Refactorització

Refactorització

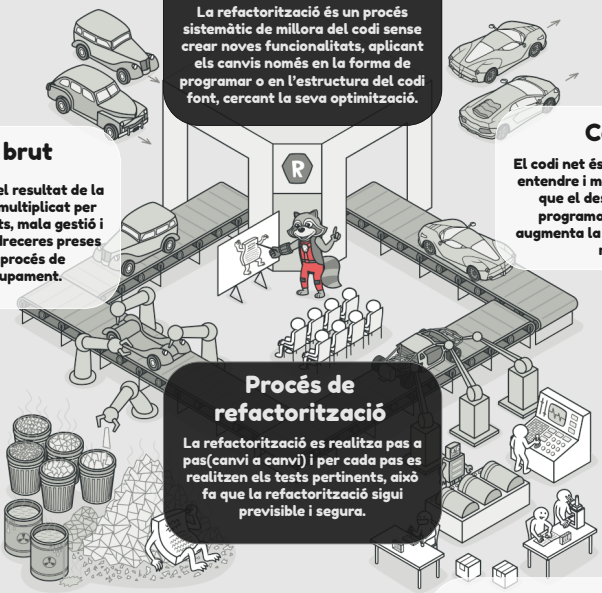
La refactorització és un procés sistemàtic de millora del codi sense crear noves funcionalitats, aplicant els canvis només en la forma de programar o en l'estructura del codi font, cercant la seva optimització.

Codi brut

El codi brut és el resultat de la inexperiència multiplicat per terminis ajustats, mala gestió i desagradable dreceres preses durant el procés de desenvolupament.

Codi net

El codi net és un codi fàcil de llegir, entendre i mantenir. El codi net fa que el desenvolupament de programari sigui previsible i augmenta la qualitat del producte resultant.



Procés de refactorització

La refactorització es realitza pas a pas(canvi a canvi) i per cada pas es realitzen els tests pertinents, això fa que la refactorització sigui previsible i segura.

El codi fa olor

Es diu que el codi fa olor quan tot i que no s'ha identificat cap error i el seu funcionament és correcte, està escrit sense atendre els principis de disseny de programari o fent ús de pràctiques gens recomanables.

Tècniques de refactorització

Les tècniques de refactorització descriuen les passes a seguir. La majoria de tècniques de refactorització tenen el seu pros i contres. Per tant, cada refactorització hauria de tenir una motivació adequada i ser aplicada amb precaució.

Que es la efactorització?

Codi net

L'objectiu principal de la refactorització és combatre el deute tècnic. Transforma un desgavell en codi net i un disseny senzill.

Bonic! Però, què és el codi net? Aquí hi ha algunes de les seves característiques:

El codi net és obvi per a altres programadors.

Nom pobre per les variables, les classes i els mètodes inflats, els "números màgics", tot això fa que el codi sigui descuidat i difícil d'entendre.

El codi net no conté codi duplicat.

Cada vegada que heu de fer un canvi en un codi duplicat, heu de recordar fer el mateix canvi a cada instància. Això augmenta la càrrega cognitiva i alenteix el progrés.

El codi net conté te el mínim de classes possibles.

Menys codi té menys coses per mantenir al cap. Menys codi és menys manteniment. Menys codi són menys errors. El codi és responsabilitat, sigui breu i senzill.

El codi net supera totes les proves.

El vostre codi està brut quan només passa el 95% dels Tests.

El codi net és més fàcil i econòmic de mantenir.

Deute tècnic

Tothom fa tot el possible per escriure un codi excel·lent des de zero. Probablement no hi hagi cap programador que escrigui intencionadament codi impur en detriment del projecte. Però, en quin moment el codi net queda impur?

Ward Cunningham va suggerir originalment la metàfora del "deute tècnic" pel que fa al codi impur.

