# SOFTWAROVÉ METRIKY – STABILITA SYSTÉMU

**Vladimír Vaněk**
Ostravská Univerzita, vanna@email.cz

**ABSTRAKT:**
Vývojové týmy po celém světě se v dnešní době potýkají s velmi podobnými problémy. Není přitom vůbec důležité, o vývoj jakého produktu jde nebo v jaké technologii vývoj probíhá.

Snažíme se dosáhnout domluvy se zákazníkem na požadavcích a na funkčnosti aplikace, aplikaci naimplementovat, otestovat a to vše ve stanoveném čase a ve stanoveném rozpočtu. Vývoj softwaru tedy obsahuje hned několik naprosto odlišných disciplín [Rup08]:

- Byznys model (Business modeling)
- Správa požadavků (Requirement management)
- Analýza (Analysis)
- Design
- implementace (Implementation)
- Testování (Test)
- Nasazení systému (Deployment)
- Konfigurační management a změnové řízení (Configuration & Change management)
- Řízení projektu (Project management)

Existuje nespočet různých metodik, které nám nabízejí řešení. Některé z nich existují již desítky let (Waterfall [Wat08]), jiné jsou žhavou novinkou (OpenUP [Opp08]). Metodiky jsou vytvářeny soukromým sektorem, ale i jako mezinárodní standardy (ISO, CMMI, …). Všechny tyto metodiky mají jedno společné: popisují procesy, případně doporučené praktiky (tzv. best practices).

Aby byl náš projekt úspěšný, je důležité sledovat naše procesy a praktiky v každé disciplíně. Například špatné chování programu může být důsledkem špatné specifikace, špatné implementace a nedostatečného testování nebo prostě špatným návrhem aplikace.
Objektivní sledování procesů je sledování procesů založené na měřitelných, a tedy objektivních hodnotách. Tak jako existuje spousta metodik pro vývoj softwaru, tak existuje i spousta možností, jak měřit kvalitu procesů a kvalitu výsledného produktu.
Propojení procesů a metrik je přitom oblast, která je až trestuhodně zanedbávána, a to jak v praxi, tak ve školní výuce. Přitom jsou tyto dvě oblasti navzájem logicky provázány. Velmi těžko například zjistíme, zda je pro nás daný proces dostatečně efektivní nebo ne, pokud nemáme definována kritéria úspěšnosti a způsob měření.

V následujícím textu se podíváme na praktickou stránku použití metriky pojmenované *Stabilita* na dvou reálných projektech.

**KLÍČOVÁ SLOVA:**
Softwarové metriky, Scrum, CMMI, RUP, Stabilita systému

## VLASTNÍ PŘÍSPĚVEK

V tomto příspěvku se budeme věnovat praktickým zkušenostem při zavádění metrik na dvou projektech. Projekty jsou podobného charakteru a oba následují stejný procesní model upravený pro konkrétní situaci. Sledovali jsme vždy vývoj jedné verze produktu.

**Projekt A**
Vývoj nové aplikace pro telekomunikační společnost. Cílem bylo vytvořit multiplatformní aplikaci na straně serveru, která by nabízela služby – aplikace nemá uživatelské rozhraní.
Projekt následoval praktiky RUP a pracovalo na něm 15 – 20 lidí. Jednalo se o vývoj v JAVĚ.
Pro plánování projektu byl použit nástroj Microsoft Project a pro plánování úkolů JIRA [Jira] issue tracking. Jelikož JIRA byla napojena na databázi MySQL, jsou reporty v dalších kapitolách tvořeny pomocí nástroje BIRT [Birt].

**Projekt B**
Vývoj nové aplikace pro mediální společnost. Cílem bylo vytvořit aplikaci pro správu aktivit společnosti. Cílová platforma je MS Windows.
Projekt postupně adoptoval základní praktiky RUP a pracovalo na něm také 15 – 20 lidí. Aplikace byla vyvíjena v jazyce C++ a C#.
Plánování probíhalo v nástroji Gemini [Gem], který pracuje s databází MS SQL, reporty jsou vytvářeny pomocí reportovacích služeb nástroje MS SQL.

