Metodologi Desain Sistem

Kuliah#2 TSK-612 Sistem Embedded Terdistribusi - TA 2011/2012

Eko Didik Widianto

Teknik Sistem Komputer - Universitas Diponegoro

Review Kuliah

- Pokok bahasan di kuliah #1
  - Penjelasan tentang sistem embedded terdistribusi
  - Deskripsi, tujuan, sasaran dan materi kuliah TSK612 Sistem Embedded Terdistribusi
- Umpan balik
  - Jelaskan tentang sistem embedded!
  - Jelaskan tentang sistem terdistribusi!
  - Jelaskan tentang sistem embedded terdistribusi!
- Link
  - Website: http://didik.blog.undip.ac.id/2012/03/06/kuliah-tsk-612-sistem-embedded-terdistribusi-2011/
  - Email: didik@undip.ac.id
- Acknowledgement:
  - Beberapa gambar yang ada di slide ini diambil dari http://www.ece.cmu.edu/~ece649/[ECE649]

Tentang Kuliah

- Pokok bahasan di kuliah #2
  - Metodologi desain sistem: waterflow, v-model, agile
  - Penerapan metodologi untuk mengembangkan sistem embedded terdistribusi
- Kompetensi dasar
  - [C2] mahasiswa akan memahami metodologi desain secara umum, meliputi waterflow, v-model, spiral dan agile
  - [C3] mahasiswa akan mampu mengaplikasikan metodologi tersebut dalam mendesain suatu sistem embedded terdistribusi
- Link
  - Website: http://didik.blog.undip.ac.id/2012/03/06/kuliah-tsk-612-sistem-embedded-terdistribusi-2011/
  - Email: didik@undip.ac.id

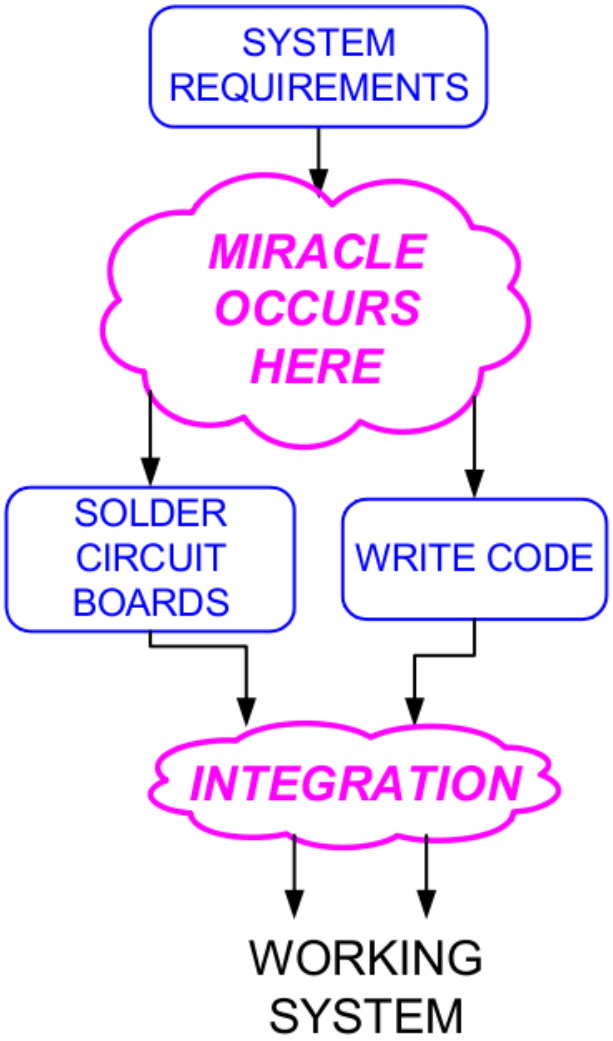
Bahasan

General Quote

- Mengetahui cara untuk mensolder tidak akan membuat seseorang menjadi insinyur elektronika
- Mengetahui cara untuk menuliskan kode tidak akan membuat seseorang menjadi insinyur software
- Mengetahui cara untuk mensolder dan menulis kode tidak akan membuat seseorang menjadi insinyur sistem embedded