Si obteniu un préstec d'un banc, això us permetrà fer compres més ràpidament. Pagueu més per agilitzar el procés; no només pagueu el principal, sinó també els interessos addicionals del préstec. No cal dir que fins i tot podeu obtenir tants interessos que la quantitat d'interès supera els vostres ingressos totals, cosa que impossibilita la devolució total.

El mateix pot passar amb el codi. Podeu accelerar temporalment sense escriure proves de noves funcions, però això alentirà progressivament el vostre progrés cada dia fins que finalment pagueu el deute mitjançant la redacció de proves.

Causes del deute tècnic

Pressió empresarial

De vegades, les circumstàncies empresarials poden obligar-vos a desplegar funcions abans que acabin completament. En aquest cas, apareixeran apedaçaments al codi per amagar les parts inacabades del projecte.

Manca de comprensió de les conseqüències del deute tècnic

De vegades, el vostre empresari pot no entendre que el deute tècnic té "interessos" en la mesura que frena el ritme de desenvolupament a mesura que s'acumula. Això pot fer que sigui massa difícil dedicar el temps de l'equip a la refactorització perquè la direcció no hi veu el valor.

Manca de proves

La manca de retroalimentació immediata afavoreix solucions ràpides, però arriscades. En el pitjor dels casos, aquests canvis s'implementen i s'implementen directament a producció sense cap prova prèvia. Les conseqüències poden ser catastròfiques. Per exemple, una solució d'aspecte innocent pot enviar un missatge de prova estrany a milers de clients o, encara pitjor, eliminar o corrompre tota una base de dades.

Manca de documentació

Això alenteix la introducció de noves persones al projecte i pot frenar el desenvolupament si les persones claus abandonen el projecte.

Manca d'interacció entre els membres de l'equip

Si la base de coneixement no es distribueix a tota l'empresa, la gent acabarà treballant amb una comprensió obsoleta dels processos i la informació general del projecte. Aquesta situació es pot agreujar quan els desenvolupadors júnior són incorrectament formats pels seus mentors.

Desenvolupament simultani a llarg termini en diverses branques

Això pot conduir a l'acumulació de deute tècnic, que després s'incrementa quan es fusionen els canvis. Com més canvis es facin ailladament, major serà el deute tècnic total.

Retard de refactorització

Els requisits del projecte canvien constantment i, en algun moment, pot resultar evident que algunes parts del codi estan obsoletes, s'han tornat feixugues i s'han de redissenyar per complir els nous requisits.

D'altra banda, els programadors del projecte escriuen cada dia un nou codi que funciona amb les parts obsoletes. Per tant, com més es retardi la refactorització, el codi més dependent haurà de ser reelaborat en el futur.

Falta de control de compliment

Això passa quan tothom que treballa al projecte escriu el codi que creu convenient (és a dir, de la mateixa manera que va escriure l'últim projecte).

Incompetència

És quan el desenvolupador no sap escriure un codi decent.

Quan refactoritzar?

Regla de tres

Quan feu alguna cosa per primera vegada, només cal que ho feu un pic.

Quan feu una codi semblant per segona vegada, cal haver de repetir, cap problema.

Quan feu alguna cosa per tercera vegada, comenceu a refactoritzar!!

En afegir una funció

La refactorització us ajuda a entendre el codi d'altres persones. Si heu de tractar el codi brut d'una altra persona, proveu de refactoritzar-lo primer. El codi net és molt més fàcil d'entendre. No el millorareu només per a vosaltres mateixos, sinó també per a aquells que l'utilitzen després de vosaltres.

La refactorització facilita afegir noves funcions. És molt més fàcil fer canvis en el codi net.

En solucionar un error

Els errors del codi es comporten com els de la vida real: viuen als llocs més foscos i bruts del codi. Neteja el codi i els errors pràcticament es descobriran.

Els administradors agraeixen la refactorització proactiva, ja que elimina la necessitat de tasques especials de refactorització més endavant. Els caps feliços fan feliços als programadors!

Durant una revisió de codi

La revisió del codi pot ser l'última oportunitat per ordenar el codi abans que estigui disponible per al públic.