**Definice: Stabilita systému**

Je poměr mezi počtem vyřešených a počtem známých chyb.

```
Stabilita = Vyřešené chyby / Všechny (známé) chyby
```

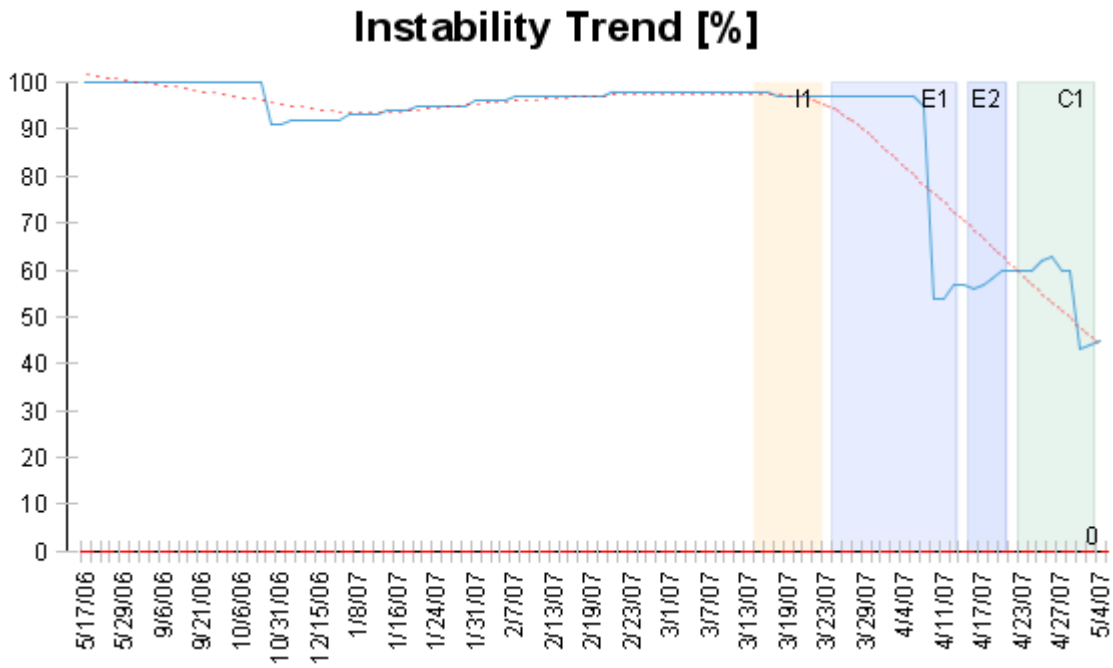
Stabilita nám říká, kolik chyb v produktu se nám podařilo odstranit a jak velkou část systému ještě musíme opravit. Všechny nevyřešené chyby nám tak zvyšují nestabilitu systému, proto se velmi často používá také pojmu nestabilita systému.

**Definice: Nestabilita systému**

```
Nestabilita = 1 – Stabilita
```

Podle potřeby tak můžeme bez obav zaměňovat stabilitu a nestabilitu, což je obzvláště vhodné pro prezentační účely. Používání termínu nestabilita působní psychologicky více varovně než použití termínu stabilita.

