Pannon Egyetem

Matematikai Tanszék

Programtervező Informatikus BSC

SZAKDOLGOZAT

Release Recommendation Tool

Suborits Péter

Zágon Ferenc

Témavezető: Lipovits Ágnes

Konzulens:Both Botond, Horák Krisztina

2017

Contents

[**1.** **Feladat ismertetése** 3](#_Toc499131222)

[**2.Szükséges programok,dokumentumok ismertetése** 4](#_Toc499131223)

# Feladat ismertetése

A Continental Automotive Hungary veszprémi telephelyén dolgozom diákmunkásként. A csoportvezetőknek kell ellenőrizni a fejlesztők tevékenységeit, ez nagyon sok időt vesz el a munkaidejükből, amit dokumentációk átnézésével kell tölteniük. Ez napi szinten 20-30 perc, ami egy hónapban akár már 10 óra is lehet. Ezzel a szoftverrel, amit Zágon Ferenccel készítünk, próbáljuk napi szinten körülbelül 5 percre vagy még kevesebbre redukálni. Ezen dokumentumok ellenőrzésének nagy része automatizálható és mi ezt igyekszünk megvalósítani. Persze az idő csökkentésével semmit sem érünk el, ha megbízhatatlan lesz a program és rosszul értékeli ki az adatokat, ezért nagyon fontos szempont lesz az is, hogy a dokumentumokat, adatokat helyesen értékelje ki az alkalmazás. Ha az applikáció hibásan dolgozik, maradhatnak a rendszerben nem megfelelő dokumentációk és szoftverkódok, amelyek javításra szorulnának, és ha ez csak a vásárlóknál derülne ki, az a vállalatnak hatalmas károkat tudna okozni. Például ha egy autót rossz szoftverrel kezdenek el gyártani és ez csak forgalomba helyezésük után derül ki az összes gépjárművet vissza kell hívni és javítani a hibákat, ráadásul fennáll annak a veszélye is, hogy a hiba balesetet, súlyosabb esetben emberéletet is követelhet. Ez pedig a cég számára jelentős veszteséget okozna mind hírnévben, mind pedig pénzügyileg.

Munkánk bonyolultságát fokozza, hogy a cégen belül nem egy szoftverstruktúrával dolgoznak. Két nagyobb szoftverstruktúrát különböztethetünk meg a TB és a Vision Controls. Az induló projectek mindig a legújabbat használják vásárlótól és verziótól függően, mivel a TB és a Vision Controls is folyamatos fejlesztés alatt áll. Ezek nagyrészt eltérően állítják elő a kiértékelendő dokumentumokat, így valószínűleg minden dokumentumfajtára két algoritmust kell kidolgozni. Szerencsére szoftverstruktúrán belül nincsenek nagy eltérések, ezért lehetséges az automatizálás.

A probléma megoldására régebben már voltak próbálkozások, viszont egyik sem volt sikeres. A korábbi verziók átláthatatlanok, kezelhetetlenek voltak és nagyon sok redundáns adatokat tartalmaztak, néhol ami kellett volna az hiányzott belőle, valamint egyik sem volt igazán stabil. Mivel Ferenc már teljes munkaidőben dolgozik, és az Egyetem után én is szeretnék teljes munkaidőben a cégnél dolgozni, ezáltal mindig naprakészen tudnánk tartani a programot, illetve ha valami új dolgot kellene beépíteni, ezt el tudnánk végezni.

