

Praktikum Rekayasa Perangkat Lunak

Pertemuan 11

→ Hari ini belajar apa??

**Fundamental Pengujian
Perangkat Lunak**

Apa itu pengujian?

Metodologi Pengujian

Teknik, level, tipe pengujian

Merancang Pengujian

TDD, strategi testing, test case



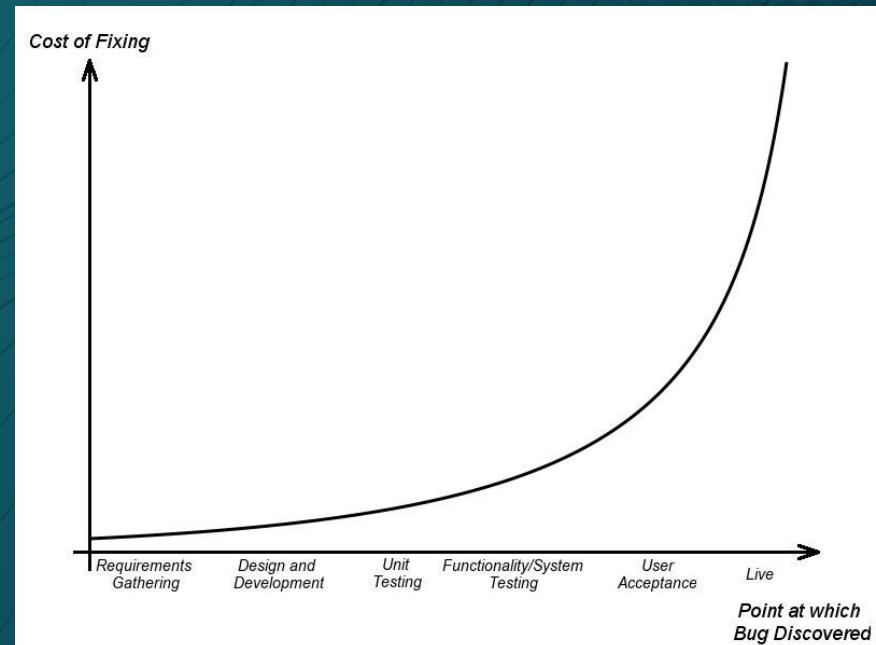
1. Fundamental Pengujian Perangkat Lunak

Memahami Pengujian Perangkat Lunak

- Pengujian (*testing*) adalah proses pemeriksaan aplikasi untuk memastikan aplikasi telah memenuhi **persyaratan** yang dirancang dan memenuhi **kualitas** yang diharapkan.
- Pengujian ditujukan untuk mengukur kualitas aplikasi atau proyek.
- *Developer* perlu meyakini adanya **bug** atau **defect** (kecacatan) yang belum ditemukan.
- Pengujian membantu menemukan dan memperbaiki cacat tersebut.
- **Bug** adalah error dalam kode atau logika program yang menyebabkan program tidak berfungsi atau memproduksi hasil yang tidak sesuai.

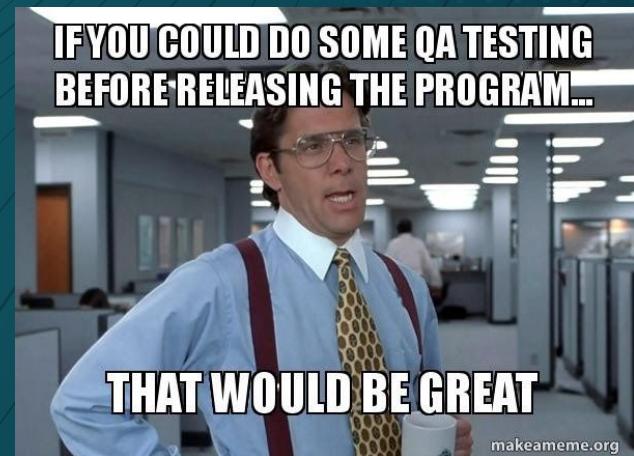
Mengapa pengujian penting?