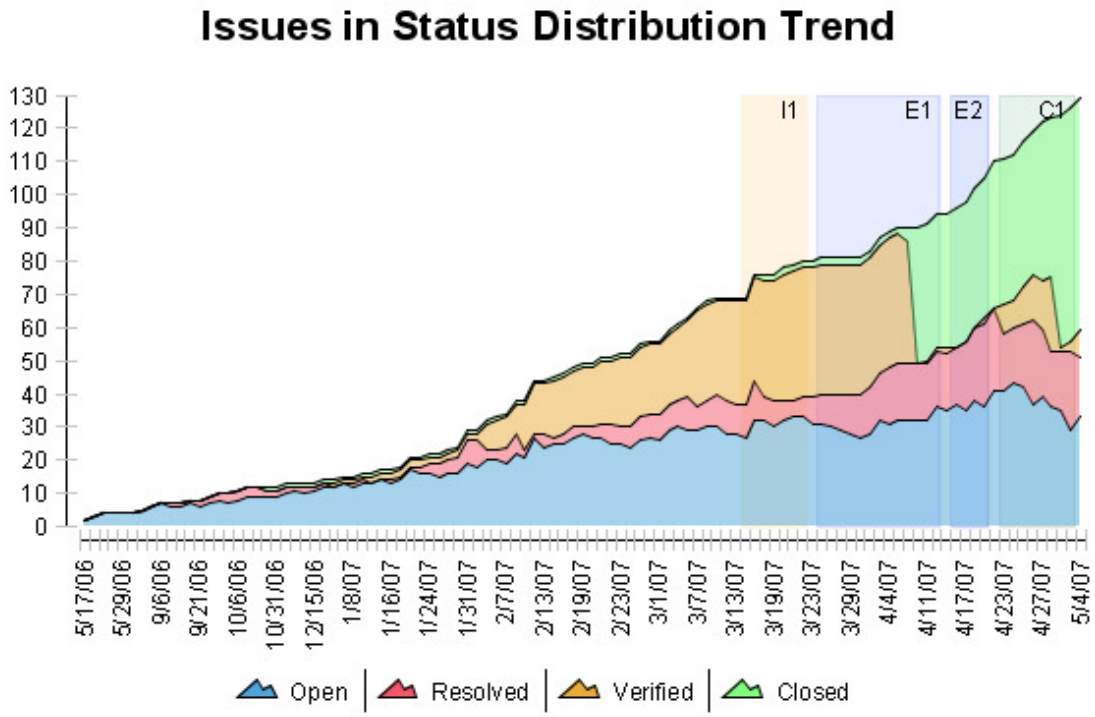
**Projekt A**




**Obr. 1: Nestabilita - trend (zobrazená modře)**

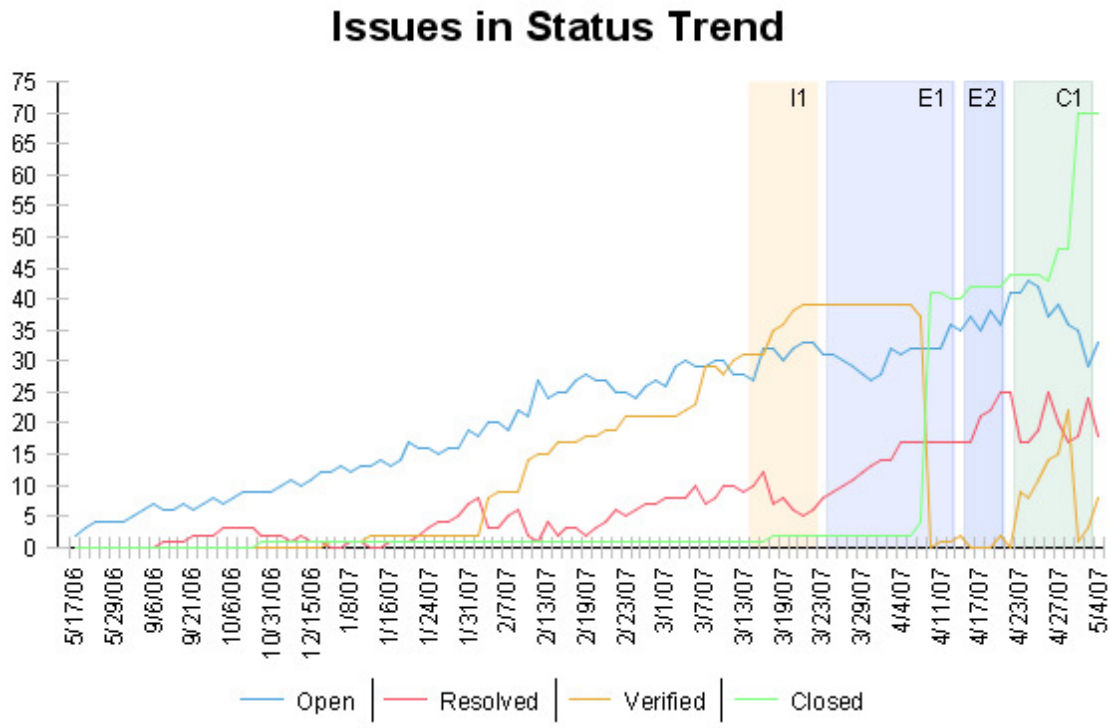
Na začátku projektu A je stabilita velmi nízká, to je však očekávaný stav. Největší prioritou v této fázi je dokončit a ověřit architekturu a kritické požadavky na systém. Oprava všech nalezených chyb není tudíž prioritní věc. Na konci projektu ovšem chceme mít v systému chyb co nejméně, ale nesnažíme se o 100% stabilitu.
Vždy nám v aplikaci nějaké chyby zůstanou. Snažíme se ale docílit toho, aby chyby nebyly v kritických částech aplikace a aby chyby, které zůstanou v aplikaci, měly minimální dopady na práci systému.