# 2. Szükséges programok

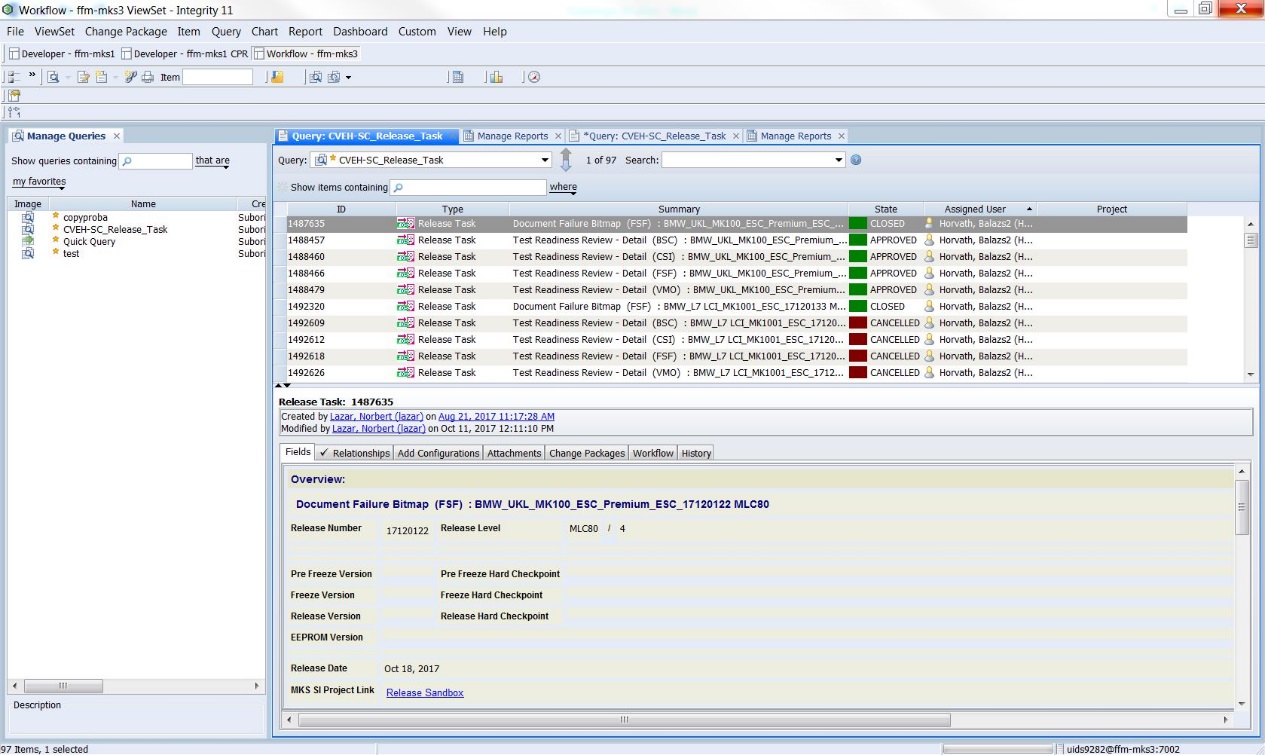
## 2.1 Fejlesztési környezet

A szoftver megvalósítását Java programnyelven tervezzük, NetBeans fejlesztői környezetet használva. A dokumentációk kiértékeléséhez elképzelhető, hogy reguláris kifejezéseket is alkalmaznunk kell és ebben a nyelvben lehetőségünk van ezeket egyszerűen kezelni. A cég által használt verziókövető rendszer (MKS) is Java nyelven készült és tartalmaz egy manual-t amiben különböző parancsok vannak, hogy hogyan is lehet külső program (parancssor) segítségével elérni, dokumentumokat letölteni illetve lekérdezéseket írni és jelentéseket készíteni. Azért választottuk ezt a nyelvet, illetve fejlesztői környezetet, mivel felhasználóbarát, könnyen átlátható kezelőfelületet tartalmaz, valamint kényelmes használni. Mivel az Egyetemen órai keretek között már használtam a programot, ezért nem ismeretlen számomra.