- Mengurangi biaya pengembangan program.
 - Perkiraan untuk biaya untuk memperbaiki kecacatan yang tidak terdeteksi sampai program beroperasi sekitar 40-100 kali lebih mahal dibanding memperbaikinya di awal.
- Memastikan aplikasi berperilaku persis seperti yang dimaksudkan.



Mengapa pengujian penting bagi user?

- Pengujian di awal akan menghasilkan *software* dengan fungsi dan keandalan yang lebih baik serta *cost of ownership* yang lebih rendah.
 - *User* tidak akan menemukan *bug* ketika menggunakan *software*.
 - Mengurangi *downtime* akibat perbaikan dan penginstalan *update*.
 - Mengurangi *user support* dan *training*.
 - Meningkatkan kepuasan *user*.

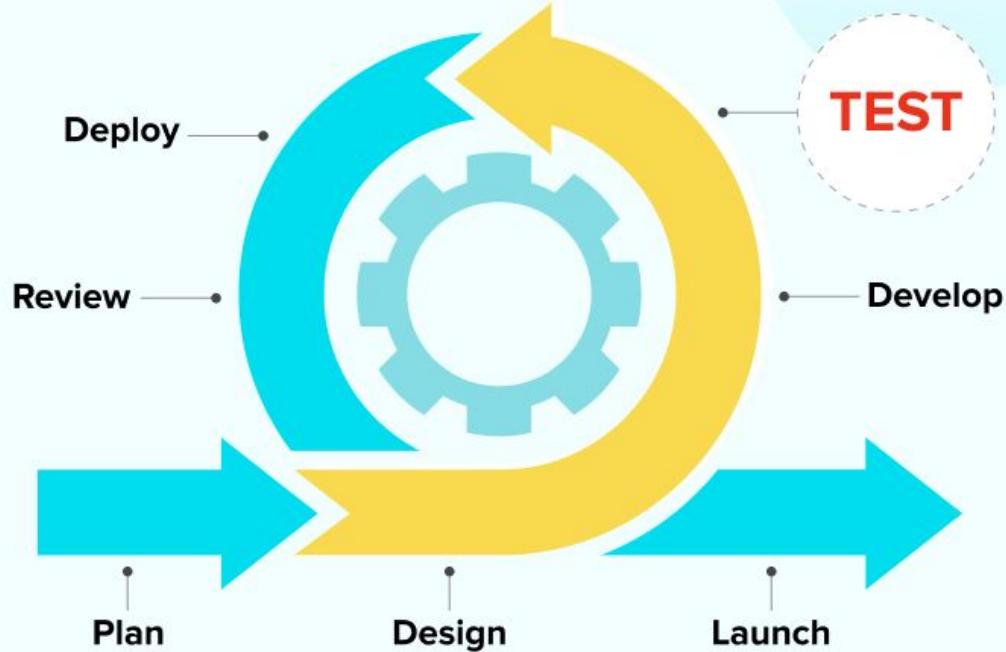


Mengukur Kualitas Software

Beragam jenis metrik/ *metrics* (standar pengukuran) untuk mengukur kualitas perangkat lunak. Metrik yang umum antara lain:

- *Performance metrics*, seperti waktu yang dibutuhkan untuk menyelesaikan suatu proses atau jumlah *space* yang dibutuhkan oleh aplikasi.
- *Reliability metrics*, seperti jumlah total bug, atau "*defect density*", yaitu jumlah bug dibagi jumlah baris kode.

