Pannon Egyetem

Matematikai Tanszék

Programtervező Informatikus BSC

SZAKDOLGOZAT

Release Recommendation Tool

Suborits Péter

Zágon Ferenc

Témavezető: Lipovits Ágnes

Konzulens:Both Botond, Horák Krisztina

2017

Tartalomjegyzék

[1. Feladat ismertetése 3](#_Toc500604348)

[2. Szükséges programok 5](#_Toc500604349)

[2.1 Előkészületek 5](#_Toc500604350)

[2.2 Fejlesztési környezet 5](#_Toc500604351)

[2.3 Verziókövető rendszer 6](#_Toc500604352)

[3. Release munka 9](#_Toc500604353)

[3.1 Project életciklus- Szoftver oldalról 9](#_Toc500604354)

[3.2 Release szintek 11](#_Toc500604355)

[3.3 Freeze/Release idővonal/fázis 11](#_Toc500604356)

[3.4 Release munka 12](#_Toc500604357)

[4. Release dokumentáció 14](#_Toc500604358)

[4.1 Critical Change Report 14](#_Toc500604359)

[4.2 Compiler Report-CRR 15](#_Toc500604360)

[4.3 QA-C report 15](#_Toc500604361)

[4.4 Object Compare report 16](#_Toc500604362)

[4.5 Change Documentation report 16](#_Toc500604363)

[4.6 Change System Relevant Code report 16](#_Toc500604364)

# 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 (Technology Branch) 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.

# Szükséges programok

## 2.1 Előkészületek

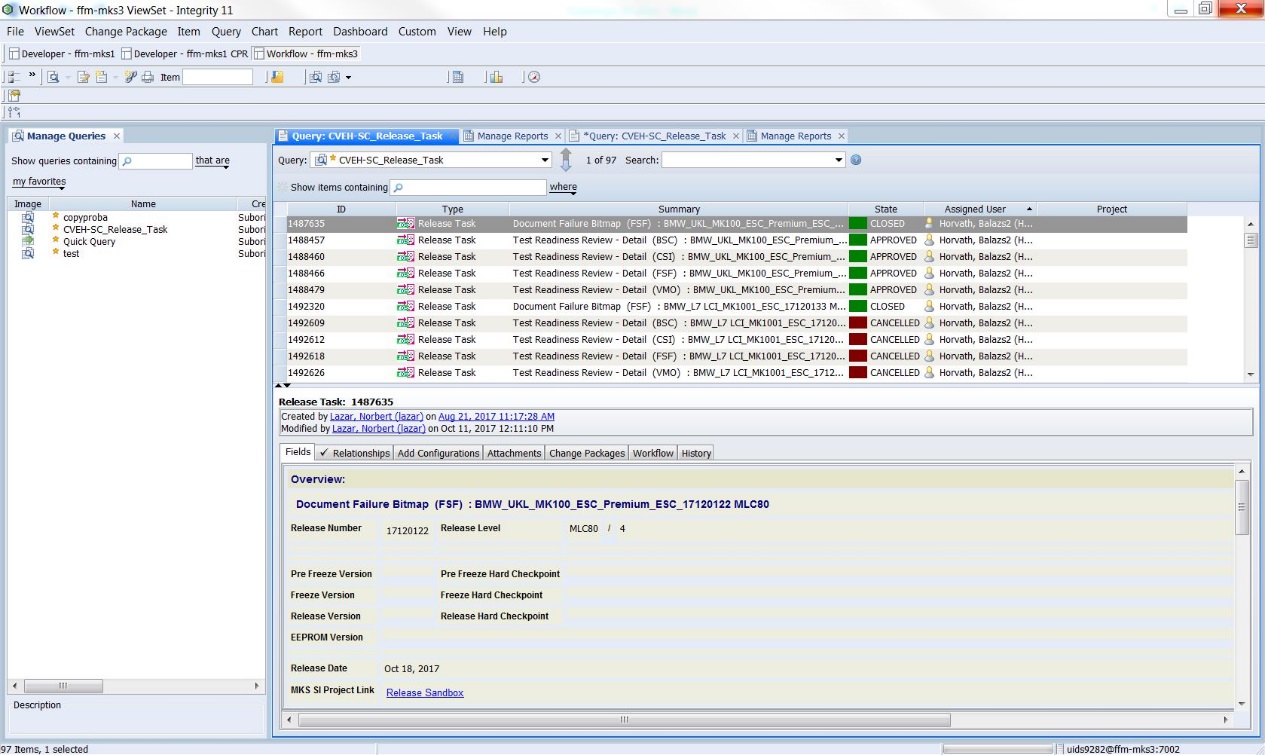
A rendszer készítése során folyamatosan beszélünk a Continental-on belüli illetékesekkel, vezetőinkkel, hogy a projektünk állandóan nyomon követhető legyen, illetve az esetleges hibákat, félreértéseket elkerüljük. A megvalósítást próbáljuk ésszerűen felépíteni, pontról-pontra haladni, kezdve elsődlegesen 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 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.