A rendszer készítése során folyamatosan beszélünk a Continental-on belüli illetékesekkel, vezetőinkkel, hogy a projektünk állandóan nyomon követhető legyen, illetve az esetleges hibákat, félreértéseket elkerüljük. A megvalósítást próbáljuk ésszerűen felépíteni, pontról-pontra haladni, kezdve elsődlegesen az információszerzéssel, hogy a leendő felhasználóknak milyen igényei vannak a szoftverrel kapcsolatosan. Ezután az adminisztrációs munkához kellő dokumentumokról szerzünk információkat, hogy milyen szempontok alapján kell kiértékelni őket, illetve mi az ami kell belőle és mi az ami nem, és mi elfogadható és mi nem az adott dokumentumoknál. Miután megkaptunk minden információt az elvárásokkal kapcsolatban, és tudjuk a pontos definíciókat, szabályokat egy-egy adott ellenőrzési folyamathoz, Ferenccel szétosztjuk egymás között ezeket a kiértékeléseket. Ezzel párhuzamosan Ferenc a felhasználói interface-t fogja megtervezni, jómagam pedig a Continental belső verziókövető rendszeréből való lekérdezéseket illetve dokumentum letöltését, valamint az verziókövető rendszerrel való kommunikációt fogom implementálni.

## 2.2 V**erziókövető rendszer**

A szoftverfejlesztők egy verziókövető rendszert (MKS) használnak a dokumentációk és programkódok tárolására. Továbbá ezt az egész rendszert nem csak itt a magyarországi vállalatnál használják, hanem a többi lokáción is.



*1.1 ábra*

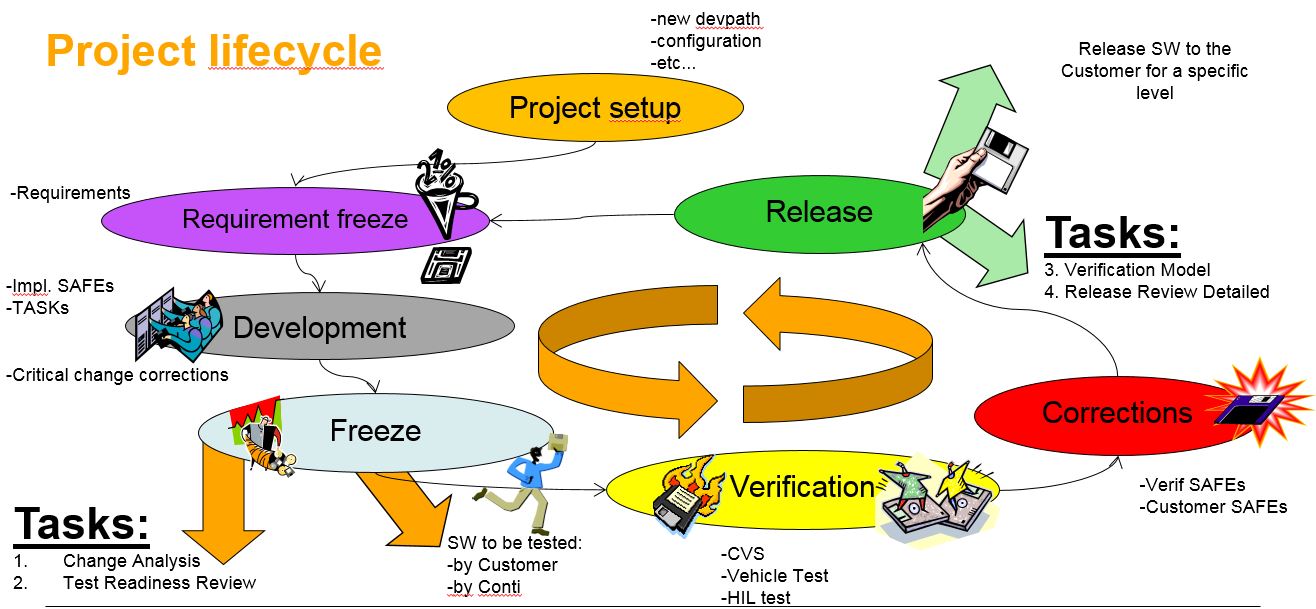
A rendszer tartalmazza a felhasználói felületet, amely az 1.1-es ábrán látható és egy kis gyakorlással egyszerűen lehet használni. Két szervert használnak a projekt munkával kapcsolatosan az adatok tárolásához, az egyiken vannak a dokumentációk, a másikon pedig a programkódok, módosítások. Ebből a szempontból is érdekes lesz az adatok megfelelően módon történő kinyerése, mivel nem elég, ha a programunk kommunikál a szerverrel, ráadásul a programunkon keresztül tudnia kell kommunikálnia a két szervernek is. Nekünk csak arra lesz szükségünk, hogy a szükséges fájlokat kiexportáljuk és feldolgozzuk, ezért előreláthatólag csak két funkcióját fogjuk használni a verziókövető rendszernek. Az egyik ilyen funkció a lekérdezés (query) lesz, ami a rendszeren belül egy szűrő. Logikai műveletekből épül fel amit tudunk kombinálni a mezőkkel, hogy milyen sorrendben keresse a megadott feltételeket. A program szempontjából ezeknek a szűréseknek félig dinamikusaknak kell lenniük, mivel lesz bennük, ami fix, és lesz, ami a felhasználótól fog függni. A felhasználok, mint korábban említettem a csoportvezetők lesznek, és a dinamikusság alatt arra kell gondolni, hogy minden vezetőhöz más-más csoport vagy csoportok tartoznak, amiket át kell nézniük. A másik funkció a generálás (report) lesz, amely lényegében az exportálásért felelős. Itt megtudjuk adni beállításként, hogy milyen formátumba szeretnénk exportálni (xml, html), formázást tudunk adni, számolt cellákat tudunk hozzáadni, de ezekre nagyon kell figyelni, mivel a futási időre nagy hatással van. Például ha a számított celláknál nem a megfelelő képleteket használjuk, akkor ez a lefutás megszakításához is vezethet, mivel a rendszernek van egy időkorlátja, amit ha túllép, automatikusan leállítja a futó folyamatot. Tehát nagyon fontos az optimalizálás, mivel ha csak egy report nem fut le az már hatással lehet az egész kiértékelésre, hibás adatokat, információkat kaphatnak a csoportvezetők.

