan dapat digunakan tanpa memaksa para ilmuwan untuk mengadopsi proses perangkat lunak yang kaku.

Karena sumber daya perangkat keras selalu terbatas dan karena kinerja sangat penting, terutama di HPC, penerapan pendekatan SPE semacam itu untuk mengoptimalkan efisiensi runtime perangkat lunak ilmiah secara sistematis adalah bidang yang menjanjikan untuk pekerjaan di masa depan.

Menguji Perangkat Lunak Ilmiah

Seperti yang telah kita diskusikan, rekayasa perangkat lunak untuk ilmu komputasi harus menarik perhatian pemrogram akan pentingnya kebenaran perangkat lunak.[33]Pengujian perangkat lunak biasanya membutuhkan oracle, yang merupakan mekanisme untuk memeriksa apakah program yang diuji menghasilkan keluaran yang diharapkan ketika dijalankan menggunakan serangkaian kasus uji. Namun, mendapatkan oracle yang andal untuk program ilmiah itu menantang, karena sebagian besar persyaratannya tidak jelas di awal karena sifat eksplorasi pengembangan perangkat lunak ilmiah. Pengujian berbasis model membutuhkan persyaratan yang terdefinisi dengan baik dan stabil untuk mengembangkan model; dengan demikian, pengujian berbasis model tidak mudah diterapkan pada perangkat lunak ilmiah. Sebagai gantinya, diperlukan pendekatan untuk melakukan pengujian yang efektif tanpa nubuat yang telah ditentukan sebelumnya.[57]Pendekatan baru seperti yang disebut pengujian metamorfik bermaksud untuk memecahkan tantangan pengujian program nondeterministik yang tidak memiliki nubuat,[58]misalnya, melalui teknik pembelajaran mesin untuk mendeteksi hubungan metamorf secara otomatis.[59]

Tantangan lain adalah mengintegrasikan regresi otomatis dan pengujian penerimaan untuk perangkat lunak ilmiah, misalnya, dalam pengaturan integrasi berkelanjutan. Pengujian regresi memungkinkan perbandingan keluaran saat ini dengan keluaran sebelumnya untuk mengidentifikasi kesalahan atau anomali kinerja yang diperkenalkan saat kode dimodifikasi. Berbagai alat dan pendekatan sedang dikembangkan untuk mengatasi tantangan pengujian dan debugging perangkat lunak yang dirancang untuk berjalan pada sistem terdistribusi. Beberapa kerangka kerja pengujian unit secara langsung mendukung MPI dan telah berhasil digunakan oleh banyak komunitas. Pendekatan lain adalah untuk mengolok-olok MPI dan menguji/men-debug komponen simulasi secara terpisah.[60]

Untuk perangkat lunak ilmiah, kesulitan utama untuk pengujian regresi otomatis disebabkan oleh biaya komputasi pengujian yang tinggi. Untuk memastikan cakupan kode yang tinggi, serangkaian konfigurasi pengujian yang berpotensi eksponensial harus dijalankan. Solusi untuk tantangan ini dapat berupa modularisasi perangkat lunak yang tepat sehingga komponen perangkat lunak dapat diuji secara terpisah. Pendekatan modularisasi seperti layanan mikro memungkinkan skalabilitas,[61]serta kelincahan dan kehandalan.[62] Modularisasi semacam itu mungkin juga memfasilitasi pengujian regresi otomatis dari perangkat lunak ilmiah.

Rekayasa Persyaratan

Beberapa pendekatan rekayasa perangkat lunak untuk ilmu komputasi, seperti proyek Advises oleh Sarah Thew dan rekan-rekannya[63]dan pendekatan oleh Javier Garcia dan rekan,[64]fokus pada teknik rekayasa kebutuhan. Garcia dan rekan-rekannya[64]memperkenalkan metode berbasis komponen dan berorientasi aspek untuk pengembangan perangkat lunak ilmiah. Pendekatan mereka berfokus pada rekayasa persyaratan formal untuk memungkinkan penggunaan kembali komponen perangkat lunak yang ada dan integrasinya melalui teknik pemrograman berorientasi aspek. Idenya adalah bahwa begitu persyaratan aplikasi ilmiah diketahui, itu dapat dibangun hanya dengan mengidentifikasi komponen fungsional yang sesuai yang sudah ada dan hubungan ketergantungan di antara mereka.