## 2.3 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.



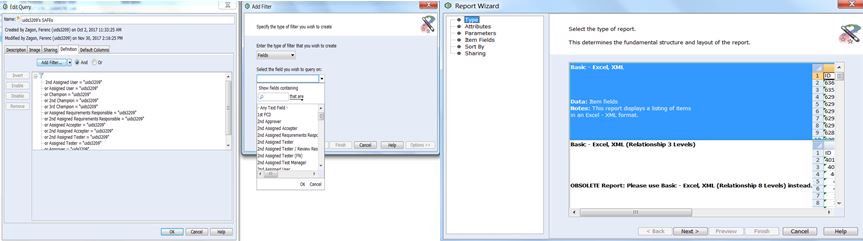
*1.1 ábra*

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. Azért van eltérés, mert ahogy már korábban is említettem, 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. Alább az 1.2-es ábrán látható egy-egy példa a lekérdezésre(query), és a generálásra(report), hogy a készítésük során mennyi funkció közül lehet választani.

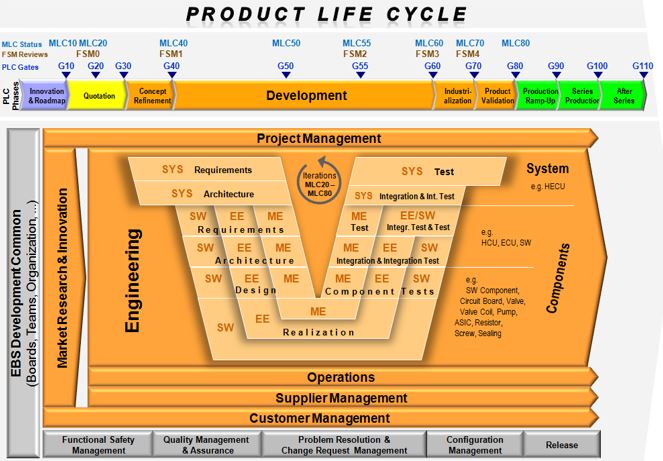


*1.2 ábra*

# 3. Release munka

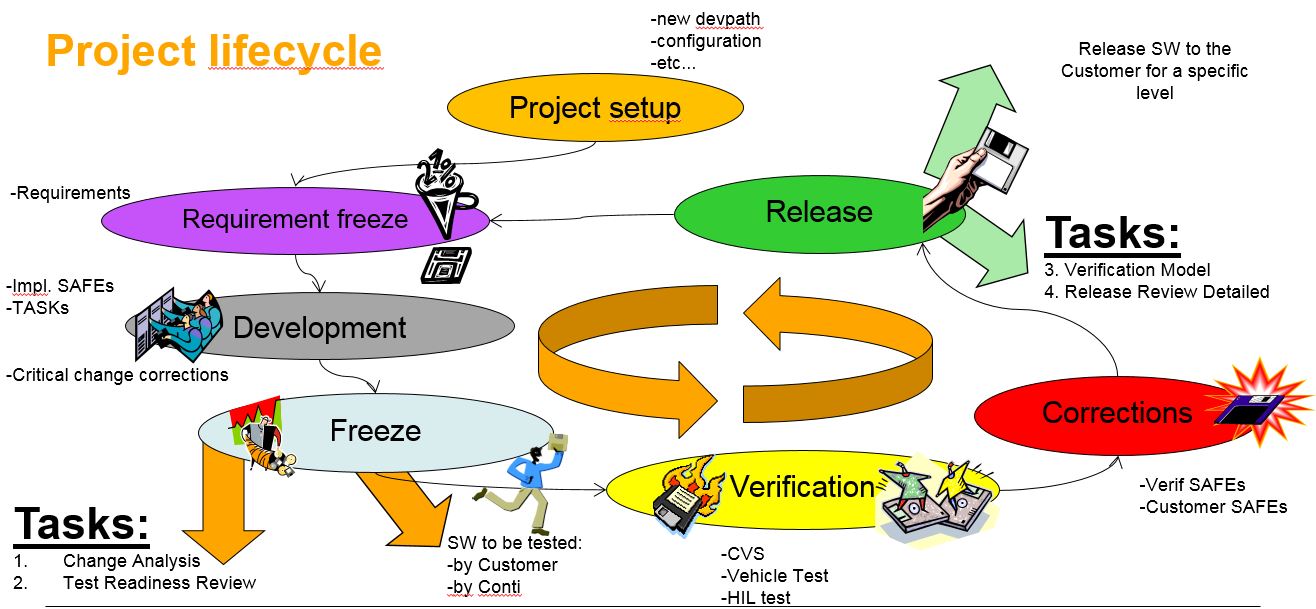
## 3.1 Project életciklus- Szoftver oldalról

A continetnál-nál V-modell alapján építik fel a tennivalókat, ez alapján dolgoznak, ami a vízesés modellnek egy újragondolt verziója. Szoftver oldalról nézve először megfogalmazzák a rendszerkövetelményeket, majd kidolgozzák hozzá az architektúrát. Ez után ugyanígy tesznek a szoftverkövetelményekkel is. A fejlesztők elkezdik a megtervezni a kódot, majd implementálják. Ha minden megtörtént minden egyes metódust külön-külön tesztelnek, és ha valamelyik fázisnál probléma lép fel akkor visszaugranak arra a szintre, ahol probléma volt és attól a szinttől lefele újrakezdenek mindent, javítják a hibát,hibákat, és újra a legalsó szinttől kezdve kezdődik a tesztelés. Az alábbi 1.3-es ábrán látható ez a folyamat.



*1.3 ábra*

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, amelyben olyan dolgokat határoznak meg, mint például hogy milyen rendszer lesz , milyen struktúrára fog épülni, kik fognak dolgozni a projekten, és létrehozzák ezek alapján a projektet a verziókövető rendszerben. A következő lépésben a fejlesztőcsapat elkezdi programozni a kért funkciókat. Release szintenként meghatározott időintervallumok lejárta után, fagyasztják a szoftverfejlesztést. A release szintekről a következő fejezetben fog szó esni. A freeze(fagyasztás) alatt a szoftver tesztelése zajlik mind a Continentál, mind pedig a vásárlónál. Hogyha nem 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. Ez a folyamat addig ismétlődik, míg el nem érik a végső Release szintet, vagy ameddig kritikus hibák vannak a szoftverben.



