## Bab 2

### Metode Formal

* Apa itu metode formal (Definisi, Penemu, Filosofi)
* Metode Formal merupakan sebuah teknik penerapan prinsip-prinsip matematika dalam memodelkan dan menganalisis sistem ICT (Teknologi Komunikasi dan Informasi) (Baier, 2008). Dengan kata lain, Metode Formal didefinisikan sebagai ilmu matematika untuk sistem perangkat keras dan perangkat lunak komputer (Holloway, 1997). Menurut Monin, sebenarnya istilah “Teknik Formal” lebih pantas untuk digunakan dibandingkan “Metode Formal” karena teknik ini belum memiliki metodologi yang baku. Namun karena istilah Metode Formal lebih populer maka untuk selanjutnya dalam tulisan ini istilah Metode Formal lah yang akan digunakan (Monin, 2003).
* Kenapa metode formal [Holloway1997](#fn-holloway1997)
* Metode Formal merupakan salah satu teknik verifikasi yang “sangat direkomendasikan” untuk pengembangan perangkat lunak dan sistem kritikal oleh standar praktik terbaik yang dimiliki oleh IEC (Komisi Elektroteknik Internasional) dan ESA (Agensi Antariksa Eropa) (Baier, 2008). Hasil laporan investigasi yang dilakukan oleh FAA (Otoritas Penerbangan Federal Amerika) dan NASA (Administrasi Aeronautika dan Antariksa Nasional Amerika) mengenai penggunaan Metode Formal menunjukkan bahwa Metode Formal haruslah menjadi bagian dari pendidikan seluruh ilmuwan komputasi dan insinyur sistem perangkat lunak, sepertih bagaimana ilmu matematika terapan merupkan ilmu yang wajib dimiliki oleh setiap insinyur lainnya.
* Kesalahan pada unit pembagian bilangan titik mengambang (*floating point*) milik Intel pada tahun 90an menyebabkan kerugian mencapai 470 juta dolar Amerika untuk mengganti seluruh prosesor yang cacat dan juga merusak reputasi Intel sebagai pembuat chip komputer yang bisa diandalkan. Kesalahan perangkat lunak dalam sistem penanganan bagasi memundurkan pembukaan sebuah bandara di Denver selama 9 bulan yang menyebabkan kerugian sekitar 1.1 juta Dolar Amerika per hari. Kesalahan selama 24 jam pada sebuah sistem pemesanan tiket daring internasional akan menyebabkan kebangkrutan perusahaan tersebut karena hilangnya pesanan. BIla kesalahan terjadi pada sistem keamanan, akibat yang ditimbulkan bisa menjadi bencana. Kecacatan fatal pada perangkat lunak kontrol pada misil Ariadne-5, wahana antariksa Mars Pathfinder, serta pesawat-pesawat milik Airbus sudah menjadi berita utama di seluruh dunia dan menjadi kasus-kass yang terkenal. Perangkat lunak juga digunakan pada kontrol proses sistem yang kritikal pada pabrik kimia, pembangkit listrik tenaga nuklir, sistem lalu lintas, sistem penghalau badai, serta sistem penting lainnya yang bisa menimbulkan bencana dan kerugian besar jika terjadi kesalahan pada sistem tersebut. Salah satu contohnya adalah sebuah kecacatan perangkat lunak pada mesin terapi radiasi Therac-25 yang menyebabkan kematian 6 pasien kanker di antara 1985 dan 1987 disebabkan oleh overdosis paparan radiasi. Meningkatnya kebergantungan aplikasi kritikal pada pemrosesan informasi menyebabkan pentingnya meningkatkan reliabilitas dalam proses desain sistem ICT.
* Software Engineering jauh lebih tidak reliable dari other eng
* Pengembangan perangkat lunak saat ini sudah terkenal sebagai sebuah proses yang lambat dalam memberikan hasil serta sulit diprediksi dan tidak bisa diandalkan dalam operasi (Holloway, 1997). Menurut sebuah artikel yang ditulis pada 1994 oleh Wyatt Gibbs, “Studi menunjukkan bahwa untuk setiap 6 sistem perangkat lunak skala besar baru yang dioperasikan, 2 akan dibatalkan. Rata-rata waktu pengembangan perangkat lunak melampaui jadwal yang ditentukan sebanyak 50%. Perangkat lunak yang lebih besar bahkan membutuhkan waktu yang lebih lama lagi. 75% dari seluruh sistem skala besar memiliki kesalahan dalam operasi yang menyebabkan mereka tidak berfungsi sebagaimana mestinya atau bahkan tidak digunakan sama sekali.” Dibandingkan dengan disiplin keinsinyuran lainnya, teknik perangkat lunak terlihat sangat buruk. Namun ini tidak terlalu mengejutkan karena setidaknya dalam dua aspek, perangkat lunak berbeda dengan objek fisik, material, dan sistem yang ditangani oleh ilmu teknik pada umumnya.
* Pertama, pada sistem fisik perubahan yang halus pada masukan akan menghasilkan perubahan yang halus pada keluaran. Dengan kata lain, sistem fisik merupakan sebuah sistem yang kontinu. Hal ini memungkinkan perilaku sistem untuk ditentukan hanya dengan memberikan beberapa masukan yang kemudian diterjemahkan menggunakan interpolasi dan extrapolasi untuk menentukan perilaku sistem pada masukan yang tidak dites. Sistem perangkat lunak, berbeda dengan sitem fisik, merupakan sebuah sistem yang diskontinu. Perubahan kecil pada masukan bisa menghasilkan perubahan yang signifikan pada beberapa penentuan keputusan dalam perangkat lunak dan menyebabkan keluaran yang sangat berbeda. Hasilnya, interpolasi atau ekstrapolasi tidak bisa digunakan untuk memprediksi keluaran dari masukan yang tidak dites karena resiko yang tinggi dan bisa menyebabkan hasil yang tidak diinginkan. Selain itu sistem komputer semakin hari memiliki kompleksitas yang semakin tinggi. Dengan naiknya kompleksitas maka semakin banyak pula kemungkinan kecacatan desain yang akan terjadi. Pada sisi baiknya, sistem perangkat lunak cenderung tidak memiliki keausan tidak seperti sistem fisik. Sehingga jika sistem perangkat lunak sudah dibuktikan reliabilitasnya maka reliabilitas itu bisa bertahan selama bertahun-tahun tanpa mengalami keusangan.
* Kenapa metode formal jarang dipake
* Walalupun sudah nyata bahwa pengembangan perangkat lunak memiliki masalah yang serius dalam menangani kecacatan dalam produk dan Metode Formal bisa menjadi solusi untuk menyelesaikan masalah tersebut, pada kenyataanya Metode Formal tidaklah diajarkan di universitas-universitas dan setelah 40 tahun Metode Formal masih sangat jauh dari pengaplikasian pada pemrograman sehari-hari. Menurut Ricardo Pena (Pena, 2016), ada beberapa alasan atas situasi ini:
* 1. Dibutuhkan waktu dan usaha yang cukup besar untuk memformalisasi spesifikasi pada kebutuhan dengan menuliskan prekondisi dan poskondisi untuk setiap kebutuhan.
* 2. Dibutuhkan usaha yang lebih besar lagi untuk menentukan *loop invariant* (variabel yang selalu konstan selama sebuah *loop* berjalan) serta menentukan asumsi lain yang kritikal dalam program.
* 3. Bahkan setelah menuliskan seluruh hal tersebut, untuk menuliskan pembuktian program tersebut secara manual dibutuhkan ruang bahkan mencapai 5 sampai 10 kali panjang program yang dibuktikan.
* Pada umumnya, verifikasi formal bisa memberikan manfaat yang jelas namun investasi usaha yang dibutuhkan untuk melakukan hal tersebut sangatlah tinggi. Hal ini menyebabkan Metode Formal jarang digunakan dan hanya dilakukan untuk program kritikal yang akan memberikan kerugian yang sangat besar untuk setiap kesalahan yang terjadi sehingga investasi yang besar untuk melakukan Metode Formal bisa dijustifikasi. Usaha besar yang dibutuhkan untuk melakukan Metode Formal ini harus dikurangi untuk bisa memungkinkan Metode Formal untuk digunakan lebih luas lagi kepada lebih banyak pemrogram yang membutuhkannya.
* Contoh metode formal yang sudah dipake
* Kelemahan Metode Formal