Bahasan

Metodologi Desain secara Umum



- Kebutuhan sistem merupakan bentuk penugasan projek yang perlu ditulis
  - Asumsi: sempurna
  - Tapi, tidak akan pernah lengkap dan/atau sempurna saat memulai projects
- Terdapat banyak langkah antara kebutuhan dan implementasi
  - Akankah memulai menyolder/menyambungkan jalur tanpa skematik?
  - Akankah menulis kode atau VHDL tanpa sebuah desain?
- Sebagian besar projek, integrasi dapat diselesaikan dalam waktu 50% dari jadwalnya

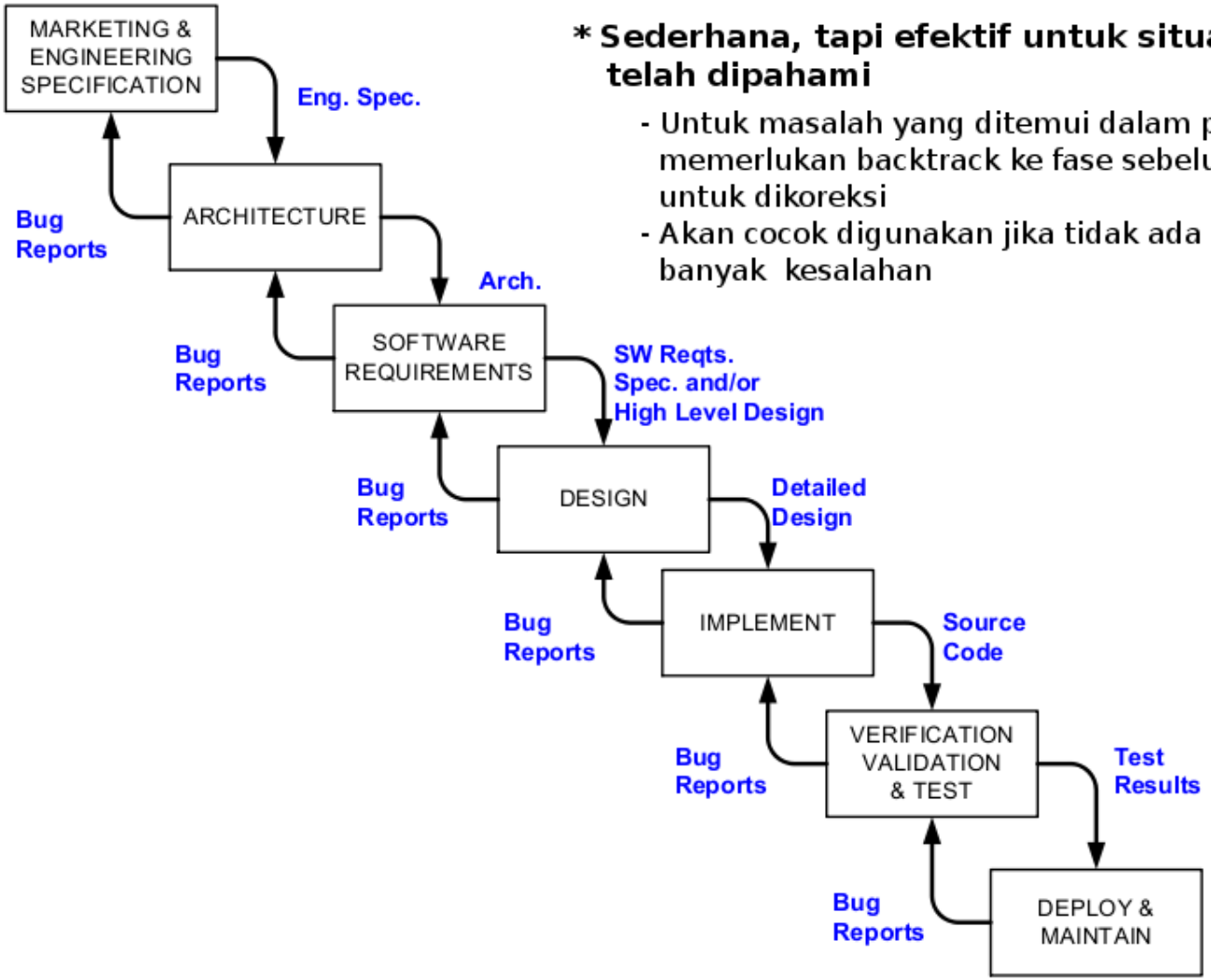
Metodologi Pengembangan

- Rencana tertulis yang menyatakan bagaimana pengembangan sistem akan dilakukan
  - Pendekatan yang diambil
  - Bagaimana requirement dibuat dan dikelola
  - Bagaimana arsitektur didefinisikan dan dikembangkan
  - Bagaimana perancangan dan implementasi akan dilakukan dan didokumentasikan
  - Bagaimana pengujian direncanakan dan dieksekusi
  - Bagaimana evaluasi dilakukan
  - Aspek lain: pemeliharaan, manajemen projek, pengelolaan SDM, dan lain-lainnya
- Tiap langkah mendefinisikan aktivitas, input dan output