El millor és fer aquestes ressenyes en parella amb un autor. D'aquesta manera podríeu solucionar problemes senzills ràpidament i avaluar el temps per solucionar els problemes més difícils.

Com refactoritzar?

La refactorització s'hauria de fer com una sèrie de petits canvis, cadascun dels quals fa que el codi existent sigui lleugerament millor mentre es deixa el programa en bon estat.

Llista de comprovació que la refactorització es realitza correctament

El codi hauria de ser més net.

Si el codi continua sent tan impur després de refactoritzar-lo ... bé, ho sento, però acaba de perdre una hora de la vostra vida. Intenta esbrinar per què va passar això.

Sovint passa quan us allunyeu de la refactorització amb petits canvis i barregeu un munt de refactoritzacions en un gran canvi. Per tant, és molt fàcil perdre el focus, sobretot si teniu un límit de temps.

Però també pot passar quan es treballa amb un codi extremadament descuidat. Qualsevol cosa que milloreu, el codi continua sent un desastre.

En aquest cas, val la pena pensar en reescriure completament parts del codi. Abans, però, hauríeu d'haver escrit proves i reservar una bona part del temps. En cas contrari, acabareu amb el tipus de resultats dels quals hem parlat al primer paràgraf.

No s'haurien de crear noves funcionalitats durant la refactorització.

No barregeu la refactorització i el desenvolupament directe de noves funcions. Intenteu separar aquests processos almenys dins dels límits de les comissions individuals.

Totes les proves existents han de passar després de la refactorització.

Hi ha dos casos en què les proves es poden descompondre després de la refactorització:

- **Heu comès un error durant la refactorització. Aquesta no és una obvietat: seguiu endavant i corregiu l'error.**
- **Les vostres proves eren massa baixes. Per exemple, provaves mètodes privats de classes.**

En aquest cas, les proves són les culpables. Podeu refactoritzar les proves o escriure un conjunt completament nou de proves de nivell superior.

El codi fa olor

Es diu que el codi fa olor quan tot i que no s'ha identificat cap error i el seu funcionament és correcte, està escrit sense atendre els principis de disseny de programari o fent ús de pràctiques gens recomanables.

Bloaters

Els bloaters són codis, mètodes i classes que han augmentat fins a proporcions tan enormes que són difícils de treballar. Normalment, aquestes olors no apareixen de seguida, sinó que s'acumulen amb el pas del temps o mesura que el programa evoluciona (i sobretot quan ningú no fa cap esforç per eradicar-les).

- Mètode llarg
- Classe gran
- Obsessió primitiva
- Llarga llista de paràmetres
- Grups de dades

Abusadors d'orientació d'objectes

Totes aquestes olors són una aplicació incompleta o incorrecta dels principis de programació orientats a objectes.

- Classes alternatives amb diferents interfícies
- Llegat rebutjat
- Instruccions de commutació
- Camp temporal

Preventors de canvis

Aquestes olors volen dir que si heu de canviar alguna cosa en un lloc del codi, també heu de fer molts canvis en altres llocs. Com a resultat, el desenvolupament del programa es fa molt més complicat i car.

- Canvi divergent
- Jerarquies d'herència paral·leles
- Cirurgia disbalancejada

Dispensables

Un prescindible és quelcom inútil i innecessari l'absència de la qual farà que el codi sigui més net, eficient i fàcil d'entendre.

- Comentaris
- Codi duplicat
- Classe de dades
- Codi Mort
- Lazy Class
- Generalitat especulativa

Acoblaments

Totes les olors d'aquest grup contribueixen a un excés d'acoblament entre classes o mostren què passa si l'acoblament és substituït per una delegació excessiva.

