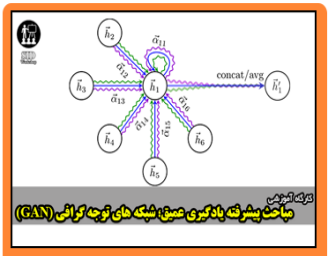
کارگاه های آموزشی مرکز اطلاعات علمی جهاد دانشگاهی

کارگاه آنلاین مکالمه روزمره انگلیسی

کارگاه آنلاین آموزش استفاده از وب آو ساینس



مباحث پیشرفته یادگیری عمیق؛
شبکه های توجه گرافی
(Graph Attention Networks)

INTERNATIONAL CONFERENCE ON
RESEARCH IN SCIENCE AND TECHNOLOGY
14 December 2015, Kualalumpur –Malaysia

بررسی تاثیر طراحی نرم افزار نوع-حالت گرا در خواص کیفی نرم افزار از نظر پنهان سازی اطلاعات

ابراهیم گنجعلی پور
دانشگاه صنعتی امیر کبیر (پلی تکنیک تهران)
ganjali@aut.ac.ir

چکیده

اشیاء با رفتار و خصوصیات جهان را مدل می کنند اما اشیاء جهان واقعی دارای حالت هستند و انتقال حالت دارند. در زبان های برنامه نویسی شیء گرا برای شیء نوع داده حالت تعریف نمی شود و فیلد ها حالت شئ را تعیین می کنند. نوع-حالت گرایی با در نظر گرفتن انواع حالت برای اشیاء، گونه برنامه نویسی جدیدی را معرفی می کند که در آن هر شیء به مانند ماشین حالت می تواند در حالت های تعریف شده به کار رود به طوری که در هر حالت متد ها وخواصی از شیء که در آن مجاز است دسترس پذیر خواهد بود و امکان انتقال حالت با فراخوانی توابع وجود دارد. این تحقیق نشان می دهد که در نوع-حالت گرایی نحوه اتصال میان اشیاء و قراردادهای استفاده از آنها موجب بیشتر شدن مخفی سازی اطلاعات (مخفی سازی فیلد ها و متد ها) نسبت به شی گرایی شده است و همچنین تاثیر این گونه طراحی در کیفیت نرم افزار (عملکرد، قابلیت استفاده، قابلیت استفاده مجدد، قابلیت تعمیر و نگهداری، کارایی و قابلیت حمل) مورد تحلیل قرار گرفته است.

واژگان کلیدی:
نوع-حالت – برنامه نویسی نوع-حالت گرا – طراحی نرم افزار شئ گرا – فاکتور مخفی سازی اطلاعات – خواص کیفی نرم افزار

مقدمه

با بلوغ صنعت نرم افزار، توسعه نرم افزار از پیاده سازی ساختارهای داده و الگوریتم ها به سمت سیستم های با اجزا قابل استفاده مجدد نوعی، حرکت کرده است. این تغییر دستاوردهای بزرگی در تولید نرم افزار داشته است. اما بهره گیری کامل از مزایای مهندسی نرم افزار مبتنی بر اجزا قابل استفاده مجدد نوعی، نیازمند مدل های بهتری می باشد (Aldrich et al, 2009). شیء گرایی با معرفی کلاس به عنوان عنصر مرتبه اول برنامه نویسی و نیز ارائه ویژگی هایی چون پنهان سازی اطلاعات، تجرید داده، کپسول بندی، چند ریختی و وراثت توانسته مدل برنامه نویسی مطلوبی را ارائه دهد. اشیاء با رفتار و خصوصیات، جهان را مدل می کنند اما اشیاء جهان واقعی دارای حالت هستند و انتقال حالت دارند. این در حالی است که زبان های شیء گرا وجود حالت برای شیء را پشتیبانی نمی کنند. به نظر می رسد که با در نظر گرفتن انواع حالات برای اشیاء، تغییرات مطلوبی در روش برنامه نویسی صورت پذیرد. محققان با ارائه زبان Plaid، برنامه نویسی نوع-حالت گرا را معرفی کردند. در این گونه برنامه نویسی، نوع-حالت نشان دهنده این است که چطور با توجه به حالت شیء در زمان اجرا ، عملگرهای قابل اجرا بر روی آن شیء تغییر می کند. نوع-حالت یک تجرید قدرتمند است که به برنامه نویس برای مدل کردن و استفاده مجدد از اشیاء به صورت صحیح، کمک می کنند (Aldrich et al, 2011).