  - Berupa diagram kotak dan garis panah dengan labelnya masing-masing
  - Label garis panah terkait dengan luaran (misalnya dokumen), yang menjadi tolak ukur untuk melangkah ke aktivitas berikutnya

Siklus Pengembangan Sistem

- Metodologi pengembangan tersebut dapat dilakukan dengan pendekatan berikut:
  1. Waterfall: berurutan mulai dari spesifikasi sampai pemasangan dan pemeliharaan
  2. V-model: versi waterfall yang dimodifikasi untuk mengejar modularity subsistem
  3. Spiral model: mengkombinasikan elemen dalam tahap desain dan prototyping sebagai upaya memanfaatkan kelebihan pendekatan top-down dan bottom-up
  4. Agile model: self-organizing dan cross-functional team. Pengembangan iterative dan incremental

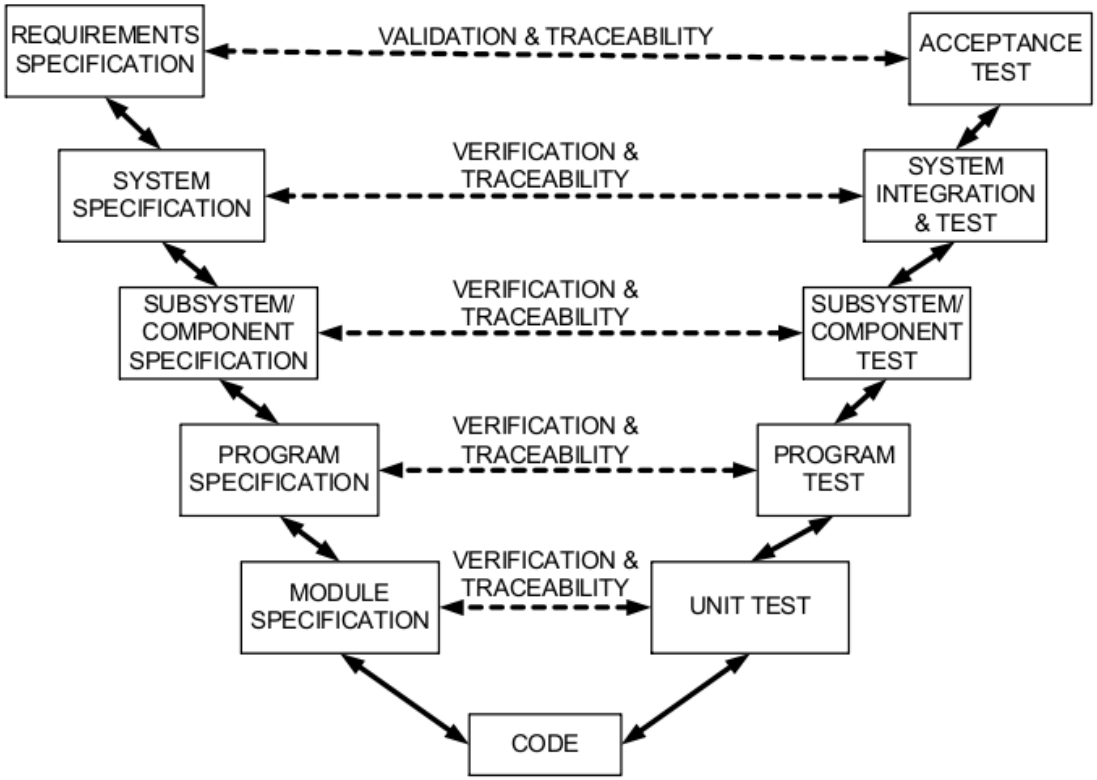
Model Pengembangan Waterfall



* **Sederhana, tapi efektif untuk situasi yang telah dipahami**
  - Untuk masalah yang ditemui dalam proses, memerlukan backtrack ke fase sebelumnya untuk dikoreksi
  - Akan cocok digunakan jika tidak ada terlalu banyak kesalahan