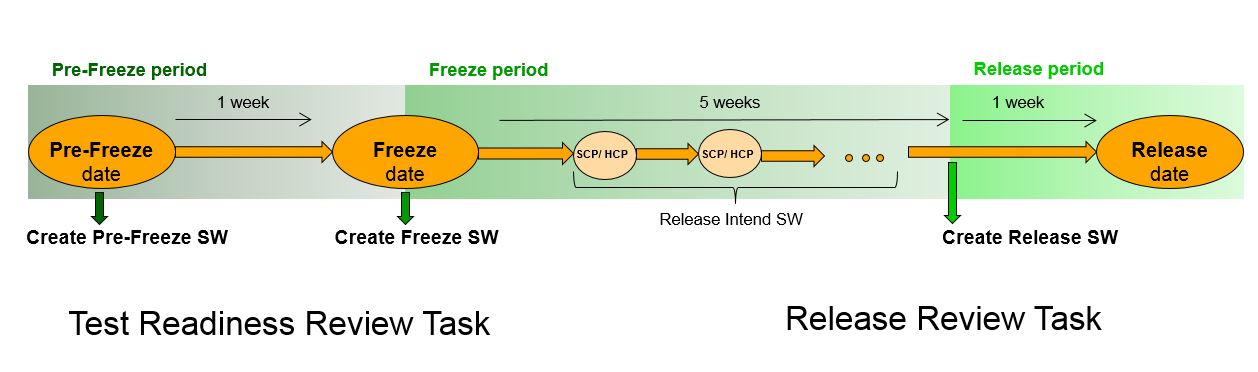
*1.4 ábra*

Az 1.4-as ábrán ennek a folyamatnak az életciklusa látható.

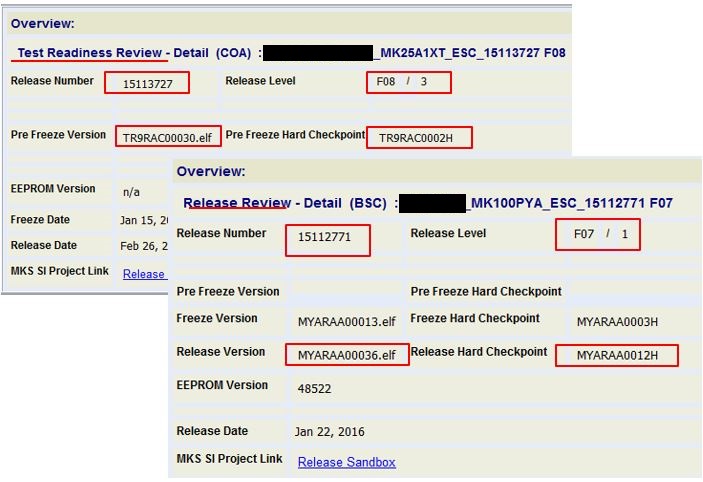
## 3.2 Release szintek

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, ezek az elnevezések csak a Continentál-on belül léteznek, 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. MLC80-nál A rendszert jóváhagyják tömeggyártásra, ekkor már befejeződtek a hosszú távú tesztelések sikeresen.

## 3.3 Freeze/Release idővonal/fázis

*1.5 ábra*

A szoftver kiadására készülés időszakát három fázisra lehet bontani, Pre-Freeze, Freeze, és Release, ez az 1.5-ös ábrán látható. A Pre-Freeze periódusban felkészülnek a tesztre illetve a Freeze fázisra, mindenki ellenőrzi a saját projektjét. Itt a legfontosabb dolog, hogy a task-okban ne maradjon Critical Change, ezeket meg kell csinálni a Freeze periódus előtt és a Test Readiness Review Taskokban jóvá kell hagyni. A Critical Change-t bővebben a negyedik fejezetben fogom kifejteni. A Freeze fázisban elkezdődnek a tartós tesztek, közben az összes hibát(bug-ot) folyamatosan fixálni javítani kell a fejlesztőknek. A Release periódusban a fejlesztők megkapják a Release Review task-okat, amelyekben meg kell nézniük, hogy minden fejlesztés lezárult-e a verziókövető rendszer egyes szerverén. Majd ha ezek megtörténtek a periódus végén a csoportvezetők is kapnak egy úgynevezett supervisor task-ot és ebben nekik is jóvá kell hagyniuk azt, amit a fejlesztőjük elfogadott. Itt a vezető ugyanazt nézi át mint a fejlesztő, csak nem nézi a technikai hátteret, hanem a dokumentumokat és task-okat, hogy ezek mind levannak-e zárva, illetve a fejlesztés befejeződött-e. Egy csoportvezetőhöz több funkció is tartozhat.