Proyek Advises mengakui bahwa dalam komputasi ilmiah, biasanya tidak mungkin untuk menentukan persyaratan perangkat lunak yang terperinci di muka. Mengingat situasi ini, mereka mengusulkan proses rekayasa persyaratan di mana insinyur perangkat lunak menggunakan teknik seperti wawancara tidak terstruktur dan observasi pengguna untuk secara iteratif memperoleh persyaratan rinci. Atas dasar persyaratan ini, para insinyur perangkat lunak seharusnya mengembangkan perangkat lunak untuk ilmuwan domain.

Meskipun evaluasi positif dari proyek Advises dalam epidemiologi, masih belum jelas apakah pendekatan semacam itu, yang berfokus pada rekayasa persyaratan dan insinyur perangkat lunak yang mengimplementasikan perangkat lunak untuk ilmuwan domain, dapat diterapkan ke cabang ilmu komputasi lainnya. Studi Segal[23] menunjukkan bahwa ini mungkin tidak berlaku untuk cabang seperti HPC.

KESIMPULAN

Atas dasar pemeriksaan perkembangan historis hubungan antara rekayasa perangkat lunak dan ilmu komputasi (masa lalu), kami mengidentifikasi 13 karakteristik utama pengembangan perangkat lunak ilmiah dengan meninjau literatur yang diterbitkan (saat ini). Kami menemukan bahwa karakteristik unik pengembangan perangkat lunak ilmiah mencegah para ilmuwan menggunakan alat dan metode rekayasa perangkat lunak yang canggih. Situasi ini menciptakan jurang antara rekayasa perangkat lunak dan ilmu komputasi, yang mengakibatkan krisis produktivitas dan kredibilitas dari disiplin ilmu yang terakhir. Kami memeriksa upaya untuk menjembatani kesenjangan untuk mengungkapkan kekurangan dari solusi yang ada dan untuk menunjukkan arah penelitian lebih lanjut, seperti penggunaan DLS dan teknik pengujian tanpa nubuat yang telah ditentukan sebelumnya (kemungkinan masa depan). Namun,