Az adatok mappás elrendezésben vannak jelen a rendszerben, viszont itt is lehetnek gondok, ha nem a megfelelő mappastruktúrába kerültek, ezt az okozza, amit már korábban is említettem, hogy kétféle szoftverstruktúrával dolgoznak a fejlesztők vásárlótól függően, és ez projekttől függetlenül minimálisan, de eltérhetnek. A programunknak ezt is le kell tudnia kezelnie, hogy eredményesen futhasson le a kiértékelés mind a két struktúrára.

*Ide egy query-t és egy reportos képet berakni!!!!*

## 2.3 Release munka

## 2.3.1 Project életciklus- Szoftver oldalról

Egy project nagyon sok részből áll, és rengeteg idő, míg elkészül. Most ezt fogom ismertetni, hogy hogyan is áll össze, milyen részei vannak, ezáltal könnyebb lesz megérteni a későbbiekben a programot is működés szempontjából. Az egész egy konfigurálással kezdődik, amiben a vásárló meghatározza, milyen funkciókra lesz szüksége. Miután a két fél meg tudott egyezni, hogy ezek közül mit lehetséges megvalósítani ezeket a követelmények befagyasztják, úgymond freeze-lik. A következő lépésben a fejlesztőcsapat elkezdi programozni a kért funkciókat, itt már keletkeznek dokumentációk is. Release szintenként meghatározott időintervallumok lejárta után, fagyasztják a szoftverfejlesztést, itt is jönnek létre dokumentációk. A freeze(fagyasztás) alatt a szoftver tesztelése zajlik mind a Continentál, mind pedig a vásárlónál. Ha a teszt alatt nem volt semmilyen probléma, ami miatt nem lehetne tesztelni a szoftvert a gépjárműben, akkor a következő fázisban tesztelik a szoftvereket autóban is, hogy olyan körülmények között hogyan viselkedik. Hogyha itt sem lép fel semmi gond, akkor a szoftvert átadják a vásárlónak a Release szintnek megfelelő specifikációval, ha gond volt, akkor javítják a szoftvert amit újra kell majd tesztelni, ezt természetesen külön dokumentálják. Ez a folyamat addig ismétlődik, míg el nem érik a végső Release szintet, vagy ameddig kritikus hibák vannak a szoftverben.