روش طراحی نوع حالت گرا با هدف افزایش سلاست و انعطاف پذیری برنامه نویسی و نیز بهبود قابلیت اطمینان نرم افزار بررسی شده است. در این تحقیق معیار های طراحی شئ گرا (MOOD) مورد مطالعه قرار گرفته اند و از میان این معیار ها، فاکتور مخفی سازی اطلاعات مورد تمرکز واقع شده است. این فاکتور احتمال مخفی بودن اطلاعات در برنامه را نشان می دهد. در اینجا با قوایین احتمال ریاضی بهبود این فاکتور در نوع حالت گرایی نسبت به شی گرایی نشان داده شده است و نهایتا اثر مثبت طراحی نوع حالت گرا در کیفیت نرم افزار از نظر مخفی سازی اطلاعات تحلیل شده است.

برنامه نویسی شئ گرا

در برنامه نویسی شیء گرا شئ توسط ایجاد الگویی برای حالت محلی(داده هایش) و متدهایش، تعریف می شوند. این الگو کلاس نامیده می شود و شبیه یک نوع داده است. به عبارت دیگر شیء نوعی ماژول شامل داده و زیرروال است. شیء نوعی موجودیت است که یک حالت داخلی دارد (داده هایش) و می تواند به پیام ها (فراخوانی زیر روالهایش) پاسخ دهد و اشیاء با فرستادن پیام با یکدیگر تعامل دارند. در این گونه برنامه نویسی از اشیاء و تعامل بین آن ها به منظور طراحی برنامه ها استفاده می شود. تکنیک های برنامه نویسی با این روش شامل ویژگی هایی مانند تجرید داده، کپسول بندی، چند ریختی، وراثت و ماژولاریتی می باشد (Churcher et al, 1995).

برنامه نویسی نوع-حالت گرا

نوع-حالت نشان دهنده این است که چطور با توجه به حالت شیء در زمان اجرا، عملگرهای قابل اجرا بر روی آن شی تغییر می کنند. چک کننده حالت داده ایستا می تواند تضمین کند که یک متد تنها در زمانی که شی در حالت تعریف شده برای آن متد قرار دارد، فراخوانی گردد. (Wolff et al, 2011) یک زبان شی گرا صوری با حالت تغییر پذیر که تغییر نوع-حالت و چک کردن نوع-حالت را با یکدیگر ادغام کرده است، فرموله کرده اند. این زبان مفهوم نوع تدریجی را برای نوع-حالت توسعه داده است چک کردن نوع-حالت تدریجی با افزودن خودکار چک کردن در زمان اجرا، چک کردن ایستا و پویا را به صورت یکپارچه با همدیگر ترکیب کرده است. این روش به هدف افزایش سلاست و انعطاف پذیری برنامه نویسی شیء گرا به منظور بهبود