Na grafu na obrázku 1 můžeme vidět zavlečené chyby z minulých verzí aplikace, které musí být právě v této verzi vyřešeny. Proto je počáteční nestabilita 100%.
Během inception I1 a elaboration E1, E2 (viz. [Rup08]) je nestabilita stále vysoká. Zde se snažíme o kontinuální opravování chyb, ale přece jenom jde hlavní úsilí směrem k vývoji nových vlastností aplikace, takže se snažíme hlavně opravovat a řešit kritické chyby systému.
Další fáze je construction C1. Zde by již naším cílem mělo být, aby nestabilita rapidně klesala (tj. stabilita rostla).
Za povšimnutí a hlubší rozbor určitě stojí dva skoky v E1 a C1, pojďme se na ně nyní blíže podívat.
K tomu využijeme podobný graf, který nám umožní dešifrovat skoky v minulém grafu nestability – je jím graf s kumulativním rozdělení issues dle statusu.



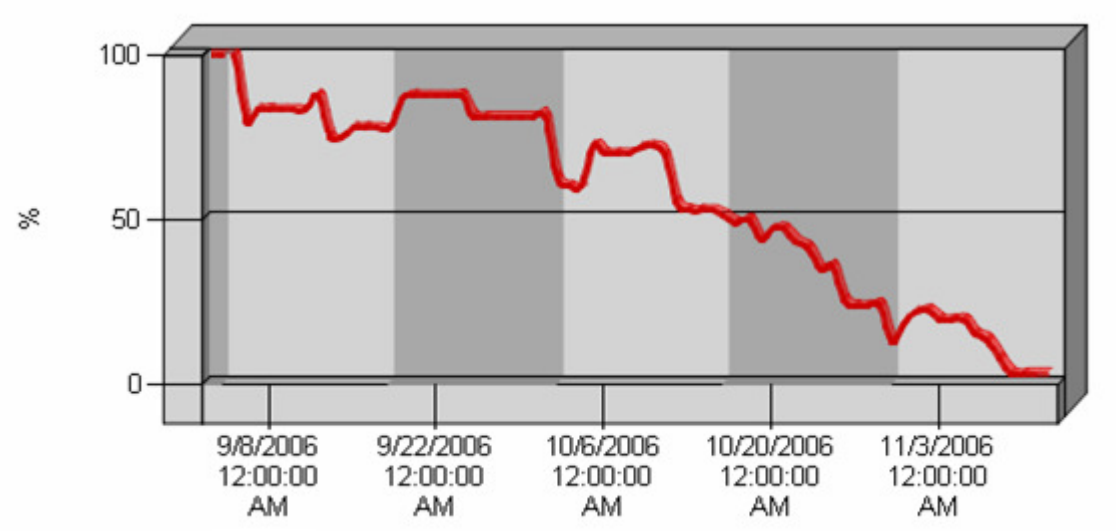

**Obr. 2: Kumulativní rozdělení issues podle statusu**

V grafu na obrázku 2 můžeme zcela snadno identifikovat stejné dva skoky jako v grafu na obrázku 1 a vidíme, že, jak v prvním, tak v druhém případě, byly zapříčiněny přepínáním issue mezi stavy *Verified* a *Closed*. Tento úkol stojí na bedrech Change Control Board [Rup08], takže už nyní přesně víme, kde tento problém můžeme řešit.