Posisi Pengujian dalam Agile Methodology





2. Metodologi Pengujian



Metodologi Pengujian

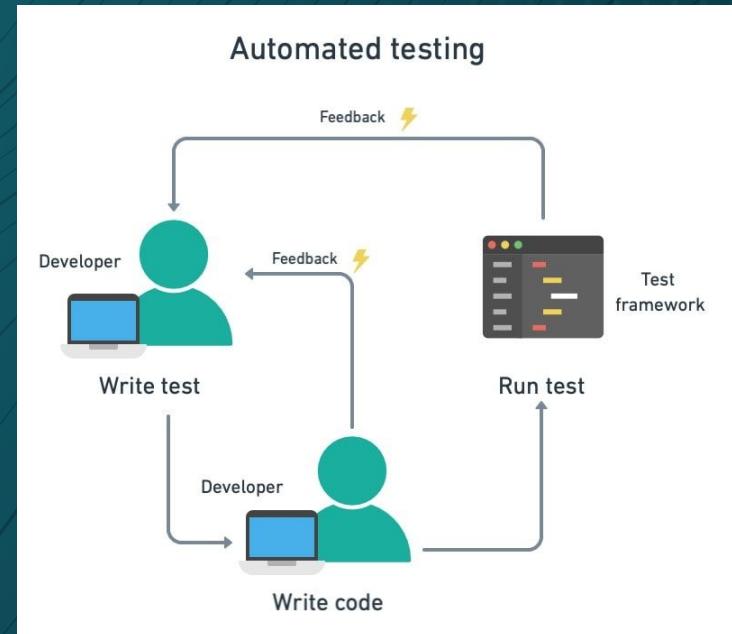
1. Teknik Pengujian

Pengujian Manual [Manual Testing]

- Merupakan pengujian secara tradisional dengan mencoba programnya dan melihat apakah hasilnya sesuai.
- Dalam pengujian manual, penguji berperan sebagai *user* dan bertugas memeriksa perilaku yang tidak diharapkan.
- Pengujian secara manual dapat menggunakan **rencana pengujian/ test plan** dengan **kasus uji/ test case** tertentu untuk memastikan pengeksplorasiyan dilakukan secara menyeluruh.
- Penguji dapat merupakan *developer* maupun *non-developer*.

Pengujian Otomatis [Automated Testing]

- Pengujian otomatis menggunakan *software* khusus untuk pengujian (misalnya Microsoft Test Manager, Selenium, Katalon Studio, dst) untuk mengontrol dan melacak satu atau beberapa pengujian yang dilakukan secara otomatis.
- Pengujian dapat dibuat dan dikonfigurasi untuk dijalankan setiap kali ada versi baru (atau segmen *source code*) yang dibuat.



Black Box Testing

- Merupakan pengujian yang dilakukan **tanpa mengetahui cara kerja internal** sistem yang sedang diuji.
- Mensimulasikan pengalaman *end user*.
- Penguji **tidak mengetahui** cara kerja kode. Penguji hanya akan memberikan *input* dan memeriksa *output*. Penguji **tidak perlu tahu** cara memprogram.

Contoh skenario:

- Menguji apakah *user interface* memenuhi semua persyaratan dan berfungsi.
- Menguji berbagai jenis *input* (termasuk *input* di luar kisaran yang diharapkan, seperti memasukkan angka negatif).
- Melakukan *load testing* atau *stress testing* ke sistem.
- Menguji keamanan proyek atau sistem.

White Box Testing

- Dilakukan dengan **memeriksa source code aplikasi** terhadap skenario kegagalan.
- Dibuat oleh seseorang yang **memahami** kode aplikasi dan menyiapkan *test case* untuk memastikan bahwa komponen kode berfungsi sesuai dengan spesifikasi.

Contoh skenario:

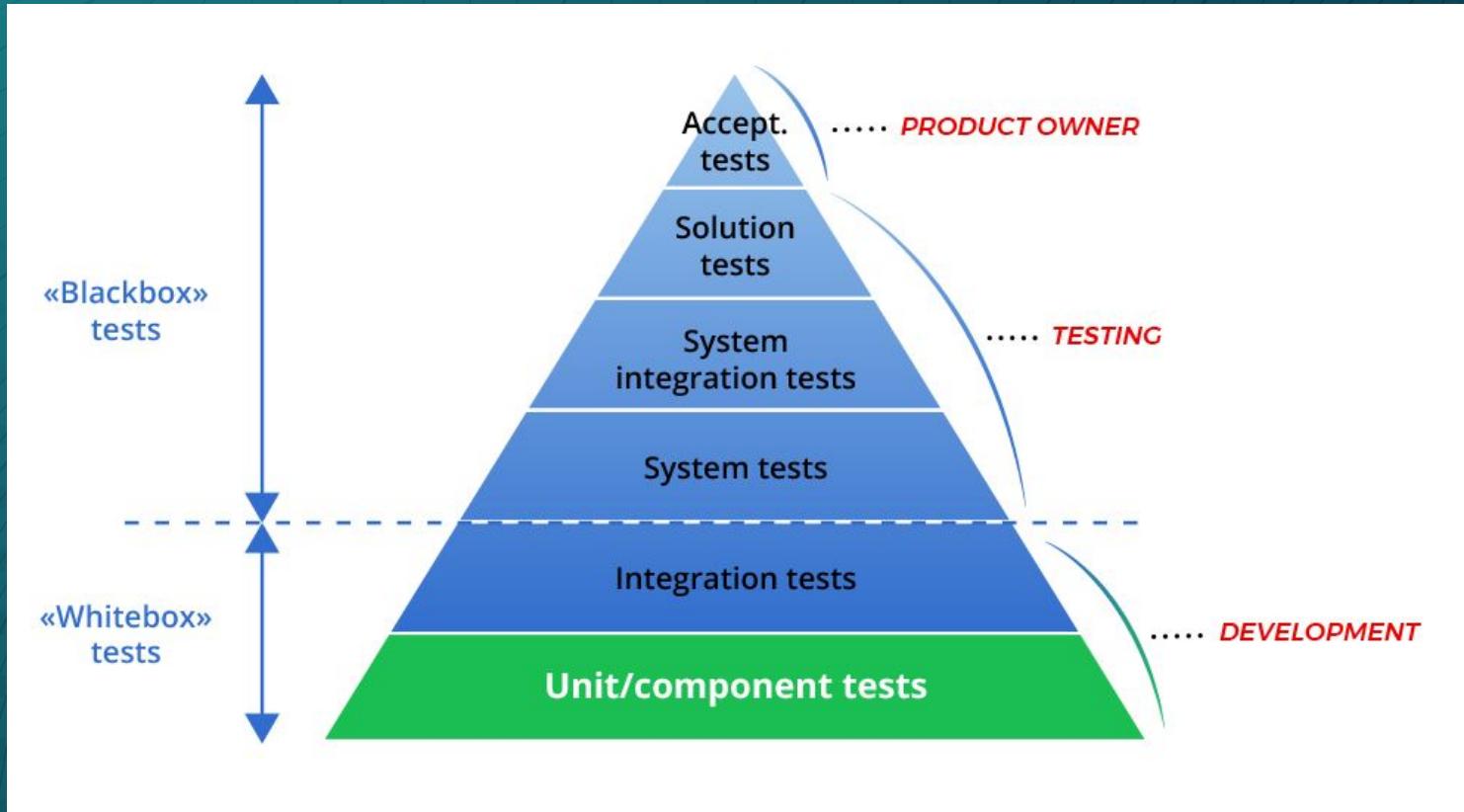
- Menguji *internal subroutine* atau *functions* pada *source code*.
- Menguji tingkah laku *loop* dan *conditional statement*.
- *Performance testing* terhadap jalur kode atau algoritma yang dibuat.



Metodologi Pengujian

2. Level Pengujian

Level Pengujian



Unit Testing

- Merupakan pengujian otomatis untuk memverifikasi fungsionalitas pada tingkat ***component, class, method, atau property.***
- Tujuannya adalah untuk mengambil bagian terkecil dari kode yang dapat diuji dalam aplikasi, mengisolasinya dari sisa kode, dan menyimpulkan apakah bagian tersebut berperilaku persis seperti yang diharapkan.
- Setiap unit diuji secara **terpisah** sebelum diintegrasikan ke dalam komponen-komponen untuk menguji *interface* antar unitnya.
- *Unit test* harus ditulis **sebelum** (atau **segera setelah**) *method* ditulis dan biasanya dibuat oleh *developer*.