REFERENSI

1. DE Post, "Wajah yang Berubah dari Komputasi Ilmiah dan Rekayasa,"*Komputasi dalam Sains & Eng.*, vol. 15, tidak. 6, 2013, hlm. 4–6.
2. D. Heaton dan JC Carver, "Klaim tentang Penggunaan Praktik Rekayasa Perangkat Lunak dalam Sains: Tinjauan Literatur Sistematis,"*Teknologi Informasi dan Perangkat Lunak*, vol. 67, 2015, hlm. 207–219.
3. P. Prabhu et al., "A Survey of the Practice of Computational Science,"*Prok. Konferensi Internasional Komputasi, Jaringan, Penyimpanan, dan Analisis Performa Tinggi*(SC 11), 2011, hal. 19:1.
4. JE Hannay et al., "Bagaimana Ilmuwan Mengembangkan dan Menggunakan Perangkat Lunak Ilmiah?," *Prok. Perangkat Lunak Bengkel Eng. untuk Ilmu Komputasi dan Eng.*(BAGIAN 09), 2009, hlm. 1–8.
5. S. Faulk et al., "Gangguan Produktivitas Komputasi Ilmiah: Bagaimana Rekayasa Perangkat Lunak Dapat Membantu,"*Komputasi dalam Sains & Eng.*, vol. 11, 2009, hlm. 30–39.
6. PE Ceruzzi,*Sejarah Komputasi Modern, edisi ke-2.*, MIT Press, 2003.
7. A. Newell, A. Perlis, dan H. Simon, "Surat kepada Redaksi,"*Sains*, vol. 157, 1967, hlm. 1373–1374.
8. I. Vessey, "Masalah versus Solusi: Peran Domain Aplikasi dalam Perangkat Lunak," *Prok. Workshop ke-7 tentang Studi Empiris Programmer*, 1997, hlm. 223– 240.
9. P. Naur dan B. Randell,*Rekayasa Perangkat Lunak: Laporan Konferensi yang Disponsori oleh Komite Sains NATO*, laporan, NATO, 1969.
10. D. Kelly, "Sebuah Jurang Perangkat Lunak: Rekayasa Perangkat Lunak dan Komputasi Ilmiah,"*Perangkat Lunak IEEE*, vol. 24, tidak. 6, 2007.
11. L. Hatton dan A. Roberts, "Seberapa Akurat Perangkat Lunak Ilmiah?,"*IEEE Trans. Perangkat Lunak Eng.*, vol. 20, tidak. 10, 1994, hlm. 785–797.
12. SH Fuller dan LI Millett, "Kinerja Komputasi: Game Over atau Level Selanjutnya?," *Komputer*, vol. 44, tidak. 1, 2011, hlm. 31–38.
13. A. Buttari dkk., "Dampak Multicore pada Perangkat Lunak Matematika,"*LNCS*, vol. 4699, Springer, 2007.
14. MR Benioff dkk.,*Ilmu Komputasi: Memastikan Daya Saing Amerika*, laporan teknis, Komite Penasihat Teknologi Informasi Presiden (PITAC), 2005.
15. Z. Merali, "Ilmu Komputasi: Kesalahan, Mengapa Pemrograman Ilmiah Tidak Menghitung,"*Alam*, vol. 467, tidak. 7317, 2010, hlm. 775–777.
16. J. Brown, MG Knepley, dan BF Smith, "Runtime Extensibility and Librarization of Simulation Software,"*Komputasi dalam Sains & Teknik*, vol. 17, tidak. 1, 2015, hlm. 38–45.
17. GV Wilson, "Di mana Hambatan Sebenarnya dalam Komputasi Ilmiah?,"*Ilmuwan Amerika*, vol. 94, tidak. 1, 2006, hlm. 5–6.
18. T. Storer, "Menjembatani Jurang: Survei Praktik Rekayasa Perangkat Lunak dalam Pemrograman Ilmiah,"*Survei Komputasi ACM*, vol. 50, tidak. 4, 2017, hal. 47:1.
19. DE Post dan LG Votta, "Ilmu Komputasi Menuntut Paradigma Baru," *Fisika Hari Ini*, vol. 58, tidak. 1, 2005, hlm. 35–41.