Pro zajímavost si můžeme ukázat, že nás ke stejnému výsledku dovede i nekumulativní verze stejného grafu.




**Obr. 3: Rozdělení issues podle statusu**

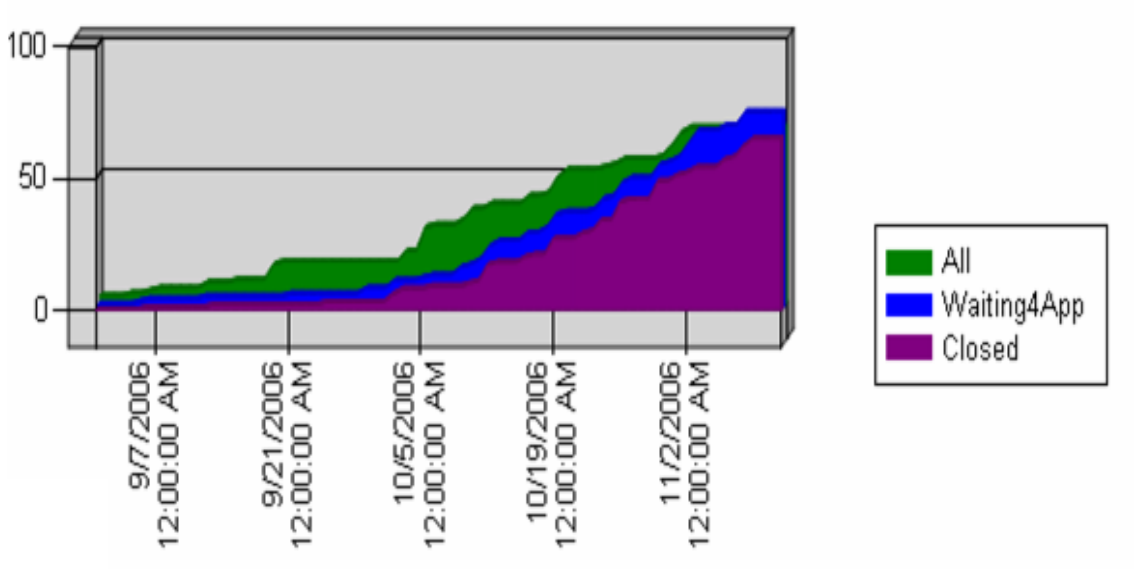
## Projekt B




**Obr. 5: Nestabilita**

Pro projekt B máme k dispozici data až do konce projektu, kdy aplikace byla nasazena v praxi. V první části vidíme rozumnou míru nestability systému, což je velmi dobré.
Bohužel dojde k nepříjemnému skoku směrem nahoru a poté se již nestabilita snižuje příliš pomalu. Dokonce těsně před dokončením se počet nevyřešených problémů lehce blíží 30%, což je příliš. Také bychom v tuto chvíli potřebovali vidět rozdělení chyb podle jejich závažnosti. Pokud by totiž tyto chyby měly nízkou závažnost, pak by výsledek byl stále ještě dobrý. Bohužel realita tak příznivá nebyla a vesměs se jednalo o chyby se střední a vysokou závažností. V grafu na obrázku 6 můžeme vidět, že opravy některých minoritních chyb, které

se nestihly otestovat, byly přesunuty do následujících verzí. (Proto je na konci projektu počet issue ve stavu „waitting for approval“ vyšší než ve stavu closed).