قابلیت اطمینان نرم افزار مورد بررسی واقع شده است. نوع-حالت مستقیما در زبان معرفی شده است و هم چنین این زبان از نوع تغییر پذیر نیز پشتیبانی می کند. نوع یک شیء تعیین می کند که چه متدهایی را می تواند فراخوانی کند. در بیشتر زبان های برنامه نویسی نوع شیء در مدت حیات شیء ثابت است، اما در عمل در زمان اجرا با فراخوانی یک شیء، حالت آن شیء در زمان اجرا تغییر می کند (به عنوان مثال یک فایل باز شده نمی تواند مجددا باز شود) (Wolff et al, 2011) . در سطح وسیع تر، نوع خاصیتی است که بدون تغییر باقی می ماند، بنابراین نمی توان مورد بررسی قرار داد که چطور با تغییر حالت، خواص یک شی تغییر می کند. برای غلبه بر این مشکل مفهوم نوع-حالت را معرفی کرده اند (Strom et al, 1986). به عنوان مثال، یک نمونه تجرید فایل شی گرا را در نظر بگیرید. مقدار فیلد توصیفگر یک فایل بسته شده تهی[1] است. بنابراین متد بستن فایل، ابتدا مقدار توصیفگر فایل را چک می کند که آیا تهی است یا خیر، و در صورت تهی بودن، درخواست بستن فایل را رد می کند. این روش دقت و فهم برنامه را کم می کند و تضمینی وجود ندارد که در برنامه عمگرهای غیر مجاز اجرا نشوند. برای غلبه بر این مشکل، زبان نوع-حالت گرا [2] به طور مستقیم از تعریف نوع-حالت پشتیبانی می کند (Aldrich et al, 2009). برای نمونه در یک زبان مبتنی بر کلاس (مانند Smalltalk) که تغییر پویای کلاس شی را پشتیبانی می کند، نوع-حالت می تواند به عنوان یک کلاس نشان داده شود و به صورت پویا به روز رسانی گردد. اشیاء می توانند واسط ها، رفتارها و ارائه های وابسته به نوع-حالت داشته باشند. نقض پروتکل در چنین زبانی منجر به خطای "متد یافت نشد"[3] می گردد با چک حالت ایستا دشوار می‌توان اجتناب از چنین خطاهایی را تضمین کرد. چک نوع-حالت ایستا به خصوص در حضور نام مستعار[4] دشوار است. فوگو (Fahndrich and DeLine, 2004) اولین سیستم نوع-حالت ماژولار برای نرم افزارهای شی گراست. این سیستم اشیاء را به عنوان "نا مستعار" و "محتملا مستعار" در نظر می گیرد و تنها اشیاء نا مستعار می توانند تغییر حالت داشته باشند. Jonathan و همکاران این روش را با اضافه کردن مفهوم اجازه دسترسی به منظور اجازه تغییر حالت حتی در حالتی که اشیاء نام مستعار دارند، توسعه دادند (Bierhoff and Aldrich, 2007) و زبان نوع حالت گرای Plaid را ارائه کردند. در ادامه به معرفی مختصر این زبان می پردازیم.

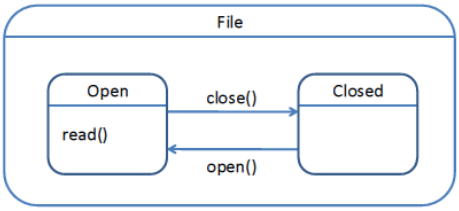
در زبان Plaid قابلیت تعریف حالات برای شی وجود دارد و در نظر گرفتن عمل تغییر به حالت جدید برای انتقال بین حالات ممکن می باشد. انتقال حالت میان اشیا می تواند متدها و فیلدها را اضافه، کم و یا به روز رسانی کند و در نتیجه بر ارائه و رفتار شی تاثیر می گذارد. برای مثال یک شی فایل را در نظر بگیرید. بعضی فایل ها باز و بعضی بسته هستند. یک فایل باز با صدا زدن متد close بسته و یک فایل بسته با صدا زدن متد open باز می شود.

[1] null

[2] typestate-oriented programming (TSOP) language

[3] method not found

[4] aliasing



شکل(۱) : فضای حالت فایل.

فضای حالت فایل در پلاید می تواند مطابق شکل (۱) پیاده سازی شود. کلمه ی state برای تعریف حالت استفاده می شود. فایل شامل فیلدها و متدهایی است که بین فایل های باز و بسته مشترک هستند. در این موارد، تنها نام فایل به اشتراک گذاشته می شود. فیلدها با کلید val تعریف می شوند.

مطابق کد (۱)، OpenFile و ClosedFile متدها و فیلدهایی را تعریف می کنند که مختص حالات باز و بسته هستند و هر دو زیرحالت فایل هستند. کلمه ی case برای تعریف تخصصی استفاده می شود. متدها با کلمه ی کلیدی method تعریف می شوند. فایل های باز متد read برای خواندن فایل و متد close برای بستن فایل را دارند و فایل های بسته تنها متد open را دارند. قواعد نحوی داخل متدها کمی متفاوت است. یک شی با متغیر x می تواند با دستور x<-S به حالت S تغییر حالت دهد.

```
1  state File {
2     val filename;
3  }
4  state OpenFile case of File = {
5     val filePtr;
6     method read() { ... }
7     method close() { this <- ClosedFile; }
8  }
9  state ClosedFile case of File {
10    method open() { this <- OpenFile; }
11 }
```

شکل(۱): حالات فایل در زبان پلاید.

فایل کلاینت در کد (۲) نشان داده شده است. متد readClosedFile یک فایل ورودی را می گیرد، آن را باز می کند، می خواند، می بندد، و مقدار خوانده شده از فایل را بر می گرداند. اگر فایل ورودی بسته باشد، همه ی صدا زدن های متدها به

```
1  method readClosedFile(f) {
2     f.open();
3     val x = f.read();
4     f.close();
5     x; //return
6  }
```

کد(۲): کلاینت فایل در زبان پلاید.