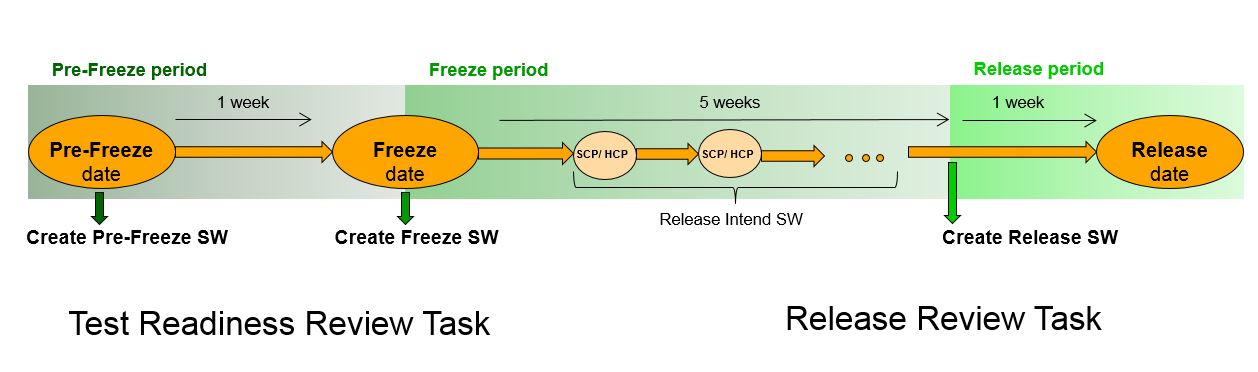
*1.3 ábra*

Az 1.3-as ábrán ennek a folyamatnak az életciklusa látható.

## 2.3.2 Release szintek

Mint említettem ez az életciklus Release szintek alapján ismétlődik újra és újra, mindig egy kicsivel többet hozzárakva az addigiakhoz. A Release szinteket MLCxx formátumban nevezik el, illetve ezek a szintek is al-szintekre bontódhatnak amelyet MLCxx/x néven definiálnak. Az MLC00-ás szint azt jelenti, hogy a cég megkapta a projectet. Az MLC40-nél a Continental belsőleg osztja ki a feladatokat a megfelelő csoportoknak. Az MLC50 az a szint, ahol már egy alapszoftver rendelkezésre áll, hogy beépítsék prototípusokba, ezáltal készen áll az első tesztvezetésre, de ezt még csak tesztpályán tehetik meg, közútra nem mehetnek vele. Az MLC55-nél a szoftver funkciók is implementálásra kerülnek, amelyet engedélyeznek és funkcionálisan tesztelnek. Ekkor a szoftver már elérhető lesz a vásárló számára is, és ez a verzió már tesztelhető közúton is. Az MLC60-as fázisnál áll úgy össze a rendszer, hogy a cég már tud biztosítani egy viszonylag fix hardvert és szoftvert, ezáltal a vásárló már beszerezheti a nagymennyiségű termeléshez szükséges eszközöket. Az MLC70-nél a rendszer már készen áll kisebb sorozatgyártásra, itt a szoftver tartalmazza már a funkcionalitást, és a vásárló specifikus követelményeit, és a tesztek is sikeresek voltak, ez után már nem kell a funkciókat tovább fejleszteni. MLC80-nál A rendszert jóváhagyják tömeggyártásra, ekkor már befejeződtek a hosszú távú tesztelések sikeresen.