#### Justifikasi Logis [Huth2004](#fn-huth2004)

* Aturan Logis
* Logika Propositional
* Logika Predikat
* SAT and SMT Solver

#### Spesifikasi Program

* Desain sebelum kode
* Test Driven Development

#### Verifikasi Program

* Verifikasi dengan Model Checking (Model Based)
* Verifikasi dengan Semantic Entailment (Proof Based)

### Pemrograman Fungsional

* Paradigma Pemrograman [Harper2017](#fn-harper2017)
* Paradigma Pemrograman Fungsional [Sabry1998](#fn-sabry1998)
* Paradigram Pemrograman Fungsional Murni
* Kenapa Fungsional Murni itu excellent buat Metode Formal [Turner1985](#fn-turner1985) [Hughes1989](#fn-hughes1989) [Butler1995](#fn-butler1995)
* Perbandingan dengan Paradigma lain

#### Haskell [O'Sullivan2008](#fn-o'sullivan2008)

* Definisi, Penemu, Filosofi [Hudak2007](#fn-hudak2007)
* Fungsional Murni
* Lazy Computing
* Hard Typing
* Program yang dibuat menggunakan Haskell (Hackage)
* How to Specify and Verify (Haskell for Specification)

#### Liquid Haskell [Pena2017](#fn-pena2017)

* Definisi, Penemu, Filosofi  
  Liquid Haskell merupakan sebuah type-checker untuk bahasa fungsional Haskell yang bertujuan untuk membantu memudahkan pemrogram untuk memverifikasi program mereka dengan usaha sesedikit mungkin. Untuk itu type-checker ini menggunakan sebuah teknologi bernama Liquid Types yang merupakan singkatan dari Logically Qualified Data Types. Pada dasarnya, teknologi ini menggunakan SMT Solver untuk secara instan memverifikasi sebuah program yang sudah dituliskan spesifikasinya dalam Liquid Types. Dengan itu, tidaklah lagi diperlukan waktu yang panjang untuk memverifikasi program dengan menuliskan persamaan matematika dengan tangan karena hal itu sudah bisa ditangani oleh komputer.
* Liquid Types
* Usage of SMT Solver in Liquid Haskell
* Kelemahan Liquid Haskell
* Refinement, Inference, and Polymorphism
* Liquid haskell Case Studies

#### Concurrency in Haskell