درستی کار می کنند. اگر فایل باز ورودی داده شود، صدا زدن متد open شکست می خورد. نویسندگان کتابخانه نیازی به نوشتن کد کنترل خطا برای کنترل این شرایط ندارند. این مزیت مهم پلاید می باشد (Joshua et al, 2011).

روش طراحی نوع حالت گرا

در یک طراحی نوع حالت گرا موجودیت های طراحی نرم افزار حالت کلاس ها هستند که در ارتباط با یکدیگر طرح نرم افزار را شکل می دهند. در ادامه انواع ارتباطات را مرور می کنیم .

۱. رابطه انجمنی: این رابطه زمانی برقرار می شود که یک کلاس در یکی از حالاتش به منظور سرویس دادن به کلاس دیگر عضو آن باشد.
۲. رابطه تجمعی: رابطه شمول، و یا جز به کل بودن دو کلاس نسبت به هم را نشان می دهد. در این رابطه، یک کلاس شامل کلاس دیگری در یک حالت اولیه خواهد بود.
۳. رابطه ترکیب: دو کلاس زمانی ترکیب می شوند که یکی مالک دیگری باشد. این رابطه، مشابه رابطه تجمعی خواهد بود.
۴. رابطه وابستگی: این رابطه زمانی برقرار می شود که یک حالت از کلاس، از کلاس دیگری در یکی از حالاتش نمونه سازی کندیا اینکه ارجاع داشته باشد.
۵. رابطه تعمیم: یک حالت از یک کلاس، همه یا حالاتی از یک کلاس دیگر را ارث بری کند و یا اینکه در یک کلاس حالات از هم ارث بری کنند.

معیار های طراحی نرم افزار

معیار های نرم افزاری مقادیر کمی هستند که درک بازدهی فرایند تولید نرم افزار را امکان پذیر می سازند. در این تحقیق معیار های طراحی شئ گرا (MOOD) مورد مطالعه قرار گرفته اند. این معیار های عبارتند از فاکتور مخفی سازی متد یا فیلد، فاکتور وراثت متد یا فیلد، فاکتور پلی مورفیسم و فاکتور اتصال. در ادامه روی فاکتور مخفی سازی متمرکز می شویم.

فاکتور مخفی سازی و تاثیر طراحی نوع حالت گرا در خواص کیفی نرم افزار

فاکتور مخفی سازی نشان دهنده امکان فراخوانی تابع یا فیلد در برنامه است . اندازه گیری مخفی سازی اطلاعات در طراحی نرم افزار شامل دو فاکتور مخفی سازی متد ها (MHF) و مخفی سازی فیلدها (AHF) می باشد.

MHF احتمال دسترس پذیری یک تابع است که در طراحی شئ گرا به طور زیر محاسبه می شود. اگر NF_{total} متغیر تعداد توابع موجود در یک کلاس باشد و $NF_{visible}$ متغیر تعداد توابع قابل دسترس باشد انگاه برای کلاس c داریم.

$$MHF(c) = 1 - NF_{visible}(c) / NF_{total}(c)$$

AHF احتمال دسترس پذیری فیلد هاست و بطور مشابه محاسبه می شود. اگر NA_{total} تعداد فیلدهای موجود در کلاس باشد و $NA_{visible}$ تعداد فیلد های دسترس پذیر باشد انگاه برای کلاس c داریم.

$$AHF(c) = 1 - NA_{visible}(c) / NA_{total}(c)$$

اما در طراحی نوع حالت گرا، حالات فیلدها یا توابع در احتمال دسترس پذیری انها موثر است. و احتمال دسترس پذیر بودن تابع یا فیلد برابر خواهد بود با احتمال اینکه تابع یا فیلد در حالت مجاز باشد و قابل دسترس باشد. بنابراین اگر NS تعداد حالات کلاس باشد انگاه داریم.

$$MHF_{TSOP}(c) = 1 - (1/NS) \times (NF_{visible}(c) / NF_{total}(c))$$
$$AHF_{TSOP}(c) = 1 - (1/NS) \times (NA_{visible}(c) / NA_{total}(c))$$

از مقایسه MHF یا AHF در طراحی شئ گرا با MHF_{TSOP} یا AHF_{TSOP} در طراحی نوع حالت گرا به این نتیجه می رسیم که در طراحی نوع حالت گرا مخفی سازی اطلاعات بیشتر است و این تاثیر مثبتی در کیفیت نرم افزار خواهد داشت که در ادامه به بررسی دقیق تر می پردازیم.