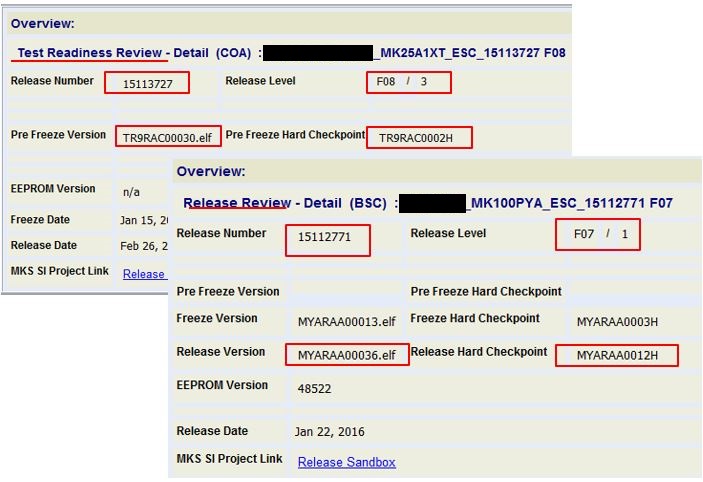
## 2.3.3 Freeze fázisok

*1.4 ábra*

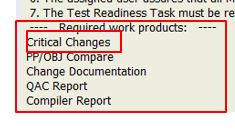
A Freeze-t három fázisra lehet bontani, Pre-Freeze, Freeze, és Release. A Pre-Freeze periódusban a szoftverproject managerek elkészítik az EEPROM fájlokat, csinálnak soft és hard checkpoint-okat, valamint legenerálják a reportokat, ezek pedig a következők:critical changes, QA-C, CRR, Change doc, Object Compare, Change System Relevant Code, Compiler Bug Report. A fejlesztők átnézik és kommentezik ezeket a reportokat, módosíthatják a kódot, paramétert állíthatnak, illetve el kell fogadják a Test Readiness task-ot. A Freeze fázisban a manager újragenerálja az EEPROM fájlokat, új soft és hard checkpoint-ot hoz létre, legenerálja a szükséges munkatermékeket, illetve átrakja az adott verziót Freeze állapotba ami azt jelenti hogy ez idő alatt nem lehet módosítani a kódokon és tesztelik a verziót. Miután ezek megvannak A Release periódusban a manager újra legenerálja a EEPROM fájlokat, report-okat, és Release Review Taskot hoz létre. A fejlesztők átnézik majd kommentezik, és ez után a manager dönt a report-ok alapján, hogyha nincs benne olyan hiba ami miatt nem lehet kiadni akkor jóváhagyja.

## 2.3.4 Task-ok

A task-okon nagyon sok információ van, amit a managereknek össze kell hasonlítaniuk a hozzájuk tartozó dokumentumokkal. Ilyen például a release number és a release level, többek között pedig azt is ellenőrizni kell, hogy a különböző Freeze verziókhoz esetleges módosítások után le lettek-e generálva a megfelelő pdf állományok. A task-ok tartalmazzák továbbá, hogy milyen munkatermékeket kellett elkészíteni vagy módosítani, ezek meglétét is ellenőrizni kell.