20. JC Carver dan T. Epperly, “Rekayasa Perangkat Lunak untuk Ilmu dan Teknik Komputasi,”*Komputasi dalam Sains & Eng.*, vol. 16, tidak. 3, 2014, hlm. 6–9.
21. J. Segal dan C. Morris, “Mengembangkan Perangkat Lunak Ilmiah,”*Perangkat Lunak IEEE*, vol. 25, tidak. 4, 2008, hlm. 18–20.
22. JC Carver dkk., “Lingkungan Pengembangan Perangkat Lunak untuk Perangkat Lunak Ilmiah dan Rekayasa: Serangkaian Studi Kasus,”*Prok. Konferensi Internasional ke-29 Perangkat Lunak Eng.* (ICSE 07), 2007, hlm. 550–559.
23. J. Segal, “Ketika Insinyur Perangkat Lunak Bertemu Ilmuwan Riset: Studi Kasus,” *Perangkat Lunak Empiris Eng.*, vol. 10, tidak. 4, 2005, hlm. 517–559.
24. SM Easterbrook dan TC Johns, “Merekayasa Perangkat Lunak untuk Memahami Perubahan Iklim,”*Komputasi dalam Sains & Eng.*, vol. 11, tidak. 6, 2009, hlm. 65–74.
25. L. Hochstein et al., “Produktivitas Programmer Paralel: Studi Kasus Pemrogram Paralel Pemula,”*Prok. ACM/IEEE Conf. superkomputer*(SC 2005), 2005, hlm. 35–43.
26. J. Segal, “Beberapa Masalah Pengembang Pengguna Akhir Profesional,”*Prok. Sim. Bahasa Visual dan Komputasi Berpusat pada Manusia*(VL/HCC 07), 2007, hlm. 111–118.
27. R. Sanders dan DF Kelly, "Berurusan dengan Risiko dalam Pengembangan Perangkat Lunak Ilmiah," *Perangkat Lunak IEEE*, vol. 25, tidak. 4, 2008, hlm. 21–28.
28. VR Basili et al., “Memahami Komunitas Komputasi Kinerja Tinggi: Perspektif Insinyur Perangkat Lunak,”*Perangkat Lunak IEEE*, vol. 25, tidak. 4, 2008, hlm. 29–36.
29. J. Segal, “Model Pengembangan Perangkat Lunak Ilmiah,”*Prok. Lokakarya Internasional Pertama tentang Software Eng. untuk Ilmu Komputasi dan Eng.*(BAGIAN 08), 2008, hlm. 1–7.
30. U. Kanewala dan JM Bieman, “Menguji Perangkat Lunak Ilmiah: Tinjauan Literatur Sistematis,” *Teknologi Informasi dan Perangkat Lunak*, vol. 56, tidak. 10, 2014, hlm. 1219–1232.
31. F. Shull et al., "Desain Studi Empiris di Area Komputasi Kinerja Tinggi (HPC),"*Prok. Gejala Internasional Perangkat Lunak Empiris Eng.*, 2005, hlm. 1–10.
32. JC Carver dkk., “Pengamatan tentang Pengembangan Perangkat Lunak untuk Komputasi Kelas Atas,”*CTWatch Triwulanan*, vol. 2, tidak. 4A, 2006, hlm. 33–38.