Contoh Unit Testing

- Misalnya aplikasi *mobile* yang menerima tinggi dan berat pengguna untuk menghitung *Body Mass Index* (BMI) dan memberikan rekomendasi terhadap kesehatan pengguna dari hasil BMI tersebut. Penghitungan BMI menggunakan tinggi dalam satuan inci, namun pengguna akan memasukkan tinggi dalam satuan kaki plus inci (mis. 5'6").
- Unit testing akan dilakukan pada *method* atau *function* yang mengonversi kaki plus inci menjadi total inci.
- *Method* untuk menghitung BMI akan dimasukkan ke unit test yang berbeda.

Integration/ Component Testing

- Setiap unit akan membentuk komponen yang lebih besar dalam sistem. Dua unit atau lebih yang telah diuji digabungkan menjadi komponen terintegrasi dan dilakukan pengujian.
- Bertujuan untuk mengidentifikasi masalah yang terjadi **saat unit digabungkan**.
- Error yang muncul bisa jadi terkait dengan antarmuka antar unit daripada di dalam unit itu sendiri sehingga dapat memudahkan *developer* untuk mencari dan memperbaiki *defects*.

Contoh Integration Testing

- Misalnya pada aplikasi *mobile* tadi kita sudah selesai membuat *method* untuk menghitung total inci serta *method* lain untuk menghitung BMI. Kemudian kita membuat *method* ketiga yang akan mengambil input pengguna, memanggil *method* pertama, lalu memanggil *method* kedua.
- Integration testing akan dilakukan pada *method* ini yang berisi integrasi dari tiap-tiap unit di dalamnya.

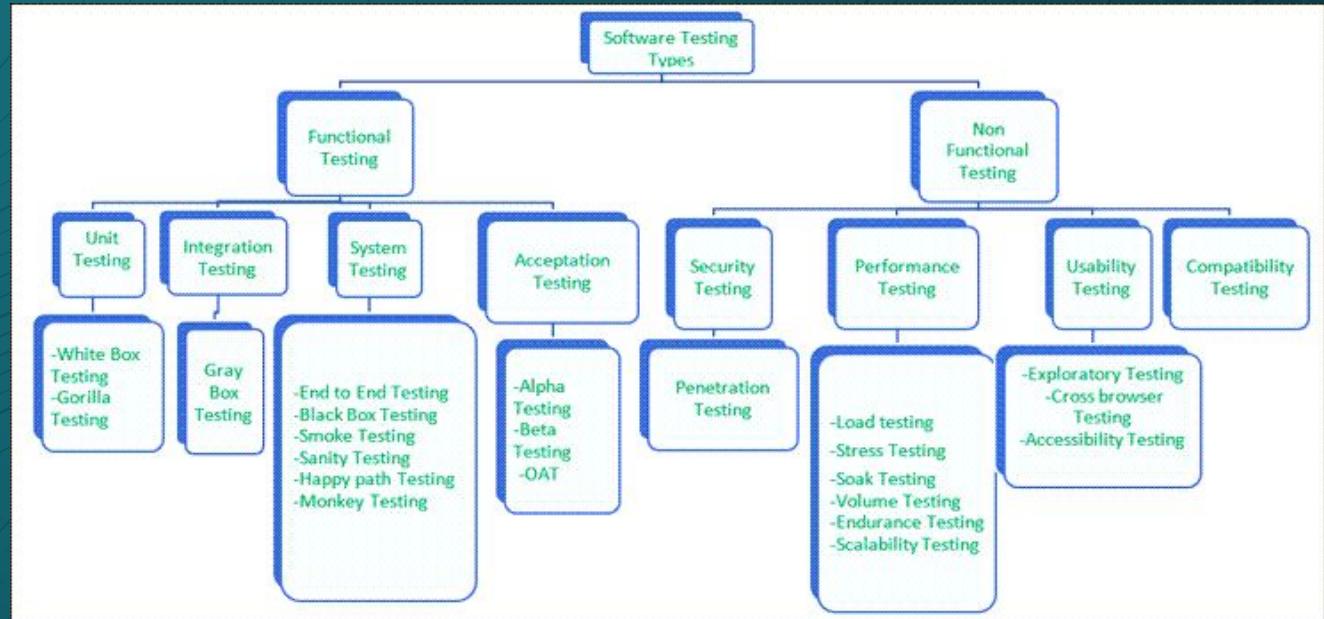


Metodologi Pengujian

3. Tipe Pengujian

Macam-macam tipe pengujian

- Functional
- Performance
- Regression
- Security
- Stress
- Accessibility
- Usability
- Localization