طراحی شیء گرا	مقایسه	طراحی نوع حالت گرا
MHF_{OOP}	$\geq$	MHF_{TSOP}
AHF_{OOP}	$\leq$	AHF_{TSOP}

پس از مطالعه تحقیقات انجام شده در خصوص رابطه میان خواص کیفی نرم افزار (عملکرد، قابلیت استفاده، قابلیت استفاده مجدد، قابلیت تعمیر و نگهداری، کارایی و قابلیت حمل) و معیار های طراحی نرم افزار نتایج زیر در خصوص فاکتور مخفی سازی اطلاعات استخراج شد.

	Quality Factors	MHF	AHF		Quality Factors	MHF	AHF
Efficiency	Time Behavior	↑	↑	Portability	Extensibility	↑	↑
	Resource Behavior	↑	↑		Adaptiveness	↑	↑
	Reply Time	-	-		replaceability	↑	↑
	Processing Speed	↑	↑		Transferability	↑	↑
	Execution Efficiency	↑	↑	Reliability	Maturity	↑	↑
	Robust	↑	↑		Fault Tolerance	↑	↑
Maintainability	Analyzability	↑	↑		Recoverability	↑	↑
	Changeability	↑	↑		Simplicity	↑	↑
	Stability	↑	↑	Usability	Attractiveness	↓	↓
	Testability	↑	↑		Understandability	↑	↑
	Adaptiveness	↑	↑		Learnability	↑	↑
	Understandability	↑	↑		Operability	-	-
	Error Debugging	↑	↑		Documentation	-	-
	Reusability	↓	↓				

جدول (۱): تاثیر فاکتور مخفی سازی اطلاعات در خواص کیفی نرم افزار

جدول (۱) به طور کامل چگونگی تاثیر فاکتور مخفی سازی (MHF و AHF) در خواص کیفی نرم افزار را نشان می دهد. ایتم های کیفی نرم افزار عبارتند از :

- کارایی (Efficiency) : بالا بودن فاکتور مخفی سازی نشان دهنده افزایش رفتار زمانی، بهره وری منابع، کارایی اجرایی و ... خواهد شد و درنتیجه نشان دهنده بیشتر بودن کارایی است (Sadaf et al, 2010) .

- قابلییت نگهداری (Maintainability) : بالا بودن فاکتور مخفی سازی نشان دهنده افزایش قابلیت تست، پایداری، قابلیت تحلیل، قابیلت تغییر، خطایابی و ... خواهد بود که موجب بیشتر شدن قابلییت نگهداری است (Henry et al, 1993).

- قابلیت حمل (Portability) : بالا بودن فاکتور مخفی سازی نشان دهنده افزایش قابلیت توسعه، جابجایی، انطباق و ... است که موجب بیشتر شدن قابلیت حمل است (Dallal, 2013).

- قابلیت اطمینان (Reliability) : افزایش مخفی سازی باعث افزایش شاخص بلوغ نرم افزار، تحمل خطا و ... می شود که در نتیجه قابلیت اطمینان نیز بیشتر خواهد شد (Victor et al, 1996).

- قابلیت استفاده (Usability) : بالا بودن مخفی سازی موجب افزایش قابلیت درک، یادگیری، مستندسازی و ... می شود که در نتیجه ان قابلیت استفاده افزایش خواهد یافت (Fatma, 2002).

بحث و نتیجه‌گیری

در شی گرایی حالت اشیا با استفاده از فیلدها توصیف می شود. به این معنی که برای شی فیلد هایی تعریف می شود و مقادیر مختلف برای فیلد ها نشان دهنده حالات شیء است. حال اگر حالات مختلف شیء عملکرد های متفاوتی نیاز داشته باشند، طراحی شیء گرا برای حالاتی که عملکرد متفاوت دارند کلاس های جدیدی تعریف می کند. برنامه نویسی نوع حالت گرا با معرفی نوع حالت ها راه حل طبیعی تری ارائه می دهد به این صورت که نوع حالت ها با استفاده از توابع خود عملکرد های متفاوت برای حالات اشیا را پشتیبانی می کنند. دراین تحقیق نشان دادیم که روش برنامه نویسی و طراحی نوع حالت گرا موجب بیشتر بودن فاکتور مخفی سازی نسبت به شئ گرایی است و از طرفی نتایج استخراج شده از تحقیقات بیان گر رابطه مستقیم میان خواص کیفی نرم افزار (عملکرد، قابلیت استفاده، قابلیت استفاده مجدد، قابلیت تعمیر و نگهداری، کارایی و قابلیت حمل) و فاکتور مخفی سازی است. بنابراین میتوان نتیجه گیری کرد که طراحی نوع حالت گرا از دیدگاه مخفی سازی اطلاعات، می تواند کیفیت بیشتری در طراحی نرم افزار ایجاد کند.