33. K. Hinsen, "Menara Perkiraan dalam Ilmu Komputasi: Mengapa Menguji Perangkat Lunak Ilmiah Itu Sulit,"*Komputasi dalam Sains & Eng.*, vol. 17, tidak. 4, 2015, hlm. 72–77.
34. R. Kendall et al., “Pengembangan Kode Prakiraan Cuaca: Studi Kasus,”*Perangkat Lunak IEEE*, vol. 25, tidak. 4, 2008, hlm. 59–65.
35. MA Heroux dkk., “Ikhtisar Proyek Trilinos,”*ACM Trans. Perangkat Lunak Matematika*, vol. 31, tidak. 3, 2005, hlm. 397–423.
36. M. Ragan-Kelley et al., “Arsitektur Jupyter/IPython: Pandangan Terpadu Penelitian Komputasi, dari Eksplorasi Interaktif hingga Komunikasi dan Publikasi,”*Abstrak Pertemuan Musim Gugur AGU*, 2014.
37. S. Killcoyne dan J. Boyle, “Mengelola Kekacauan: Pelajaran yang Dipetik dari Mengembangkan Perangkat Lunak dalam Ilmu Hayati,”*Komputasi dalam Sains & Eng.*, vol. 11, tidak. 6, 2009, hlm. 20–29.
38. JC Carver et al., "Persepsi diri tentang Rekayasa Perangkat Lunak: Survei Ilmuwan dan Insinyur,"*Komputasi dalam Sains & Eng.*, vol. 15, tidak. 1, 2013, hlm. 7– 11.
39. J. Segal, “Beberapa Tantangan yang Dihadapi Insinyur Perangkat Lunak Mengembangkan Perangkat Lunak untuk Ilmuwan,”*Prok. Workshop ICSE tentang Software Eng.for Computational Science and Eng.*, 2009, hlm. 9–14.
40. U. Goltz dkk., “Desain untuk Masa Depan: Evolusi Perangkat Lunak Terkelola,”*Ilmu Komputer–Penelitian dan Pengembangan*, vol. 30, tidak. 3, 2015, hlm. 321–331.
41. RD Peng, "Penelitian yang Dapat Direproduksi dalam Ilmu Komputasi,"*Sains*, vol. 334, tidak. 6060, 2011, hlm. 1226–1227.
42. RJ LeVenque et al., "Penelitian yang Dapat Direproduksi untuk Komputasi Ilmiah: Alat dan Strategi untuk Mengubah Budaya,"*Komputasi dalam Sains & Eng.*, vol. 14, tidak. 4, 2012, hal. 13.
43. DC Ince, L. Hatton, dan J. Graham-Cumming, "Kasus untuk Program Komputer Terbuka,"*Alam*, vol. 482, tidak. 7386, 2012, hlm. 485–488.
44. S. Krishnamurthi dan J. Vitek, “Krisis Perangkat Lunak Nyata: Pengulangan sebagai Nilai Inti,”*Komunikasi ACM*, vol. 58, tidak. 3, 2015, hlm. 34–36.
45. GV Wilson, “Pertukangan Perangkat Lunak: Pelajaran yang Dipetik,”*F1000Penelitian*, vol. 3, 2014, hlm. 1–11.