*1.6 ábra*

Itt az 1.6-ös ábrán látható a Test Readiness Review és a Release Review-ra egy példa, hogy hogyan is néznek ki ezek a task-ok.

## 3.4 Release munka

A feladatok task-okban nyílnak ezek a különböző periódusok ellenőrzésére szolgálnak. A release sheet az adott task fontosabb adatait tartalmazza. Egy-egy taskhoz tartozhatnak al-task-ok is amiket a fejlesztők kapnak, ezek a Test Readiness Review Task és a Release Review Task. Mint korábban említettem a csoportvezetők is kapnak egy gyűjtő task-ot, amiben szerepel az összes olyan funkció, ahol a funkció fejlesztő a beosztottja, úgynevezett SuperVisor Task-ok. A program szempontjából ez is egy fontos szempont, hogy megfelelően ki tudja listázni ezekhez a funkciókhoz tartozó task-okat a szoftvert használók számára. Továbbá előfordulhatnak olyan kivételes esetek is, minthogy nm nyílik Release Review Task, mivel éppen nem volt rá szükség, a programnak ezt is tudnia kell kezelnie, hogy ne keresse, ha nincs.

# Release dokumentáció

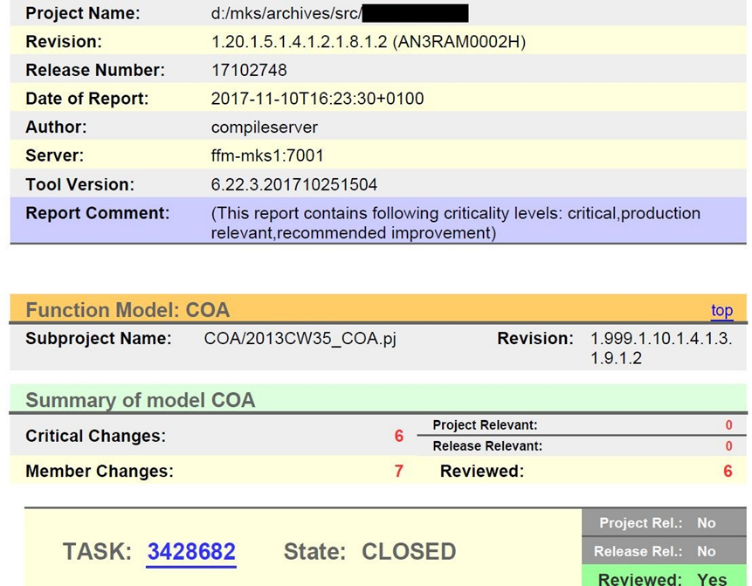
A szoftver legfontosabb funkciója a Release dokumentációk megfelelő kiértékelése lesz. Ezzel szeretnénk támogatni a csoportvezetők munkáját, hogy a program tudja nézni a verziókövető rendszerben az első szerveren a fejlesztési munkákat, és hogy ezek kapcsolódnak-e a hármas szerveren levő dokumentumokkal, illetve a SuperVisor task-ok elemzése is az applikáció feladata lesz.

## 4.1 Critical Change Report

A Critcal Change.-t csak a Test Readiness Review-nál kell nézni. Itt a programnak tudnia kell ellenőrizni a következőket:

* van-e olyan elem amely még nincs átnézve a fejlesztő által
* a report jó szoftverre készült-e
* a checkpoint és a release number megfelelő-e
* van-e benne olyan hiba, amivel nem mehet ki a release

Ezekre a kérdésekre a szoftverünknek mind-mind választ kell adnia. Alább látható egy Critical Change Report az 1.7-es ábrán.