فضا های تحقیق زیادی در مورد زبان های نوع–حالت گرا وجود دارد. برای نمونه یکی از مسائل مطرح مشکل هم نامی (aliasing) است (Stephan, 2008)، این مشکل زمانی پیش می آید که شیء بیش از یک ارجاع داشته باشد که یکی از چالش ها درخصوص طراحی و پیاده سازی زبان های نوع–حالت گرا است. همچنین ارائه الگو های طراحی نوع–حالت گرا برای نرم افزارهای با حالات زیاد و ارزیابی مفصل تر از عملکرد این روش طراحی می تواند موضوعات جدیدی برای تحقیق باشند.

مراجع

Aldrich, J., Sunshine, J., Saini, D., & Sparks, Z. (2009, October). Typestate-oriented programming. In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. ACM.

Aldrich, J., Bocchino, R., Garcia, R., Hahnenberg, M., Mohr, M., Naden, K & Wolff, R. (2011, October). Plaid: a permission-based programming language. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion . ACM.

Shepperd, M. J., Chidamber, S., & Kemerer, C. F. (1995). Comments on" A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, 21(3), 263-265.

Wolff, R., Garcia, R., Tanter, É., & Aldrich, J. (2011). Gradual typestate. InECOOP 2011–Object-Oriented Programming . Springer

Strom, R. E., & Yemini, S. (1986). Typestate: A programming language concept for enhancing software reliability. Software Engineering, IEEE Transactions on, (1), 157-171.

Fink, S. J., Yahav, E., Dor, N., Ramalingam, G., & Geay, E. (2008). Effective typestate verification in the presence of aliasing. ACM Transactions on Software Engineering and Methodology (TOSEM)

Bierhoff, K., & Aldrich, J. (2007). Modular typestate checking of aliased objects (Vol. 42, No. 10). ACM.

DeLine, R., & Fähndrich, M. (2004). Typestates for objects. In ECOOP 2004–Object-Oriented Programming. Springer.

Sunshine, J., Naden, K., Stork, S., Aldrich, J., & Tanter, É. (2011, October). First-class state change in plaid. In ACM SIGPLAN Notices (Vol. 46, No. 10). ACM.

Khalid, S., Zehra, S., & Arif, F. (2010, October). Analysis of object oriented complexity and testability using object oriented design metrics. In Proceedings of the 2010 National Software Engineering Conference. ACM.

Basili, Victor R., Lionel C. Briand and Walcélio L. Melo, (1996) "A validation of object-oriented design metrics as quality indicators", Software Engineering, IEEE Transactions on 22, no. 10.

Li, W and Henry, S., (1993) "Maintenance metrics for the object oriented paradigm," Software Metrics Symposium, Proceedings, First International.

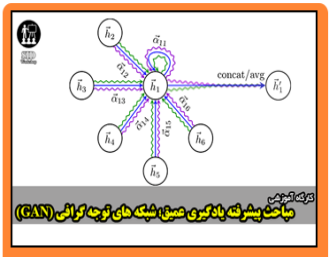
Jehad Al Dallal, (2013) "Object-oriented class maintainability prediction using internal quality attributes", Information and Software Technology 55, no. 11.

Dandashi Fatma, (2002) "A method for assessing the reusability of object-oriented code using a validated set of automated measurements", In Proceedings of the 2002 ACM symposium on applied computing ACM.

کارگاه های آموزشی مرکز اطلاعات علمی جهاد دانشگاهی

کارگاه آنلاین مکالمه روزمره انگلیسی

کارگاه آنلاین آموزش استفاده از وب آو ساینس



مباحث پیشرفته یادگیری عمیق؛ شبکه های توجه گرافی (Graph Attention Networks)