- Funció Envoja
- Intimitat inadequada
- Home Mitjà
- Cadenes de missatges
- Classe de biblioteca incompleta

Tècniques de refactorització

Mètodes de composició

Gran part de la refactorització es dedica a la composició correcta de mètodes. En la majoria dels casos, els mètodes excessivament llargs són l'arrel de tot mal. Els cabricis del codi dins d'aquests mètodes disimulen la lògica d'execució i fan que el mètode sigui extremadament difícil d'entendre i fins i tot més difícil de canviar.

Les tècniques de refactorització d'aquest grup racionalitzen els mètodes, eliminen la duplicació de codi i obren el camí a futures millores.

Mètode d'extracció | Mètode en línia | Extreure la variable | Temp en línia

Substituir Temp per consulta | Algorisme substitut

Variable temporal dividida | Eliminar tasques o paràmetres

Substituir el mètode per l'objecte Mètode

Moure funcionalitat entre objectes

Fins i tot si hau distribuit la funcionalitat entre diferents classes d'una manera poc efectiva, encara hi ha esperança.

Aquestes tècniques de refactorització mostren com moure de manera segura la funcionalitat entre classes, crear classes noves i amagar els detalls de la implementació de l'estil públic.

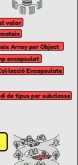
Eliminar Middle Man | Moure mètode | Moc al camp | Extracta de classe

Classe en línia | Amaga delegat | Introduir el mètode estrenger

Introduïu l'extensió local

Organització de dades

Aquestes tècniques de refactorització ajuden amb el tractament de dades, substituint les primitives per funcionalitats de classe rics. Això té molts resultat important és el desenredament de les associacions de classes, que fa que les classes siguin més portàtils i reutilitzables.



Canviau el valor per referència | Canviau la referència al valor

Dades observables duplicades | Camp encapsulat per al mateix

Substituir el valor de les dades per objecte | Substituïu Array per Object

Canviau l'Associació Unidireccional a Bidireccional | Camp encapsulat

Canviau l'Associació Bidireccional a Unidireccional | Col·lecció Encapsulada

Substituir el número màgic per constant simbòlica

Substituir el codi de tipus per classe | Substituir el codi de tipus per subclasses

Substituir el codi de tipus per estat / estratègia

Substituïu la subclasse per camps

Simplificació d'expressions condicionals

Els condicionants tendeixen a ser code vegada més complicats en la seva lògica amb el pas del temps, i encara hi ha més tècniques per combatre-ho.

Consolidar l'expressió condicional | Consolidar fragments condicionals duplicats

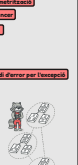
Descomponeu Condicional | Substituïu Condicional per Polimorfisme

Elimina el senyal de control | Introduïu l'objecte nul | Introduïu l'afirmació

Substituir la condició imbricada per cláusules de guàrdia

Simplificació de trucades de mètodes

Aquestes tècniques fan que les trucades de mètodes siguin més senzilles i fàcils d'entendre. Això, el seu torn, simplifica les interfícies per a la interacció entre classes.



Afegeix un paràmetre | Elimina el paràmetre | Canvia el nom del mètode

Separeu la consulta del modificador | Mètode de parametrització

Introduïu l'objecte Parameter | Conserver l'objecte sencer

Elimina el mètode de configuració | Amaga el mètode

Substituir el paràmetre per mètodes explícits

Substituir el paràmetre per mètode de trucada

Substituïu l'excepció per la prova | Substituir el codi d'error per l'excepció

Substituir el constructor pel mètode de fàbrica

Tractament de la generalització

L'abstracció té el seu propi grup de tècniques de refactorització, principalment associades amb el moviment de funcionalitats al llarg de la jerarquia d'herències de classes, la creació de noves classes i interfícies i la substitució de l'herència per delegació i viceversa.

Camp cap amunt | Mètode Pull Up | Tireu el cos del constructor

Premeu el camp cap avall | Mètode de baixada | Extreu la subclasse

Extracta de superclasse | Extreu la interfície | Col·lapse de jerarquia

Mètode de plantilla de formulari | Substituïu l'herència per delegació

Substituïu la delegació per l'herència