*1.5 ábra*



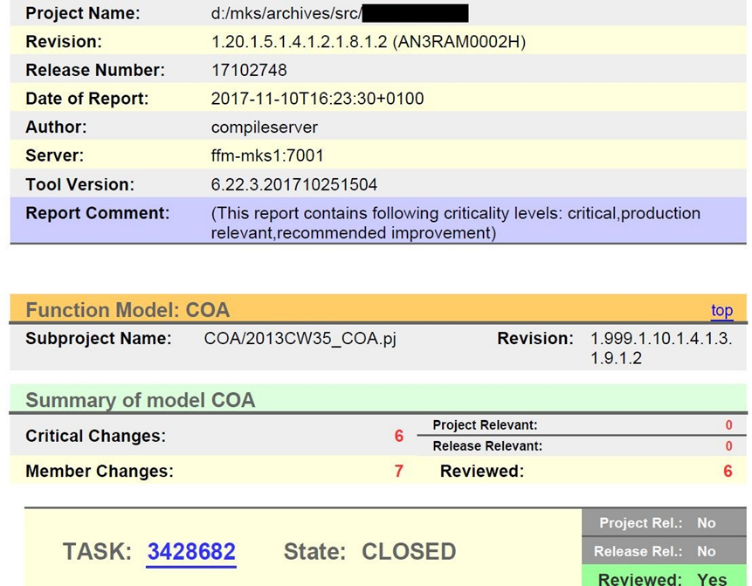
*1.6 ábra*

Az 1.5 illetve az 1.6-os ábrákon látható erre egy-egy példa.

## 2.3.5 Report-ok

## 2.3.5.1 Critical Change Report

Ha a report-ban van hiba, akkor azt a fejlesztőknek át kell nézni, javítani az esetleges hibákat majd újragenerálni a report-ot. Ennek a report-nak tartalmaznia kell az új task-ot az új verziójú szoftverrel ellátva. Ezen felül kell hozzá készíteni egy Test Readiness Review-t és egy Release Review-t. Majd ellenőrizni kell, hogy ez a task releváns-e a projectre, avagy sem. Ehhez meg kell néznie a leírást, a kommenteket, és hogy kik azok akik dolgoznak rajta vagyis a tagokat. Ha ez megtörtént, akkor Reviewed állapotba kell átraknia, ha pedig releváns is a projectjére, akkor pedig Relevant állapotot is meg kell jelölnie, probléma esetén kommentet kell hozzáfűznie, amiben el kell döntenie, hogy a probléme project vagy release releváns, tehát hogy blokkolja-e vagy sem a kiadását. Alább látható egy report az 1.7-es ábrán.