Regression Testing

- Setiap kali ada perubahan yang dilakukan pada proyek, ada kemungkinan bahwa kode yang ada mungkin tidak lagi berfungsi dengan baik, atau *bug* yang sebelumnya tidak ditemukan akan muncul dengan sendirinya. *Bug* semacam ini disebut **regresi**.
- Untuk menangkap kecacatan ini, seluruh proyek harus dilakukan pengujian regresi, yakni **pengujian ulang lengkap** dari program yang dimodifikasi dibanding hanya menguji unit yang dimodifikasi untuk memastikan bahwa tidak ada error yang terjadi **setelah** modifikasi.

Stress Testing/ Load Testing

- Pengujian berskala kecil misalnya pada satu *user* yang menjalankan aplikasi atau penggunaan database dengan transaksi *record* sedikit mungkin tidak mengungkapkan masalah yang mungkin terjadi saat aplikasi digunakan dalam kondisi IRL.
- *Stress testing* mendorong batas fungsional sistem dengan menempatkan sistem pada **kondisi ekstrim**, seperti volume transaksi data yang sangat tinggi atau sejumlah besar pengguna mengakses aplikasi secara bersamaan
- *Automated test* memungkinkan *stress testing* yang ketat tanpa menggunakan banyak tenaga manual.

Performance Testing

- Berfungsi untuk mengukur ***responsiveness, throughput, reliability, dan scalability*** dari sistem di bawah beban kerja tertentu. Dalam aplikasi web, pengujian kinerja seringkali terkait erat dengan *stress testing* untuk mengukur lag dan respons di bawah kondisi beban yang berat.
- Di aplikasi lain (misalnya aplikasi *desktop* dan *mobile*), pengujian ini mengukur kecepatan dan utilisasi sumber daya seperti ruang penyimpanan dan memori.

Security Testing

- Bertujuan memvalidasi **keamanan** aplikasi dan mengidentifikasi **potensi kerentanan sistem**.
- Biasanya dilakukan menggunakan pendekatan *black box* dari pakar keamanan yang tidak memiliki pengetahuan tentang internal sistem untuk menyelidiki lubang dan kelemahan pada aplikasi.

Usability Testing

- Bertujuan mengevaluasi proyek dengan mempelajari bagaimana **user menggunakan aplikasi** yang dibangun. Contohnya adalah:
 - Mengukur berapa lama waktu yang dibutuhkan pengguna untuk menyelesaikan sebuah *task*.
 - Melacak berapa banyak klik atau tindakan yang diperlukan untuk menyelesaikan *task* atau mengakses fitur.

Localization Testing

- Lokalisasi yakni menerjemahkan UI dan mengubah beberapa pengaturan agar **cocok untuk wilayah user yang berbeda.**
 - Misalnya, proyek yang dilokalkan untuk negara Perancis berarti mengubah bahasa ke Perancis dan unit pengukuran ke sistem metrik.
- Bertujuan memeriksa kualitas pelokalan produk untuk wilayah target tertentu.
- Berfokus pada UI dan area lain yang terpengaruh oleh pelokalan (persiapan dan pemutakhiran di lingkungan yang dilokalkan, kompatibilitas perangkat keras berdasarkan wilayah target, dll.).

Accessibility Testing

- Difungsikan untuk memvalidasi **dukungan** aplikasi terhadap **user dengan kebutuhan khusus**.
- Cakupan aksesibilitas yakni:
 - *Compliance* - apakah memenuhi kebutuhan terhadap peraturan aksesibilitas?
 - *Effectiveness* - dapatkah *user* dengan disabilitas menggunakan aplikasi?
 - *Usefulness* - apakah aplikasi menyediakan fungsionalitas yang cukup untuk penyandang disabilitas?
 - *Satisfaction* - bagaimana persepsi penggunaan aplikasi oleh *user* disabilitas?
- Dapat melibatkan *usability testing* terhadap *user* disabilitas dengan perangkat bantu.