Model Pengembangan Kurva-V

- Merupakan versi modifikasi dari model waterfall
  - Untuk mengeksploitasi modularitas subsistem
  - Sisi kiri: definisi projek, sisi kanan: pengujian dan integrasi, bagian bawah: implementasi



Perspektif Rekayasa Software

- Mengapa perlu metode rekayasa software?
  - Pembuatan software telah menjadi komponen biaya utama
  - Metode dan pendekatan khusus diperlukan untuk mengelola kompleksitas software
    - Untuk mengurangi cost
    - Untuk menjadi kesesuaian jadwalnya
    - Untuk meningkatkan kualitas sehingga defect software tidak merugikan perusahaan
  - Aplikasi embedded memerlukan standar yang lebih tinggi daripada software desktop
  - Perspektif yang perlu diambil
    - Proses yang baik diperlukan untuk mendapatkan software yang bagus
      Review desain merupakan aktivitas penting untuk memulai mengerjakan sesuatu dengan benar (*doing right*)

Perlunya Requirement

- Memastikan sistem melakukan fungsinya dengan benar (*the right things*)
- Memastikan sistem tidak melakukan fungsi yang salah
- Menuliskan daftar semua konstrain yang harus dipenuhi oleh produk
  - realtime, sumber daya listrik, ukuran, berat, etc
- Memastikan sistem cukup kompleks, namun melebihi yang diperlukan
  - Fungsionalitas dipenuhi
  - Meminimalkan biaya/meminimalkan kompleksitas
  - Dengan ekstensi, requirement yang sederhana lebih baik

Sudut Pandang Kebutuhan

- Requirement Sistem/Bisnis
  - Fungsional: lift harus dapat mengangkat semua orang dengan kecepatan yang memadai
    - Lift mungkin harus lebih cepat dari lift kompetitor
  - Non-fungsional: lift harus meningkatkan profit kontrak pemeliharaan
  - Constraint: produk harus kompetibel dengan jaringan pemeliharaan gedung yang ada
- Requirement Marketing/Engingeering
  - Fungsional: lift harus mempunyai kecepatan puncak 11.3 meter/detik (jika kecepatan tercepat kompetitor 11.2 m/dt)
  - Non-fungsional: lift harus mempunyai antarmuka diagnostik
  - Contraint: lift harus kompatible dengan jaringan standar (BACnet)

Requirement Teknis

- Kebutuhan fungsional (FRS, functional requirement specs)
  - Pelepasan tombol X sebaiknya menghidupkan lampu Y dalam waktu 200ms
- Kebutuhan non-fungsional
  - “Mean time between unsafe operating situations for each rail signal shall be greater than 250,000 years” (example from a subway system)
  - “Mean time to repair a single component failure shall be less than 30 minutes after arrival of mechanic bringing standard tool set.” (typical jet aircraft)
  - “Elevator shall deliver at least 1000 passenger*floors/minute at up-peak”
- Constraints
  - Specifies a required technical or other approach
    - “.NET technology shall be used”
  - Specifies regulatory or other constraints on solution space/design process
    - “System shall conform to requirements of DO-178b” (FAA software process)
- Assumptions/operating conditions
  - “Assume no power outage lasts more than 30 minutes”

Proses Pengembangan Kebutuhan

- Proses pengembangan kebutuhan di sistem embedded
  1. Elicitation: Identify business/system requirements (B100)
  2. Create architecture / allocate functions to subsystems (B200-OVS)
  3. Create scenarios / use cases (B200-SWS)
  4. Create detailed behavioral requirements (B200-FRS)
- Requirements Traceability & Risk Management
  1. Create traceability matrices to trace:
     - System req., behavioral req., implementations, integration tests, System req., acceptance tests
  2. Simulate system to check global/emergent behaviors
  3. Prototype system to check allocation-based properties

1. Requirement Elicitation

Identify business/system requirements

- Customer may provide requirements in *request for quote* (RFQ)
- Vendor may need to interview customer and extract requirements
  - Requirements phase may precede design phase under a separate contract
- You might have engineering judgment (“guessing”)
  - “I’ll know it when I see it”
  - “Same as last time except better”
- This is one place where being a good writer + clear thinker really helps!
  - It can help a lot to have a professional technical writer on the requirements team

