Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Javaban írt webalkalmazások tesztelési módszerei

Kovács Ádám (C9TSBJ), IV. évf, (BSc) mérnök inf. szakos hallgató
Külső konzulens: Herman Tamás, MI Software Kft.
Belső konzulens: Bartha Tamás egyetemi docens, MIT
Informatikai technológiák szakirány, Rendszertervezés ágazat
Önálló laboratórium 1 összefoglaló
2009/10. II. félév

Azért választottam ezt a témát, mert a munkahelyemen indult egy Java EE projekt, és megbíztak a tesztelés megtervezésében, valamint a konfiguráció összeállításában.

A félév első felében megismerkedtem a Java EE alkalmazások tesztelésének alapjaival. Eldőlt, hogy Spring framework alapon fejlesztjük az alkalmazást, így az irodalomkutatást a Spring tesztelési mechanizmusaira koncentráltam.

Először megismerkedtem a Spring Source környezettel és a rendszer JUnit 4-gyel való együttműködésével.

Megismertem a DBUnit nevű, JUnit-ra épülő adatbázis tesztelő eszközt, egy példaalkalmazáson kipróbáltam az adatbázis fájlból való feltöltését, fájban definiált állapottal való összehasonlítását, és a többi funkcióját (a Spring Framework-kel együttműködve).

A tesztelés tervezetét három fő fázisban építettem fel: Specifikáció, Tesztek futtatása, Riportgenerálás.

A Specifikáció fázisban rögzítjük a követelményeket (mit várunk el az alkalmazástól funkcionálisan, teljesítményben), és ezek alapján létrehozzuk a teszteseteket. Mind a követelmények, mind a tesztesetek leírása jól feldolgozható formában kell, hogy legyen, ezért XML fájlokban írom le (ehhez egy XSD sémát is létrehoztam, az egységesség miatt).

A tesztelésnek két alapvető megközelítése van. Az egyik szerint a kódba beépítjük az elvárásainkat (assert), a másik megközelítés szerint definiálunk egy elvárt kimenetet (gold file), és azzal hasonlítjuk össze a teszt kimenetét. Mindkét megközelítést alkalmazni fogom, a megfelelő követelményeket az annak megfelelő módon fogom tesztelni. Az automatizált tesztek futtatásában nagy segítséget jelentenek a Continuous Build rendszerek, melyek figyelik a repository változásait, és a forráskód megváltozása esetén automatikusan lefuttatják a megfelelő teszteket, majd értesítik a fejlesztőket az eredményről. Egyes teszteket nem éri meg automatizálni, előfordulhat, hogy a teszt bonyolultságához képest túlságosan komplikált felparaméterezni a tesztelő eszközt.

A harmadik fázis a tesztek dokumentálása, a Riportgenerálás. Fontos, hogy később is felhasználható jelentéseket gyártsunk a tesztek futásáról, ezeket a riportokat szintén generáljuk a tesztek futtatásainak eredményeiből. Felsoroljuk a teszteseteket és kimenetelüket (hibamátrix), valamint a követelmények lefedettségét (traceability matrix), és magukat a futási eredményeket. Kódminőség-biztosítási szempontokat is vizsgálunk és riportolunk (kód tesztekkel lefedettsége, dokumentáció megléte, és különféle minőségbiztosítási metrikák). Az egész riportgenerálás automatizált, a létrejött dokumentum könnyen böngészhető, és használható kell, hogy legyen.

A félév során megismerkedtem a részfeladatokra alkalmas eszközökkel, ezekből összeállítottam egy konfigurációt. Sajnos idő szűkében nem volt alkalmam minden eszközt behatóan kipróbálni, így maradtak nyitott kérdések, néhány feladatról még nem dőlt el, hogy maradjunk a már bevált eszköznél, vagy az alternatív, újabb megoldás lesz a hatékonyabb. A következő félévben remélhetőleg alkalmam lesz minden tesztelési lépés gyakorlati alkalmazására, és a projekt is már olyan szintű lesz, hogy lehessen rajta rendszer szintű teszteket végrehajtani, valós helyzetben, nem példaalkalmazás szintjén.