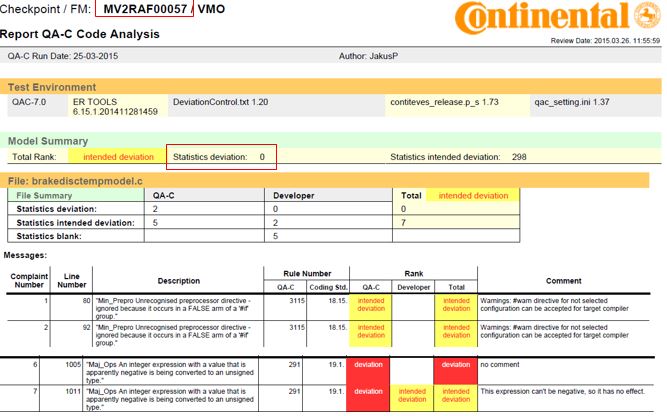
*1.7 ábra*

## 2.3.5.2 Compiler Report-CRR

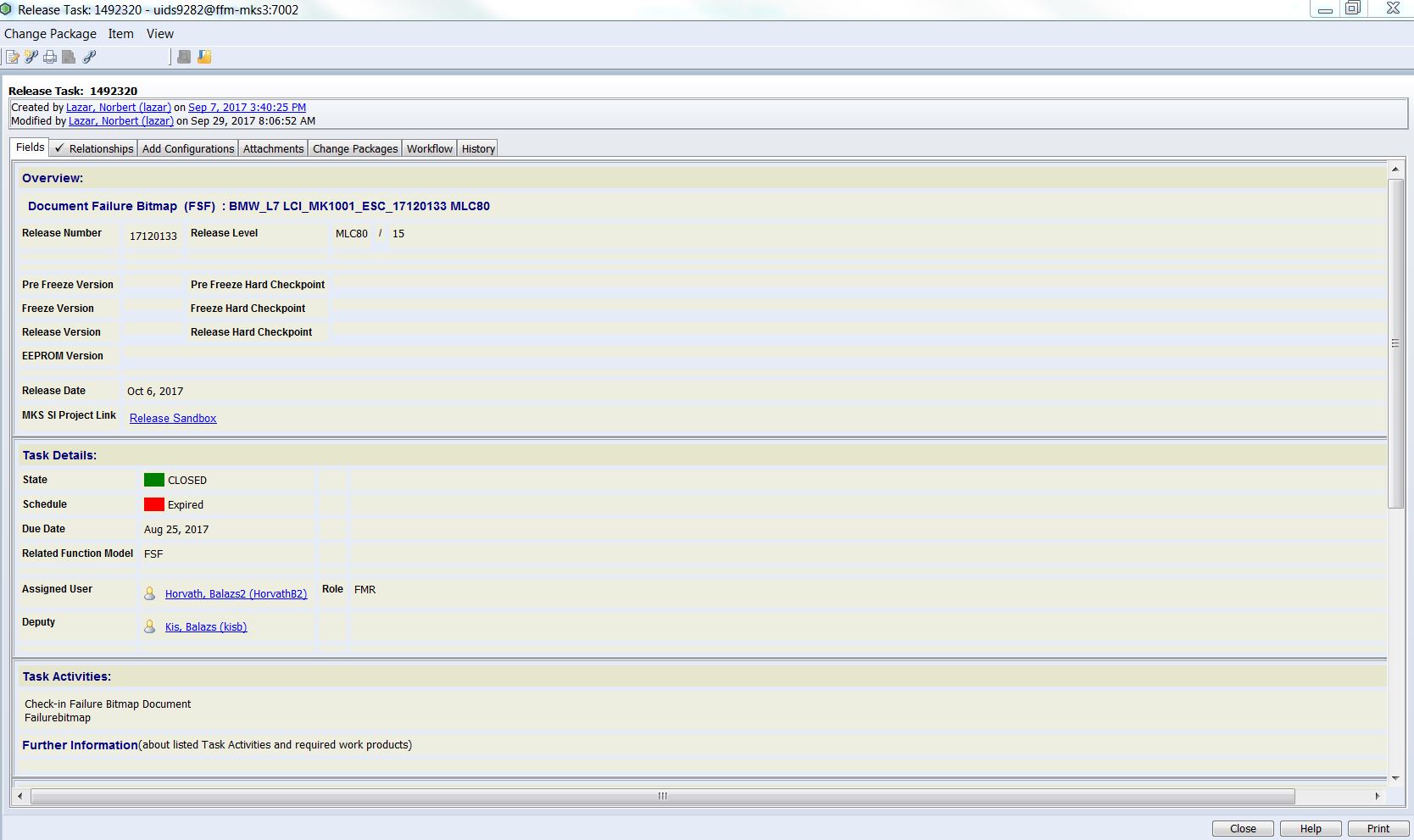
A compiler report-oknál a szoftver warning-ok szerepelnek, amelyek nincsenek hatással a működésre, csak valamilyen oknál fogva a fordító figyelmeztet. Ekkor a fejlesztőknek meg kell nézni az összes warning-ot, és mindegyikhez kommentet kell fűzniük.

## 2.3.5.3 QA-C report

Statikus kód analízist jelent. Ez az eszköz segíti a fejlesztőket, hogy eltüntessék a figyelmeztetéseket, eltéréseket. Ha egy-egy eltérést, figyelmeztetést, átállítanak szándékossá, akkor azt kommentben meg kell indokolni. Ezt a végén ellenőrizniük is kell, hogy nem hiányzik, komment megfelelő a checkpoint is a kommentek a szabályoknak megfelelően van-e kitöltve és sikerült-e eltérés mentessé tenni a kódot. Erre egy példa az 1.8-as ábrán látható.



*1.8 ábra*



*1.2 ábra*

Az 1.2 ábrán látható egy task, ami számos fontos adatot tartalmaz a szoftverünk szempontjából. Egy ilyen task rengeteg információt tartalmaz, ami alapján például egy query-t is össze tudunk állítani. A fontos adatokra a későbbiekben fogok kitérni.