Keyword

- Behaviors/Constraints:
  - Shall = system has to do it 100% of the time unless specifically excepted
  - Should = desirable that system does it whenever reasonable to do so
  - Can = system can do something, but no particular incentive to implement
- User Actions:
  - Must = user has to do this (same as “shall”, except for user, not computer)
  - May = user can exhibit this behavior, but does not have to
- Environment words:
  - Will = designer can count on environment being this way
  - Might = designer has to accommodate situation, but can’t count on it

Contoh: High-Level Requirement

- (**What does it do?**): All passengers shall be delivered to desired destination floor in a timely manner.
- (**Safety**): Any unsafe condition shall cause an emergency stop
  - An emergency stop should never occur
- (What metrics matter?): Performance should be optimized to extent possible, including customer-specified weightings for:
  - Delivery times:
    - Maximum end-to-end passenger delivery time
    - Maximum passenger waiting time
    - Average end-to-end passenger delivery time
    - Average passenger waiting time
- Note: SHALL = mandatory, SHOULD = desirable

2. Membuat Arsitektur

- Class/Component diagram:
  - Objects and interfaces
  - However, architecture is both objects AND interfaces, so there is more to it
- Assume you have a list of objects in the system
  - Elevator car
  - Doors
  - Hall call buttons
  - Car call buttons
  - Directional lanterns

3. Membuat Skenario / Use Case

- First: high level flows through the system
  - Idea is to create a flow chart of actions experienced by actors / “object lifecycles”
    - Elevator passenger from initial entry into system until final delivery
    - Driver from entering car to leaving car
  - Each block of flow chart yields one or more use cases
    - (Object-oriented/other approaches possible, but flow charts are usual practice)
- Second: “building-block” use cases catching snippets of functionality
  - Multiple scenarios possible for each use case (elevator example):
    - Passenger calls car
      Scenario: Presses button once
      Scenario: Button has already been pressed by someone else
    - Passenger enters car
      Scenario: Successful first try
      Scenario: Gets bumped by door and exits instead of entering, then retries
    - Passenger calls destination
      ...

4. Membuat Detail Kebutuhan Perilaku

Behavioral Requirement:

- 2.3.1 When the first probe temperature exceeds 80 degrees C +/- 1 degree C, the heating element shall turn off within 100 msec.
  Rationale: this is a software safety mechanism to avoid actuating safety relief valve
  ....

Constraint:

- 5.7.1 Program memory shall be no more than 60% full at initial product release.
  Rationale: this leaves room for software expansion and avoids software costs caused by nearly-full memory.

Membuat Detail Kebutuhan Perilaku

1. List subsystems
   - In a fine-grain distributed system, this is sensors+actuators+objects
     - “Other objects” are usually compute-nodes such as dispatcher
   - Actors included to provide environmental model and interface
2. Create a database of behavioral requirements for each subsystem
   - Replication
   - Instantiation
     - Initial conditions
     - When are dynamic objects are created?
   - Assumptions about how other modules behave
   - I/O interface (list of sensor/actuator interfaces)
   - State (private variables useful for describing behavioral memory)
   - Constraints
     - Non-functional requirements; safety interlocks
   - Behaviors
     - Functional requirements

Kebutuhan Perilaku

1. Event-driven system (think “interrupts”):
2. Time-triggered system is different (think “polled I/O”):

Lisensi

Creative Common Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

- Anda bebas:
  - untuk **Membagikan** — untuk menyalin, mendistribusikan, dan menyebarkan karya, dan
  - untuk **Remix** — untuk mengadaptasikan karya
- Di bawah persyaratan berikut:
  - **Atribusi** — Anda harus memberikan atribusi karya sesuai dengan cara-cara yang diminta oleh pembuat karya tersebut atau pihak yang mengeluarkan lisensi.
  - **Pembagian Serupa** — Jika Anda mengubah, menambah, atau membuat karya lain menggunakan karya ini, Anda hanya boleh menyebarkan karya tersebut hanya dengan lisensi yang sama, serupa, atau kompatibel.
- Lihat: Creative Commons Attribution-ShareAlike 3.0 Unported License