**Obr. 6: Nestabilita**

## Závěr

Elementární informaci, kterou nám tato metrika může přinést, zjistíme ze sledování jejího trendu. Trend stability by měl mít stoupající charakter (v případě nestability sestupný), tak, ať ke konci projektu jsme schopni doručit produkt dobré kvality. Pokud trend není stabilní a vykazuje razantní změny je důležité takové změny detailně prostudovat a hledat tzv. root cause, neboli hlavní příčiny těchto skoků.

## LITERATURA

[NI02] NITS: Software Errors Cost U.S 2002.
http://www.nist.gov/public_affairs/releases/n02-10.htm
[Br00] Bruegge, B., Dutoit, A., H.: Object-oriented software engineering. 1st edition. Prentice-Hall 2000. ISBN 0-13-017452-1.
[Rup08] RUP http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process
[Wat08] Waterfall http://en.wikipedia.org/wiki/Waterfall_model
[B05] Buchalcevová, A.: Metodiky vývoje a údržby informačních systémů. 1. vydání. Grada. Praha 2005. ISBN 80-247-1075-7.
[Ca01] Carda, A., Kunstová, R.: Workflow: Řízení firemních procesů. Grada. Praha. 2001. ISBN 80-247-0200-2.
[Do95] Dori, D.: Object-Process Analysis: Maintaining the Balance Between System Structure and Behavior. Journal of Logic and Computation 5, page 227-249. 1995.
[Ga94] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
[Hl97] Hlavenka, J., a kolektiv: Výkladový slovník výpočetní techniky a komunikací. 3. vydání. Computer Press. Praha 1997. ISBN 80-7226-023-5.
[IDS] IDS Scheer: Úvod k procesnímu řízení a tvorbě procesního modelu v ARIS. prezentace IDS Scheer Ostrava, Brno. 2004.

[K03] Kruchten, P.: The Rational Unified Process: An Introduction. 3rd Edition. Addison Wesley 2003. ISBN 0-321-19770-4.
[LPV] Lukasík, P., Procházka, J., Vaněk, V.: Procesní řízení. Učební text pro distanční studium. Ostrava 2005. Přf. Ostravská univerzita.
[OPM] Object Process Methodology home page. http://www.objectprocess.org/
[O06] OMG home page. www.omg.com
[Ře99] Řepa, V.: Analýza a návrh informačních systémů. 1. vydání. Ekopress. Praha 1999. ISBN 80-86119-13-0.
[So92] Soin, S., S.: Total quality control essentials: key elements, methodologies, and for managing success. McGraw-Hill, Inc. 1992. ISBN 0-07-113664-9.
[Von00] Vondrák, I.: Business Process Studio, version 3.0. Elektronický manuál. 1998-2000. http://vondrak.cs.vsb.cz/download/bpstudio_eval/manual.pdf.
[Von02] Vondrák, I.: Úvod do softwarového inženýrství. Učební text. VŠB-TU. Ostrava. 2002.
[Von04] Vondrák, I.: Metody byznys modelování. Učební text. VŠB-TU. Ostrava. 2004.
[Opp08] OpenUp http://en.wikipedia.org/wiki/OpenUP/Basic
[Jira] Jira homepage: http://www.atlassian.com/software/jira/
[Birt] http://www.eclipse.org/birt/phoenix/