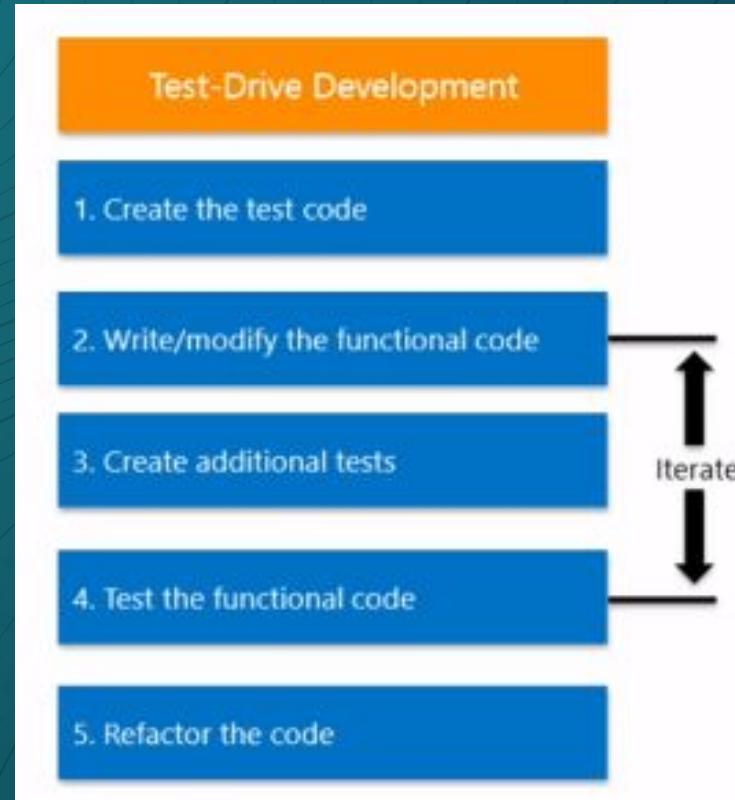


3. Merancang Pengujian

Test-driven Development (TDD)

- Teknik yang menggunakan ***unit test*** untuk memandu pembuatan ***software***. *Developer* perlu membuat *test case* **sebelum** membuat kode. *Developer* kemudian mengimplementasi fungsionalitas agar melewati *test case* yang dibuat.
- Memiliki banyak manfaat, diantaranya:
 - Baik untuk fase pengembangan dimulai dengan kebutuhan yang belum jelas.
 - Mendorong kode yang *loosely coupled*, yakni setiap unit berdiri sendiri karena memang disusun independen dari unit lain.
 - *Test case* bertindak sebagai dokumentasi.
 - Penggunaan *automated testing* yang intensif akan mengurangi waktu pengujian dan memungkinkan *regression test* yang cepat.

Proses Test-driven Development



Strategi dan Lingkup Testing

- Seawal mungkin, pengembang perlu mendiskusikan dan menetapkan strategi pengujian proyek. Strategi biasanya ditentukan atau dipengaruhi oleh **metodologi** pengembangan yang digunakan, seperti pengembangan *waterfall* atau *agile*.
- Berdasarkan strategi ini, tim mengembangkan satu atau lebih rencana pengujian. Setiap rencana pengujian memiliki ruang lingkupnya sendiri:
 - Seluruh bagian proyek dapat memiliki rencana pengujian.
 - Dalam pengembangan *agile*, **setiap sprint** harus memiliki rencana pengujian.
 - Untuk model pengembangan lainnya, pengujian seringkali didasarkan pada fitur dibanding *sprint*.