46. Z. DeVito et al., “Liszt: Bahasa Khusus Domain untuk Membangun Pemecah PDE Berbasis Mesh Portabel,”*Prok. Konferensi Internasional Komputasi, Jaringan, Penyimpanan, dan Analisis Performa Tinggi*, 2011, hlm. 1–12.
47. R. Ewald dan AM Uhrmacher, “SESSL: Bahasa Khusus Domain untuk Eksperimen Simulasi,”*ACM Trans. Pemodelan dan Simulasi Komputer*, vol. 24, tidak. 2, 2014, hal. 11.
48. E. Schnetter et al., "Chemora: Kerangka Pemecahan PDE untuk Arsitektur Komputasi Kinerja Tinggi Modern,"*Komputasi dalam Sains & Eng.*, vol. 17, tidak. 2, 2015, hlm. 53–64.
49. M. Palyart et al., “MDE4HPC: Sebuah Pendekatan untuk Menggunakan Model-Driven Engineering dalam Komputasi Kinerja Tinggi,”*LNCS*, vol. 7083, 2012, hlm. 247–261.
50. M. Palyart et al., "HPCML: Bahasa Pemodelan yang Didedikasikan untuk Komputasi Ilmiah Berkinerja Tinggi,"*Prok. Lokakarya Internasional Pertama tentang Model-Driven Eng. untuk Performa Tinggi dan Komputasi Awan*, 2012, hlm. 1–6.
51. M. Almorsy et al., "Suite Bahasa Visual Khusus Domain untuk Pemodelan Aplikasi Perangkat Lunak Ilmiah,"*Prok. Sim. Bahasa Visual dan Komputasi Berpusat pada Manusia*(VL/HCC 13), 2013, hlm. 91–94.
52. AN Johanson et al., “SPRAT: Model Ekosistem Laut Eksplisit Spasial Berdasarkan Persamaan Keseimbangan Populasi,”*Pemodelan Ekologis*, vol. 349, 2017, hlm. 11–25.
53. AN Johanson dan W. Hasselbring, "Kombinasi Hirarki Bahasa Khusus Domain Internal dan Eksternal untuk Komputasi Ilmiah,"*Prok. Konferensi Eropa Lokakarya Arsitektur Perangkat Lunak*(ECSAW 14), 2014, hal. 17:1.
54. S. Gunther, M. Haupt, dan M. Splieth, “Agile Engineering of Internal Domain-Specific Languages with Dynamic Programming Languages,”*Prok. Konferensi Internasional ke-5 Perangkat Lunak Eng. Rayuan*, 2010, hlm. 152–168.
55. AN Johanson dan W. Hasselbring, “Efektivitas dan Efisiensi Bahasa Khusus Domain untuk Simulasi Ekosistem Laut Berkinerja Tinggi: Eksperimen Terkendali,”*Perangkat Lunak Empiris Eng.*, vol. 22, tidak. 4, 2017, hlm. 2206–2236.
56. AB Bondi,*Dasar-dasar Rekayasa Kinerja Perangkat Lunak dan Sistem: Proses, Pemodelan Kinerja, Persyaratan, Pengujian, Skalabilitas, dan Praktik*, Addison-Wesley, 2014.
57. D. Kelly, S. Smith, dan N. Meng, "Rekayasa Perangkat Lunak untuk Ilmuwan,"*Komputasi dalam Sains & Eng.*, vol. 13, tidak. 5, 2011, hlm. 7–11.
58. R. Guderlei dan J. Mayer, "Program Pengujian Pengujian Metamorfik Statistik dengan Output Acak dengan Uji Hipotesis Statistik dan Pengujian Metamorfik," *Prok. Konferensi Internasional ke-7 Perangkat Lunak Berkualitas*(QSIC 07), 2007, hlm. 404–409.
59. U. Kanewala dan JM Bieman, "Menggunakan Teknik Pembelajaran Mesin untuk Mendeteksi Hubungan Metamorfik untuk Program tanpa Uji Oracle,"*Gejala Internasional ke-24. Keandalan Perangkat Lunak Eng.*(ISSRE 13), 2013, hlm. 1–10.
60. T. Clune, H. Finkel, dan M. Rilee, “Menguji dan Men-debug Aplikasi Exascale dengan Mengejek MPI,”*Prok. Lokakarya Internasional ke-3 tentang Software Eng. untuk Komputasi Kinerja Tinggi dalam Ilmu Komputasi dan Eng.*(SE-HPCCSE 15), 2015, hlm. 5–8.
61. W. Hasselbring, “Layanan Mikro untuk Skalabilitas: Abstrak Keynote Talk,”*Prok. ACM/SPEC ke-7 di Konferensi Internasional. Kinerja Eng.*(ICPE 16), 2016, hlm. 133–134.
62. W. Hasselbring dan G. Steinacker, “Arsitektur untuk Skalabilitas, Kelincahan, dan Keandalan dalam E-Commerce,”*Prok. IEEE Internasional Conf. Lokakarya Arsitektur Perangkat Lunak* (ICSAW 17), 2017, hlm. 243–246.
63. S. Thew et al., "Rekayasa Persyaratan untuk e-Science: Pengalaman dalam Epidemiologi,"*Perangkat Lunak IEEE*, vol. 26, tidak. 1, 2009.
64. JC Garcia et al., “Pengembangan Aplikasi Ilmiah dengan Komputasi Kinerja Tinggi melalui Metodologi Berbasis Komponen dan Berorientasi Aspek,”*Ilmu Komputer Tingkat Lanjut Int'l J.*, vol. 3, tidak. 8, 2003, hlm. 400–408.

TENTANG PENULIS

Arne Johansonadalah sebagai ilmuwan data di XING Marketing Solutions GmbH. Penelitiannya berfokus pada mengadaptasi teknik rekayasa perangkat lunak untuk ilmu komputasi. Johanson menerima gelar PhD dalam ilmu komputer dari Universitas Kiel. Hubungi dia di arj@informatik.unikiel.de.

Wilhelm Hasselbringadalah profesor rekayasa perangkat lunak di Universitas Kiel. Minat penelitiannya meliputi rekayasa perangkat lunak dan sistem terdistribusi. Hasselbring menerima gelar PhD dalam ilmu komputer dari University of Dortmund. Dia adalah anggota dari ACM, IEEE Computer Society, dan Asosiasi Jerman untuk Ilmu Komputer. Hubungi dia di hasselbring @email.uni-kiel.de.