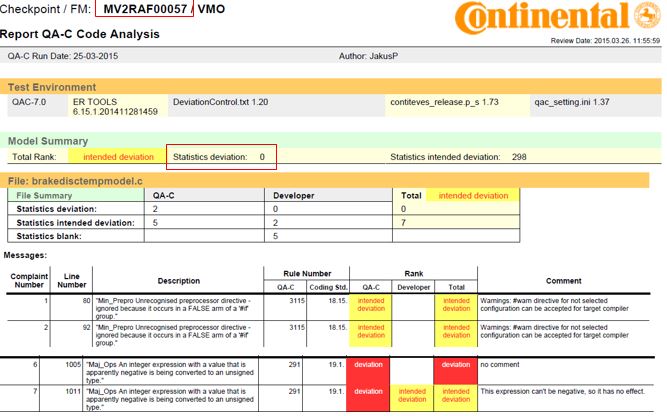
*1.7 ábra*

## 4.2 Compiler Report-CRR

A compiler report-oknál a szoftver warning-ok szerepelnek, amely fordítási hibát nem okozott. Ekkor a fejlesztőknek meg kell nézni az összes warning-ot, és mindegyikhez kommentet kell fűzniük. A fejlesztőknek itt ellenőrizni kell, hogy ezeknek a warning-oknak van-e funkcionális hatása. A programnak pedig azt kell vizsgálnia, hogy a report jó szoftverre készült-e, a megfelelő checkpoint-okat tartalmazzák, és, hogy minden warning-hoz tartozik-e komment. Ezt mind a Test Readiness Review mind pedig a Release Review-nál is nézni kell.

## 4.3 QA-C report

Ez egy statikus kód analízis, Misra C nyelvet használnak a Continental-ban. Az eltéréseket, figyelmeztetéseket el lehet nyomni a szoftverben, ezt intended deviation-ként tüntetik fel a reportok-ban. Release Review-nál végig kell néznie a fejlesztőnek, hogy ami deviációként szerepel annak van-e funkcionális hatása a szoftverre. Ha nincs, akkor azt kommentben meg kell indokolni normális magyarázattal, gondolok itt arra, hogy „no comment” és hasonló érdektelen kommentek nem elfogadottak, a programnak ezt is tudnia kell kezelni. Továbbá mint az előző kettő report-nál is elenőrizni kell, hogy a report jó szoftverre készült-e, megfelelőek-e a checkpoint-ok, minden QA-C deviációnál van-e komment, és figyelni kell arra is, hogy amelyik már a TB 2013-as verziója után készült azokban már itt sem lehet „no comment”. Test Readiness Review és Release Review-nál is nézni kell. QA-C report példa az 1.8-as ábrán látható.



*1.8 ábra*

## 4.4 Object Compare report

Az előző release szoftvert hasonlítja össze a jelenlegivel objektum szinten. A csoportvezető nézi, hogy létezik-e ez a dokumentum, a fejlesztő pedig azt hogy a változtatások szándékosak voltak-e az előző release óta.

## 4.5 Change Documentation report

Tulajdonképpen ugyanazt csinálja mint az object compare, ugyanúgy az előzőt hasonlítja össze a mostanival, annyi különbséggel hogy itt fájl revízió szinten végzi.

## 4.6 Change System Relevant Code report

Vannak olyan fájlok amelyek változtatása a rendszerre is hatással lesz, és ezért az ilyen típusú fájlokat olyan fejlesztőknek kell jóváhagynia, akik expert/base szinten vannak. Itt ellenőrizni kell, hogy van-e aláírás/jóváhagyás az ilyen fájlokon, jó szoftverre készült-e release number és checkpoint azonosító megfelelő-e.