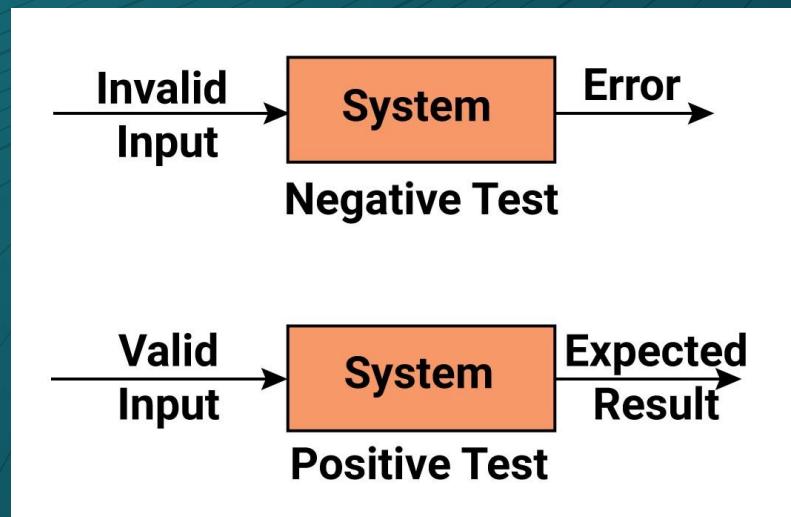
Pertimbangan Desain Test Case

- Dalam banyak situasi, kemungkinan permutasi data aplikasi dan kemungkinan jalur menyusuri kode ataupun fitur sangat besar.
- Desain *test case* yang baik adalah memilih *test case* yang memiliki probabilitas tinggi untuk menangkap sebagian besar kecacatan.



Positive dan Negative Testing

- Positive testing dilakukan untuk mengecek output yang diharapkan (kondisi normal) terhadap input
- Negative testing dilakukan untuk mengecek output error jika input *invalid*.



Happy Path Test Case

- Dalam TDD (termasuk model agile), *test case* ditulis **sebelum** kode unit dibuat. Biasanya pengujian awal yang dibuat mengacu pada *output* yang diinginkan.
 - Misalnya pada antarmuka login dimulai dengan pengujian dengan kombinasi *username* dan *password* yang valid sehingga test diharapkan berhasil.
- Test case semacam ini disebut ***Happy Path test case*** (hanya menerapkan **positive testing**).
- Kita juga perlu mempertimbangkan pengujian terhadap ***input yang invalid***.

Contoh Test Case

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	<ol style="list-style-type: none">1. Go to site http://demo.guru99.com2. Enter UserId3. Enter Password4. Click Submit	Userid = guru99 Password = pass99	User should Login into an application	As Expected	Pass
TU02	Check Customer Login with invalid Data	<ol style="list-style-type: none">1. Go to site http://demo.guru99.com2. Enter UserId3. Enter Password4. Click Submit	Userid = guru99 Password = glass99	User should not Login into an application	As Expected	Pass

Membuat test case yang baik

- Simpel dan transparan.
- Membuat *test case* sesuai kebutuhan pengguna (lihat kembali ***user story!***)
- Hindari repetisi *test case*. Gunakan kolom *pre-condition* untuk *test case* yang perlu dijalankan dahulu.
- Jangan berasumsi terkait fungsionalitas diluar dokumen spesifikasi.
- Pastikan *test case* 100% mencakup kebutuhan software yang dibuat.
- Gunakan id agar *test case* mudah diidentifikasi.
- Implementasi teknik testing (BVA, EP, dll). Referensi.
- *Self-cleaning*, yakni mengembalikan *state* sebelum dilakukan pengujian.
- Dapat dilakukan berulang dengan hasil yang selalu sama (pass/ fail).
- Lakukan *peer review* dengan rekan tim.

Literatur Tambahan

<https://www.guru99.com/test-case.html>

<https://www.softwaretestinghelp.com/how-to-write-effective-test-cases-test-cases-procedures-and-definitions/>

<https://learn.microsoft.com/en-us/training/modules/visual-studio-test-concepts/>



Tugas

adalah keharusan yang tidak terelakkan

Tugas

Buat **dokumen rencana testing (*test cases*)** aplikasi sesuai seluruh kebutuhan fungsional yang telah dibuat. (Tidak perlu ada kolom *actual results*). Gunakan template ini.

Kumpulkan di Google Classroom

Format:

Testing_namaprojek.xls

Deadline : H-1 Praktikum Berikutnya, 23.59 (**KUMPUL PERWAKILAN SAJA!**)

Terima kasih

malu bertanya sesat di kelas